Discrete Methods

F. Oggier

These notes serve as teaching material for the course MAS 711, Discrete Methods, taught in the Division of Mathematical Sciences, Nanyang Technological University, Singapore.

My main reference has been notes by my colleague Bernhard Schmidt who taught this course before I did.

For Pólya's enumeration theorem, I have used the slides by Gordon Royle [12].

I have consulted the notes [9] for discussing Edmonds-Karp Algorithm, and the slides [3] for the min cost flow, and have followed closely the exposition by Amaury Pouly for the min cost flows [10].

For Floyd-Warshall algorithm, I have used the references [4, 1].

For linear programming, the references are [11, 7, 13, 5].

For the network simplex algorithm, I have followed the notes by Alessandro Agnetis [2].

For semi-definite programming, I have used the introduction by R. M. Freund [6], and the notes by A. Gupta and R. O'Donnell [8] on duality for semi-definite programming, which contain a proof for strong duality.

Some other references that I have consulted have been lost during a hard drive crash, I hope that I did not miss any significant one.

Pictures are taken from Wikipedia, for educational purpose.

I would like to thank Wong Dai Quan who typed in latex some of the notes on graph theory and in particular trees, and Ong Jenn Bing who typed in latex some of the notes on network flow and max flow min cut.

As for all my notes, I try to make them as bugfree as possible, but chances are that there might still be some bugs and typos.

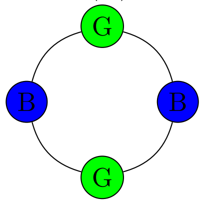
Contents

Chapter 1

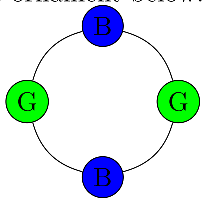
Pólya's Enumeration Theorem

1.1 Counting Necklaces

Consider a decorative ornament that consists of n coloured beads, arranged on a circular loop of strings. This can be represented as a word of length n over a suitable alphabet of colours. For example, for $n = 4$, and the colours blue (B) and green (G), we could have $GBGB$:



We could also label the above ornament $BGBG$, and similarly label $GBGB$ the ornament below:

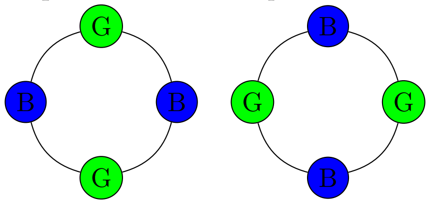


This is because the second ornament is drawn as a rotated version of the first ornament, but they are in fact the same ornament. The labels should thus correspond to the same word, and we say that two words that differ uniquely by a rotation of letters represent the same ornament, and they are called *equivalent*: $GBGB \equiv BGBG$. One may easily check that this indeed defines an equivalence relation (take the identity rotation for reflexivity, the reverse rotation for symmetry, and the combination of rotations for transitivity).

Definition 1.1. An (n, k)-*necklace* is an equivalence class of words of length n

over an alphabet of size k, under rotation.

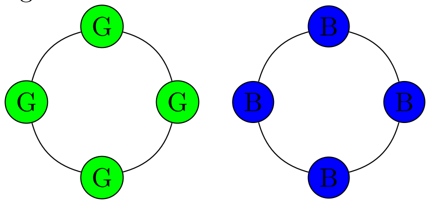
Example 1.1. The examples above form a $(4, 2)$-necklace:



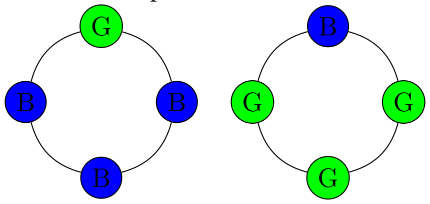
As usual when dealing with equivalent classes, one picks one representative per class.

Problem 1. **[Necklace Enumeration Problem]** Given n and k, how many (n, k)-necklaces are there?

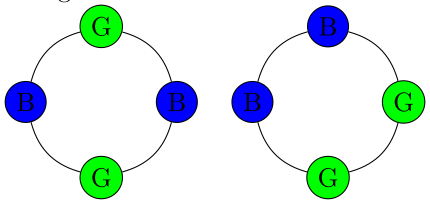
Example 1.2. Suppose $n = 4$ and $k = 2$ as above. Let us try to count how many necklaces with 4 beads and two colours there are. We have necklaces with a single colour: $BBBB$ and $GGGG$.



Then we have necklaces with only one bead B, and those with only one bead G, and their respective rotations which are not counted as different necklaces:



Then we have necklaces with exactly two beads of each colour, which could be contiguous or not.



This gives us a total of 6 necklaces.

This was easy by hand. Suppose now we have $n = 6$ beads, but $k = 5$ colours, what would be the answer? It is probably not advised to try this case by hand, since the claim is that there are 2635 such necklaces. We will develop next the tools to be able to prove this.

The problem involves two objects of different natures. The set X of words of length n over an alphabet of size k only has a set structure, it has size $|X| = k^n$. The set of n rotations (or rotations of angle $2\pi s/n$, $s = 0, \ldots, n-1$) has the structure of (is isomorphic to) the cyclic group of order n, which we denote by C_n.

How the group structure of rotations gives structure to the set X is formally captured by the notion of group action on a set.

Definition 1.2. A (left) group action φ of a group G on X is a function

$$\varphi : G \times X \to X,\ (g, x) \mapsto \varphi(g, x) = g * x$$

that satisfies:

- **identity.** $1 * x = x$ for all $x \in X$.
- **compatibility.** $(gh) * x = g * (h * x)$ for all $g, h \in G$, for all $x \in X$.

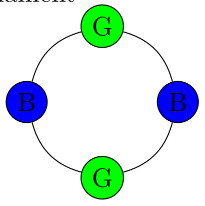
We can check here that our necklace setting does fit the framework of group action: 1 is the rotation that does not do anything on a word, so that the identity property is satisfied. As for compatibility, if we compose two rotations first, and apply the result on a word, it does give the same thing as applying the first rotation, and then applying the second one.

Definition 1.3. For a group G acting on a set X, the orbit $\mathrm{Orb}(x)$ of $x \in X$ is by definition

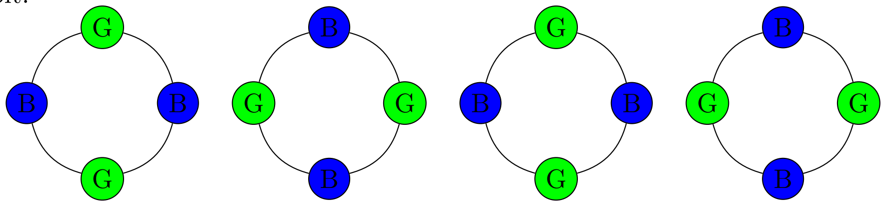
$$\mathrm{Orb}(x) = \{g * x,\ g \in G\}.$$

In terms of necklaces, $x \in X$ is a word of length n over an alphabet of size k, and g are rotations. An orbit for x is thus obtained by taking the chosen word x and applying on it all possible rotations in C_n.

Example 1.3. For $n = 4$ and $k = 2$, we have 4 rotations (by $\pi/2$, π, $3\pi/2$ and the identity), this is isomorphic to the cyclic group C_4. Then consider the ornament



and apply the 4 rotations, starting from the identity, to get the following orbit:



which thus consists of two distinct colourings.

The set of orbits is usually denoted by X/G:

$$X/G = \{\mathrm{Orb}(x),\ x \in X\}.$$

It is useful to notice that orbits partition X (and in that, the group action of G on X does tell us something about the structure of X).

Lemma 1.1. *Orbits under the action of the group G partition the set X.*

Proof. Firstly, the union of orbits is actually the whole of X:

$$\bigcup_{x\in X} \mathrm{Orb}(x) = X.$$

This is happening because since G is a group, it contains an identity element 1, so $1 * x \in \mathrm{Orb}(x)$, then the "identity" property of the group action implies that $x \in \mathrm{Orb}(x)$ for every orbit. Next two orbits $\mathrm{Orb}(x), \mathrm{Orb}(y)$ are either disjoint, or they are the same. Suppose that they are not disjoint, then there exists an element z that lives in both the orbits $\mathrm{Orb}(x)$ and $\mathrm{Orb}(y)$, then

$$z = g * x = g' * y \Rightarrow x = g^{-1}g' * y \in \mathrm{Orb}(y).$$

We note that we used twice group axioms, once to invert g, and once to say that $g^{-1}g' \in G$. We just showed that $x \in \mathrm{Orb}(y)$ and thus $\mathrm{Orb}(x) \subseteq \mathrm{Orb}(y)$. By repeating the same argument, we show the reverse inclusion. $\square$

Based on what we just defined, we can rephrase Problem 1:

Problem 2. [Necklace Enumeration Problem] Given n and k, how many orbits of X under the action of C_n are there?

At this point, we may wonder why it was worth the effort to take our counting necklace problem and translate it into a problem of counting orbits under a group action, which does not seem an easier formulation (at first). The point of the reformulation is the result called *Burnside Lemma* (even though it was not proven by Burnside, so other authors call it Cauchy Frobenius Lemma).

Proposition 1.2. [Burnside Lemma] *Let G be a finite group action on a set X. Then*

$$|X/G| = \frac{1}{|G|}\sum_{g\in G} |\mathrm{Fix}(g)|,\ \mathrm{Fix}(g) = \{x \in X,\ g * x = x\}.$$

Before giving a proof of this result, let us see how useful it is for our necklace problem.

Example 1.4. Suppose $n = 6$ and $k = 2$. We could of course follow the same approach as in Example 1.2, that is, list ornaments based on how many beads are of the same colour:

- $BBBBBB$ and $GGGGGG$,

Figure 1.1: William Burnside (1852-1927)

- $GBBBBB$ and $BGGGGG$,
- $GGBBBB$, $GBGBBB$, $GBBGBB$, and the same pattern with reversed colours, $BBGGGG$, $BGBGGG$, $BGGBGG$,
- $GGGBBB$, $GGBGBB$, $GGBBGB$, $GBGBGB$ (note that the reversed colours do not give anything new up to rotation).

This list looks ok, but how do we make sure we got the right list? A first simple observation is that for $n = 4$, we had only $2^4 = 16$ possible words to check, in this case, we would have $2^6 = 64$ possible words, if we want to check them all, then we need to make sure we remove all the words equivalent up to rotation. So let us try Burnside Lemma. The group action on X is C_6, it has a generator g, which in cycle notation is $g = (1, 2, 3, 4, 5, 6)$. Then

$$\begin{aligned}
g^2 &= (135)(246)\\
g^3 &= (14)(25)(36)\\
g^4 &= (153)(264)\\
g^5 &= (165432)\\
g^6 &= (1)(2)(3)(4)(5)(6)
\end{aligned}$$

and we need to compute $\text{Fix}(g^i)$ for each i, that is we want ornaments which are invariant under rotation by g^i. Now g fixes only 2 words, $BBBBBB$ and $GGGGGG$, so $|\text{Fix}(g)| = 2$. Then g^2 fixes words with the same color in position 1,3,5 and in position 2,4,6, these are $BBBBBB$, $GGGGGG$, $BGBGBG$ and $GBGBGB$ (yes, the last two are obtained by rotation of each other, but remember that there is also an average by the number of elements of the group in the final formula), so $|\text{Fix}(g^2)| = 4$. We observe in fact that within one cycle, all the beads have to be of the same color, thus what matters is the number of cycles. Once this observation is made, we can easily compute:

$$\begin{array}{rcl}
g = & (123456) & |\mathrm{Fix}(g)| = 2^1 \\
g^2 = & (135)(246) & |\mathrm{Fix}(g^2)| = 2^2 \\
g^3 = & (14)(25)(36) & |\mathrm{Fix}(g^3)| = 2^3 \\
g^4 = & (153)(264) & |\mathrm{Fix}(g^4)| = 2^2 \\
g^5 = & (165432) & |\mathrm{Fix}(g^5)| = 2^1 \\
g^6 = & (1)(2)(3)(4)(5)(6) & |\mathrm{Fix}(g^6)| = 2^6
\end{array}$$

and we see that the number of necklaces is

$$\frac{1}{6}(2 + 2^2 + 2^3 + 2^2 + 2 + 2^6) = 14.$$

This does not really tell whether our above list was correct, but this shows that we got the right number of necklaces.

The above example shows that the number k of colours does not play a role but for being the basis of the exponents, so for $n = 6$ beads in general, we have

$$\begin{array}{rcl}
g = & (123456) & |\mathrm{Fix}(g)| = k \\
g^2 = & (135)(246) & |\mathrm{Fix}(g^2)| = k^2 \\
g^3 = & (14)(25)(36) & |\mathrm{Fix}(g^3)| = k^3 \\
g^4 = & (153)(264) & |\mathrm{Fix}(g^4)| = k^2 \\
g^5 = & (165432) & |\mathrm{Fix}(g^5)| = k \\
g^6 = & (1)(2)(3)(4)(5)(6) & |\mathrm{Fix}(g^6)| = k^6
\end{array}$$

and we see that the number of necklaces is

$$\frac{1}{6}(2k + 2k^2 + k^3 + k^6).$$

We can thus set $k = 5$ in this formula to obtain the number of necklaces with 5 colours and 6 beads, if we want to give an answer to the question asked after Example 1.2.

This formula for counting necklaces with 6 beads already shows why the formulation in terms of group action was a good idea. Thanks to Burnside Lemma, it becomes easy to compute a quantity which grows pretty quickly as a function of k.

Now this formula supposes that $n = 6$, and we would like to have a general formula, that is a formula valid for an arbitrary n. We saw above that the number of words fixed by an element $g \in C_n$ is determined by the number of cycles in its cycle decomposition: if g has $c(g)$ cycles, then it fixes $k^{c(g)}$ words, and the number of (n, k)-necklaces is

$$\frac{1}{n}\sum_{g \in C_n} k^{c(g)}.$$

Let us thus try to understand $c(g^m)$ for g^m a rotation of angle $2\pi m/n$. Such a rotation, in cycle notation, will look like

$$(i, i+m, i+2m, \dots, i+(k_m-1)m)$$

where i is the first index of the cycle, and $k_m = n/\gcd(m,n)$. Indeed k_m must be such that $k_m m \equiv 0 \pmod n$, which is the case. But also, there cannot be a smaller k' such that $k'm \equiv 0 \pmod n$: we need to multiply m by an integer such that the product is $0 \pmod n$, and the smallest integer is obtained by multiplying m by the prime factors that are missing to m to obtain n, and only those, which is exactly what k_m does.

Now that we know that each cycle has length k_m (note that the reasoning does not depend on i), and since the union of all cycles must give n, we get that the number $c(g^m)$ of cycles in the decomposition of g^m is

$$\frac{n}{k_m} = \frac{n}{\frac{n}{\gcd(m,n)}} = \gcd(m,n).$$

We thus have an answer to our Problems 1 and 2 of Necklace Enumeration.

Theorem 1.3. *Given n, k two positive integers, the number of (n,k)-necklaces is*

$$\frac{1}{n}\sum_{i=1}^{n} k^{\gcd(n,i)}.$$

When $n = p$ is prime, this simplifies to

$$\frac{1}{p}((p-1)k + k^p).$$

Corollary 1.4. *Let ϕ be the Euler totient function. Then the number of (n,k)-necklaces can be alternatively written as*

$$\frac{1}{n}\sum_{d|n}\phi(d)k^{n/d}.$$

See Exercise 2 for a proof.

1.2 Pólya's Enumeration Theorem

We saw in the previous section that Burnside Lemma is a powerful tool, and in fact it was the key to solve the problem of counting (n,k)-necklaces. It deserves to spend some time to go through its proof, which furthermore contains a useful counting technique.

[**Burnside Lemma**] Let G be a finite group action on a set X. Then

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|\text{Fix}(g)|,\ \text{Fix}(g) = \{x \in X,\ g * x = x\}.$$

Proof. We have

$$\begin{aligned}
\sum_{g\in G}|\mathrm{Fix}(g)| &= \sum_{g\in G}|\{x\in X,\ g*x=x\}| \\
&= |\{(g,x)\in G\times X,\ g*x=x\}| \\
&= \sum_{x\in X}|\{g\in G,\ g*x=x\}| \\
&= \sum_{x\in X}\frac{|G|}{|\mathrm{Orb}(x)|},\ \text{this is a claim} \\
&= |G|\sum_{x\in X}\frac{1}{|\mathrm{Orb}(x)|} \\
&= |G|\sum_{A\in X/G}\sum_{x\in A}\frac{1}{|A|},\ \text{since } X=\bigcup_{A\in X/G}A \\
&= |G|\sum_{A\in X/G}1 \\
&= |G||X/G|.
\end{aligned}$$

Note that the first lines are a nice combinatorial trick of counting the same set in two different manners. Also the 6th equality uses Lemma 1.1. We are thus left to prove the claim. □

The set involved in the claim to be proven is called *stabilizer* of x:

$$\mathrm{Stab}(x)=\{g\in G,\ g*x=x\}.$$

We prove the claim separately, it is called the Orbit-Stabilizer Theorem.

Proposition 1.5. [Orbit-Stabilizer Theorem] *Let G be a finite group acting on a set X. Then*

$$|\mathrm{Stab}(x)|=\frac{|G|}{|\mathrm{Orb}(x)|}.$$

Proof. Fix $x\in X$, consider $\mathrm{Orb}(x)$, the orbit of x, which contains the elements $g_1*x,\dots,g_n*x$ for $G=\{g_1,\dots,g_n\}$. Look at g_1*x, and gather all the g_i*x such that $g_i*x=g_1*x$, and call A_1 the set that contains all the g_i. Do the same process with g_2*x (assuming g_2 is not already included in A_1), to obtain a set A_2, and iterate until all elements of G are considered. This creates m sets $A_1,\dots,A_m$, which are in fact equivalence classes for the equivalence relation $\sim$ defined on G by $g\sim h \iff g*x=h*x$. We have $m=|\mathrm{Orb}(x)|$, since there is a distinct equivalence class for each distinct $g*x$ in the orbit, and since $A_1,\dots,A_m$ partition G

$$|G|=\sum_{i=1}^m|A_i|.$$

Now $|A_i|=|\mathrm{Stab}(x)|$ for all i. Indeed, fix i and $g\in A_i$. Then

$$h\in A_i \iff g*x=h*x \iff x=g^{-1}h*x \iff g^{-1}h\in\mathrm{Stab}(x) \iff h\in g\mathrm{Stab}(x).$$

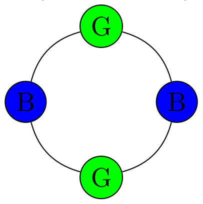
This shows that $|A_i| = |g\text{Stab}(x)| = |\text{Stab}(x)|$, the last equality being a consequence of g being invertible.

Thus

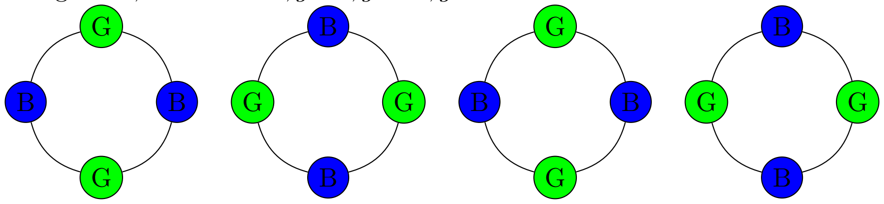
$$|G| = \sum_{i=1}^{m} |A_i| = m|\text{Stab}(x)| = |\text{Orb}(x)||\text{Stab}(x)| \Rightarrow |\text{Orb}(x)| = \frac{|G|}{|\text{Stab}(x)|}.$$

□

Example 1.5. We already saw an illustration of this theorem in Example 1.3. For $n = 4$ and $k = 2$, we considered the 4 rotations (by $\pi/2$, π, $3\pi/2$ and the identity, denoted by $g, g^2, g^3, g^4 = 1$). Then consider the ornament x



on which we apply the 4 rotations, starting from the identity, to get the following orbit, formed of $x, g * x, g^2 * x, g^3 * x$:



Then Stab(x) is given by g^2 and $g^4 = 1$, and $|\text{Stab}(x)| = 2 = \frac{|G|}{|\text{Orb}(x)|}$ since the orbit contains only 2 distinct colourings.

The same example can be used to illustrate the proof of the Orbit-Stabilizer Theorem. Let us look again at these 4 ornaments, given by $x, g*x, g^2*x, g^3*x$. Since x and $g^2 * x$ give the same colouring, group $1, g^2$ into a set A_1, and since $g * x$ and $g^3 * x$ give the same colouring, group g, g^3 into a set A_2. Then $|G| = |A_1| + |A_2|$. We also see that A_1 is actually the stabilizer of x, and that A_2 is $g\text{Stab}(x)$, thus $|A_1| = |A_2| = |\text{Stab}(x)|$, and the number of A_i is the number of distinct colourings in Orb(x), so $|G| = 2|\text{Stab}(x)| = |\text{Orb}(x)||\text{Stab}(x)|$.

If we look back at the problem of counting necklaces, we used the fact that necklaces can be seen as orbits under the action of a group of rotations, after which we used Burnside's lemma to count the number of orbits. So one can now imagine that the same principle could apply to other counting scenarios, where the group acting is not necessarily the cyclic group. But then, the reasoning on cycles remains the same, so one would have to capture the cycle decomposition of the group elements involved. This is captured by the notion of cycle index.

Definition 1.4. Given a permutation g on n elements, let $c_i(g)$ be the number of cycles of length i in its cycle decomposition. Then the *cycle index* of a permutation group is a polynomial that summarizes the information about the cycle types of all the elements of the group:

$$P_G(X_1,\dots,X_n)=\frac{1}{|G|}\sum_{g\in G}X_1^{c_1(g)}X_2^{c_2(g)}\cdots X_n^{c_n(g)}.$$

Analogies could be that of a weight enumerator for linear codes, or of theta series for lattices. Note that we consider group actions on words of length n, so since the action of g sends a word of length n to another word of length n, g can always be seen as a permutation on n elements (elements in a finite group can always be seen as permutations, remember Cayley Theorem).

Example 1.6. We continue Example 1.4, where $G = C_6$, and we have $n = 6$ beads. Then

$$\begin{array}{rcc}
g = & (123456) & c_6(g)=1\\
g^2 = & (135)(246) & c_3(g^2)=2\\
g^3 = & (14)(25)(36) & c_2(g^3)=3\\
g^4 = & (153)(264) & c_3(g^4)=2\\
g^5 = & (165432) & c_6(g^5)=1\\
g^6 = & (1)(2)(3)(4)(5)(6) & c_1(g^6)=6
\end{array}$$

and

$$P_G(X_1,\dots,X_6)=\frac{1}{6}(X_6+X_3^2+X_2^3+X_3^2+X_6+X_1^6)=\frac{1}{6}(2X_6+2X_3^2+X_2^3+X_1^6).$$

We may want to pay attention to the information contained in this polynomial. For example, $2X_6$ tells us that we have 2 patterns corresponding to a cycle of length 6 (one is associated to the cycle (123456) and one is associated to the cycle (165432)). Then $2X_3^2$ tells us that we have 2 patterns corresponding to 2 cycles of length 3 (one is associated to the cycle (135)(246), the other to the cycle (153)(264)).

Now let us add the number k of colors. For $2X_6$, we have 2 patterns, each can be of any of the k colours, so this counts $2k$ necklaces. For $2X_3^2$, we also have 2 patterns, but each pattern contains 2 cycles, and each cycle can take one colour, so the number of necklaces is $2k^2$. Iterating the process, we get that the number of $(6,k)$-necklaces is

$$P_G(k,\dots,k)=\frac{1}{6}(2k+2k^2+k^3+k^6),$$

as we already know.

The polynomial $P_G(X_1,\dots,X_n)$ does contain a lot of information, but the difficulty lies in actually finding it.

To state Pólya's Enumeration Theorem in a more general setting than counting coloured necklaces, we will consider D,C two finite sets, G a finite group acting on D, and we will let G act on C^D, the set of functions $f : D\to C$, by

$$(g*f)(d)=f(g^{-1}*d)$$

Figure 1.2: George Pólya (1887-1985). The popularity of the enumeration theorem is in particular due to its applications to chemistry (enumeration of chemical compounds).

(see Exercise 4 for a discussion on this action). The choice of the letters D, C corresponds to a colouring (C) of a domain (D), since a popular application of the theorem is to enumerate coloured objects. Functions $f : D \to C$ by definition of a function must assign a value $f(d) \in C$ for every $d \in D$, so it is the same thing as having $|D|$ slots, and for each slot, assigning a value from C, or said otherwise, we are looking at words of length $|D|$ with alphabet C. For (n,k)-necklaces, $D = \{1,\dots,n\}$ and $C = \{1,\dots,k\}$.

We assign a weight to each element $c \in C$, call it $w(c)$.

Definition 1.5. The weight $W(f)$ of a function $f \in C^D$ is the product

$$W(f) = \prod_{d\in D} w(f(d)).$$

We first notice that functions which belong to the same orbit under the action of G have the same weight, and for that reason, we call these orbits *patterns* (in the necklace setting, an orbit, or pattern, is a necklace). Indeed, suppose f_1, f_2 are in the same orbit under the action of G, that is, there exists $g \in G$ for which $f_2(d) = g * f_1(d)$ for all d:

$$\begin{aligned} W(f_2) &= \prod_{d\in D} w(f_2(d)) = \prod_{d\in D} w(g * f_1(d)) = \prod_{d\in D} w(f_1(g^{-1} * d)) \\ &= \prod_{d\in D} w(f_1(d)) = W(f_1). \end{aligned}$$

The second line equality is saying that when d runs through all values of D, so does $g^{-1} * d$ (if this was not the case, then $g^{-1} * d$ would not go through all possible values of D, and there would exist $d \neq d' \in D$ with $g^{-1} * d = g^{-1} * d'$ which is not possible because g^{-1} is invertible). Thus if F denotes a pattern (an orbit under the action of G on C^D), instead of considering weights $f \in F$, it is enough to consider the weight $W(F)$ of F, which is then $W(f)$ for any choice of f in F.

Example 1.7. Consider again the $(4,2)$-necklace problem of Example 1.2. We have the set of functions $f : D = \{1,2,3,4\} \to C = \{c_1, c_2\}$, with $|D| = 4$, and $|C| = 2$. Choose weights for $c \in C$, e.g.

$$w(c_1) = R, \; w(c_2) = B,$$

where the weights capture the property of colouring that is of interest in the necklace problem. Pick the function f_1, given by:

$$f_1(1) = c_2, \; f_1(2) = c_2, \; f_1(3) = c_1, \; f_1(4) = c_1,$$

then $w(f_1) = \prod_{i=1}^4 w(f_1(i)) = B^2R^2$. Then pick the function f_2 given by:

$$f_2(1) = c_1, \; f_2(2) = c_1, \; f_2(3) = c_2, \; f_2(4) = c_2,$$

with $w(f_2) = \prod_{i=1}^4 w(f_2(i)) = B^2R^2$. It has the same weights as f_1, and if we take g to be a rotation of 180 degrees clockwise, we get $g * f_1 = (c_1, c_1, c_2, c_2) = f_2$, thus f_1 and f_2 belong to the same pattern. Now pick the function f_3, given by

$$f_3(1) = c_2, \; f_3(2) = c_1, \; f_3(3) = c_2, \; f_3(4) = c_1,$$

with $w(f_3) = \prod_{i=1}^4 w(f_3(i)) = B^2R^2$, which is the same weight as that of f_1, f_2, however f_3 does not belong to the same pattern. To see this, notice that the permutation needed to send f_3 to f_1 cannot be obtained by rotation.

The example illustrates that there is one weight assigned to a pattern because every function in this orbit has the same weight, however several orbits (or patterns) could have the same weight. In a sense, this is saying that the weight is a coarser characterization of functions than patterns.

Theorem 1.6. **[Pólya's Enumeration Theorem]** *Let D, C be two finite sets, and let G be a finite group acting on C^D. We assign a weight $w(c)$ to each element $c \in C$. The patterns F have induced weights $W(F)$. Then the* pattern inventory *is*

$$\sum_F W(F) = P_G\left(\sum_{c\in C} w(c), \sum_{c\in C} w(c)^2, \sum_{c\in C} w(c)^3, \dots\right),$$

where P_G is the cycle index.

Corollary 1.7. *If all the weights are chosen to be equal to 1, then the number of patterns (or orbits of G on C^D) is given by $P_G(|C|, \dots, |C|)$.*

We first prove the corollary.

Proof. If all weights are equal to 1, then $W(F) = \prod_{d\in D} W(f(d)) = 1$ and $\sum_F W(F) = \sum_F 1$ just counts the number of patterns. Also $\sum_{c\in C} w(c)^i = \sum_{c\in C} 1 = |C|$ for $i \geq 1$. □

We next prove the theorem.

Proof. To evaluate $\sum_F W(F)$, we need to consider all possible weights ω of F, and for each ω, count how many F we have such that $W(F) = \omega$:

$$\sum_F W(F) = \sum_\omega \omega |\{F,\ W(F) = \omega\}|.$$

Counting $|\{F,\ W(F) = \omega\}|$ means counting the number of orbits F under the action of G, restricting to functions of given weight, that is we restrict the action of G on

$$S_\omega = \{f \in C^D, W(f) = \omega\}$$

and $|\{F,\ W(F) = \omega\}| = |S_\omega/G|$. Let us count how many patterns (or orbits) are in S_ω using Burnside Lemma:

$$|S_\omega/G| = \frac{1}{|G|}\sum_{g\in G} |\mathrm{Fix}_\omega(g)|,\ \mathrm{Fix}_\omega(g) = \{f \in C^D,\ W(f) = \omega,\ g * f = f\}.$$

Thus

$$\sum_F W(F) = \sum_\omega \omega \frac{1}{|G|}\sum_{g\in G} |\mathrm{Fix}_\omega(g)| = \frac{1}{|G|}\sum_{g\in G}\sum_\omega \omega |\mathrm{Fix}_\omega(g)|.$$

Since

$$\begin{aligned} \mathrm{Fix}(g) &= \{f \in C^D,\ g * f = f\} \\ &= \sqcup_\omega \{f \in C^D, W(f) = \omega,\ g * f = f\} \\ &= = \sqcup_\omega \mathrm{Fix}_\omega(g), \end{aligned}$$

the sum $\sum_\omega \omega |\mathrm{Fix}_\omega(g)|$ exactly captures all the weights that appear in $\mathrm{Fix}(g)$ for a given g, with their multiplicity.

But this can also be counted as follows: if $x \in \mathrm{Fix}(g)$, then by definition $g * f = f$ and the elements of a cycle of g must be given the same value c by f. A cycle of length i will contribute a factor $\sum_{c\in C} w(c)^i$, capturing that over each of the i terms, $w(c)$ must be the same, thus over the cycle of length i, we have $w(c)^i$, and any choice of $c \in C$ is possible thus $\sum_{c\in C} w(c)^i$. Recalling that $c_i(g)$ denotes the number of cycles of length i, we then have

$$\begin{aligned} \sum_F W(F) &= \frac{1}{|G|}\sum_{g\in G}\left(\sum_{c\in C} w(c)\right)^{c_1(g)} \cdots \left(\sum_{c\in C} w(c)^n\right)^{c_n(g)} \\ &= P_G\left(\sum_{c\in C} w(c), \sum_{c\in C} w(c)^2, \ldots, \sum_{c\in C} w(c)^n\right) \end{aligned}$$

where $n = |D|$. □

Example 1.8. We illustrate the proof using the example of $(4,2)$-necklaces. There are 2^4 possible such necklaces before considering their equivalence up to rotation. We list them by weight:

B^4	RB^3	R^2B^2	R^3B	R^4
$BBBB$	$RBBB$	$RRBB$	$RRRB$	$RRRR$
	$BRBB$	$RBRB$	$RRBR$	
	$BBRB$	$BRRB$	$RBRR$	
	$BBBR$	$RBBR$	$BRRR$	
		$BRBR$		
		$BBRR$		

What the proof of Pólya's Enumeration Theorem does, is to look at each column, they correspond to S_ω, for ω ranging from B^4 to R^4. Now looking at the action of $G = C_4$ on each column, we see that $|S_\omega/G| = 1$ except for $\omega = R^2B^2$, in which case $|S_\omega/G| = 2$. Thus we have 5 different weights, but 6 different patterns, since two patterns have the same weight. This illustrates:

$$\sum_F W(F) = \sum_\omega \omega |S_\omega/G| = B^4 + RB^3 + 2R^2B^2 + R^3B + R^4.$$

This can be computed differently, by looking at every weight in $\mathrm{Fix}(g)$. For a given g, if $x \in \mathrm{Fix}(g)$, it must be that the weights are the same for every elements of a cycle of g:

- for a cycle of length 1, the possible weights are $R + B$,
- for a cycle of length 2, the possible weights are $R^2 + B^2$,
- for a cycle of length 4, the possible weights are $R^4 + B^4$.

Let us thus look at the possible group elements, and their cycle information ($c_i(g)$ is the number of cycles of length i in the cycle decomposition of g):

1	$c_1(1) = 4$	$(R+B)^4$
g	$c_4(g) = 1$	$R^4 + B^4$
g^2	$c_2(g) = 2$	$(R^2+B^2)^2 = R^4 + 2R^2B^2 + B^4$
g^3	$c_4(g) = 1$	$R^4 + B^4$

- For 1, we have 4 cycles of length 1, thus $(R+B)^4$. Also, $(R+B)^4 = (R^2+2RB+B^2)^2 = R^4 + 4R^3B + 6R^2B^2 + 4RB^3 + B^4$ gives all possible weights of length 4 involving 2 colours.
- For g and g^3, we have 1 cycle of length 4, thus $(R^4 + B^4)$. This means that g and g^3 fix only $RRRR$ and $BBBB$.
- For g^2, we have two cycles of length 2, thus $(R^2+B^2)^2$: g^2 does fix $RRRR$ and $BBBB$ but also 2 patterns of weight R^2B^2.

If we sum over all $g \in G$ their corresponding weight, we get:

$$(R^4 + 4R^3B + 6R^2B^2 + 4RB^3 + B^4) + 2(R^4 + B^4) + (R^4 + 2R^2B^2 + B^4)$$

which simplifies to

$$4R^4 + 4R^3B + 8R^2B^2 + 4RB^3 + 4B^4$$

as expected.

Let us next just apply the theorem. The cycle index is

$$P_G(X_1, \dots, X_4) = \frac{1}{|G|}\sum_{g \in G} X_1^{c_1(g)} \cdots X_1^{c_4(g)} = \frac{1}{|G|}(X_1^4 + 2X_4 + X_2^2).$$

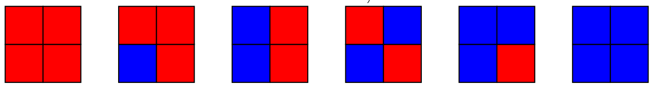
We have two colours R and B, then $\sum_c w(c)^i = R^i + B^i$, and we just need to evaluate

$$\begin{aligned} & \frac{1}{4}P_G(B+R, B^2+R^2, B^3+R^3, B^4+R^4) \\ = & \frac{1}{|G|}((B+R)^4 + 2(B^4+R^4) + (B^2+R^2)^2) \\ = & R^4 + R^3B + 2R^2B^2 + RB^3 + B^4. \end{aligned}$$

This tells us for example that there are two possible necklaces with two beads of each colour.

Let us see if we can use Pólya's Theorem for counting something else than (n, k)-necklaces.

Example 1.9. Consider an $n \times n$ chessboard, $n \geq 2$, where every square is either colored by blue (B) or red (R). How many different colourings are there, if different means that one cannot obtain a colouring from another by either a rotation or a reflection? For $n = 2$, we have



We first to observe that:

- There are 4 rotations, r the rotation by 90 degrees clockwise, r^2 the rotation by 180 clockwise, r^3 the rotation by 270 degrees clockwise, and r^4 the identity.
- There are 4 reflections: one with respect to the vertical axis, one with respect to the horizontal axis, and two with respect to each of the two diagonals. Let m be the reflection with respect to the horizontal axis. Note that rm, that is applying first m and then r, gives a reflection with respect to the left diagonal, that r^3m gives a reflection with respect to the right diagonal, and that r^2m gives a reflection with respect to the vertical axis.

- The 8 reflections and rotations are thus given by

$$\{1, r, r^2, r^3, m, rm, r^2m, r^3m\}.$$

In order to apply Pólya's enumeration Theorem, we need the action of a group G, for which we need to show that $G = \{1, r, r^2, r^3, m, rm, r^2m, r^3m\}$ is a group (see Exercise 5). We then need to compute $\text{Fix}(g)$ for each $g \in G$, for which we write the cycle decomposition. Note that we label the 4 squares of the 2×2 chessboard clockwise:

1	2
4	3

so that a permutation by r is of the form (1234). We get

g		$\lvert\text{Fix}(g)\rvert$
1	(1)(2)(3)(4)	2^4
r	(1234)	2
r^2	(13)(24)	2^2
r^3	(1432)	2
m	(14)(23)	2^2
r^2m	(12)(34)	2^2
rm	(24)(1)(3)	2^3
r^3m	(13)(2)(4)	2^3

Using Burnside lemma, we get:

$$\frac{1}{8}(2^4 + 2 + 2^2 + 2 + 2^2 + 2^2 + 2^3 + 2^3) = \frac{48}{6} = 6.$$

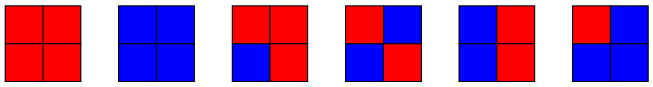
We can compute the cycle index:

$$P_G(X_1, X_2, X_3, X_4) = \frac{1}{8}(X_1^4 + 2X_1^2X_2 + 3X_2^2 + 2X_4)$$

which evaluated in $X_1 = X_2 = X_3 = X_4 = 2$ gives 6 as expected. Now using Pólya's enumeration Theorem:

$$\begin{aligned}
& P_G((R+B), (R^2+B^2), (R^3+B^3), (R^4+B^4)) \\
= \; & \frac{1}{8}((R+B)^4 + 2(R+B)^2(R^2+B^2) + 3(R^2+B^2)^2 + 2(R^4+B^4)) \\
= \; & \frac{1}{8}(R^4 + 4R^3B + 6R^2B^2 + 4RB^3 + B^4) + \\
& \frac{1}{8}(2R^4 + 2R^2B^2 + 4R^3B + 4RB^3 + 2B^2R^2 + 2B^4) + \\
& \frac{1}{8}(3R^4 + 3B^4 + 6R^2B^2 + 2R^4 + 2B^4) \\
= \; & R^4 + B^4 + R^3B + 2R^2B^2 + RB^3.
\end{aligned}$$

This enumerates the 6 colourings of the chessboard:



The case $n = 2$ could be done by hand, this is much harder in general (see Exercise 6).

1.3 Exercises

Exercise 1. Compute the number of $(6, 3)$-necklaces, that is the number of necklaces with 6 beads and 3 colours.

Exercise 2. Prove that

$$\frac{1}{n}\sum_{d|n}\phi(d)k^{n/d}$$

counts the number of (n, k)-necklaces, where ϕ is Euler totient function.

Exercise 3. Compute the cycle index $P_{C_n}(X_1, \ldots, X_n)$.

Exercise 4. Let D, C be two finite sets, let G be a group acting on D, and let C^D be the set of functions $f : D \to C$. Show that

$$(g * f)(d) = f(g^{-1} * d)$$

is indeed a group action on C^D.

Exercise 5. Show that

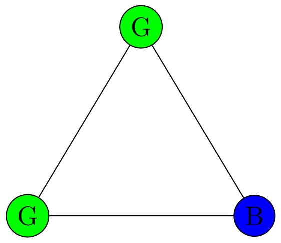
$$\{1, r, r^2, r^3, m, rm, r^2m, r^3m\}$$

form a group with respect to composition of maps, where r is a rotation by 90 degrees (clockwise) and m is a reflection through the horizontal axis.

Exercise 6. Consider an $n \times n$ chessboard, $n \geq 2$, where every square is either colored by blue (B) or red (R). How many different colorings are there, if different means that one cannot obtain a coloring from another by either a rotation (by either 90, 180 or 270 degrees) or a reflection (along the vertical and horizontal axes, and the 2 diagonals)?

Exercise 7. Consider $(4, 3)$-necklaces, that is the number of necklaces with 4 beads and 3 colours, say blue (B), green (G) and red (R) . Use Pólya's Enumeration Theorem to list the different necklaces involving at least two blue beads.

Exercise 8. Consider an equilateral triangle whose vertices are coloured, they can be either blue or green. Here is an example of colouring:



Two colourings of the vertices are considered equivalent if one can be obtained from another via a rotation or reflection of the triangle.

1. List all the rotation(s) and reflection(s) of the equilateral triangle and argue they form a group.
2. Compute the cycle index polynomial for the group of rotations and reflections of the equilateral triangle.
3. Use Polya's Enumeration Theorem to list the different colourings using two colours of the equilateral triangle.

We know that a (n, k)-necklace is an equivalence class of words of length n over an alphabet size k, under rotation. Consider now an (n, k)-bracelet, that is an equivalence class of words of length n over an alphabet of size k, under both rotation (as necklaces), but also under reversal, which means for example that the bracelet $ABCD$ is equivalent to the bracelet $DCBA$: $ABCD \equiv DCBA$.

Exercise 9. We know that a (n, k)-necklace is an equivalence class of words of length n over an alphabet size k, under rotation. Consider now an (n, k)-bracelet, that is an equivalence class of words of length n over an alphabet of size k, under both rotation (as necklaces), but also under reversal, which means for example that the bracelet $ABCD$ is equivalent to the bracelet $DCBA$: $ABCD \equiv DCBA$.

1. For (4,2)-necklaces, they are orbits under the action of the group of rotations by $(2\pi/4)k, k = 0, 1, 2, 3$. For (4,2)-bracelets, they are also orbits under the action of some group G. What is this group G? List its elements.
2. Compute the cycle index polynomial for the group G.
3. Use Polya's Enumeration Theorem to list the different (4,2)-bracelets.

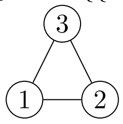
Chapter 2

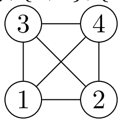
Basic Graph Theory

2.1 Some Basic Definitions

A *graph* $G = (V, E)$ consists of a set V of *vertices* (or *nodes*), and a set E of *edges*. Formally E is a set containing 2-subsets of V, and for u, v two vertices of V, an edge between u and v is denoted by $\{u, v\}$.

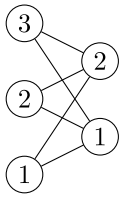
Example 2.1. The graph $K_n = (\{1, \dots, n\}, \{\{i, j\},\ 1 \le i < j \le n\})$ is called a *complete graph*. The term "complete" captures the fact that every vertex is connected to every other vertex. For $n = 3$, K_3 has $\{1, 2, 3\}$ for vertices, and $\{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$ for edges. For K_4, the vertices are $\{1, 2, 3, 4\}$ and the edges are $\{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$:





Definition 2.1. A graph $G = (V, E)$ is called *bipartite* if $V = A \sqcup B$ and every edge has one vertex in A and one vertex in B.

Example 2.2. Consider the graph $K_{m,n} = (V, E)$ with $V = A \sqcup B$, $|A| = n$, $|B| = m$, and $E = \{\{a, b\},\ a \in A,\ b \in B\}$, which is called a *complete bipartite graph*. For example, for $n = 2$ and $m = 3$, we have $K_{2,3}$:



This graph is bipartite, because $V = A \sqcup B$, and every edge has one vertex in A and one vertex in B. It is called complete bipartite because every edge allowed to exist by the definition of bipartite is present.

An edge $\{i, j\}$ is undirected. This is captured by the set notation, which means there is no ordering on i, j. In a *directed graph (digraph)*, edges are ordered 2-tuples, not 2-subsets, and we write (i, j):

$$\begin{tikzcd} \boxed{i} \arrow[r, dash] & \boxed{j} & \boxed{i} \arrow[r] & \boxed{j} \end{tikzcd}$$

It is however possible to encounter the notation (i, j) for an undirected graph, if it is written that G is undirected, and then the notation (i, j) is used, it is to be understood as a pair where the ordering however does not matter.

Two vertices a, b are *adjacent* if $\{a, b\}$ (or (a, b)) is an edge.

Definition 2.2. Let G be a graph with n vertices, say $V = \{1, \ldots, n\}$. An *adjacency matrix* A of G is an $n \times n$ matrix defined by $A_{ij} = 1$ if i and j are adjacent, and 0 otherwise.

Examples 2.3. An adjacency matrix of K_3 is

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

An adjacency matrix of $K_{2,3}$ is

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

We remark that an undirected graph has a symmetric adjacency matrix.

Definition 2.3. The *degree* $\deg(v)$ of a vertex v of a graph G is the number of vertices adjacent to v.

For directed graphs, we can define similarly the in-degree and out-degree.

Definition 2.4. A graph is called *k-regular* if all its vertices have degree k.

Example 2.4. The following famous graph, called *Petersen graph*, is 3-regular.

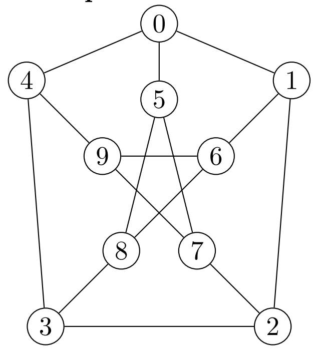


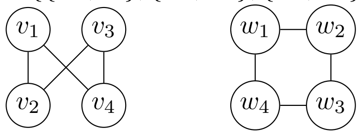
Figure 2.1: Julius Petersen (1839-1910), a Danish mathematician whose contributions to mathematics led to the development of graph theory.

2.2 Graph Isomorphisms and Automorphisms

Definition 2.5. Let $G = (V, E)$ and $G' = (V', E')$ be graphs. An *isomorphism* between G and G' is a bijection $\alpha : V \to V'$ such that $\{u, v\} \in E \iff \{\alpha(u), \alpha(v)\} \in E'$, for all $\{u, v\} \in E$.

In words, an isomorphism of graphs is a bijection between the vertex sets of two graphs, which preserves adjacency. An isomomorphism between G and itself is called an *automorphism*. Automorphisms of a given graph G form a group $\mathrm{Aut}(G)$ (see Exercise 10).

Example 2.5. Consider the following two graphs: $G = (V, E)$, with vertices $V = \{v_1, v_2, v_3, v_4\}$ and edges $E = \{\{v_1, v_2\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_3, v_4\}\}$, and another graph $G' = (V', E')$, with vertices $V' = \{w_1, w_2, w_3, w_4\}$ and edges $E' = \{\{w_1, v_2\}, \{w_1, w_3\}, \{w_3, w_4\}, \{w_1, w_4\}\}$:



These two graphs are isomorphic. To check this, we first establish the bijection α between vertices.

$$\alpha : v_1 \mapsto w_1,\ v_4 \mapsto w_2,\ v_2 \mapsto w_4,\ v_3 \mapsto w_3.$$

Now let us apply this bijection on every edge of G:

$$\begin{aligned}\{\alpha(v_1), \alpha(v_2)\} = \{w_1, w_4\},\ \{\alpha(v_1), \alpha(v_4)\} = \{w_1, w_2\},\\ \{\alpha(v_2), \alpha(v_3)\} = \{w_4, w_3\},\ \{\alpha(v_3), \alpha(v_4)\} = \{w_3, w_2\}\end{aligned}$$

which gives the 4 edges of E' as desired.

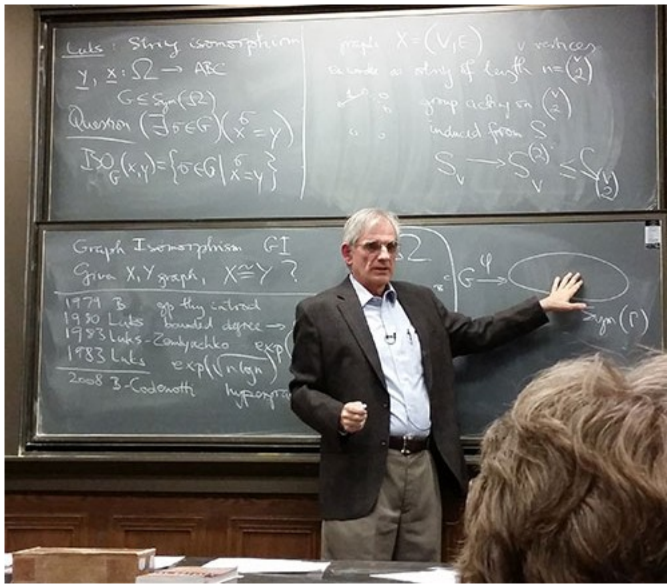
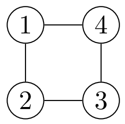


Figure 2.2: László Babai (born in 1950), a Hungarian professor of computer science and mathematics at the University of Chicago, claimed on January 9 2017 to have a quasi-polynomial algorithm with running time $2^{O((\log n)^c)}$, for n the number of vertices and $c > 0$ a fixed constant.

This looks like a pretty tedious procedure, one can imagine that once the number of vertices and edges grows, this may become hard to check. As it turns out, the problem of determining whether two graphs are isomorphic is hard. To make the term “hard” more precise, as of 2017, the question “can the graph isomorphism problem be solved in polynomial time?” is still open.

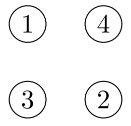
Example 2.6. Compute the automorphism group Aut(G) of the graph G given by:



Let us try a naive approach, by just applying the definition. We first need a bijection $\alpha : G \to G$, which is thus a permutation of the vertices. There are $4! = 24$ permutations, so we can still list them. There are many ways to list them systematically. We list them by looking at what happens with position 1. The first column in the table below looks at the cases where 1 is not permuted, and thus remains at position 1. The second column looks at the cases where 2 is in position 1, etc. The idea is that once we know what happens with position 1, there are only 6 cases left for the permutations of the other 3 elements:

1234	2134	3124	4123
1324	2314	3214	4213
1423	2413	3142	4312
1243	2143	3412	4132
1342	2341	3241	4231
1432	2431	3421	4321

For example, the second row 1324 of the first column means that we are looking at the labelling



Now the question is, which of these permutations are preserving adjacency? We have edges $E = \{\{1,2\},\{1,4\},\{2,3\},\{3,4\}\}$, so let us try out 1324. In this case, $1 \mapsto 1$, $2 \mapsto 3$, $3 \mapsto 2$, $4 \mapsto 4$, we get $\{\{1,3\},\{1,4\},\{3,2\},\{2,4\}\}$, and so this is not preserving adjacency, since $\{1,3\}$ is not an edge. Said differently, this permutation keeps 1 where it is, 1 is connected to 2 but not to 3, so switching the roles of 2 and 3 is not consistent with the adjacency structure.

So let us look at the first column, and see which permutation is an automorphism. The identity 1234 is obviously one. Now once 1 is fixed, since 1 is connected to both 2 and 4, we can switch these 2. But then we have no choice, 3 remains where it is, and it gives 1432 as the other automorphism of the first column.

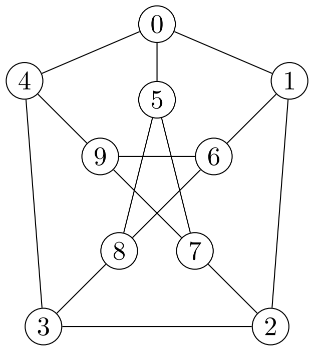
Now that we get a grip on what is going on, let us look at the second column. If the vertices 2 and 1 are switched, then we have no choice than switching 3 and 4, this is 2143. Then when 2 goes in position 1, apart switching 1 and 2, the other scenario that can happen is that 1 goes to 4, in which case, there is no choice either, 4 goes to 3, and we have the permutation 2341.

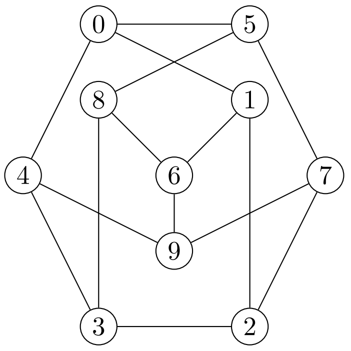
Applying the same reasoning to the other two columns gives the following list of group automorphisms: $1234, 1432, 2143, 2341, 3214, 3412, 4123, 4321$.

Since we know that automorphisms of G form a group, one may wonder what group it is in the case of the above example. It turns out to be a dihedral group (see Exercise 12).

It is thus quite tempting to connect $\mathrm{Aut}(G)$ with geometric symmetries (rotations, reflections). However it is dangerous to rely on graph representations when working with graphs: sometimes visualizing a graph does help, but sometimes, it can do the other way round.

Example 2.7. The following two graphs (one being the Petersen graph already encountered in Example 2.4) can be shown to be isomorphic (see Exercise 13), though this is not obvious.





On the left hand side graph, we "see" a rotation of the vertices, written in cycle notation as (01234)(56789), but while this is indeed an automorphism of G, it cannot be seen on the right hand side graph, which is the same graph up to a different way of drawing it.

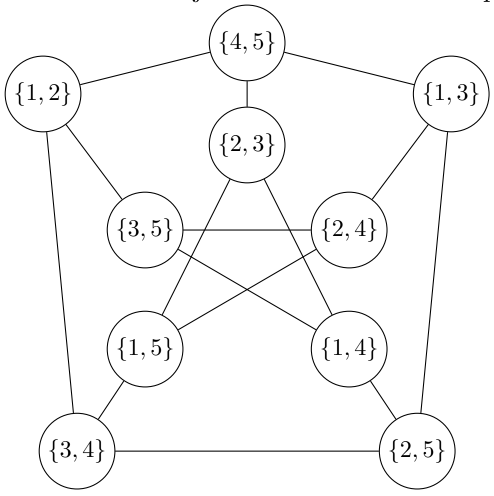
Next, let us compute the automorphism group $\mathrm{Aut}(G)$ for G the Petersen graph. To do so, let us construct a labelling of each of its $n = 10$ vertices, where each label is a pair $\{x, y\}$ with $x, y \in \{1, 2, 3, 4, 5\}$. Note that if we choose any 2 distinct numbers out of $\{1, 2, 3, 4, 5\}$, we do have 10 choices:

$$\begin{aligned}&12, 13, 14, 15\\&23, 24, 25\\&34, 35\\&45.\end{aligned}$$

Now we want this labelling to follow the rule:

$$\{x, y\} \sim \{z, w\} \iff \{x, y\} \cap \{z, w\} = \varnothing$$

where $\sim$ means "adjacent". Here is an example of such a labelling:



Now any such a labelling describes exactly the Petersen graph: the adjacency structure of the graph is fully characterized by the rule $\{x,y\} \sim \{z,w\} \iff \{x,y\} \cap \{z,w\} = \varnothing$, or in other words, if one takes 10 nodes, labels them with the 10 pairs $12, 13, 14, 15, 23, 24, 25, 34, 35, 45$, and draws edges according to the intersection rule above, this will give the Petersen graph (a 3-regular graph on 10 vertices, because given xy, it will be connected to exactly 3 other vertices).

Let σ be a permutation on 5 elements ($\sigma \in S_5$, where the symmetric group S_n is by definition the group of all permutations on n elements). We have

$$\{x,y\} \cap \{z,w\} = \varnothing \iff \{\sigma(x),\sigma(y)\} \cap \{\sigma(z),\sigma(w)\} = \varnothing$$

because if this were not true, then there would be an element in the intersection, say $\sigma(x) = \sigma(z)$, but then, since σ is invertible, this would mean that $x = z$, a contradiction.

Thus every $\sigma \in S_5$ gives a valid automorphism of G, and we just showed that $S_5 \subseteq \mathrm{Aut}(G)$ and in particular $|S_5| = 5! = 120 \leq |\mathrm{Aut}(G)|$.

We will show next that $|\mathrm{Aut}(G)| = 120$, which in turn will prove that

$$\boxed{\mathrm{Aut}(G) \simeq S_5 \text{ for } G \text{ the Petersen graph.}}$$

Step 1. Consider the vertex $v_1 = \{1,2\}$. The Orbit-Stabilizer Theorem (see Proposition 1.5) tells us that when $\mathrm{Aut}(G)$ acts on v_1, then

$$|\mathrm{Aut}(G)| = |\mathrm{Orb}(v_1)||\mathrm{Stab}_{\mathrm{Aut}(G)}(v_1)| = 10|\mathrm{Stab}_{\mathrm{Aut}(G)}(v_1)|.$$

We know that $|\mathrm{Orb}(v_1)| = 10$ because $S_5 \subseteq \mathrm{Aut}(G)$, and when we apply every permutation on $\{1,2\}$, we end up getting every one of the 10 possible labels.

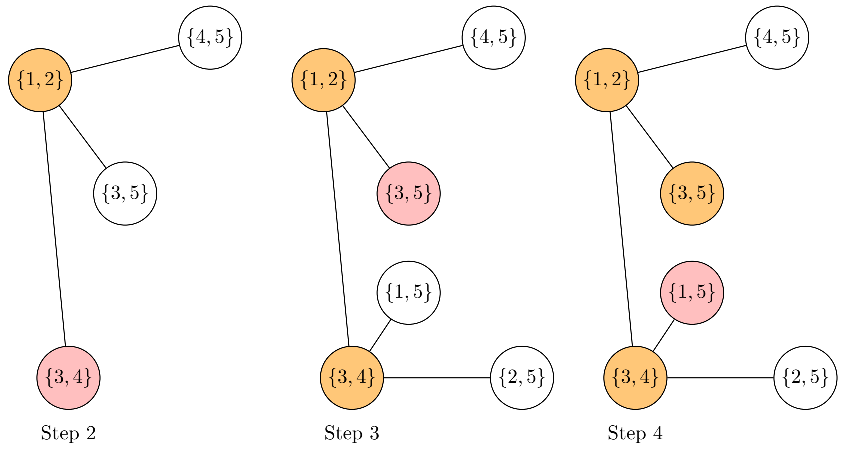
Remark. We can also repeat the Orbit-Stabilizer Theorem computation when S_5 acts on v_1. In that case, the orbit is the same and $|\mathrm{Orb}(v_1)| = 10$. For computing $\mathrm{Stab}_{S_5}(v_1)$, we need to count the permutations that are sending v_1 to itself, and since v_1 is connected to $\{4,5\}$,$\{3,4\}$ and $\{3,5\}$, there are $3! = 6$ permutations that permute $3,4,5$ without touching $1,2$, and then either $1 \mapsto 1, 2 \mapsto 2$, or $1 \mapsto 2, 2 \mapsto 1$, for a total of 12 permutations, and as desired $12 \cdot 10 = 120$.

Step 2. Consider next the vertex $v_2 = \{3,4\}$, which is adjacent to $v_1 = \{1,2\}$. In order to compute $|\mathrm{Stab}_{\mathrm{Aut}(G)}(v_1)|$, we notice that this is subgroup of $\mathrm{Aut}(G)$, and so now, we can just look at the action of this subgroup on v_2, and invoke again the Orbit-Stabilizer Theorem:

$$|\mathrm{Stab}_{\mathrm{Aut}(G)}(v_1)| = |\mathrm{Orb}(v_2)||\mathrm{Stab}_{\mathrm{Stab}_{\mathrm{Aut}}(G)(v_1)}(v_2)|.$$

To compute $|\mathrm{Orb}(v_2)|$, we look only at automorphisms that are fixing v_1, and then apply them on v_2. But if an automorphism is fixing $v_1 = \{1,2\}$, since v_1 is connected to $\{4,5\}$,$\{3,4\}$ and $\{3,5\}$, it has no choice than to permute them, so $|\mathrm{Orb}(v_2)| = 3$, and

$$|\mathrm{Aut}(G)| = 10|\mathrm{Stab}_{\mathrm{Aut}(G)}(v_1)| = 30|\mathrm{Stab}_{\mathrm{Stab}_{\mathrm{Aut}}(G)(v_1)}(v_2)|.$$

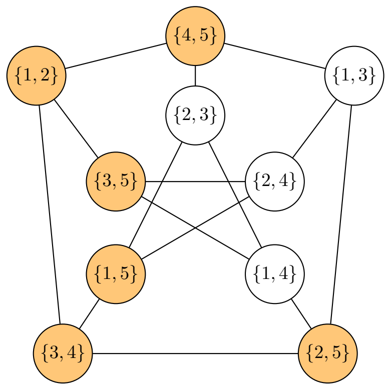


Step 3. We repeat the process once more. To compute $|\mathrm{Stab}_{\mathrm{Stab}_{\mathrm{Aut}(G)(v_1)}}(v_2)|$, we take a 3rd vertex $v_3 = \{3, 5\}$, and let the group $\mathrm{Stab}_{\mathrm{Stab}_{\mathrm{Aut}(G)(v_1)}}(v_2)$ act on it. The notation is not very friendly, but it just says that among the automorphisms that are fixing $v_1 = \{1, 2\}$, we restrict to those fixing $v_2 = \{3, 4\}$, and then we see how they act on $v_3 = \{3, 5\}$. Since $\{4, 5\}$,$\{3, 4\}$ and $\{3, 5\}$ could be permuted, but now, we further fix $\{3, 4\}$, the only choice left is to permute $\{4, 5\}$ and $\{3, 5\}$, so the orbit of v_3 is now of size 2.

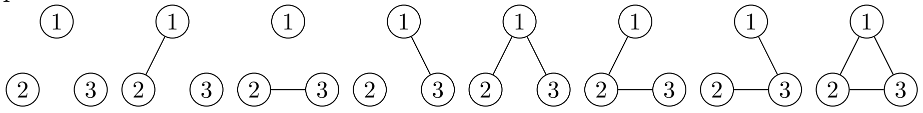
Step 4. We repeat the process one last time with $v_4 = \{1, 5\}$. Once v_1, v_2, v_3 are fixed, the orbit of v_4 contains only 2 elements, v_4 and $\{2, 5\}$, which give

$$|\mathrm{Aut}(G)| = 30 \cdot 4 = 120.$$

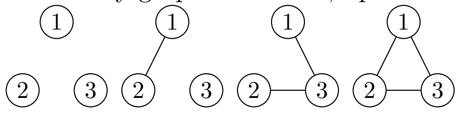
The process is finished, because: by now, $\{1, 2\}$, $\{3, 5\}$, $\{3, 4\}$ are fixed. Then $\{4, 5\}$ is fixed because it is connected to $\{1, 2\}$ whose 2 other adjacent nodes are fixed. Then we should be looking at automorphisms further fixing $\{1, 5\}$, which means that $\{2, 5\}$ is fixed, since it is connected to $\{3, 4\}$, whose 2 other adjacent nodes are fixed. Now $\{1, 4\}$ is connected to both $\{3, 5\}$ and $\{2, 5\}$, both of them being fixed, so $\{1, 4\}$ can only be permuted with another vertex also connected to both $\{3, 5\}$ and $\{2, 5\}$, so $\{1, 4\}$ is fixed and using the same argument, so are the other vertices.



We are next interested in counting isomorphism classes of graphs. Consider graphs with 3 vertices:

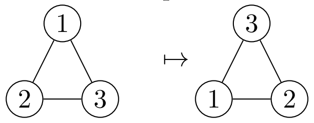


How many graphs are there, up to isomorphism? We have 4 of them:



We have an action of S_3 (the group of permutations on 3 elements) which permutes the edges, but note that permutations of vertices and edges are tied up in the sense that a permutation of vertices induces one on edges. Let us make this more precise on the above example.

Consider the permutation (123) of vertices:

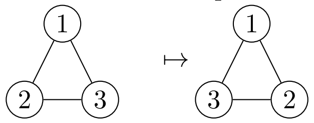


This permutation of vertices induces a permutation on the edges, because

$$\{1,2\} \mapsto \{2,3\},\ \{2,3\} \mapsto \{1,3\},\ \{1,3\} \mapsto \{1,2\}.$$

So the 3 edges are mapped to each other by a cycle of length 3. If we think in terms of the cycle index (see Definition 1.4), this corresponds to X_3 (one cycle of length 3).

Let us do the computation once more with the permutation (23) of vertices:



This permutation of vertices induces a permutation on the edges:

$$\{1,2\} \mapsto \{1,3\},\ \{1,3\} \mapsto \{1,2\},\ \{2,3\} \mapsto \{3,2\}.$$

So one edge is mapped to itself, and two edges are swapped by a cycle of length 2. If we think in terms of the cycle index, this corresponds to X_1X_2 (one cycle of length 1, one cycle of length 2).

Let us thus summarize the permutations in S_3, and the corresponding cycle index in terms of induced permutations on the edges:

S_3	$P_{S_3}(X_1, X_2, X_2)$
(1)(2)(3)	X_1^3
(1)(23)	X_1X_2
(2)(13)	X_1X_2
(3)(12)	X_1X_2
(123)	X_3
(132)	X_3

The cycle index is thus

$$P_{S_3}(X_1, X_2, X_2) = \frac{1}{6}(X_1^3 + 3X_1X_2 + 2X_3).$$

Now if an edge does not exist, give 1 as its weight, and if an edge does exist, give E as its weight. Using Pólya's Enumeration Theorem:

$$P_{S_3}(1+E, 1+E^2, 1+E^3) = \frac{1}{6}((1+E)^3+3(1+E)(1+E^2)+2(1+E^3)) = E^3+E^2+E+1.$$

This tells us that the number of graphs on 3 vertices up to isomorphism is 1 if there are 3 edges (E^3), 1 if there are 2 edges (E^2), 1 if there is one edge (E) and 1 if there is none (1).

The computations can be repeated for $n = 4$ vertices (see Exercise 16). The case $n = 4$ is arguably more interesting, because for $n = 3$, we work with 3 nodes and 3 edges, and not only the number of vertices and edges are the same, but the permutations induced on the edges are also matching those on the vertices. For $n = 4$, we have a different number of vertices (4) and edges (6), and the permutations induced on the edges are also different from those on the vertices. One could obviously consider all possible edge permutations, but this would be a lot of extra work: first, we would need to consider 6! permutations, and then we would have to anyway remove those which are not giving out a graph isomorphism, so it is less work to look at the permutations of vertices, and how they induce permutations on the edges.

2.3 Trees

We start with a series of definitions, that will allow us to define formally what is a tree.

Definition 2.6. A *walk* in a graph is a sequence $(v_1, \ldots, v_n)$ of vertices such that $\{v_i, v_{i+1}\}$ is an edge, for each $i = 1, \ldots, n-1$.

Definition 2.7. A *trail* is a walk where none of the edges occurs twice.

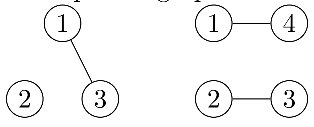
Definition 2.8. A *cycle* is a trail $(v_1, \ldots, v_n)$ such that $v_1 = v_n$.

Definition 2.9. A *path* is a trail where all vertices are distinct.

Definition 2.10. The *distance* between two vertices u and v, denoted by $d(u, v)$, is the length of the shortest path between them. If no such a path exists, then their distance is defined to be infinite, i.e $d(u, v) := \infty$.

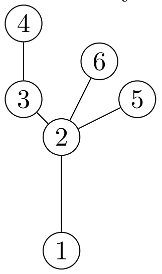
Definition 2.11. A graph is *connected* if $d(u, v) < \infty$ for all $u, v \in V$.

Examples of graphs which are not connected are:

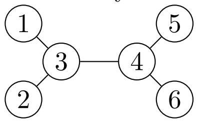


Definition 2.12. A *tree* is a connected graph that contains no cycle.

A tree may look the way we expect a tree to be, e.g

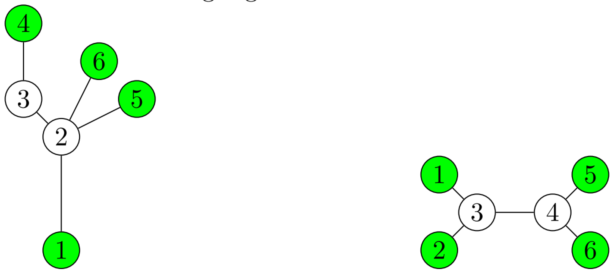


but it may also look different:



Definition 2.13. A *leaf* is a vertex of degree 1.

The leaves are highlighted in the trees below:



We will next prove a series of lemmas, the goal is to get intermediate results to help us prove equivalent definitions of trees.

Lemma 2.1. *Every finite tree with at least two vertices has at least two leaves.*

Proof. Take a finite tree with at least two vertices. Since a tree is by definition connected, one can go from any vertex u to any vertex $v \neq u$ in this tree. Among all these possible paths, take a maximum path. Since this graph has no cycle, the endpoints of a maximum path have only one neighbour on the path, and they are the two leaves whose existence we needed to prove. $\square$

Lemma 2.2. *Deleting a leaf from an n-vertex tree produces a tree with $n-1$ vertices.*

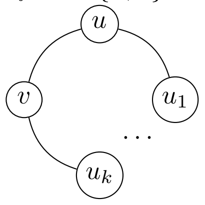
Proof. Let v be a leaf and $G' = G \backslash \{v\}$. We want to show that G' is connected and has no cycle.

Let $u, w \in V(G')$, then no u, w-path P in G can pass through the vertex v, since it is of degree 1. So, P is also present in G' and thus G' is connected.

Also G' has no cycle since deleting a vertex cannot create a cycle. $\square$

Lemma 2.3. *An edge contained in a cycle is not a* cut-edge *(that is, an edge whose deletion disconnects the graph).*

Proof. Let $\{u, v\}$ be an edge belonging to a cycle $\{u, u_1, u_2, \ldots, u_k, v\}$:



Then any path from x to y which uses $\{u, v\}$ can be replaced by the walk not using $\{u, v\}$ as follows:

$$(x \to \cdots \to u \to v \to \cdots y) \quad \rightsquigarrow \quad (x \to \cdots \to u \to \underbrace{u_1 \cdots \to u_k}_{\text{other part of cycle}} \to v \to \cdots y)$$

□

Theorem 2.4. *Let G be a graph with n vertices. Then the following are equivalent:*

(a) G is connected and has no cycle.

(b) G is connected and has $n - 1$ edges.

(c) G has $n - 1$ edges and has no cycle.

(d) For all vertices u, v in G, there is exactly one u, v-path.

Proof. $\underline{(a) \implies (b), (c)}$: We want to show that G has $n-1$ edges and we proceed by induction on n, the number of vertices.

For $n = 1$, the graph has clearly no edge.

Suppose $n > 1$ and the induction hypothesis holds for graphs with less than n vertices. We pick a leaf v and consider $G' = G\backslash\{v\}$, which by Lemma 2.2 is a tree with $n-1$ vertices, thus, using the induction hypothesis, it has $n-2$ edges. Adding back v gives $n - 1$ edges. This proves (b) because G' being connected, adding v keeps it connected, and it also proves (c), because adding a leaf cannot create a cycle.

$\underline{(b) \implies (a), (c)}$: Suppose that G is connected and has $n - 1$ edges. To show: G has no cycle.

A priori, G could have cycles. We delete edges from cycles of G one by one until the resulting graph G' has no cycle. Then, G' is connected by Lemma 2.3 and has no cycle. Now G' has n vertices, no cycle and is connected. Since we already showed that $(a) \Rightarrow (b)$, G' has $n - 1$ edges. But G initially had $n - 1$ edges, which implies that no edge was deleted. So $G = G'$ has no cycle.

$\underline{(c) \implies (a), (b)}$: Suppose that G has $n - 1$ edges ($|E(G)| = n - 1$) and has no cycle. We want to show that G is connected.

Suppose that G has k connected components, each with n_i vertices where $i = 1, \ldots, k$. Since every component satisfies (a), and $(a) \Rightarrow (b)$ was already proven, each of them has $n_i - 1$ edges. So,

$$|E(G)| = \sum_{i=1}^{k}(n_i - 1) = n - k \overset{!}{=} n - 1 \quad \Longrightarrow \quad k = 1.$$

Therefore there is only one component and so G is connected.

We are left to prove the last equivalence. Suppose G is connected and has no cycle. We want to prove that this is equivalent to: for all vertices u, v in G, there is exactly one u, v-path.

$\underline{(a) \Rightarrow (d)}$: Suppose there are two distinct u, v-paths, say P and Q. Let $e = \{x, y\}$ be an edge in P but not in Q. Concatenate P to the reverse of Q; this is a closed walk where e appears exactly once. Hence $(PQ^{-1})\backslash\{e\}$ is an x, y-walk not containing e. This walk from x to y contains a path from x to y (see Exercise 17). So this path together with e gives a cycle, which is a contradiction. Therefore, there is exactly one u, v-path.

$\underline{(d) \Rightarrow (a)}$: Now, suppose for all vertices u, v in G, there is exactly one u, v-path. Then G is clearly connected. If G had a cycle, then there would be two distinct paths for every pair of vertices in the cycle. So, G has no cycle. □

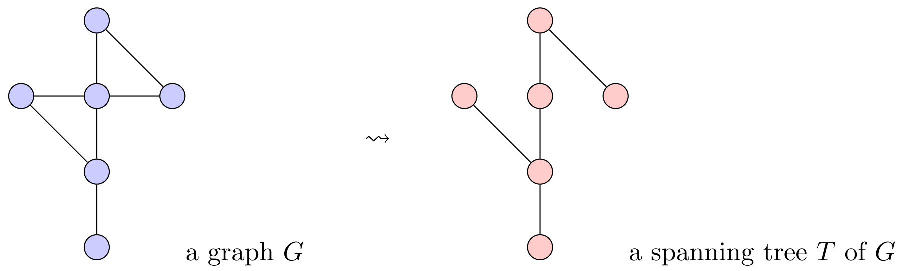
2.4 Minimum Spanning Trees

Definition 2.14. A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$.

Definition 2.15. A subgraph is called a *spanning subgraph* if $V = V'$.

Definition 2.16. A *spanning tree* is a spanning graph which is also a tree.

Example 2.8.



Definition 2.17. A *weighted graph* is a graph $G = (V, E)$ together with a function $w : E \to \mathbb{R}_{\geq 0}$ and $w(e)$ is called the weight of e.

Definition 2.18. The *weight* of $M \subseteq E$ in a weighted graph $G = (V, E)$ is $\sum_{e \in M} w(e)$.

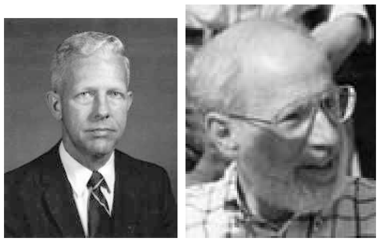


Figure 2.3: Prim's algorithm was developed in 1930 by V. Jarník, and rediscovered by Robert Prim (shown on the left) in 1957, and then by E.W. Dijkstra in 1959. Prim was working at Bell Laboratories with J. Kruskal (on the right), who developed Kruskal's algorithm.

Problem 3 (Minimum Spanning Tree (MST) Problem)**.** Given a weighted, connected graph G, find a spanning tree for G of minimum weight.

An algorithm for finding a minimum spanning tree is given below:

Algorithm 1 the MinTree algorithm

```
Input:  $G = (V, E)$ a connected graph, $V = \{1, 2, \dots, n\}$
        $w : E \to \mathbb{R}_{\geq 0}$ its weight function
Output: edge set $T$ of a minimum spanning tree
1: $V_i \leftarrow \{i\}$ for each $i, 1 \leq i \leq n$.
2: $T \leftarrow \varnothing$.
3: for $k = 1$ to $n - 1$ do
4:     Choose $i$ where $V_i \neq \varnothing$.
5:     Choose $e = \{u, v\}$, $u \in V_i$, $v \notin V_i$ such that $w(e)$ is minimal among edges
   $e' = \{u', v'\}$, $u' \in V_i$, $v' \notin V_i$.
6:     Let $j$ be the index such that $v \in V_j$.
7:     $V_i \leftarrow V_i \cup V_j$.
8:     $V_j \leftarrow \varnothing$.
9:     $T \leftarrow T \cup \{e\}$.
10: end for
11: return $T$
```

Prim's algorithm always chooses $i = 1$ in the MinTree Algorithm.

Proposition 2.5. *Let T be the edge set constructed by the above algorithm at any intermediate step. Then there is a MST for G which contains T.*

Note that k goes from 1 to $n - 1$, thus when $k = n - 1$, T has $n - 1$ edges. Thus the MST which itself has $n - 1$ edges and contains T must be T itself.

Proof. We proceed by induction on k, the iteration step.

When $k = 0$ (that is before we start the loop), T is the empty edge set and thus belongs to some MST of G.

At some iteration $k > 0$, we have T which is an edge set of some MST M by the induction hypothesis. At step $k+1$, add the edge e to T according to the algorithm, thus $T \cup \{e\}$ is surely an edge set. We want to show that $T \cup \{e\}$ is contained in some MST M'.

If e is in the MST M, then $M' = M$ and we are done. Otherwise, consider what would have happened if e were added to M. This would have added a cycle to the tree M (see Exercise 18). Since e has one endpoint in T and one endpoint not in T, there exists e' with one endpoint in T and the other not in T, to close the cycle. The algorithm at step $k+1$ could have chosen e' but it picked e. Thus, $w(e) \leq w(e')$.

Since $M \cup \{e\}$ is a spanning subgraph with n edges, remove e' to get the graph $M \backslash \{e'\} \cup \{e\}$ which is connected (removing an edge from a cycle does not disconnect the graph, see Lemma 2.3) and has $n-1$ edges, thus is a spanning tree. This gives a new spanning tree whose total weight is at most that of M, thus we found M' a MST which contains $T \cup \{e\}$. ☐

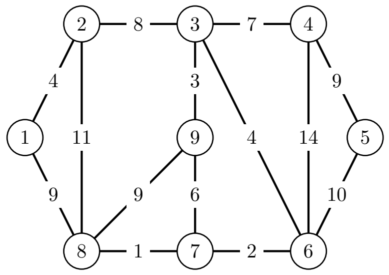
Another algorithm to compute the MST of a graph is Kruskal's algorithm.

Algorithm 2 Kruskal's algorithm

```
Input:  $G = (V, E)$ a connected graph, $V = \{1, 2, \ldots, n\}$
        $w : E \to \mathbb{R}_{\geq 0}$ its weight function
Output: edge set $T$ of a minimum spanning tree
 1: Sort all edges such that $w(e_1) \leq \cdots \leq w(e_m)$.
 2: $T = \varnothing$.
 3: for $k = 1$ to $m$ do
 4:     if $|T| = n - 1$ then
 5:         break;
 6:     end if
 7:     if $T \cup \{e_k\}$ contains no cycle then
 8:         $T \leftarrow T \cup \{e_k\}$;
 9:     end if
10: end for
11: return $T$
```

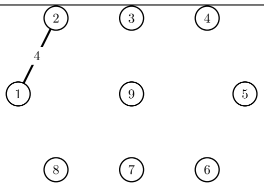
This algorithm can actually be seen as a particular case of the MinTree algorithm. In the MinTree algorithm, Kruskal's algorithm always chooses i such that there is an edge $e = \{u, v\}$, $u \in V_i$, $v \notin V_i$ such that e has minimum weight among all edges which do not have both vertices inside any set V_i, $i = 1, \ldots, n$.

Example 2.9. Compute the minimum spanning tree (MST) of the following graph:

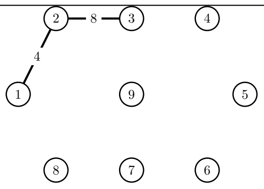


Using Prim's algorithm, the progression of T is as follows.

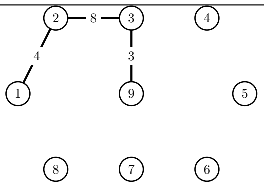
1. Choose $V_1 = \{1\}$, and $e = \{u, v\}$ with $u \in V_1$ and v outside, with minimal weight. Between weights 4 and 9, choose 4.



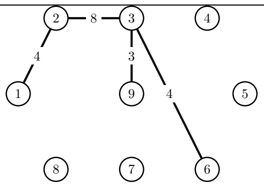
2. Then $V_1 = \{1, 2\}$ and $T = \{\{1, 2\}\}$. To pick the next edge, the choices for the weights are $9, 11, 8$, so we pick 8.



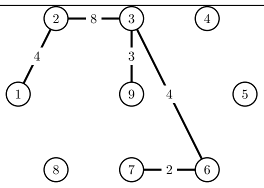
3. Then $V_1 = \{1, 2, 3\}$, $T = \{\{1, 2\}, \{2, 3\}\}$. For the next edge, we have weights 9, 11, 3, 4, and 7. So we choose 3.



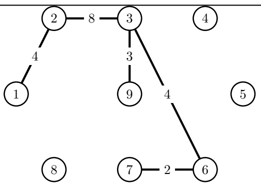
4. Then $V_1 = \{1, 2, 3, 9\}$, $T = \{\{1, 2\}, \{2, 3\}, \{3, 9\}\}$ and for the next edge, we can choose among weights 9, 11, 4, 7, and 6 so we pick 4.



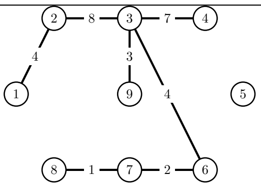
5. Then $V_1 = \{1, 2, 3, 9, 6\}$, $T = \{\{1, 2\}, \{2, 3\}, \{3, 9\}, \{3, 6\}\}$. The weights for the next edge are 9, 11, 7, 6, 2, 14 and 10, so we pick 2.



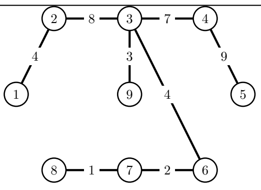
6. Then $V_1 = \{1, 2, 3, 9, 6, 7\}$, $T = \{\{1, 2\}, \{2, 3\}, \{3, 9\}, \{3, 6\}, \{6, 7\}\}$. The weights are now 9, 11, 7, 6, 14, 10, and 1. So we pick 1.



7. Then $V_1 = \{1, 2, 3, 9, 6, 7, 8\}$, $T = \{\{1, 2\}, \{2, 3\}, \{3, 9\}, \{6, 9\}, \{6, 7\}, \{7, 8\}\}$. The weights are now 9, 11, 7, 14, 10. So we pick 7. Note a weight of 6 that cannot be used because $\{9, 7\}$ has both its endpoints in V_1.

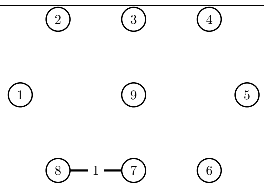


8. Then $V_1 = \{1, 2, 3, 9, 6, 7, 8, 4\}$, $T = \{\{1, 2\}, \{2, 3\}, \{3, 9\}, \{6, 9\}, \{6, 7\}, \{7, 8\}, \{3, 4\}\}$. The weights are now 9, 11, 14, 10, so we pick 9. This adds $\{4, 5\}$ to T and the algorithm stops since we have 8 edges.

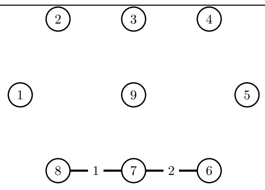


To use Kruskal's algorithm, we first sort the edges by weight: $w(\{7, 8\}) = 1$, $w(\{6, 7\}) = 2$, $w(\{3, 9\}) = 3$, $w(\{1, 2\}) = 4$, $w(\{3, 6\}) = 4$, $w(\{7, 9\}) = 6$, $w(\{3, 4\}) = 7$, $w(\{2, 3\}) = 8$, $w(\{1, 8\}) = 9$, $w(\{4, 5\}) = 9$, $w(\{8, 9\}) = 9$, $w(\{2, 8\}) = 11$, $w(\{4, 6\}) = 14$. A progression of T is then as follows:

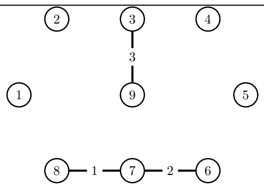
1.



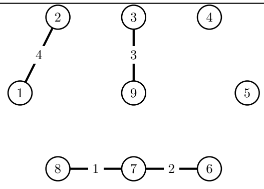
2.



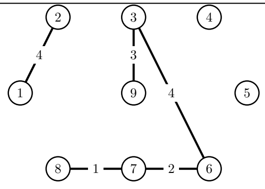
3.



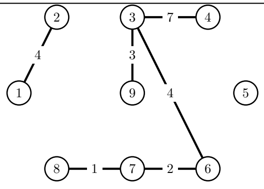
4.



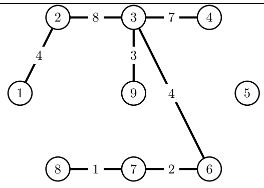
5.



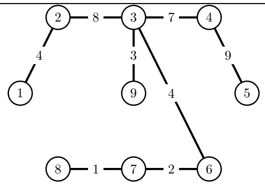
6.

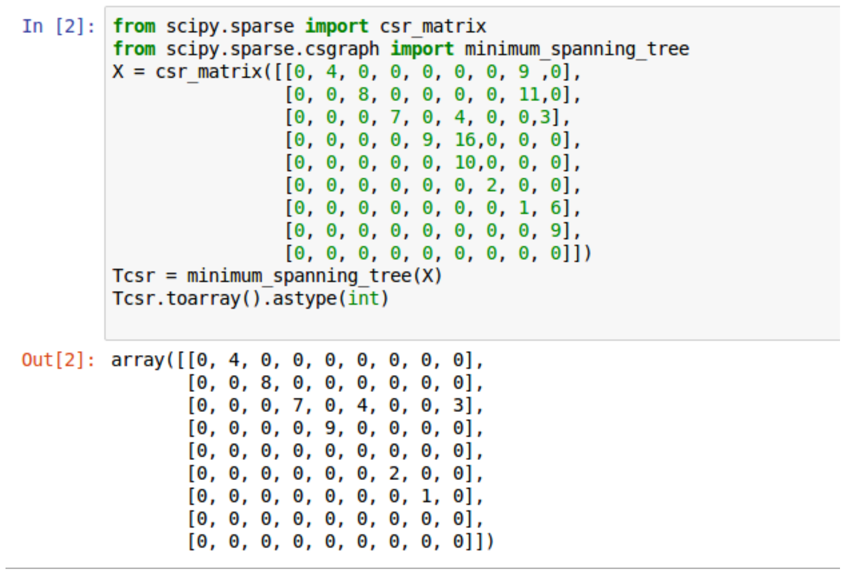


7.



8.





```
In [2]: from scipy.sparse import csr_matrix
        from scipy.sparse.csgraph import minimum_spanning_tree
        X = csr_matrix([[0, 4, 0, 0, 0, 0, 0, 9 ,0],
                        [0, 0, 8, 0, 0, 0, 0, 11,0],
                        [0, 0, 0, 7, 0, 4, 0, 0,3],
                        [0, 0, 0, 0, 9, 16,0, 0, 0],
                        [0, 0, 0, 0, 0, 10,0, 0, 0],
                        [0, 0, 0, 0, 0, 0, 2, 0, 0],
                        [0, 0, 0, 0, 0, 0, 0, 1, 6],
                        [0, 0, 0, 0, 0, 0, 0, 0, 9],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0]])
        Tcsr = minimum_spanning_tree(X)
        Tcsr.toarray().astype(int)

Out[2]: array([[0, 4, 0, 0, 0, 0, 0, 0, 0],
               [0, 0, 8, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 7, 0, 4, 0, 0, 3],
               [0, 0, 0, 0, 9, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 2, 0, 0],
               [0, 0, 0, 0, 0, 0, 0, 1, 0],
               [0, 0, 0, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

Figure 2.4: Minimum spanning tree algorithms are usually implemented by default in most languages, e.g. above in Python.

Note that $\{7, 9\}$ has weight 6, but it was skipped, because it would have created a cycle. Similarly, there are 3 edges of weight 9, but two of them create a cycle.

Sometimes, edges have the same weight, and unlike in the above example, several of them are valid. In that case, one could choose an edge at random, or choose the lexicographic order.

Remark. We do not discuss the complexity of these algorithms, because it depends on the data structure used.

Remark. You may be wondering about the unicity of spanning trees, and how these algorithms behave with respect to this. See Exercise 21 for a discussion on this.

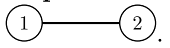
Minimum spanning tree algorithms can be used as part of more complicated algorithms, but they have also applications of their own. Here is an example of application to clustering. Suppose that you have n items, and you know the distance between each pair of items. You would like to cluster them into k groups, so that the minimum distance between items in different groups is maximized. The idea is to create a graph whose vertices are the n items, and whose edges have for weight the distance between the pair of items. Then consider for clusters a set of connected components of the graph, and iteratively combine the clusters containing the 2 closest items by adding an edge between them. This process stops at k clusters.

What we have just described is actually Kruskal's algorithm: the clusters are the connected components that Kruskal's algorithm is creating. In the language of clustering, this process is also called a single linkage agglomerative clustering.

2.5 Labeled Trees

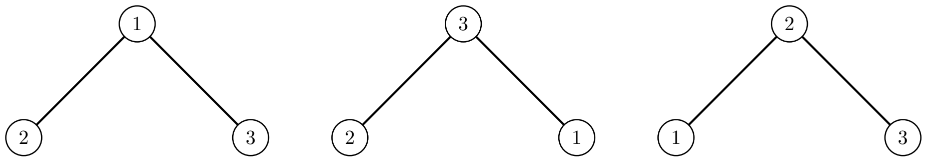
We next study trees whose n vertices are labeled from 1 to n. Formally, let $l : V \to \{1, \ldots, n\}$, $v \mapsto l(v)$ be the label function attached to the graph $G = (V, E)$.

Example 2.10. For $n = 2$, we have



.

For $n = 3$, we have



Definition 2.19. Two labeled graphs are *isomorphic* if their graphs are isomorphic and the labels are preserved by the isomorphism. Formally, two labeled graphs $G = (V, E, l)$ and $G = (V', E', l')$ are isomorphic if there exists a bijection $\alpha : V \to V'$ such that (1) $\{u, v\} \in E \iff \{\alpha(u), \alpha(v)\} \in E'$ for all $\{u, v\} \in E$ (edges are preserved), (2) $l(u) = l'(\alpha(u))$ for all $u \in V$ (labels are preserved).

Note that sometimes a further label could be introduced on edges, a case we are not considering here.

With this definition, we can see why the above example makes sense. For $n = 3$, two nodes have degree 1, and one node has degree 2. The node with degree 2 can take any of the 3 labels, then the leaves have for labels the remaining labels.

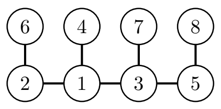
To be able to count the number of labeled trees (up to isomorphism), we will use a tree labeling called Prüfer's code, or Prüfer'sequence.

Algorithm 3 Algorithm for Prüfer's Code

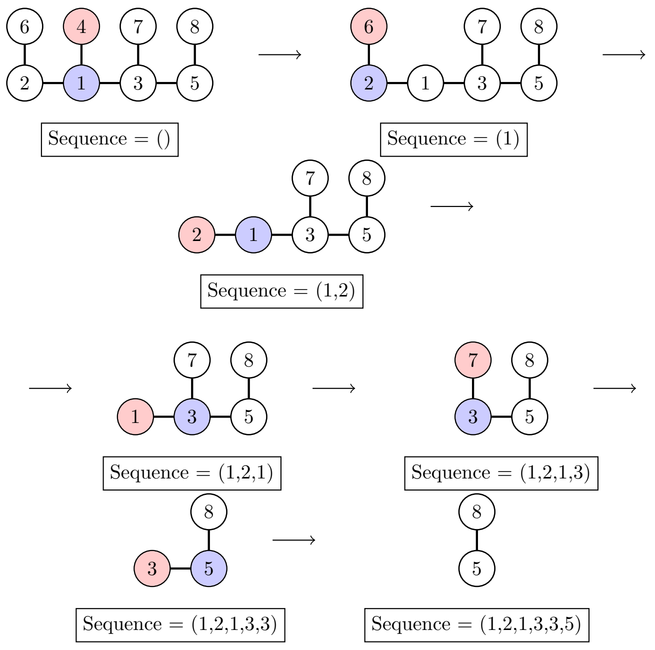
```
1: Input: A tree $T$ with vertex set $V$, $|V| = n \geq 2$.
2: Output: A sequence $(a_1, \cdots, a_{n-2}) \in V^{n-2}$.
3: Set $T_1 = T$.
4: for $i = 1, 2, \ldots, n-2$ do
5:     Let $v$ be the leaf of $T_i$ with the smallest label.
6:     Set $a_i$ to be the unique neighbour of $v$ in $T_i$.
7:     Construct $T_{i+1}$ from $T_i$ by removing $v$ and the edge $\{v, a_i\}$.
8: end for
9: if $|V| = 2$ then
10:    return Empty set
11: end if
```

Figure 2.5: The mathematician Heinz Prüfer (1896-1934) created Prüfer's code, and contributed to diverse areas of mathematics such as group theory, knot theory and Sturm-Liouville theory.

Example 2.11. Compute the Prüfer's code of the following labeled tree:



We have the following progression, where we start with the leaf 4, which is the smallest label among the leaves:



Theorem 2.6. *There is a bijection between $V^{|V|-2}$ and the set of all labeled trees with vertex set V, $|V| \geq 2$.*

Corollary 2.7. *The number of labeled trees with $n \geq 2$ vertices is n^{n-2}.*

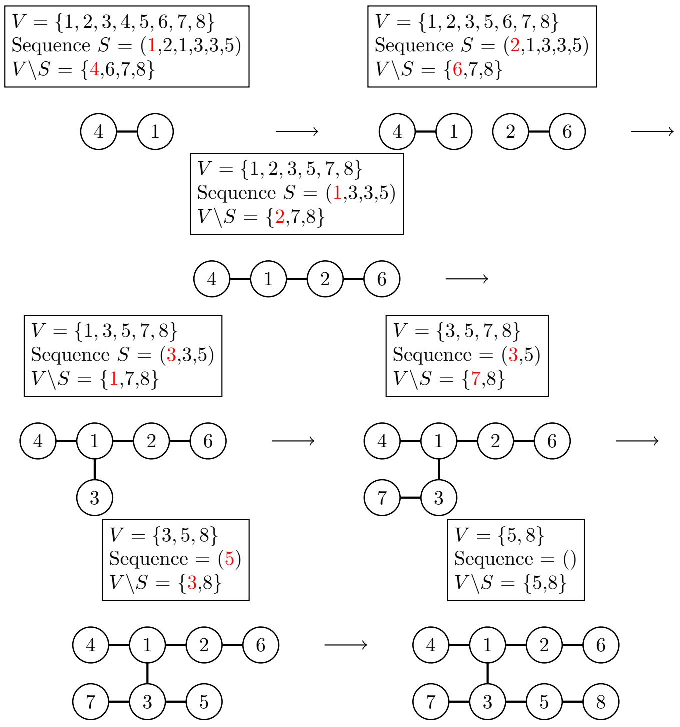
Proof. Let $|V| = n$. The preceding algorithm gives a function f which takes a tree T with n vertices and outputs $f(T) = (a_1, \cdots, a_{n-2})$. To show that f is a bijection, we need to show that every sequence $(a_1, \cdots, a_{n-2})$ defines uniquely a tree.

By induction on n: for $n = 2$, there is a single labeled tree with two vertices, so, () defines uniquely this tree. Assume that the induction hypothesis is true for all trees with less than n vertices, $n \geq 2$. Given $(a_1, \cdots, a_{n-2})$, we need to find a unique tree T such that $f(T) = (a_1, \cdots, a_{n-2})$.

None of the a_i is a leaf in T since when a vertex is set to be a_i, it is adjacent to a leaf (when the graph is left with only leaves, then it has two vertices and the algorithm terminates) and $V \backslash \{a_1, \ldots, a_{n-2}\}$ contains only nodes that are leaves.

This implies that the label of the first leaf removed is precisely the minimum element of $V \backslash \{a_1, \ldots, a_{n-2}\}$. Let v be this leaf and it has a unique neighbour a_1. By the induction hypothesis, we know that there exists a unique tree T' with vertex set $V \backslash \{v\}$ such that $f(T') = (a_2, \cdots, a_{n-2})$. Adding the vertex v and the edge $\{v, a_1\}$ to T' gives the desired tree T. □

Example 2.12. Compute the tree corresponding to the Prüfer's code (1,2,1,3,3,5). We computed this Prüfer's code in the previous example, so if all works as it should, we should get back the same labeled tree. We observe that since the code is of length 6, we are looking for a tree with $n = 8$ vertices.



$V = \{1, 2, 3, 4, 5, 6, 7, 8\}$
Sequence S = (1,2,1,3,3,5)
$V \backslash S$ = {4,6,7,8}

$V = \{1, 2, 3, 5, 6, 7, 8\}$
Sequence S = (2,1,3,3,5)
$V \backslash S$ = {6,7,8}

$V = \{1, 2, 3, 5, 7, 8\}$
Sequence S = (1,3,3,5)
$V \backslash S$ = {2,7,8}

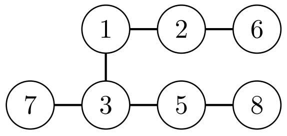
$V = \{1, 3, 5, 7, 8\}$
Sequence S = (3,3,5)
$V \backslash S$ = {1,7,8}

$V = \{3, 5, 7, 8\}$
Sequence = (3,5)
$V \backslash S$ = {7,8}

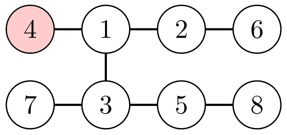
$V = \{3, 5, 8\}$
Sequence = (5)
$V \backslash S$ = {3,8}

$V = \{5, 8\}$
Sequence = ()
$V \backslash S$ = {5,8}

If we wanted to use this example to illustrate the proof of Theorem 2.6, then given the sequence $(a_2, \ldots, a_{n-2}) = (2, 1, 3, 3, 5)$, there is a unique tree T' with vertex set $V \backslash \{4\}$ corresponding to it, given by



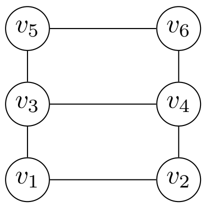
So when considering the sequence $(a_1, a_2, \ldots, a_{n-2}) = (1, 2, 1, 3, 3, 5)$, we know that $a_1 = 1$ has for neighbour a leaf v with label 4, thus we append the edge $\{v, a_1\}$ to T' to get the tree T:



2.6 Exercises

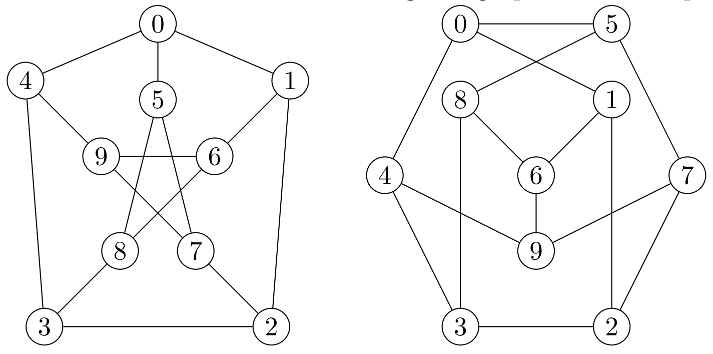
Exercise 10. Prove that the set of all automorphisms of a graph forms a group.

Exercise 11. (*) Compute the automorphism group of the following graph.



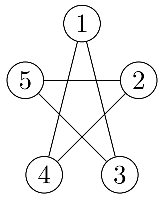
Exercise 12. Compute the automorphism group of the n-cycle graph, the graph given by n vertices $1, \ldots, n$, and n edges given by $\{i, i+1\}$ where $i, i+1$ are understood modulo n.

Exercise 13. Show that the following two graphs are isomorphic:



Exercise 14.

1. Compute the automorphism group of the following graph.



2. Give an example of a connected graph with at least 2 vertices whose automorphism group is of size 1.

3. Suppose two connected graphs G and G' have the same automorphism group, that is $\mathrm{Aut}(G) \cong \mathrm{Aut}(G')$. Does it imply that G is isomorphic to G'? Justify your answer.

Exercise 15. For all $n \geq 2$, give a graph whose automorphism group is the symmetric group S_n, that is, the group of permutations of n elements.

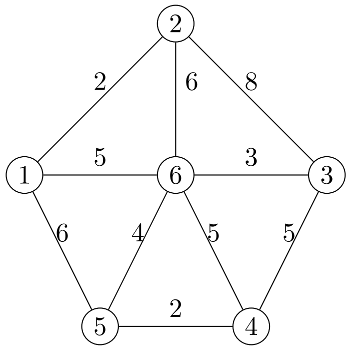
Exercise 16. Count, using Pólya's Enumeration Theorem, the number of isomorphism classes of graphs with 4 vertices, and count how many there are for each possible number of edges.

Exercise 17. Prove (by induction on the length l of the walk) that every walk from u to v in a graph G contains a path between u and v.

Exercise 18. Prove that adding one edge to a tree creates exactly one cycle.

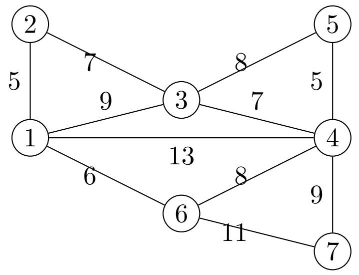
Exercise 19. Prove or disprove the following claim: if G is a graph with exactly one spanning tree, then G is a tree.

Exercise 20. Find a minimum spanning tree for this graph, using once Prim algorithm, and once Kruskal algorithm:



Exercise 21.

1. Compute a minimum spanning tree in the following graph with the method of your choice. Describe the steps of the algorithm.

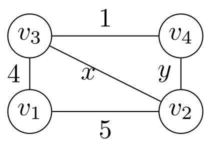


2. Construct, if possible, a connected weighted graph G with two minimum spanning trees.

3. Let G be a connected weighted graph, with m edges with respective weights $e_1 \leq e_2 \ldots \leq e_{i-1} < e_i = e_{i+1} < e_{i+2} \leq \ldots \leq e_m$, and let T_k be a minimum spanning tree found by Kruskal algorithm. Suppose that there exists a different minimum spanning tree T such that T and T_k share the same edges $e_1, \ldots e_{i-1}$, for some $i \geq 2$, but T_k contains e_i and not e_{i+1}, while T contains e_{i+1} but not e_i. Show that T can be found by an instance of Kruskal algorithm.

Exercise 22. (*)

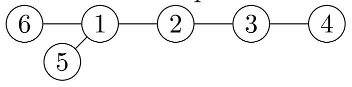
1. Consider the following weighted undirected graph, where x, y are unknown integers.



Give, if possible, values for x, y such that the graph contains (a) a single minimum spanning tree, (b) at least two minimum spanning trees.

2. Given a weighted undirected graph $G = (V, E)$, such that every edge e in E has a distinct weight. Prove or disprove the following: G contains a unique minimum spanning tree.

Exercise 23. Compute the Prüfer code of the following tree:



Construct the tree corresponding to the Prüfer code (1,1,3,5,5).

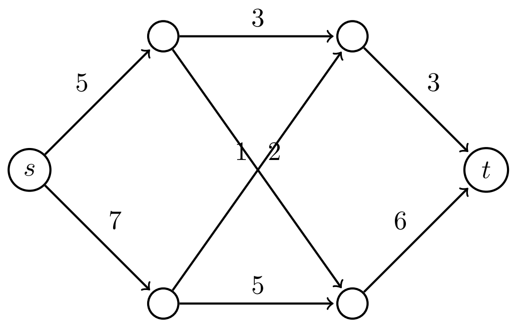
Exercise 24. Determine which trees have Prüfer codes that have distinct values in all positions.

Chapter 3

Network Flows

Definition 3.1. A *network* is a weighted directed graph with 2 distinguished vertices s (called *source*) and t (called *sink*), where s has only outgoing edges and t has only incoming edges. For a network, the weight function is called *capacity function* (denoted by c).

Example 3.1. Here is an example of network, to each edge is attached a capacity.



3.1 Maximum Flow, Minimum Cut

Definition 3.2. A *flow* on a network $G = (V, E)$ is a map $f : E \to \mathbb{R}^+$ such that

(1) $0 \leqslant f(e) \leqslant c(e), \forall\, e \in E$ $(flow\ is\ feasible)$

(2) $\displaystyle\sum_{u \in I(v)} f(u, v) = \sum_{u \in O(v)} f(v, u), \forall\, v \in V \backslash \{s, t\}$ $(flow\ conservation)$

Here, $O(v)$ (resp. $I(v)$) is the set of vertices which have an edge coming from (resp. going into) v.

Sometimes, the term “feasible” refers to both conditions instead of just the first one.

Problem 4. Given a network $G = (V, E)$, find a maximum flow.

The sum of the values of a flow f on the edges leaving the source is called the *strength* of the flow (denoted by $|f|$), that is

$$|f| = \sum_{u \in O(s)} f(s, u).$$

It can be shown (see Exercise 25) that $|f| = \sum_{u \in I(t)} f(u, t)$, that is, the sum of the values of f leaving the source is also the sum of the values of f entering the sink.

Definition 3.3. A *cut* of a network $G = (V, E)$ with source s and sink t is a decomposition $V = S \sqcup T$ of V into disjoint subsets S, T such that $s \in S$, $t \in T$. Such a cut is denoted by (S, T). The *capacity* of (S, T) is defined as

$$C(S, T) = \sum_{\substack{u \in S \\ v \in T}} c(u, v)$$

A *minimum cut* is a cut of minimum capacity.

Lemma 3.1. *For any cut (S, T), $|f| \leqslant C(S, T)$.*

Proof. First, we prove by induction on $|S|$ that

$$|f| = \sum_{\substack{u \in S \\ v \in T}} f(u, v) - \sum_{\substack{u \in S \\ v \in T}} f(v, u). \tag{3.1}$$

See Example 3.2 for an illustration of this claim. If $|S| = 1$, then S contains only the source s. Then

$$|f| = \sum_{v \in T} f(s, v).$$

Suppose true for $|S| > 1$. Now move one vertex w from T to S and we want to compute

$$\sum_{\substack{u \in S \cup \{w\} \\ v \in T \setminus \{w\}}} f(u, v) - \sum_{\substack{u \in S \cup \{w\} \\ v \in T \setminus \{w\}}} f(v, u).$$

Then we both have an increase by

$$\sum_{x \in O(w)} f(w, x)$$

(if $x \in T$ before, it adds up after, if $x \in S$ before it used to count negatively, so it also adds up after) and similarly a decrease by

$$\sum_{y \in I(w)} f(y, w),$$

that is:

$$\sum_{\substack{u\in S\cup\{w\}\\ v\in T\setminus\{w\}}} f(u,v) - \sum_{\substack{u\in S\cup\{w\}\\ v\in T\setminus\{w\}}} f(v,u) = \sum_{\substack{u\in S\\ v\in T}} f(u,v) - \sum_{\substack{u\in S\\ v\in T}} f(v,u) + \sum_{x\in O(w)} f(w,x) - \sum_{y\in I(w)} f(y,w).$$

But flow conservation must hold. So the total change

$$\sum_{x\in O(w)} f(w,x) - \sum_{y\in I(w)} f(y,w)$$

after moving w from T to S is 0 which concludes the proof by induction.

Now

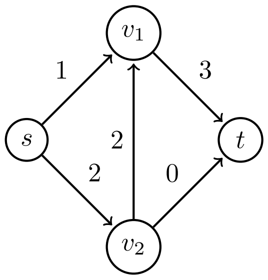
$$|f| = \sum_{\substack{u\in S\\ v\in T}} f(u,v) - \sum_{\substack{u\in S\\ v\in T}} f(v,u) \leqslant \sum_{\substack{u\in S\\ v\in T}} f(u,v) \leqslant \sum_{\substack{u\in S\\ v\in T}} c(u,v) = C(S,T)$$

□

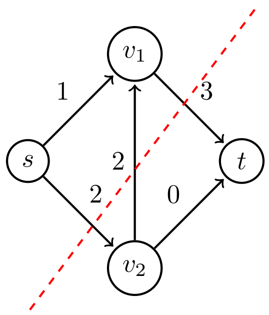
In particular, if we pick a minimum cut in the above lemma, then we get an upper bound on the maximum flow:

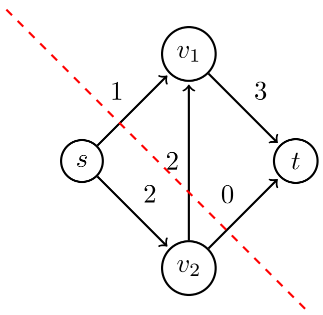
$$\max|f| \leq \min_{S,T} C(S,T).$$

Example 3.2. Different cuts of a network whose flow f is described on the network edges each gives the strength $|f| = 3$ according to (3.1).



Cut 1 : $|f| = 3 + 2 - 2 = 3$ Cut 2 : $|f| = 1 + 2 + 0 = 3$



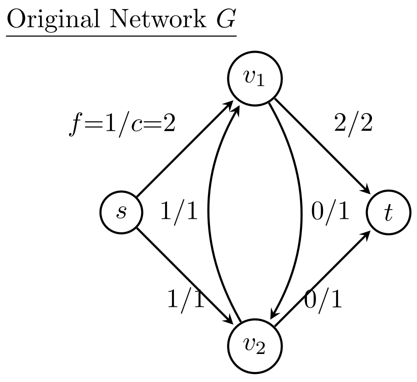


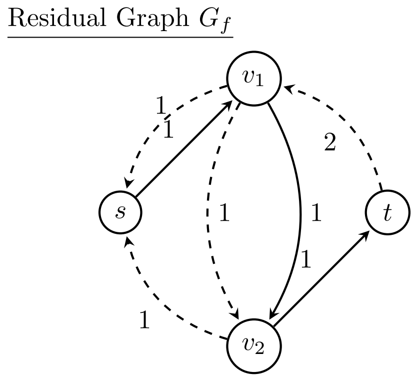
Definition 3.4. Given a flow f on a graph (network) $G = (V, E)$, the corresponding *residual graph* G_f has V for vertices, and

(1) edges have capacities $c(v, w) - f(v, w)$, only edges with $c_f := c(v, w) - f(v, w) > 0$ are shown, and they are shown in the same direction as (v, w),

(2) if $f(v, w) > 0$, place an edge with capacity $c_f(v, w) = f(v, w)$ in the opposite direction of (v, w).

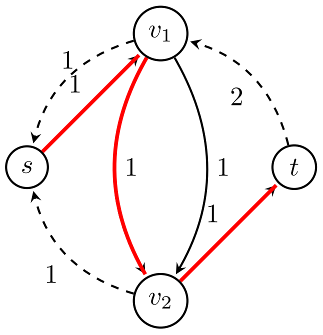
Example 3.3. We show a network G with its residual graph G_f, dashed lines are used to emphasize "backward" edges, that is edges created based on the condition (2) of Definition 3.4.





Definition 3.5. Let f be a flow in a network G. An *augmenting path* is a directed path from s to t in the residual network G_f. Alternatively, an augmenting path for f is an undirected path P from s to t such that $f(e) < c(e)$ for all "forward" edges e contained in P, and $f(e) > 0$ for all "backward" edges contained in P.

Example 3.4. An augmenting path is shown below, in the residual graph G_f computed above.

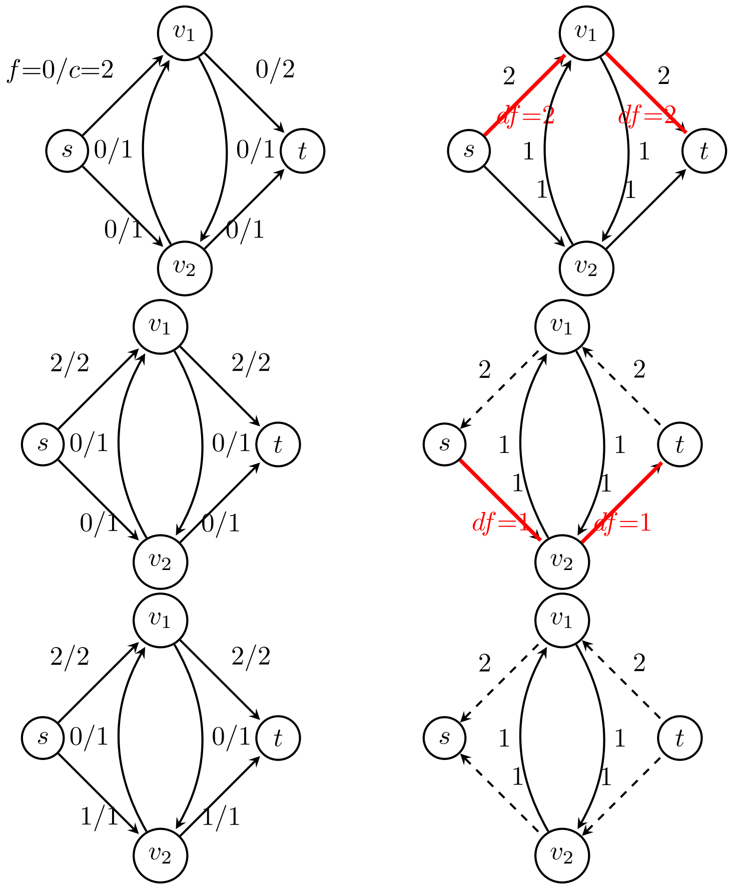


Residual networks and augmenting paths form the core of Ford-Fulkerson algorithm.

Algorithm 4 Ford-Fulkerson (1956)

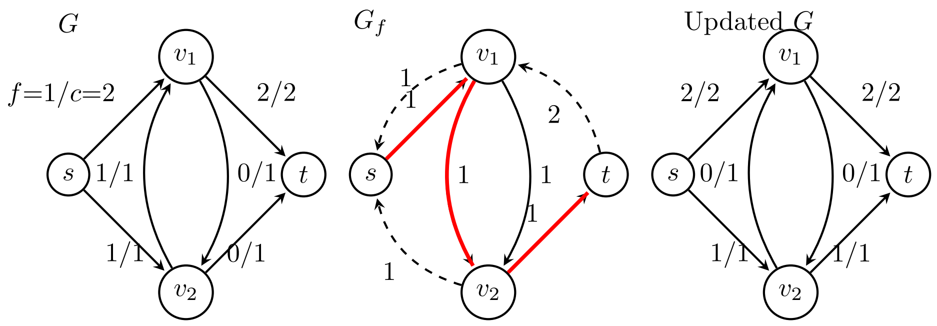
```
Input: $G = (V, E)$ a network, with capacity $c$, source $s$ and sink $t$.
Output: a maximal flow $f^*$.
1: Initialize $f(e) = 0 \ \forall\ e \in E$.
2: Compute $G_f$.
3: while (there is a path $P$ from $s$ to $t$ in $G_f$) do
4:     Set $df = \min_{e \in P} c_f(e)$ in $G_f$.
5:     Update $f$ in $G$: set $f(u, v) = f(u, v) + df$ for $(u, v) \in P$ if $(u, v) \in E$
       and $f(v, u) = f(v, u) - df$ for $(u, v) \in P$ if $(v, u) \in E$.
6:     Rebuild the residual network $G_f$.
```

Example 3.5. Ford-Fulkerson algorithm is illustrated, the left-hand side shows the network G while the right hand-side is the corresponding residual graph G_f.



Ford-Fulkerson algorithm returns the maximum flow $|f^*| = 3$ when no more path P is found in the residual graph.

Since the above example does not use any path in G_f which involves a "backward" edge, let us illustrate this case using Examples 3.3 and 3.4. Suppose that one runs the algorithm for some iterations, to get to the step constructed in Example 3.3, in which the augmenting path displayed in Example 3.4 is chosen. The example below show what would be the next updated graph G:



Theorem 3.2. If Ford-Fulkerson algorithm terminates, it outputs a maximum flow.

Proof. Suppose the algorithm terminates and outputs a flow f^*, this means there is no path from s to t in G_{f^*}. In other words, s and t are disconnected. Let S be the set of nodes reachable from s in G_{f^*}; i.e., $v \in S \iff \exists$ a path from s to v. Let $T = V \backslash S$, we claim that $|f^*| = C(S,T)$. Before proving the claim, recall that $|f^*| \leq C(A,B)$ for any cut (A,B), by Lemma 3.1. Thus when equality is reached, which is the case with $|f^*| = C(S,T)$, this means f^* is the maximum flow and $C(S,T)$ the minimum cut. We next prove the claim:

$$|f^*| = C(S,T).$$

Consider any edge e from S to T in the original network G. Edge e must not exist in G_{f^*}, or else its endpoint in T would be reachable from s, contradicting the definition of T. Thus it must be the case that $f^*(e) = c(e)$ for all such e (by construction of G_{f^*}, when $f^*(e) = c(e)$ in G, the edge is removed in its residual graph). Next consider e' from T to S in G. If $f^*(e') > 0$, there will be an edge in the opposite direction of e' in G_{f^*}; i.e., an edge from S to T, again contradicting the definition of T. We conclude that $f^*(e') = 0$ for all such e'. Now recall from (3.1) that

$$|f^*| = \sum_{\substack{u \in S \\ v \in T}} f^*(u,v) - \sum_{\substack{u \in S \\ v \in T}} f^*(v,u)$$

where for our particular cut $f^*(u,v) = c(u,v)$ and $f^*(v,u) = 0$. Thus

$$|f^*| = \sum_{\substack{u \in S \\ v \in T}} c(u,v) = C(S,T).$$

□

Corollary 3.3. (Integral Flow) *If all edge capacities are non-negative integers, then there exists an integral maximum flow.*

Proof. Since edge capacities are integral, the capacity of every edge in G_f is also integral. At each step, *df* is at least one. Thus the value of the flow f increases by at least one. Since $|f| < \infty$, $|f|$ cannot increases indefinitely, hence the algorithm stops after a finite number of steps. □

For an example of network where Ford-Fulkerson algorithm may not terminate, see Exercise 27. The above corollary tells us that this counter-example will use some edge with capacity that is not a non-negative integer.

Corollary 3.4. (Max Flow/Min Cut) *The minimum cut value in a network is the same as the maximum flow value.*

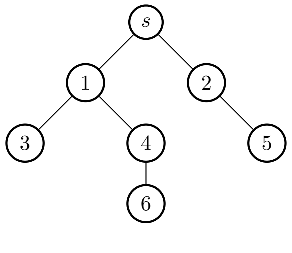
Proof. If Ford-Fulkerson algorithm terminates, as in Corollary 3.3, then we have a proof (we have a flow f^* for which $|f^*| = C(S,T)$, and equality means, as recalled in the proof of Theorem 3.2, that we have both a minimum cut and a maximum flow). Now it turns out that the algorithm always terminates, assuming a particular search order for the the augmenting paths, this is a version of the algorithm called Edmonds-Karp algorithm, which we will see below. □

Using the shortest augmenting path found by breadth-first search, instead of any augmenting path, guarantees that Ford-Fulkerson algorithm terminates.

Algorithm 5 Breadth-First Search (BFS)

```
    Input: a graph $G = (V, E)$, with start vertex $s$.
    Output: a function $d : V \to \mathbb{R}^+$, $d(v)$ is the distance from $v$ to $s$ in $G$.
    Data Structure: a queue $Q$ (first in first out)
1: $Q = \varnothing$ ; $d(s) = 0$ ; $d(v) = \infty \ \forall \ v \neq s$
2: Add $s$ to $Q$;
3: while ($Q \neq \varnothing$) do
4:     Remove first vertex $v$ from $Q$;
5:     for (all $w$ adjacent to $v$) do
6:         if ($d(w) = \infty$) do
7:             $d(w) = d(v) + 1$;
8:             add $w$ to $Q$;
```

Example 3.6. We illustrate the Breadth-First Search (BFS) algorithm.



$Q = \varnothing,\ d(s) = 0, d(1) = \ldots = d(6) = \infty$
$Q = \{s\}$
$Q = \{\}, d(1) = 1, d(2) = 1, Q = \{1, 2\}$
$Q = \{2\}, d(3) = 2, d(4) = 2, Q = \{2, 3, 4\}$
$Q = \{3, 4\}, d(5) = 2, Q = \{3, 4, 5\}$
$Q = \{4, 5\}$
$Q = \{5\}, d(6) = 3, Q = \{5, 6\}$
$Q = \{6\}$
$Q = \{\}$

The breadth first search algorithm is incorporated to Ford-Fulkerson algorithm to find an augmenting path with minimum number of edges.

Algorithm 6 Edmonds-Karp (1970)

```
    Input: $G = (V, E)$ a network, with capacity $c$, source $s$ and sink $t$
    Output: a maximal flow $f^*$.
1: Initialize $f(e) = 0 \ \forall \ e \in E$
2: Compute $G_f$.
3: while (there is a path from $s$ to $t$ in $G_f$) do
4:     Let $P$ be the shortest $s, t-$path in $G_f$ found by BFS.
5:     Set $df = \min_{e \in P} c_f(e)$ in $G_f$ along $P$.
6:     Update $f$ in $G$: set $f(u, v) = f(u, v) + df$ for $(u, v) \in P$ if $(u, v) \in E$
   and $f(v, u) = f(v, u) - df$ for $(u, v) \in P$ if $(v, u) \in E$.
7:     Rebuild the residual network $G_f$.
```

Theorem 3.5. The Edmonds-Karp algorithm terminates after at most $|E|(|V|-1)$ iterations.

Proof. The proof counts the number of while loop iterations before the algorithm stops.

At every iteration of the while loop, the algorithm uses a BFS to find the shortest augmenting path, so it builds a tree that starts at s, then the level 1 of the tree will contain a set V_1 of vertices that are at distance 1 from s, and more generally, the level i of the tree will contain a set V_i of vertices at distance i from s. A shortest path from s to t must use edges (u, v) with $u \in V_i$, $v \in V_{i+1}$ at each step, $i = 1, 2, \dots$ and a path that uses an edge u, v with $u \in V_i$, $v \in V_j$ and $j \leqslant i$ cannot be a shortest path.

We will look at how the BFS tree starting at s in G_f changes from one iteration to another. Given G_f, we find an augmenting path P, we then update G accordingly:

- If P contains a forward edge e (that is e has the same direction in G_f than in G), then in G, e will get augmented by df, so in the new G_f, the edge e either has disappeared (if $f(e) = c(e)$) or is still there with a lower capacity, and a backward edge may be added to the new G_f if it was not there before.

- If P contains a backward edge e (that is e has the reverse direction in G_f than in G), then in G, e will be diminished by df, so in the new G_f, the edge e either has disappeared, or is still there, and a forward edge may be added to the new G_f if it was not there before.

Thus every new edge that is created in the residual graph from one iteration to another is the reverse of an edge that belongs to P. Since every edge of P goes from level V_i to level V_{i+1} in the BFS tree, every new edge that gets created must go from level V_{i+1} to V_i. This shows that:

- if at a certain iteration of the algorithm, the length of a shortest path from s to t in the residual network is l, then at every subsequent iteration it is $\geq l$.

In words, the length of P from s to t can never decrease as we repeat the while loop.

Furthermore, if the length of the path from s to t in the residual network remains l from one iteration T to the next one $T+1$, then the path P in G_f at iteration $T+1$ must be using only edges which were already present in the residual network at iteration T and which were "forward" edges (from V_i to V_{i+1}). Thus, in all subsequent iterations in which the distance from s to t remains l, it is so because there is a length l path made entirely of edges that were "forward" edges at iteration T. At each iteration however, at least one of those edges is saturated and is absent from G_f in subsequent stages. So there can be at most $|E|$ iterations during which the distance from s to t stays l.

So we further established that:

- after at most $|E|$ iterations, the distance $\geq l+1$.

Since the distance from s to t is at most $|V|-1$, and it takes at most $|E|$ iterations to increase the length of the path by 1, after $|E|(|V|-1)$ iterations, the length of the shortest path becomes the total number of vertices in the network, thus it cannot increase anymore, and the algorithm terminates. $\square$

3.2 Some Applications of Max Flow-Min Cut

Results around the Max Flow-Min Cut of a network were motivated by computing a maximum flow in a network. However they can also be used to prove other results in graph theory. We give as example below Hall's Theorem. Menger's Theorem is given in Exercise 28.

Definition 3.6. Given a graph $G=(V,E)$, a *perfect matching* is a subset of E which covers all vertices but each vertex only once.

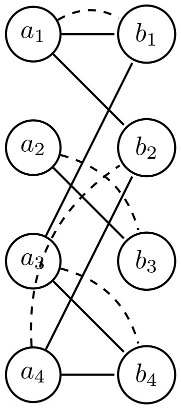
Theorem 3.6. ***[Hall's theorem]*** *Let $G=(V,E)$ be a bipartite graph with $V=A\sqcup B$ and $|A|=|B|$. For $J\subseteq A$, let $\Gamma(J)$ be the set of vertices adjacent to some vertex in J:*

$$\Gamma(J)=\{v\in B,\exists\ w\in J,\{v,w\}\in E\}.$$

Then G has a perfect matching if and only if

$$|\Gamma(J)|\geqslant|J|\ \forall\ J\subseteq A.$$

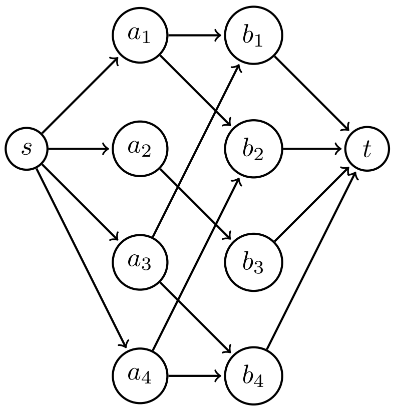
Example 3.7. In the bipartite graph below, with $A = \{a_1, a_2, a_3, a_4\}$, $B = \{b_1, b_{2,3}, b_4\}$, a perfect matching is shown using dashed lines.



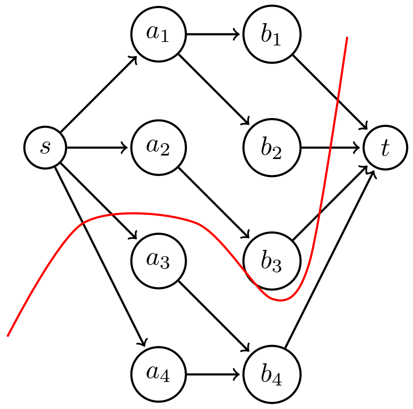
We can illustrate the condition $|\Gamma(J)| \geq |J| \forall J \subset A$. For $|J| = 1$, that is J is $\{a_i\}$ for some i, each vertex in A has at least 1 adjacent vertex in B. For $|J| = 2$, that is $J \in \{\{a_1, a_2\}, \{a_1, a_3\}, \{a_1, a_4\}, \{a_2, a_3\}, \{a_2, a_4\}, \{a_3, a_4\}\}$, then $|\Gamma(J)| = 3$. The same computations can be checked for $|J| = 3, 4$.

Proof. ($\Longrightarrow$) Suppose we have a perfect matching. The edges of a perfect matching provide a unique neighbour for each node in J. There maybe other edges and thus other adjacent nodes in B, but there are at least those provided by the matching, which makes sure that $|J| \leqslant |\Gamma(J)|$.
($\Longleftarrow$) Suppose by contradiction that G has no perfect matching but $|J| \leqslant |\Gamma(J)|$. We frame the matching problem in terms of network flow by turning G into a network N as follows:

- Direct all edges in G from A to B and give them capacity ∞.
- Add a node s and an edge (s, a) $\forall$ $a \in A$ with capacity 1.
- Add a node t and an edge (b, t) $\forall$ $b \in B$ with capacity 1.



Now G has a perfect matching $\iff$ the max flow in N is n $\iff$ the min cut in N is n. Our contradiction assumption thus implies that the min-cut is $< n$. Consider a min-cut (S, T) and let $J = S \cap A$. Then all of the edges from s to $A \backslash J$ cross the cut. These edges have total capacity $|A \backslash J|$ (since each edge has capacity 1).



The edges $(a_3, b_1), (a_4, b_2)$ were removed from the previous example to remove the perfect matching

Now all neighbours of J in G must also lie in S, that is $\Gamma(J) \subset S$, or else an edge of capacity ∞ would cross the cut. Then all of the edges from the nodes $\Gamma(J)$ to t cross the cut. These edges have total capacity $|\Gamma(J)|$. We then have:

$$\begin{cases} |J| + |A \backslash J| = |A| = n \\ |A \backslash J| + |\Gamma(J)| \leqslant C(S,T) < n \end{cases}$$

$$\implies n - |J| + |\Gamma(J)| < n \iff |\Gamma(J)| < |J|$$

which is a contradiction. □

3.3 The Min Cost Flow Problem and some Applications

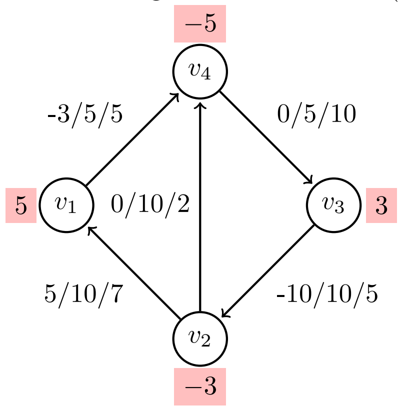
Definition 3.7. Let $G = (V, E)$ be a directed graph, and let

$$\begin{array}{ll} b : E \to \mathbb{R} & \text{(lower capacity)} \\ c : E \to \mathbb{R} & \text{(upper capacity, or just capacity)} \\ \gamma : E \to \mathbb{R} & \text{(cost function)} \\ d : V \to \mathbb{R} & \text{(demand function)} \end{array}$$

be functions (note that they take value in the whole of $\mathbb{R}$). Furthermore, the demand function is such that $\sum_{v \in V} d(v) = 0$. Then G together with the functions b, c, γ, d is called a *minimum cost flow network*.

When $d(v) > 0$, we refer to v as a *supply* node, when $d(v) < 0$, we refer to v as a *demand* node, and when $d(v) = 0$, we say that v is a transit node.

Example 3.8. Here is an example of minimum cost flow network. The demands of the nodes are written next to the nodes. We notice that $\sum_{v\in V} d(v) = 0$. The labels on the edges are of the form $b(e)/c(e)/\gamma(e)$.



Definition 3.8. A *flow* in such a network is a map $f : E \to \mathbb{R}$ with

1. $b(e) \le f(e) \le c(e)$ for all $e \in E$,
2. $d(v) = \sum_{u\in O(v)} f(v,u) - \sum_{u\in I(v)} f(u,v)$ for all $v \in V$[1].

The *cost* of such a flow is $\gamma(f) = \sum_{e\in E} \gamma(e)f(e)$.

Problem 5. The *min-cost-flow problem* consists of finding a flow in a minimum cost flow network of minimum cost.

We will show first that the maximum flow problem is a special case of the min-cost-flow problem. Indeed, let $G = (V,E)$ be a network with source s and sink t, and edge capacities given by a function c. Construct the min-cost flow network $G' = (V,E')$ where $E' = E \cup \{(t,s)\}$. We specify the 4 functions of a min-cost flow network G':

- $b'(e) = 0$ for all $e \in E'$,
- $c'(e) = c(e)$ for all $e \in E$ and $c'(t,s) = \sum_{u\in O(s)} c(s,u)$,
- $\gamma'(e) = 0$ for all $e \in E$ and $\gamma'(t,s) = -1$,
- $d'(v) = 0$ for all $v \in V$.

The functions are clearly defined on $G = (V,E)$: the lower bound is 0 for every edge in E in a max flow problem, so is the demand at every vertex but for the source and the sink. Then each edge is assigned its capacity in G. The only things to discuss are the addition of (t,s), the choice of capacity and cost for this edge, as well as the choice of demand for s and t.

[1]You may find $d(v) = -\sum_{u\in O(v)} f(v,u) + \sum_{u\in I(v)} f(u,v)$, both definitions are found.

Intuitively, if we want to maximize the flow between s and t using a min-cost formulation, this means that decreasing the cost must increase the flow. By introducing a new edge (t,s) with cost $\gamma'(t,s) = -1$, we push the flow to use this new edge as much as possible, since going through it decreases the cost (all the other costs are 0: $\gamma'(e) = 0$ for all $e \in E$). But now, the demand at the source s is 0, so the edge (t,s), which is the only incoming edge of s, must carry as much flow as the source can output, and similarly at the sink t whose demand is also 0, the edge (t,s) being its only outgoing edge, it must take in as much flow as possible.

Lemma 3.7. *With G and G' as defined above, let f be a flow in G and f' be defined by*

$$f'(e) = f(e)\ \forall e \in E,\ \ f'(t,s) = \sum_{u \in O(s)} f(s,u).$$

Then f is a maximum flow in G $\iff$ f' is a minimum cost flow in G'.

Proof. First we check that f' is indeed a flow in G'.

- We have that f is a flow in G, thus $0 \le f(e) \le c(e)$ for all $e \in E$, and f' satisfies $0 \le f'(e) = f(e) \le c(e) = c'(e)$ for all $e \in E$. Then $f'(t,s) = \sum_{u \in O(s)} f(s,u) \le c'(t,s) = \sum_{u \in O(s)} c(s,u)$ so f' is feasible.
- Then for every vertex v in V which is neither the source nor the sink, by flow conservation of f, and since $f(e) = f'(e)$ for all $e \in E$:

  $$\sum_{u \in O(v)} f'(v,u) - \sum_{u \in I(v)} f'(u,v) = 0 = d(s).$$

  For s, since (t,s) is its only incoming edge, and by definition of $f'(t,s)$:

  $$\sum_{u \in O(s)} f(s,u) - f'(t,s) = 0 = d(v).$$

  For t, since (t,s) is its only outgoing edge, by definition of $f'(t,s)$ and recalling that the flow that goes out of the source must reach the sink (see Exercise 25) :

  $$f'(t,s) - \sum_{u \in I(t)} f(u,t) = f'(t,s) - \sum_{u \in O(s)} f(s,u) = 0 = d(t).$$

Now the cost of the flow f' is

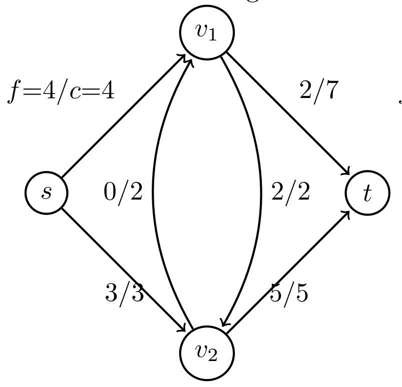
$$\gamma(f') = \sum_{e \in E'} \gamma(e) f'(e) = \gamma(t,s) f'(t,s) = -f'(t,s),$$

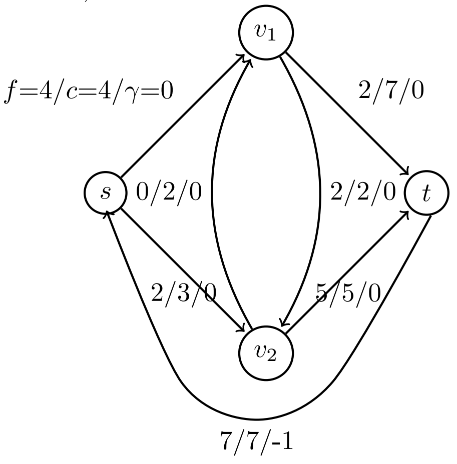
that is

$$\gamma(f') = -\sum_{u \in O(s)} f(s,u)$$

and minimizing $\gamma(f')$ maximizes $\sum_{u \in O(s)} f(s,u)$. □

Example 3.9. On the left hand-side, a graph G is shown with a maximal flow. On the right hand-side, its corresponding min-cost flow network is shown. The capacity of (t, s) is $\sum_{u\in O(s)} c(s, u)$. To minimize the cost, the edge (t, s) should be used as much as possible, constrained by its capacity which is the maximal amount of flow that can go out of the source, and come in the sink.





Next we will see that finding a shortest path is a special case of the min-cost flow too.

Let $G = (V, E)$ be a directed weighted graph with weight function $w : E \to \mathbb{R}_{>0}$, and consider two vertices s, t in V. A path $P \subseteq E$ from s to t with minimum weight $\sum_{e\in P} w(e)$ is called a *shortest path* from s to t.

We define a corresponding min-cost flow network $G' = (V, E)$ by using the same vertices and edges as that of G, and by adding the following functions:

- $b(e) = 0$ for all $e \in E$,
- $c(e) = 1$ for all $e \in E$,
- $\gamma(e) = w(e)$ for all $e \in E$,
- $d(v) = 0$ for all $v \in V\backslash\{s, t\}$, $d(s) = 1$, $d(t) = -1$.

Lemma 3.8. *Let f be a minimum cost flow in G' as defined above, such that all flow values are integral (in fact they are only 0 or 1). Then the edges with $f(e) = 1$ form a shortest path from s to t in G.*

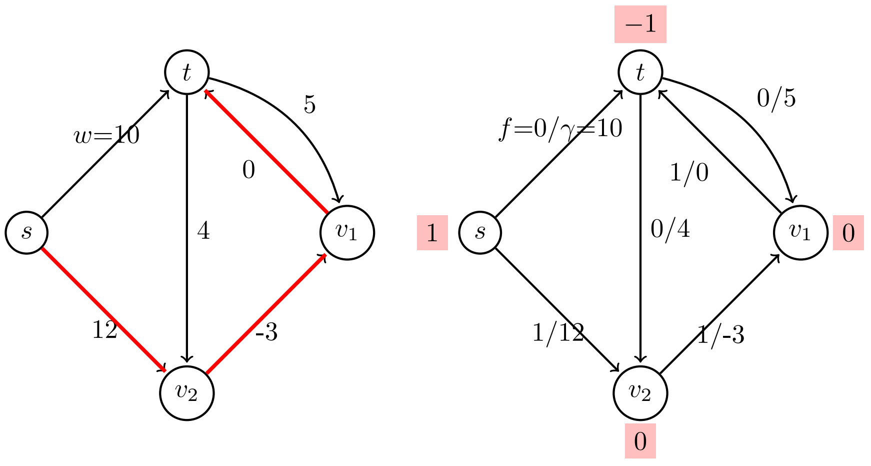
Proof. We first note that all nodes have a demand of 0, but for the start s and end t of the path, whose demands are $d(s) = 1$ and $d(t) = -1$. Thus there is a flow of 1 that has to go from s to t. Then the capacity of every edge is 1, which means that every edge is used either once or none. Also since the flow is integral, every visited vertex has at most one incoming edge, and one outgoing edge (if the flow was not integral, you could have 0.5 going of a node on two different edges). Now to minimize the cost

$$\sum_{e\in E} \gamma(e)f(e) = \sum_{e\in E} w(e)f(e),$$

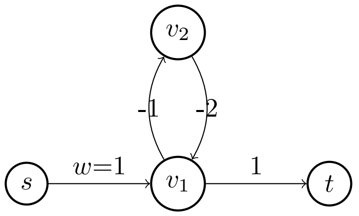
this flow of 1 will use edges with the lowest cost (the flow will be either 1 if the edge is used, or 0 else), thus finding the shortest path. Note that the output is really a path in that none of the vertices are repeated. This is because repeating a vertex means there is a cycle, and since the weights are positive, a cycle would increase the cost. □

In the above proof, we use the fact that the weights are positive. If we have negative edges, it is possible to find a shortest path as a min-cost flow, assuming that there is no cycle whose sum of weights is negative or zero. In that case, the same argument holds: adding a cycle would add a larger cost and thus the algorithm will avoid the cycle.

Example 3.10. This example shows that even with negative weights (this graph has no cycle whose sum of costs is negative), the shortest path problem can be solved using the min-cost flow problem. On the left-hand side, a graph with a shortest path from s to t is given. On the right hand-side, the equivalent min-cost flow problem is shown, whose cost is 9.



Example 3.11. The network below contains the cycle v_1, v_2, v_1 which has weight -3. The shortest path is given by s, v_1, t. Indeed, using the cycle v_1, v_2, v_1 means that v_1 is used twice, so we do not get a path. However if we were asked for a flow that minimizes the weights, we would enter the cycle, this would not create a path since v_1 would be repeated, and the iterations may not even terminate (if we do not specify a capacity).



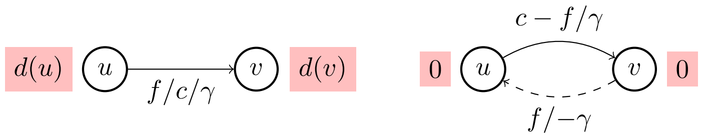
3.4 The Cycle Cancelling Algorithm

We next discuss an algorithm that actually solves the min-cost flow network problem. Similarly to Ford-Fulkerson algorithm, it relies on the notion of residual graph, only we need to define residual graph in our new context: we know how to define a residual graph that takes into account the flow and the capacity of the edges (see Definition 3.4), but we need to add what happens to the cost. The differences between the two definitions are highlighted.

Definition 3.9. Given a flow f on a graph (min-cost flow network) $G = (V, E)$, its *residual graph* G_f has V for vertices whose demands are set to 0, and

(1) edges have capacities $c(v, w) - f(v, w)$, only edges with $c_f := c(v, w) - f(v, w) > 0$ are shown, they have a cost of $\gamma_f(v, w) = \gamma(v, w)$, and they are shown in the same direction as (v, w),

(2) if $f(v, w) > 0$, place an edge with capacity $c_f(v, w) = f(v, w)$ in the opposite direction of (v, w), with a cost $\gamma_f(v, w) = -\gamma(v, w)$.



While augmenting paths were the key ingredient to compute max-flows, for min-cost flows, we look at negative cost cycles.

Definition 3.10. In a minimum cost flow network, a *negative cost cycle* is a cycle C whose edges have a cost γ such that $\sum_{e \in C} \gamma(e) < 0$.

We first give the algorithm, we will prove next why it works. For that, we will make the following assumptions:

- The lower capacity b is zero (and not written anymore): it is always possible to replace a min-cost-flow network where b is arbitrary by an equivalent min-cost-flow network where $b(e) = 0$ for all $e \in E$ (see Exercise 31).
- All upper capacities $c(e)$ are finite: if $c(e)$ is infinite for some edge e (which we can useful to describe a problem formulation), one can prove that there exists an optimal flow which is upper bounded by a quantity that only depends on the demands and the finite capacities of the graph, and this upper bound can be used as a finite capacity to replace $c(e)$.
- All costs are non-negative: it can be shown that it is possible to replace a min-cost flow network where γ can be negative (e.g. in G' in Lemma 3.7) by an equivalent min-cost-flow network where $\gamma(e) \geq 0$ for all e.
- All upper capacities c, costs γ and demands d are integral: arguments saying demands and/or flows are decreasing at each iteration and thus will reach 0 (as used later) do not hold if we deal with real values.

- Networks have no directed cycle of negative cost and infinite capacity: this scenario is ill-defined (see Example 3.11), no matter how one can transform such a network.

Algorithm 7 Cycle Cancelling

```
**Input:** $G = (V, E)$ a network, with capacity $c : E \to \mathbb{Z}_{\geq 0}$, $|c| < \infty$, demand $d : V \to \mathbb{Z}$, and cost $\gamma : E \to \mathbb{Z}_{\geq 0}$
**Output:** a minimum cost flow $f^*$, or $\varnothing$.
1: $f \leftarrow$ FeasibleFlow$(G, d, c)$
2: if $(f = \varnothing)$ do
3:     return $\varnothing$.
4: Compute the residual graph $G_f$ with cost $\gamma_f$ and capacity $c_f$.
5: Find a negative cost cycle $W$ in $G_f$, set $W = \varnothing$ if none exists.
6: while $(W \neq \varnothing)$ do
7:     Set $df = \min_{e \in W} c_f(e)$ in $G_f$ along $W$.
8:     Update $f$ in $G$: set $f(u, v) = f(u, v) + df$ for $(u, v) \in W$ if $(u, v) \in E$
       and $f(v, u) = f(v, u) - df$ for $(u, v) \in W$ if $(v, u) \in E$.
9:     Rebuild the residual network $G_f$, with cost $\gamma_f$ and capacity $c_f$.
10:    Find a negative cost cycle $W$ in $G_f$, set $W = \varnothing$ if none exists.
```

The idea is to start with a feasible flow in G (a flow that satisfies Definition 3.8)[2], then try to find a negative cost cycle in G_f. When no such a cycle exists, the claim is that we have an optimal flow. Otherwise, we augment the flow along the cycle, decrease the cost of the flow, and repeat. To find a negative cycle, one may use Floyd-Warshall algorithm (see Algorithm 9).

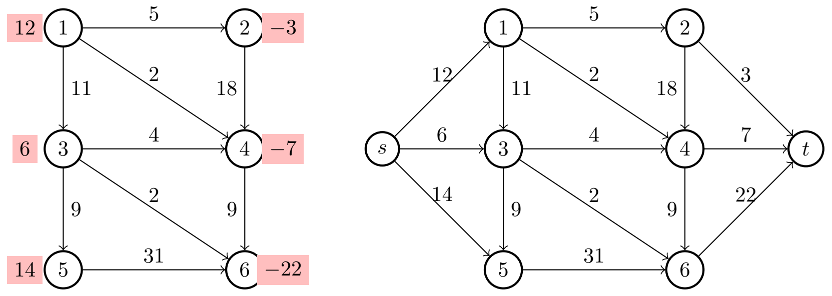
We still need to answer two questions: how do we find a feasible flow, and why does the cycle cancelling algorithm work.

Algorithm 8 FeasibleFlow

```
**Input:** $G = (V, E)$ a network, with capacity $c : E \to \mathbb{Z}_{\geq 0}$, $|c| < \infty$, demand $d : V \to \mathbb{Z}$, and cost $\gamma : E \to \mathbb{Z}_{\geq 0}$
**Output:** a feasible flow $f$, or $\varnothing$.
1: $V' \leftarrow V \cup \{s, t\}$.
2: $E' \leftarrow E \cup \{(s, v),\ d(v) > 0\}$ and set $c'(s, v) = d(v)$.
3: $E' \leftarrow E' \cup \{(v, t),\ d(v) < 0\}$ and set $c'(v, t) = -d(v)$.
4: Create a graph $G' = (V', E')$, with capacity $c'$ and $c'(e) = c(e)$ for all $e \in E$.
5: Find $f^*$ that solves the max flow problem from $s$ to $t$ in $G'$.
6: if ($f^*$ saturates the source edges) do
7:     Return $f^*|_E$.
8: else:
9:     Return $\varnothing$.
```

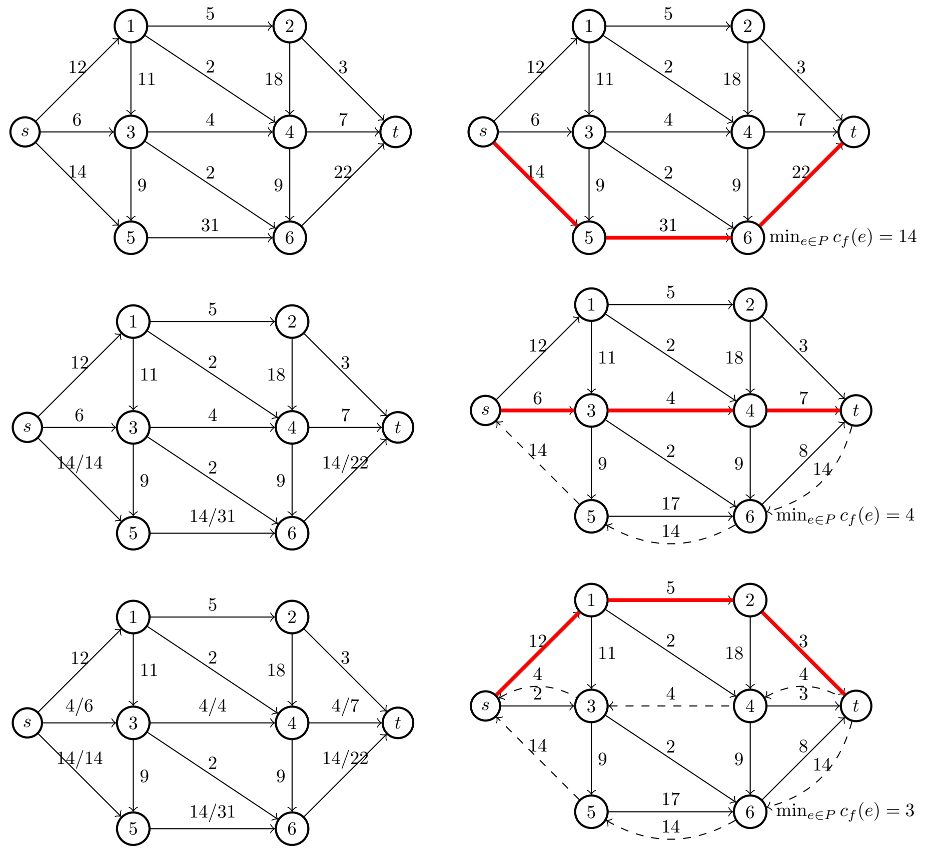
[2]For max flow problems, we use "feasible" for the condition $0 \leq f(e) \leq c(e)$, for min cost flow problems, we use "feasible" for both properties.

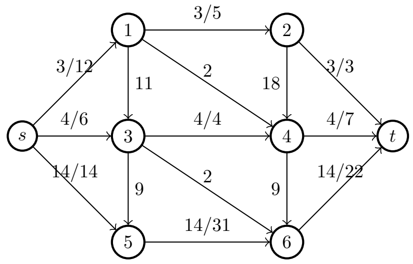
Example 3.12. Let us try to find a feasible flow for the network on the left below.

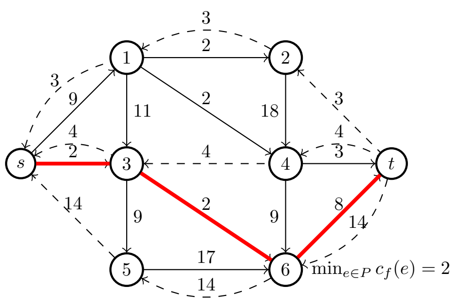


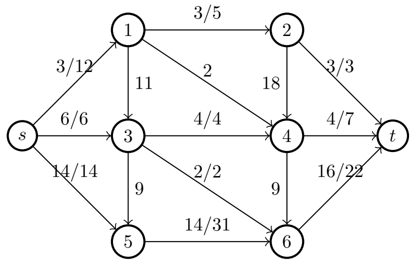
To start with, we create a graph $G' = (V', E')$ where $V' = V \cup \{s, t\}$, $E' = E \cup \{(s, v),\ d(v) > 0\} \cup \{(v, t),\ d(v) < 0\}$ and $V' = V \cup \{s, t\}$. Then the capacities are $c'(e) = c(e)$ for all $e \in E$, $c'(s, v) = d(v)$ and set $c'(v, t) = -d(v)$.

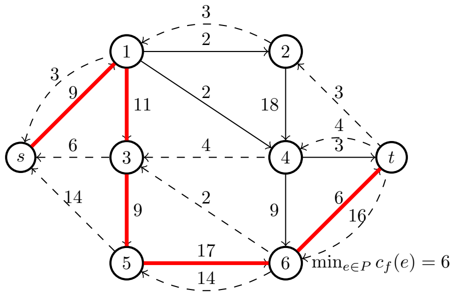
Then to solve the max-flow problem in the network G', we apply Ford-Fulkerson algorithm. We show the graph and its updates on the left, and the corresponding residual graphs with paths from s to t on the right.


$\min_{e \in P} c_f(e) = 14$
$\min_{e \in P} c_f(e) = 4$
$\min_{e \in P} c_f(e) = 3$

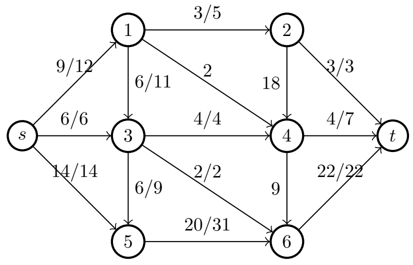


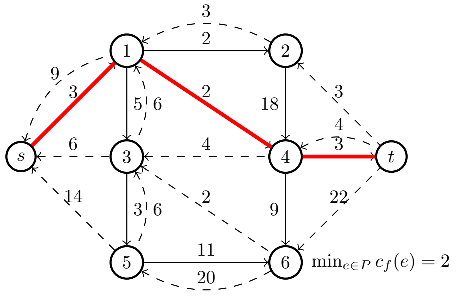


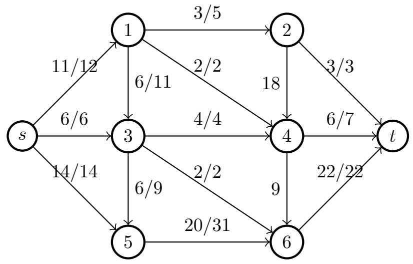


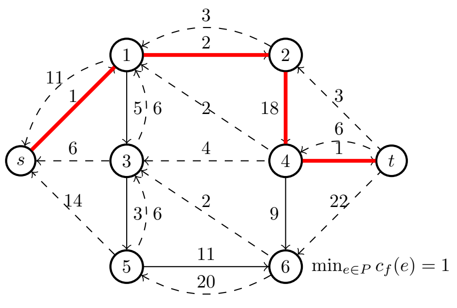


At this point, we observe that s has only one outgoing edge which is not saturated, namely the one from s to 1.

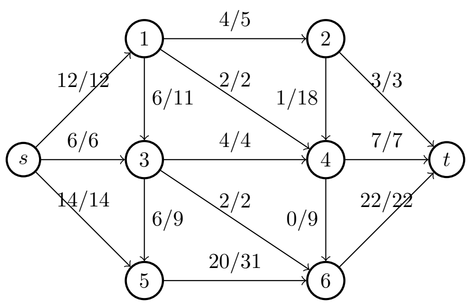




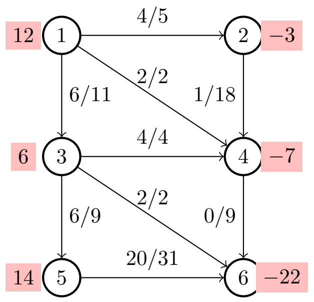




This gives us a last update:



We see that the flow obtained saturates the outgoing edges of s. We now remove s and t and the edges connected them. We can check that the flow obtained is feasible.



Let us prove that the FeasibleFlow algorithm actually works.

Lemma 3.9. *The original network G has a feasible flow if and only if every maximal flow of G' saturates all the source edges. Furthermore, if f^* is a maximal flow of G', then the restriction $f^*|_E$ to the edges of G is a feasible flow.*

Proof. Recall that a feasible flow for the min-cost-flow problem must satisfy (1) $b(e) \leq f(e) \leq c(e)$, and (2) $\sum_u f(v,u) - \sum_u f(u,v) = d(v)$. We may assume $b(e) = 0$ for all e.

($\Leftarrow$) Consider an arbitrary maximum flow f^* of G' which saturates all source edges. We will show that G has a feasible flow, given by $f^*|_E$ (this also proves the second part of the lemma statements). Clearly $f^*|_E$ satisfies (1) by definition of flow in G'. Then consider vertices v such that $d(v) > 0$ in the original network G. They are connected to s in G' with capacity $d(v)$, so since every such an edge saturates, this means that the flow on (s,v) is $d(v)$, thus $f(s,v)+\sum_{u\neq s} f(u,v) = \sum_u f(v,u)$ by definition of flow in G', and $f^*|_E$ satisfies (2) for these vertices. Next consider vertices such that $d(v) < 0$. Since f saturates the edges out of the source, by definition of demand, it also saturates the edges entering the sink. Then $\sum_u f(u,v) = \sum_{u\neq t} f(v,u) + f(v,t)$ by definition of flow in G', that is $\sum_{u\neq t} f(v,u) - \sum_u f(u,v) = -f(v,t) = d(v)$ and $f^*|_E$ satisfies (2) for these vertices.

($\Rightarrow$) Suppose we have a feasible flow for G, and a maximal flow for G'. When the manipulation that transforms the original network into a max flow problem is done, the demands have been put as edge capacities between the source and nodes of positive demand (and between nodes of negative demands and the sink with a change of sign), so a maximal flow can at most saturate the source edges, and it will do so since the flow is feasible in G. $\square$

This lemma also tells us that if we cannot find a feasible flow, then we cannot solve the min-cost-flow problem.

Next, we prove a first result that shows why it is interesting to work with a residual graph. For an edge $e = (u, v) \in E$, we write $\tilde{e} = (v, u)$, that is $\tilde{e}$ is the backward edge obtained by changing the direction of e.

Theorem 3.10. *Let f^o be a feasible flow of a network G, and let G_{f^o} be its residual graph. Then a flow f is feasible in G if and only if the flow f' defined by*

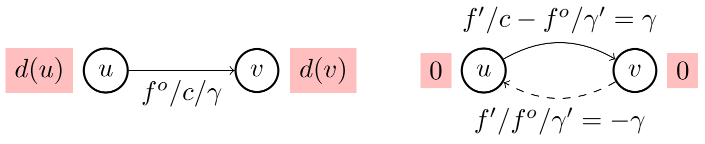
$$\begin{cases} f'(e) = f(e) - f^o(e),\ f'(\tilde{e}) = 0 & \text{if } f(e) \geq f^o(e) \\ f'(\tilde{e}) = -f(e) + f^o(e),\ f'(e) = 0 & \text{if } f(e) < f^o(e) \end{cases}$$

is a feasible flow in G_{f^o}. Furthermore

$$\sum_{e \in E} \gamma(e) f(e) = \sum_{e \in E(G_{f^o})} \gamma'(e) f'(e) + \sum_{e \in E} \gamma(e) f^o(e)$$

where γ is the cost in G and γ' is the cost in G_{f^o}.

This theorem is looking at one update of G with a feasible flow f^o, from which a residual graph G_{f^o} is built. It tells when a feasible flow f' in G_{f^o} will correspond to a feasible flow f in G, and how the cost of the flow f' in G_{f^o} will update the cost of the flow f in G with respect to the cost given by the flow f^o.



Proof. ($\Rightarrow$) Assume that f is a feasible flow, that is (1) $0 \leq f(e) \leq c(e)$ and (2) $\sum_u f(v, u) - \sum_u f(u, v) = d(v)$. We need to check that f' is feasible. We start with (1). Clearly $f'(\tilde{e}) = 0$ and $f'(e) = 0$ respectively satisfy (1).

- If $f(e) \geq f^o(e)$, then $f'(e) = f(e) - f^o(e) \geq 0$ and $f'(e) = f(e) - f^o(e) \leq c(e) - f^o(e) = c_{f^o}(e)$ since f is feasible and e is forward.
- If $f(e) < f^o(e)$, then $f'(\tilde{e}) = -f(e) + f^o(e) > 0$ and $f'(\tilde{e}) \leq f^o(e)$, the residual capacity of the backward edge $\tilde{e}$. Since f is feasible, $f(e) \geq 0$ and $f^o(e) > 0$, thus the backward edge $\tilde{e}$ appears in the residual graph.

Then we check (2) for f' in G_{f^o}, recalling that $d(v) = 0$ in G_{f^o}. For every v, we have, recalling that an edge e in G_{f^o} is either a forward edge in G, or a backward version of an edge in G:

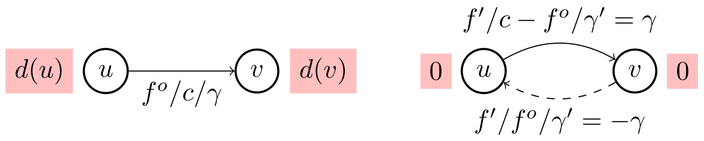
$$\begin{aligned}
&\sum_{u,(v,u)\in E(G_{f^o})} f'(v,u) - \sum_{u,(u,v)\in E(G_{f^o})} f'(u,v) && (3.2)\\
=& \sum_{u,(v,u)\in E(G)} f'(v,u) + \sum_{u,(u,v)\in E(G)} f'(v,u) - \sum_{u,(u,v)\in E(G)} f'(u,v) - \sum_{u,(v,u)\in E(G)} f'(u,v) \\
=& \sum_{u,(v,u)\in E(G)} (f'(v,u) - f'(u,v)) - \sum_{u,(u,v)\in E(G)} (f'(u,v) - f'(v,u)) && (3.3)\\
=& \sum_{u,(v,u)\in E(G)} (f(v,u) - f^o(v,u)) - \sum_{u,(u,v)\in E(G)} (f(u,v) - f^o(u,v))
\end{aligned}$$

because in the first sum, either $f'(v,u) = f(v,u) - f^o(v,u)$ and $f'(u,v) = 0$, or $f'(v,u) = 0$, and $-f'(u,v) = f(v,u) - f^o(v,u)$. Similarly, in the second sum, either $f'(u,v) = f(u,v) - f^o(u,v)$ and $f'(v,u) = 0$, or $f'(u,v) = 0$, and $-f'(v,u) = f(u,v) - f^o(u,v)$. But now, since f is feasible

$$\sum_{u,(v,u)\in E(G)} f(v,u) - \sum_{u,(u,v)\in E(G)} f(u,v) = d(v).$$

Since f^o is also feasible, (2) is proven by noting that

$$-\sum_{u,(v,u)\in E(G)} f^o(v,u) + \sum_{u,(u,v)\in E(G)} f^o(u,v) = -d(v).$$



($\Leftarrow$) For the converse, assume that f' is a feasible flow.

- If $f'(\tilde{e}) = 0$, then $f(e) = f'(e) + f^o(e)$, and $f(e) \geq 0$. Also since f' is a feasible flow in G_{f^o}, $f(e) \leq c_{f^o}(e) + f^o(e) = (c(e) - f^o(e)) + f^o(e) = c(e)$.
- If $f'(e) = 0$, then $f(e) = f^o(e) - f'(\tilde{e})$, and since f' is feasible in G_{f^o}, it must be less than the residual capacity which for a backward edge is $f^o(e)$ and $f(e) \geq 0$. Also, $f(e) \leq f^o(e) \leq c(e)$ since f^o is a feasible in G.

We then need to check that $\sum_u f(v,u) - \sum_u f(u,v) = d(v)$. We write $f(e) = f'(e) - f'(\tilde{e}) + f^o(e)$ since either $f'(e) = 0$ or $f'(\tilde{e}) = 0$, so that

$$\begin{aligned}
&\sum_{u,(v,u)\in E(G)} f(v,u) - \sum_{u,(u,v)\in E(G)} f(u,v) \\
=& \sum_{u,(v,u)\in E(G)} (f'(v,u) - f'(u,v) + f^o(v,u)) - \sum_{u,(u,v)\in E(G)} (f'(u,v) - f'(v,u) + f^o(u,v))
\end{aligned}$$

Since f^o is feasible:

$$\sum_{u,(v,u)\in E(G)} f^o(v,u) - \sum_{u,(u,v)\in E(G)} f^o(u,v) = d(v).$$

This proves what we wanted since

$$\sum_{u,(v,u)\in E(G)} (f'(v,u) - f'(u,v)) - \sum_{u,(u,v)\in E(G)} (f'(u,v) - f'(v,u)) = 0$$

using (3.3), which is equal to (3.2), which is 0 once we know f' is feasible.

Finally, we compute the cost of the flow f as a function of the cost of the flows f' and f^o. The cost γ' of f' is computed in G_{f^o}, and $\gamma' = \pm\gamma$ depending on whether the edge is forward or backward. Thus

$$\gamma'(e)f'(e) + \gamma'(\tilde{e})f'(\tilde{e}) = \gamma(e)(f'(e) - f'(\tilde{e})) = \gamma(e)(f(e) - f^o(e))$$

since $f(e) = f'(e) - f'(\tilde{e}) + f^o(e)$. Thus $\gamma(e)f(e) = \gamma'(e)f'(e) + \gamma'(\tilde{e})f'(\tilde{e}) + \gamma(e)f^o(e)$. We conclude the proof by summing over the edges $e \in E$.

$$\begin{aligned}\sum_{e\in E}\gamma(e)f(e) &= \sum_{e\in E}\gamma'(e)f'(e) + \gamma'(\tilde{e})f'(\tilde{e}) + \gamma(e)f^o(e)\\ &= \sum_{e\in E}\gamma'(e)(f'(e) - f'(\tilde{e})) + \sum_{e\in E}\gamma(e)f^o(e)\end{aligned}$$

and $\sum_{e\in E}\gamma'(e)(f'(e) - f'(\tilde{e})) = \sum_{e\in E(G_{f^o})}\gamma'(e)f'(e)$ since edges in G_{f^o} come as both forward and backward edges of G. □

The above result states how the cost of a flow changes in G based on the cost of a flow in its corresponding residual graph. The next result describes the role of cycles in a flow of G.

Theorem 3.11. [Flow Decomposition Theorem.] *Consider a graph $G = (V,E)$ with $|V| = n$ vertices and $|E| = m$ edges. Every flow f can be decomposed into cycles and paths, such that:*

1. *Every directed path with nonzero flow connects a supply node (with positive demand) to a demand node (with negative demand).*
2. *At most $n + m$ paths and cycles have nonzero flow, and out of these, at most m cycles have nonzero flow.*

Proof. Suppose v_0 is a supply vertex, that is $d(v_0) > 0$. Then there is some edge $e = (v_0, v_1)$ with nonzero flow (otherwise the flow is not feasible). If v_1 is a demand vertex, then we stop, we found a path $P = (v_0, v_1)$. Otherwise, v_1 is either another supply vertex, or transit vertex, and the property $\sum_u f(v_1,u) - \sum_u f(u,v_1) = d(v_1)$ ensures that there is another edge (v_1, v_2) with nonzero flow. We repeat this argument until either a demand node is reached, or we encounter a previously visited node. Surely, one of these two cases must occur within n steps, since n is the number of vertices. Thus either we obtain a directed path P from a supply node to a demand node, or we obtain a directed cycle W. In both cases, the path or the cycle only consists of nonzero flow edges.

If we obtain a directed path P from v_0 to v_k, set

$$\delta(P) = \min\{d(v_0), -d(v_k), \min_{e \in P} f(e)\}$$

and update $d(v_0) \leftarrow d(v_0) - \delta(P)$, $d(v_k) \leftarrow d(v_k) + \delta(P)$, and $f(e) \leftarrow f(e) - \delta(P)$, for every $e \in P$.

If instead we obtain a directed cycle W, then set

$$\delta(W) = \min_{e \in W} f(e)$$

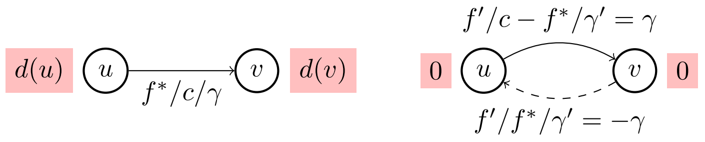
and update $f(e) \leftarrow f(e) - \delta(W)$, for every $e \in W$.

We repeat the above process with the newly obtained problem, until there are no more supply nodes, i.e., $d(v) = 0$ for all vertices v (if supply nodes are gone, by the definition of demand, the demand nodes must be gone too). This will happen because whenever a path is found, the supply demands are decreased, so they will eventually reach zero. Then we select any node with one outgoing edge with nonzero flow as a starting point, and repeat the procedure. Since $d(v) = 0$, we will find a directed cycle. The process stops when $f = 0$.

The original flow is the sum of flows on the paths and cycles we found. Then each time we find a directed path, the update $d(v_0) \leftarrow d(v_0) - \delta(P)$, $d(v_k) \leftarrow d(v_k) + \delta(P)$, and $f(e) \leftarrow f(e) - \delta(P)$, will send either a supply node, a demand node, or an edge to 0. Similarly, each time we find a directed path, the flow on some edge will be updated to 0. Therefore the process terminates after identifying at most $n + m$ cycles and paths, and identify at most m cycles. $\square$

We finally have the result that justifies the cycle cancelling algorithm.

Theorem 3.12. *A feasible flow f^* is an optimal solution of the minimum cost flow problem if and only if G_{f^*} contains no negative cost directed cycle.*



Proof.

($\Rightarrow$) Assume that f^* is an optimal flow, but that G_{f^*} contains a negative cost directed cycle W of cost $\gamma'(W) = \sum_{e \in W} \gamma'(e) < 0$. We will get a contradiction to the optimality of f^*, intuitively because we can augment the flow f' along W, which will decrease its cost value.

Formally, consider the flow f' along W in G_{f^*} given by: $f'(e) = 0$ for $e \notin W$, and for $e \in W$, use the maximal flow obtained by setting $f'(e)$ for all $e \in W$ to be the minimum residual capacity of the cycle W (this flow is surely feasible). Then by Theorem 3.10, we get a feasible flow f in G such that:

$$\sum_{e \in E} \gamma(e) f(e) - \sum_{e \in E} \gamma(e) f^*(e) = \sum_{e \in E(G_{f^*})} \gamma'(e) f'(e) = \sum_{e \in W} \gamma'(e) f'(e) < 0$$

since $f'(e) = 0$ for $e \notin W$. Also $f'(e)$ is constant, it was set to be the minimum residual capacity for every $e \in W$, thus it can be taken out of the sum, and we can then use that the cycle has a negative cost: $\gamma'(W) < 0$. Then

$$\sum_{e \in E} \gamma(e) f(e) < \sum_{e \in E} \gamma(e) f^*(e),$$

a contradiction to the optimality of f^*.

($\Leftarrow$) Now assume that f^* is a feasible flow and that G_{f^*} contains no negative cost directed cycle. We want to show that f^* is optimal. Let f^o be any feasible flow. We will show that f^* has a lesser cost than f^o.

By Theorem 3.10, with f^* a feasible flow with residual graph G_{f^*}, since f^o is also feasible in G, we can find a feasible flow f' in G_{f^*}. Now applying Theorem 3.11 to f' in G_{f^*}, we decompose f' into paths and cycles. But in G_{f^*}, all nodes have demands 0, thus the flow f' is only composed of cycles. But since G_{f^*} contains no negative cost directed cycle, the cost of any cycle must be nonnegative, and thus the cost of the flow over these cycles must be positive, and in fact, equal to

$$\sum_{e \in E(G_{f^*})} \gamma'(e) f'(e) \geq 0.$$

By Theorem 3.10, the cost in G is

$$\sum_{e \in E} \gamma(e) f^o(e) - \sum_{e \in E} \gamma(e) f^*(e) = \sum_{e \in E(G_{f^*})} \gamma'(e) f'(e) \geq 0,$$

that is

$$\sum_{e \in E} \gamma(e) f^o(e) \geq \sum_{e \in E} \gamma(e) f^*(e)$$

thus f^* is an optimal flow (that is, with minimal cost). $\square$

We conclude this chapter by discussing Floyd-Warshall algorithm. The goal of this algorithm, proposed in 1962, is to compute all shortest paths of a weighted directed graph G. It is an example of so-called *dynamic programming.* Given a path $P = (1, \dots, l)$, any vertex in P different from $1, l$ is called an *intermediate vertex.* We denote by w_{ij} the weight $w(i,j)$ of the directed edge (i,j). The weight 0 is given to w_{ii} and the weight ∞ is a convention if there is no edge between i and j. We form an $n \times n$ matrix W whose coefficients are w_{ij} for $n = |V|$. We then denote by $d_{ij}^{(k)}$ the weight of a shortest path from i to j for which all intermediate vertices are in the set $\{1, \dots, k\}$. We similarly store $d_{ij}^{(k)}$ in a matrix $D^{(k)}$. Now to compute $d_{ij}^{(k)}$ knowing $d_{ij}^{(k-1)}$, observe that there are two ways for a shortest path to go from i to j:

- not using k: then only intermediate vertices in $\{1, \dots, k-1\}$ are visited, and the weight of a shortest path is $d_{ij}^{(k-1)}$.

- using k: observe that a shortest path does not pass through the same vertex twice, so k is visited exactly once. To ensure that the algorithm actually does that, we need the assumption that there is no directed cycle whose sum of weights is negative. Since this means we go from i to k, and then from k to j, we are then using a shortest path in both cases, so the length with be $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

We then recursively define

$$d_{ij}^{(k)} = \begin{cases} w_{ij},\ k=0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}),\ k \geq 1. \end{cases}$$

To keep track of paths, denote by $\pi_{ij}^{(k)}$ the predecessor of j on a shortest path from i, with intermediate vertices in $\{1,\dots,k\}$:

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)},\ d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)},\ d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

with $\pi_{ij}^{(0)} = i$ for $i \neq j$ and $w_{ij} < \infty$ and *NIL* if $i = j$ or $w_{ij} = \infty$. Indeed, the first condition means that k was not used, and we store the predecessor of j from i, while for the second condition, k was used and the path went from k to j, so we store the predecessor of j from k.

Algorithm 9 Floyd-Warshall algorithm

Input: $G = (V, E)$ a weighted directed graph with weight w and no directed cycle whose sum of weights is negative.

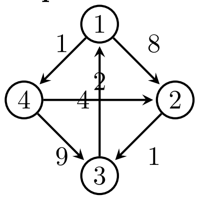
Output: $D^{(n)}$

```
1: $D^{(0)} \leftarrow W$.
2: for ($k = 1, \dots, n$) do
3:     Initialize $D^{(k)}$.
4:     for ($i = 1, \dots, n$) do
5:         for ($j = 1, \dots, n$) do
6:             $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
7:     Return $D^{(n)}$.
```

We notice that this involves storing the n matrices $D^{(k)}$, $k = 1, \dots, n$. We can instead use a single matrix D, initialized to be W as above. A matrix to contain the predecessors is initialized at the same time. Then replace $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$ by **if** $d_{ij} > d_{ik} + d_{kj}$, **then** $d_{ij} \leftarrow d_{ik} + d_{kj}$, $\pi_{ij} = \pi_{kj}$.

The path can be recovered as follows. To know the path between two nodes i and j: if π_{ij} is NIL, output i, j. Otherwise, repeat the procedure using both i and π_{ij}, this gives the path from i to the predecessor of j from i, and π_{ij} and j, this gives the rest of the path.

Example 3.13. Consider the following weighted graph.



Then the matrix D at $k = 0$ is given by

$$D = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix}, \ \Pi = \begin{bmatrix} NIL & 1 & NIL & 1 \\ NIL & NIL & 2 & NIL \\ 3 & NIL & NIL & NIL \\ NIL & 4 & 4 & NIL \end{bmatrix}.$$

At the first iteration, $k = 1$, we look at the condition $d_{ij} > d_{i1} + d_{1j}$, so for $i = 1$, $d_{1j} > d_{11} + d_{1j} = d_{1j}$ which is never true, for $i = 2$, $d_{2j} > d_{21} + d_{1j} = \infty$ which is never true either, the same holds for $i = 4$, so we just need to consider $i = 3$, that is $d_{3j} > d_{31} + d_{1j} = 4 + d_{1j}$. This gives

$$D = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix}, \ \Pi = \begin{bmatrix} NIL & 1 & NIL & 1 \\ NIL & NIL & 2 & NIL \\ 3 & \pi_{12} = 1 & NIL & \pi_{14} = 1 \\ NIL & 4 & 4 & NIL \end{bmatrix}.$$

At the second iteration, $k = 2$, we look at the condition $d_{ij} > d_{i2} + d_{2j}$, so for $i = 1$, $d_{1j} > d_{12} + d_{2j} = 8 + d_{2j}$ and for $i = 4$, $d_{4j} > d_{42} + d_{2j} = 2 + d_{2j}$, the other cases are never true. This gives

$$D = \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix}, \ \Pi = \begin{bmatrix} NIL & 1 & \pi_{23} = 2 & 1 \\ NIL & NIL & 2 & NIL \\ 3 & 1 & NIL & 1 \\ NIL & 4 & \pi_{23} = 2 & NIL \end{bmatrix}.$$

When $k = 3$, we have $d_{ij} > d_{i3} + d_{3j}$ which we consider for $i = 1, 2, 4$:

$$D = \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}, \ \Pi = \begin{bmatrix} NIL & 1 & 2 & 1 \\ 3 & NIL & 2 & 1 \\ 3 & 1 & NIL & 1 \\ 3 & 4 & 4 & NIL \end{bmatrix}.$$

Finally for $k = 4$, we have $d_{ij} > d_{i4} + d_{4j}$ which we consider for $i = 1, 2, 3$:

$$D = \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}, \ \Pi = \begin{bmatrix} NIL & 4 & 2 & 1 \\ 3 & NIL & 2 & 1 \\ 3 & 4 & NIL & 1 \\ 3 & 4 & 4 & NIL \end{bmatrix}.$$

We know from D that the shortest path from 1 to 3 is of weight 4, to know the path itself, we look at the path from 1 to $\pi_{1,3} = 2$, which gives 4, so we know that the path starts with 1,4,2, and then we look at the path from 2 to 3 which is 2, so the total path is 1,4,2,3. We can check that this path has indeed a weight of 4.

This algorithm works assuming there is no cycle whose sum of weights is negative. However, if the graph has such cycles, we can use the algorithm to actually detect them. The length of a path from i to itself is set to 0 when the algorithm starts. Now a path from i to itself can only improve if the length is less than zero, but that would mean a negative cycle. Thus once the algorithm terminates, if there is a diagonal coefficient in the matrix D which is negative, that means this node is involved in a negative cycle. Thus the presence of at least one negative diagonal coefficient reveals the presence of at least one negative cycle. As an example, one can replace the weight $w(3,1)$ in the above example to be -10, this creates a negative cycle (see Exercise 34).

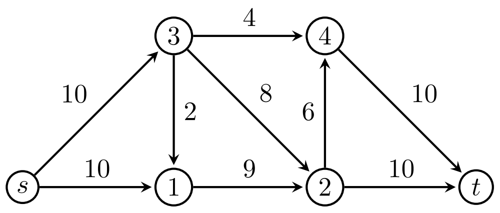
3.5 Exercises

Exercise 25. Show that

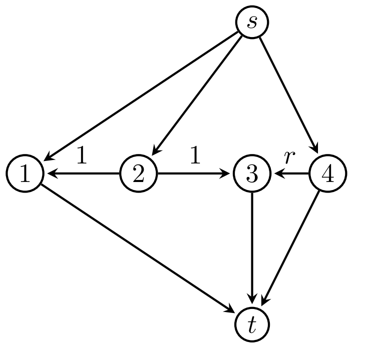
$$|f| = \sum_{u \in I(t)} f(u,t),$$

that is, the strength of the flow is the sum of the values of f on edges entering the sink.

Exercise 26. Use Ford-Fulkerson algorithm to find a maximum flow in the following network:



Exercise 27. Here is a famous example of network (found on wikipedia and in many other places) where Ford-Fulkerson may not terminate. The edges capacities are 1 for $(2,1)$, $r = (\sqrt{5}-1)/2$ for $(4,3)$, 1 for $(2,3)$, and M for all other edges, where $M \geq 2$ is any integer.



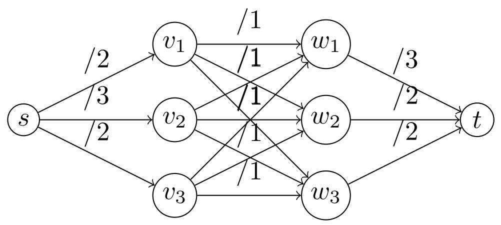
1. Explain why Ford-Fulkerson algorithm may not be able to terminate.
2. Apply Edmonds-Karp algorithm to find a maximum flow in this network.

Exercise 28. Menger's Theorem states the following. Let G be a directed graph, let u, v be distinct vertices in G. Then the maximum number of pairwise edge-disjoint paths from u to v equals the minimum number of edges whose removal from G destroys all directed paths from u to v. Prove Menger's Theorem using the Max Flow- Min Cut Theorem.

Exercise 29. Let $G = (V, E)$ be an undirected graph that remains connected after removing any $k - 1$ edges. Let s, t be any two nodes in V.

1. Construct a network G' with the same vertices as G, but for each edge $\{u, v\}$ in G, create in G' two directed edges (u, v), (v, u) both with capacity 1. Take s for the source and t for the sink of the network G'. Show that if there are k edge-disjoint directed paths from s to t in G', then there are k edge-disjoint paths from s to t in G.
2. Use the max-flow min-cut theorem to show that there are k edge-disjoint paths from s to t in G'.

Exercise 30. 1. Compute a maximal flow in the following network, where each edge (v_i, w_j) has a capacity of 1, for $i, j = 1, 2, 3$:



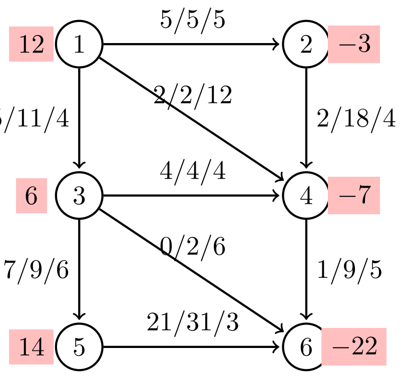
2. Interpret the above as a placement of a number of balls of different colours into bins of different capacities, such that no two balls with the same colour belong to the same bin. More generally, describe in terms of flow over a

network the problem of placing b_i balls of a given colour, for $i = 1, \dots, m$ colours, into n bins of different capacities c_j, $j = 1, \dots, n$, such that no two balls with the same colour belong to the same bin (b_i, c_j are all positive integers).

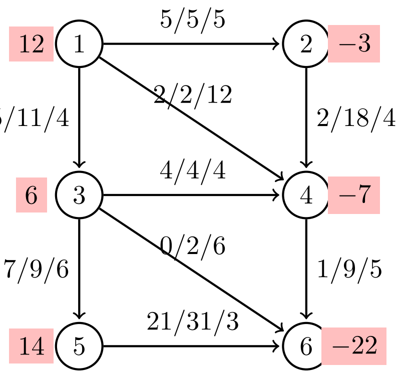
3. Give a necessary and sufficient condition on the max flow of the above graph to ensure that the ball placement problem has a solution (that is a condition (C) such that (C) holds if and only if the ball placement problem has a solution).

Exercise 31. Show that one can always consider min-cost-flow networks were lower capacities are zero, that is, if a network has lower capacities which are not zero, the network can be replaced by an equivalent network were all lower capacities are zero.

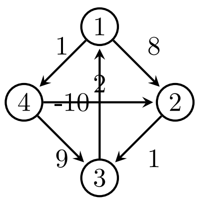
Exercise 32. Compute the residual graph G_f of the following graph G with respect to the flow given. Each edge e is labeled with $(f(e), c(e), \gamma(e))$.



Exercise 33. Consider the following graph G. Each edge e is labeled with $(f(e), c(e), \gamma(e))$. Solve the min-cost-low problem for this graph, using the flow given as initial feasible flow.



Exercise 34. Use Floyd-Warshall algorithm to detect the presence of at least one negative cycle in the graph below.


1
2
3
4
1
8
2
-10
9
1

Chapter 4

Linear Programming

We assume that vectors are defined as column vectors. Also vector inequalities are understood componentwise, that is, $x \geq 0$ for $x \in \mathbb{R}^n$ means $x_i \geq 0$ for $i = 1, \dots, n$.

Definition 4.1. A *linear program* is an optimization problem of the form

$$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

where $c, x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and A is an $m \times n$ matrix.

We call $c^T x$ the *objective function*, and $Ax \leq b$ are constraints. We refer to this linear program as (LP).

The constraint $x \geq 0$ can also be included in constraints of the form $A'x \leq b'$ by setting $A' = \begin{bmatrix} A \\ -I \end{bmatrix}$ where I_n is the identity matrix and $b' = \begin{bmatrix} b \\ 0_n \end{bmatrix}$.

Note that the above form describes a linear objective function in x with linear constraints in x without loss of generality: if some $x_j \leq 0$, then set a new variable $x'_j = -x_j \geq 0$, if some $x_j \geq d$, for some constant d, then set a new variable $x'_j = x_j - d \geq 0$, if some $x_j \leq d$, for some constant d, then set a new variable $x'_j = d - x_j \geq 0$, and if a constraint is of the form $a_j x \geq b_j$, then write instead $-a_j x \leq -b_j$, and to minimize $c^T x$, maximize $-c^T x$. Finally, a variable x_j may appear with no constraint on being positive or negative, we call such a variable a *free (or unrestricted) variable*. A free variable can be replaced by $x_j = u_j - v_j$, $u_j, v_j \geq 0$.

A linear program can also be stated as

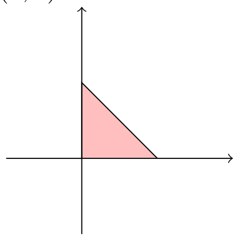
$$\begin{aligned} \min \quad & y^T b \\ s.t. \quad & y^T A \geq c^T \\ & y \geq 0 \end{aligned}$$

where $c \in \mathbb{R}^n$, $b, y \in \mathbb{R}^m$ and A is an $m \times n$ matrix. We will discuss more this form in Section 4.3.

Example 4.1. Consider the linear program:

$$\begin{aligned} \max \quad & 2x_1 + x_2 \\ s.t. \quad & x_1 + x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

where $c^T = [2, 1]$ and $A = [1, 1]$. The constraints delimit a portion of $\mathbb{R}^2$, namely a triangle whose vertices are (0,0), (0,1) and (1,0). To maximize the objective function, we notice that since x_1, x_2 should be as large as possible, we are looking at points on the line $x_1 + x_2 = 1$, and the maximum is reached at $(1, 0)$.



4.1 Feasible and Basic Solutions

Definition 4.2. The set

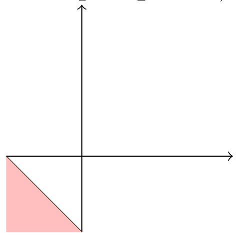
$$\{x \in \mathbb{R}^n,\ Ax \leq b, x \geq 0\}$$

is called the *feasible region* of (LP). A point x in the feasible region is called a feasible solution. An LP is said to be feasible if the feasible region is not empty, and infeasible otherwise.

Example 4.2. In the above example, the feasible region is the triangle whose vertices are (0,0), (0,1) and (1,0). Consider the linear program:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + x_2 \leq -1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

The constraints are requesting that x_1, x_2 are non-negative, but also below the line $x_1 + x_2 = -1$, this LP is thus infeasible.



Definition 4.3. A feasible maximum (respectively minimum) LP is said to be *unbounded* if the objective function can assume arbitrarily large positive (respectively negative) values at feasible points. Otherwise it is said to be *bounded.*

Example 4.3. The linear program

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1, x_2 \geq 0 \end{aligned}$$

is unbounded, no optimal solution exists.

We thus have 3 possibilities for a LP:

- It is feasible and bounded, see Example 4.1.
- It is feasible and unbounded, see Example 4.3.
- It is infeasible, see Example 4.2.

Definition 4.4. A *slack variable* is a variable that is added to an inequality constraint in $Ax \leq b$ to transform it into an equality.

Example 4.4. Consider the linear program:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + 3x_2 \leq 9 \\ & 2x_1 + x_2 \geq 8 \\ & x_1, x_2 \geq 0. \end{aligned}$$

We can rewrite it as

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + 3x_2 + s_1 = 9 \\ & 2x_1 + x_2 - s_2 = 8 \\ & x_1, x_2, s_1, s_2 \geq 0 \end{aligned}$$

and s_1, s_2 are slack variables. Indeed, $2x_1 + x_2 - s_2 = 8 \iff 2x_1 + x_2 = 8 + s_2 \geq 8$ for $s_2 \geq 0$, and similarly $x_1 + 3x_2 + s_1 = 9 \iff x_1 + 3x_2 = 9 - s_1 \leq 9$ for $s_1 \geq 0$.

With slack variables the problem

$$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

has the *standard form*

$$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

and in fact, as discussed above, any LP can be converted to this standard form. The converse is true. Suppose we have an LP of the form

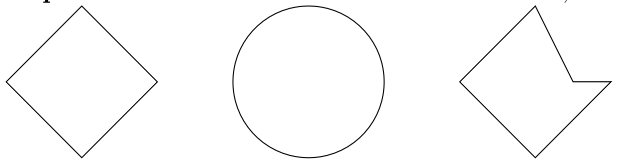
$$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

then we can replace $Ax = b$ by $Ax \leq b$ and $-Ax \leq -b$.

In a linear program in standard form, we assume that the $m \times n$ matrix A has rank m, with $m \leq n$. The case $m > n$ corresponds to having an over determined system, where the number of constraints m is more than the number of unknowns n. In this case, the system typically does not have a solution, and this requires a different approach, which we will not consider. We will see in the next chapter that sometimes having an $m \times n$ matrix A of rank less than m can be exploited to introduce degrees in freedom in how to solve the corresponding LP, but for this chapter, if we have a redundant equality, we will just assume that we remove it.

Definition 4.5. A set $S \subset \mathbb{R}^n$ is called *convex* if for all $u, v \in S$, $\lambda u + (1-\lambda)v \in S$ for all $\lambda \in]0,1[$ (the notation $(0,1)$ is also often found instead of $]0,1[$).

Example 4.5. The first two sets below are convex, the 3rd one is not.



Proposition 4.1. *The feasible region* $S = \{x \in \mathbb{R}^n,\ Ax = b,\ x \geq 0\}$ *of (LP) is convex.*

Proof. Suppose $u, v \in S$, $\lambda \in]0,1[$. Set $w = \lambda u + (1-\lambda)v$. Then

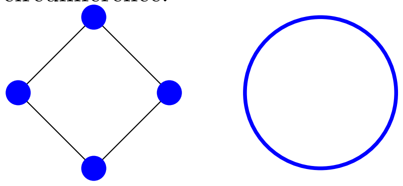
$$Aw = \lambda Au + (1-\lambda)Av = \lambda b + (1-\lambda)b = b$$

so w satisfies $Aw = b$. Then $w = \lambda u + (1-\lambda)v$ where λ lives in $]0,1[$, and $u, v \geq 0$ so $w \geq 0$. $\square$

Definition 4.6. A point x in a convex set S is called an *extreme point* of S is there are no distinct points $u, v \in S$ and $\lambda \in]0,1[$ such that $x = \lambda u + (1-\lambda)v$.

In words, this is saying that an extreme point is not in the interior of any line segment in S.

Example 4.6. In the example above, for the circle, the extreme points are on its circumference.



Extreme points of convex sets are particularly important in the context of linear programming because of the following theorem, which tells us that optimal solutions of (LP) are found among extreme points.

Theorem 4.2. *If an LP has an optimal solution (an optimal solution is a feasible solution that optimizes the objective function), then it has an optimal solution at an extreme point of the feasible set* $S = \{x \in \mathbb{R}^n,\ Ax = b, x \geq 0\}$.

Proof. Since there exists an optimal solution, there exists an optimal solution x with a minimal number of nonzero components.

Suppose x is not extreme, then by definition there exist $u, v \in S,\ u \neq v$ and $\lambda \in]0,1[$ such that

$$x = \lambda u + (1-\lambda)v \in S.$$

Since x is optimal and we want to maximize the objective function, then

$$c^T u \leq c^T x,\ c^T v \leq c^T x.$$

But also

$$c^T x = \lambda c^T u + (1-\lambda)c^T v \leq \lambda c^T x + (1-\lambda)c^T x = c^T x$$

which forces the inequality to be an equality and since $\lambda \in]0,1[$, $\lambda(c^T x - c^T u) + (1-\lambda)(c^T x - c^T v) = 0$ means that $c^T x = c^T u = c^T v$.

Now consider the line

$$x(\epsilon) = x + \epsilon(u - v),\ \epsilon \in \mathbb{R}.$$

We start by showing in (a) that the vector $x(\epsilon)$ satisfies the constraints defined by A for all ϵ, in (b) that it has the same objective function as x for all ϵ, and in (c),(d) that its coefficients are non-negative for values of ϵ around 0:

(a) $Ax = Au = Av = b$ since x, u, v all are in the feasible region, thus $Ax(\epsilon) = Ax + \epsilon(Au - Av) = b$ for all ϵ.

(b) $c^T x(\epsilon) = c^T x + \epsilon(c^T u - c^T v) = c^T x$ for all ϵ, since we showed above that $c^T x = c^T u = c^T v$.

(c) If $x_i = 0$, since $x_i = \lambda u_i + (1-\lambda)v_i$ with $u_i, v_i \geq 0$, we must have $0 = \lambda u_i + (1-\lambda)v_i$ and thus $u_i = v_i = 0$. So $x(\epsilon)_i = x_i + \epsilon(u_i - v_i) = x_i = 0$.

(d) If $x_i > 0$, then $x(0) = x$ and $x(0)_i = x_i > 0$. Also, by continuity of $x(\epsilon)_i$ in ϵ, $x(\epsilon)_i$ will remain positive for a suitable range of values of ϵ around 0.

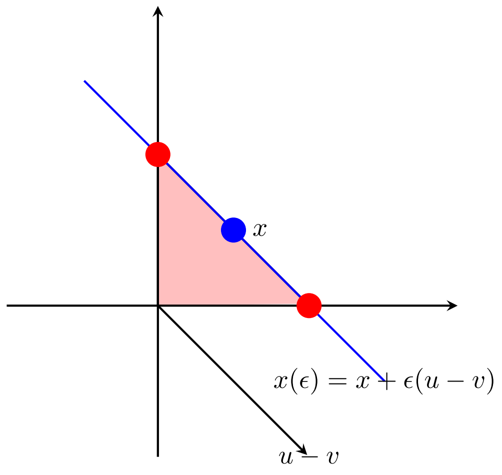
So we just showed that if x is not extreme, then it is on the line $x(\epsilon)$, and every point on this line satisfies the constraints defined by A, and has the same optimal objective function $c^T x$, furthermore, $x(\epsilon)_i > 0$ for values of ϵ around 0. Now invoking again the continuity of $x(\epsilon)_i$ in ϵ, we can increase ϵ from 0, in a positive or negative direction as appropriate (depending on the sign of $u - v$), until at least one extra component of $x(\epsilon)$ becomes 0 (since we start at $x_i > 0$ along a line, we are in the feasible region, and then need to go through 0 before getting something negative). This gives an optimal solution with fewer nonzero components than x, a contradiction, so x must be extreme.

$\square$

Example 4.7. Consider the linear program:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

where $c^T = (2, 1)$ and $A = (1, 1)$. The feasible region S is shown below. The point $x = (0.5, 0.5)$ is optimal, but not extreme. We can find two vectors $u = (1, 0), v = (0, 1) \in S$ such that $x = \frac{1}{2}u + \frac{1}{2}v$ ($\lambda = \frac{1}{2}$). Consider then the line $x(\epsilon) = x + \epsilon(u - v)$.



Definition 4.7. Given the $m \times n$ matrix A which we assumed is of rank m, select m linearly independent columns, whose indices are put in a set B. Then we can solve

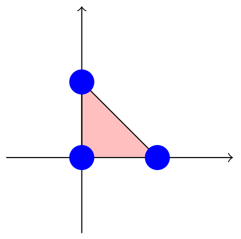
$$A_B x_B = b$$

by inverting the matrix A_B created by selecting the columns of A in B, to find an m-dimensional vector x_B, which contains the coefficients of x whose indices are in B. Then set the coefficients of x whose indices are not in B to be zero. Then x is called a *basic solution*. A basic solution satisfying $x \geq 0$ is called a *basic feasible solution (BFS)*.

Example 4.8. In Example 4.1, we considered the linear program:

$$\begin{aligned} \max \quad & 2x_1 + x_2 \\ s.t. \quad & x_1 + x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

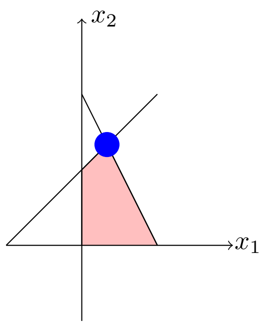
whose maximum is reached at $(1, 0)$.



Using a slack variable s_1, we can write the constraint $x_1 + x_2 \leq 1$ as $x_1 + x_2 + s_1 = 1$ for $s_1 \geq 0$ so $A = [1, 1, 1]$ and $b = 1$. We can have a single linearly independent column, so if we choose $B = \{1\}$, we get $A_B x_B = x_B = 1$ and the basic solution $[1, 0, 0]$, if we choose the second column, $B = \{2\}$, $A_B x_B = x_B = 1$ and we get the basic solution $[0, 1, 0]$ and if we choose the third column, $A_B x_B = x_B = 1$, we obtain the basic solution $[0, 0, 1]$. All the basic solutions are feasible.

Example 4.9. Consider the linear program:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & -x_1 + x_2 \leq 1 \\ & 2x_1 + x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$



Using slack variables s_1, s_2, we can write the constraints so

$$A = \begin{bmatrix} -1 & 1 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

If we choose $B = \{1, 2\}$, we get

$$A_B x_B = \begin{bmatrix} -1 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Then

$$A_B^{-1} = \frac{1}{-3} \begin{bmatrix} 1 & -1 \\ -2 & -1 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \frac{1}{-3} \begin{bmatrix} 1 & -1 \\ -2 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1/3 \\ 4/3 \end{bmatrix}$$

and $[1/3, 4/3, 0, 0]$ is a basic feasible solution.

Theorem 4.3. *We have that x is an extreme point of $S = \{x,\ Ax = b,\ x \geq 0\}$ if and only if x is a basic feasible solution (BFS).*

Proof. Let $I = \{i,\ x_i > 0\}$.

($\Leftarrow$) Suppose that x is a basic feasible solution, and write $x = \lambda u + (1-\lambda)v$ for $u, v \in S$, $\lambda \in]0, 1[$. To show that x is extreme, we need to show that $u = v$.

(a) If $i \notin I$, then $x_i = 0$ (since x is a BFS, we cannot have $x_i < 0$), which implies $x_i = \lambda u_i + (1-\lambda)v_i = 0$ and since $u_i, v_i \geq 0$ (recall that $u, v \in S$), it must be that $u_i = v_i = 0$, so we know that u and v coincide on indices not in I.

(b) Since $Au = Av = b$, we have $A(u - v) = 0$. For A_i the ith column of A, this is equivalent to say that

$$\sum_{i=1}^{n} (u_i - v_i) A_i = 0 \Rightarrow \sum_{i \in I} (u_i - v_i) A_i = 0$$

since by (a), $u_i - v_i = 0$ for all $i \notin I$. But x is a BFS, this means that the x_i which are zero may be coming either from solving the system $A_B x_B = b$ for some choice B of indices, or by being coefficients whose index is not in B, but the x_i which are not zero are necessarily coming from solving the system $A_B x_B = b$. This implies $u_i - v_i = 0$ for all $i \in I$ since the columns A_i of A_B are linearly independent.

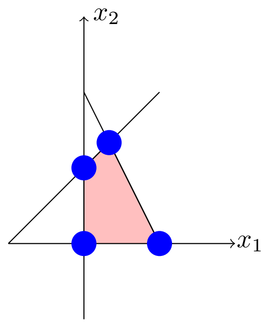
Hence $u = v$ and x is an extreme point.

($\Rightarrow$) Suppose that x is not a BFS, that is, $\{A_i,\ i \in I\}$ are linearly dependent. Then there exists $u \neq 0$ with $u_i = 0$ for $i \notin I$ such that $Au = 0$. For small enough ϵ, $x \pm \epsilon u$ are feasible since

- $A(x \pm \epsilon u) = Ax \pm \epsilon Au = Ax = b$ using that $Au = 0$ and $x \in S$,
- $x \pm \epsilon u \geq 0$ since $x \geq 0$, and when $x_i = 0$, then $i \notin I$ and $u_i = 0$ for $i \notin I$, while when $x_i > 0$, we take ϵ small enough,

and $x = \frac{1}{2}(x + \epsilon u) + \frac{1}{2}(x - \epsilon u)$, so x is not extreme. □

Example 4.10. In Example 4.9, we already saw that $B = \{1, 2\}$ gives the point $[1/3, 4/3, 0, 0]$. By choosing $B = \{1, 3\}$, we get $[1, 0, 2, 0]$, by choosing $B = \{2, 4\}$, we get $[0, 1, 0, 1]$, and for $B = \{3, 4\}$, we get $[0, 0, 1, 2]$. The other choices of B are not feasible. We thus get points $(x_1, x_2) \in \{(1/3, 4/3), (1, 0), (0, 1), (0, 0)\}$.



Corollary 4.4. *If there is an optimal solution, then there is an optimal BFS.*

Proof. By Theorem 4.2, we know that if an LP has an optimal solution, then it has an optimal solution at an extreme point of the feasible set. Then by Theorem 4.3, we have that if x is an extreme point of the feasible set $S = \{x,\ Ax = b,\ x \geq 0\}$, then x is a basic feasible solution (BFS). $\square$

In words, we have reduced our search space considerably: we started by trying to find an optimal solution for our optimization problem by looking at all points of our feasible region, while it turns out that it is enough to look at basic feasible solutions (which are extreme points of the feasible region). We also have an algorithm to do so, based on the definition of BFS. Sometimes the set B is called *a basis.*

1. Choose $n - m$ of the variables to be 0 ($x_i = 0$ for $i \notin B$). They are called the *non-basic variables.* We may group them into the vector x_N where $N = \{1, \ldots, n\} \backslash B$.

2. Look at the remaining m columns $\{A_i,\ i \in B\}$. Are they linearly independent? if so, we have an invertible $m \times m$ matrix A_B and we can solve to find x_B and thus x. We call these x_i (those in x_B), the *basic variables.*

If we try all possible choices of $n - m$ variables, we get at most $\binom{n}{m}$ of them. This is actually a bad algorithm... First of all, $\binom{n}{m}$ grows quickly when n and m grow, e.g. $\binom{20}{11} = 167960$ and solving the $m \times m$ system of equations to find x_m also costs a Gaussian elimination.

A classical method that provides a better alternative to the above algorithm is the so-called *Simplex Algorithm.*

4.2 The Simplex Algorithm

We argued above that trying all the possible basic feasible solutions is too expensive. The idea of the Simplex Algorithm is to start from one BFS (an extreme

Figure 4.1: G.B. Dantzig (1914-2005) proposed the Simplex Algorithm in 1947, on the above picture, he is awarded the National Medal of Science.

point of the feasible region), and then to try out an *adjacent* (or *neighbouring*) BFS in such a way that we improve the value of the objective function. By adjacent, we mean that the two BFS differ by exactly one basic (or non-basic) variable. This would lead to an algorithm of the following form:

Algorithm 10 General Simplex Algorithm

```
   Input: an LP in standard form max $c^T x$, such that $Ax = b,\ x \geq 0$.
   Output: a vector $x^*$ that maximizes the objective function $c^T x$.
1: Start with an initial BFS.
2: while (the current BFS is not optimal) do
3:     Move to an improved adjacent BFS.
4: return $x^*$ =BFS;
```

This needs a lot of clarifications.

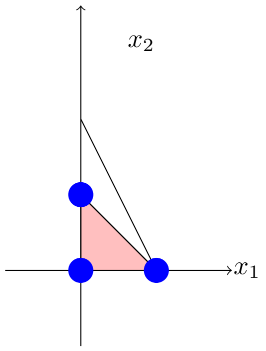
Non-Degeneracy. To start with, we are relying on the fact that a BFS corresponds to extreme points of the feasible region. We need to make sure that when we are moving from one BFS to another BFS, we actually go from one extreme point to another extreme point, the risk being that several BFS correspond to the same extreme point, and we could get stuck trying to improve a BFS which actually remains the same extreme point: this results in *stalling* if we eventually move to another solution, or worse, *cycling*, if we return to a tried degenerate BFS.

Example 4.11. Consider the linear program:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + x_2 \leq 1 \\ & 2x_1 + x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

thus, with slack variables s_1, s_2, we get

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$



We thus have the following basic feasible solutions.

$$\begin{array}{ll} B = \{1,2\} & [1,0,0,0] \\ B = \{1,3\} & [1,0,0,0] \\ B = \{1,4\} & [1,0,0,0] \\ B = \{2,4\} & [0,1,0,1] \\ B = \{3,4\} & [0,0,1,2] \end{array}$$

Definition 4.8. We say that a basic feasible solution is *degenerate* if there exists at least one basic variable which is 0. It is called non-degenerate otherwise. An LP is non-degenerate if every basic feasible solution is non-degenerate.

Indeed, in order to compute x, we are given a choice B of columns of A which are linearly independent, and

$$A_B x_B = b.$$

Suppose that we solve for x_B, and find that the ith coefficient of x_B is zero. Then this means that the ith column of A_B contributes 0 to obtain b, therefore it can be replaced by a column of A_N such that the resulting matrix $\tilde{A}_B$ remains full rank. The ith column of A_B then gets moved to form a new matrix $\tilde{A}_N$, which does not interfere in the computation of the extreme point.

This is exactly what happened in our previous example. Once the choice of $B = \{1,2\}$ gives a basic feasible solution where the basic variable $x_2 = 0$, then the second column of A_B where column 2 of A is could be replaced by the column 3 and 4 of A to create the same extreme point:

$$\begin{array}{rl} B=\{1,2\} & [1,0,0,0] \\ B=\{1,3\} & [1,0,0,0] \\ B=\{1,4\} & [1,0,0,0] \end{array}$$

We thus need non-degenerate BFS to look for improved BFS in our algorithm.

Pivoting. Once we have a non-degenerate BFS, we need to compute an adjacent BFS. We recall that two BFS are adjacent if they have $m-1$ basic variables in common (or equivalently, they differ by exactly one basic variable). Given a BFS, an adjacent BFS can be reached by increasing one non-basic variable from zero to positive, and decreasing one basic variable from positive to zero. This process is called *pivoting*. As a result, one non-basic variable “enters” B (that is, its index is put in B), while one basic variable “leaves” B.

Formally, suppose that we have a BFS x, and let x_q denote the non-basic variable of x which we would like to increase by a coefficient of $\lambda \geq 0$ (the other non-basic variables are kept to 0). Then x_N will be updated to

$$x_N + \lambda \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

where λ multiplies a vector containing only zeroes, but for a 1 in the position of x_q. To simplify the notation, let us suppose that x_q is the qth non-basic variable, and write e_q for the vector with only zero coefficients but a 1 in the qth component (we could alternatively index x_q with respect to its position in A instead of A_N).

We need to decrease correspondingly a basic variable. Since $A_B x_B + A_N x_N = b$, we have that

$$x_B = A_B^{-1}(b - A_N x_N).$$

When x_q increases by λ, we just saw above that x_N gets updated to $x_N + \lambda e_q$, and by similarly updating x_N in $A_N x_N$, we get

$$A_N(x_N + \lambda e_q)$$

and

$$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} A_B^{-1}(b - A_N x_N) \\ x_N \end{bmatrix}$$

gets updated to

$$\tilde{x} = \begin{bmatrix} A_B^{-1} b - A_B^{-1} A_N \left(x_N + \lambda e_q\right) \\ x_N + \lambda e_q \end{bmatrix}$$

and in conclusion

$$\tilde{x} = x + \lambda \begin{bmatrix} -A_B^{-1} A_N e_q \\ e_q \end{bmatrix}.$$

Write $(A_N)_q$ for the qth column of A_N. Then

$$\tilde{x} = x + \lambda \begin{bmatrix} -A_B^{-1} (A_N)_q \\ e_q \end{bmatrix}.$$

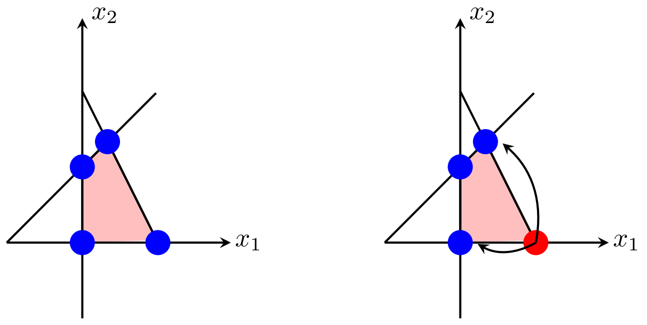
It is clear that multiplying $\tilde{x}$ by $A = [A_B, A_N]$ gives $A\tilde{x} = Ax = b$. For the non-degenerate case, $x_B > 0$, thus for $\lambda \geq 0$ small enough, $\tilde{x} \geq 0$. Note that this is not true for the degenerate case, if some $x_i = 0$, no matter how small λ is, the negative sign in $-A_B^{-1}A_N$ could create a negative coefficient. This confirms that there exist choices of $\lambda \geq 0$ such that the pivoting operation sent one BFS to a feasible solution, however do note that we still have to discuss the actual choice of λ.

Example 4.12. Let us continue Example 4.9, for which

$$A = \begin{bmatrix} -1 & 1 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

and basic feasible solutions are:

$B = \{1, 2\}$	$[1/3, 4/3, 0, 0]$
$B = \{1, 3\}$	$[1, 0, 2, 0]$
$B = \{2, 4\}$	$[0, 1, 0, 1]$
$B = \{3, 4\}$	$[0, 0, 1, 2]$



Suppose we start with the BFS $[1, 0, 2, 0]$. Then x_B, x_N satisfy

$$x_B = A_B^{-1}(b - A_N x_N) = \begin{bmatrix} 0 & 1/2 \\ 1 & 1/2 \end{bmatrix} \left(\begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ s_2 \end{bmatrix} \right).$$

We want to update x_N, so we can update x_2 or s_2.

- If we update s_2 while keeping $x_2 = 0$, the constraint $2x_1 + x_2 + s_2 = 2$ tells us that s_2 can be increased to $s_2 = 2$, which in turn sends x_1 to 0. The constraint $-x_1 + x_2 + s_1 = 1$ with $x_1 = x_2 = 0$ means that s_1 is updated to 1. Replace $s_2 = 2$ and $x_2 = 0$ in the above equation gives $x_1 = 0$ and $s_1 = 1$ as desired. Thus this instance of pivoting sends the BFS $[1, 0, 2, 0]$ to $[0, 0, 1, 2]$.

- If we update x_2 while keeping $s_2 = 0$, the constraint $2x_1 + x_2 + s_2 = 2$ tells us that $2x_1 + x_2 = 2$, but while we could increase x_2 to 2 by sending x_1 to 0, this would violate the constraint $-x_1 + x_2 + s_1 = 1$. Replacing $x_2 = 2 - 2x_1$ in this latter constraint gives $-3x_1 + s_1 = -1$ and we can send s_1 to 0, x_1 to $1/3$, and x_2 to $4/3$. Replace $x_2 = 4/3$ and $s_2 = 0$ in the above question gives $x_1 = 1/3$ and $s_1 = 0$ as desired. Thus this instance of pivoting sends the BFS $[1, 0, 2, 0]$ to $[1/3, 4/3, 0, 0]$.

Alternatively, for the BFS $x = [1, 0, 2, 0]$, use the formula

$$\tilde{x} = \begin{bmatrix} x_B \\ x_N \end{bmatrix} + \lambda \begin{bmatrix} -A_B^{-1}(A_N)_q \\ e_q \end{bmatrix}$$

for $q = 1$ (x_2) and $\lambda = 4/3$ to get

$$\tilde{x} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} + \frac{4}{3} \begin{bmatrix} -\begin{bmatrix} 0 & 1/2 \\ 1 & 1/2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 0 \\ 4/3 \\ 0 \end{bmatrix}$$

and for $q = 2$ (s_2) and $\lambda = 2$:

$$\tilde{x} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} + 2 \begin{bmatrix} -\begin{bmatrix} 0 & 1/2 \\ 1 & 1/2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 2 \end{bmatrix}$$

which gives the desired result.

We may want to look at the geometrical interpretation of the above formula: x is an extreme point, we add the vector $\begin{bmatrix} -A_B^{-1}(A_N)_q \\ e_q \end{bmatrix}$ to it, so this vector gives the direction in which we move away from x, and λ tells us how far we move away from x along the given direction. We notice that we move while remaining in the feasible set.

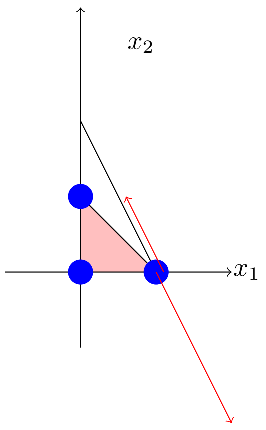
Example 4.13. Consider the linear program

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + x_2 \leq 1 \\ & 2x_1 + x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

of Example 4.11, for which we saw that

$$\begin{array}{ll} B = \{1, 2\} & [1, 0, 0, 0] \\ B = \{1, 3\} & [1, 0, 0, 0] \end{array}$$

are two basic feasible solutions corresponding to the extreme point $[1, 0, 0, 0]$.



Let us compute the vector $\begin{bmatrix} -A_B^{-1}(A_N)_q \\ e_q \end{bmatrix}$ that gives the direction in the pivoting process, for $B = \{1, 2\}, q = 1$:

$$\begin{bmatrix} -A_B^{-1}(A_N)_1 \\ e_1 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 1 \\ 0 \end{bmatrix}$$

and $B = \{1, 3\}, q = 1$:

$$\begin{bmatrix} -A_B^{-1}(A_N)_1 \\ e_1 \end{bmatrix} = \begin{bmatrix} -1/2 \\ -1/2 \\ 1 \\ 0 \end{bmatrix}$$

which since $B = \{1, 3\}$, corresponds to $[-1/2, 1, -1/2, 0]$.

This illustrates the problem that with degenerate BFS, we have no guarantee to remain within the feasible region.

Reduced Cost. We now know (or rather, almost know, we are still left to figure out how to compute λ) how to move from one BFS (which uniquely determines an extreme point) to an adjacent BFS. We next need to discuss how to look for an adjacent BFS which improves the objective function. For that, we observe how it changes from x to $\tilde{x}$:

$$\begin{aligned} c^T\tilde{x} &= c^T\left(x + \lambda \begin{bmatrix} -A_B^{-1}(A_N)_q \\ e_q \end{bmatrix}\right) \\ &= c^Tx + \lambda[c_B^T, c_N^T] \begin{bmatrix} -A_B^{-1}(A_N)_q \\ e_q \end{bmatrix} \\ &= c^Tx + \lambda((c_N^T)_q - c_B^T A_B^{-1}(A_N)_q). \end{aligned}$$

Definition 4.9. The quantity

$$r_q = (c_N^T)_q - c_B^T A_B^{-1}(A_N)_q$$

is called a *reduced cost* with respect to the non-basic variable x_q.

If $r_q < 0$, then $c^T\tilde{x} = c^T x + \lambda r_q \leq c^T x$, which improves the cost function if it is a minimization, or in our case, where we chose a maximization of the cost function, we want $r_q > 0$.

Algorithm Termination. Since the reduced cost characterizes the improvement in the cost function by moving from one BFS to an adjacent one, we expect that once the reduced cost cannot improve anymore, the algorithm terminates, and we found an optimal solution. This is under the assumption that BFS are not degenerate.

Theorem 4.5. *Given a basic feasible solution x^* with respect to a given B, that is $x^* = \begin{bmatrix} A_B^{-1}b \\ 0_{n-m} \end{bmatrix}$, if $r_q \leq 0$ for all non-basic variables x_q^* (we assume the objective function is a maximization), then x^* is optimal.*

Proof. Consider an arbitrary other feasible solution x, that is x is such that $Ax = b$ and $x \geq 0$. Write $x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}$. We have

$$\begin{aligned}\begin{bmatrix} A_B & A_N \\ 0_{(n-m)\times m} & I_{n-m} \end{bmatrix}(x - x^*) &= \begin{bmatrix} A_B & A_N \\ 0_{(n-m)\times m} & I_{n-m} \end{bmatrix}\begin{bmatrix} x_B - A_B^{-1}b \\ x_N \end{bmatrix} \\ &= \begin{bmatrix} A_B x_B - b + A_N x_N \\ x_N \end{bmatrix} = \begin{bmatrix} 0_m \\ x_N \end{bmatrix}\end{aligned}$$

since $Ax = [A_B, A_N]\begin{bmatrix} x_B \\ x_N \end{bmatrix} = b$. Thus

$$x - x^* = \begin{bmatrix} A_B & A_N \\ 0_{(n-m)\times m} & I_{n-m} \end{bmatrix}^{-1}\begin{bmatrix} 0_m \\ x_N \end{bmatrix} = \begin{bmatrix} A_B^{-1} & -A_B^{-1}A_N \\ 0_{(n-m)\times m} & I_{n-m} \end{bmatrix}\begin{bmatrix} 0_m \\ x_N \end{bmatrix} = \begin{bmatrix} -A_B^{-1}A_N \\ I_{n-m} \end{bmatrix} x_N$$

that is

$$x = x^* + \sum_{q \in N} x_q \begin{bmatrix} -A_B^{-1}(A_N)_q \\ e_q \end{bmatrix}.$$

We can now compare the objective function of both x and x^*:

$$\begin{aligned} c^T x &= c^T x^* + \sum_{q \in N} x_q [c_B^T, c_N^T] \begin{bmatrix} -A_B^{-1}(A_N)_q \\ e_q \end{bmatrix} \\ &= c^T x^* + \sum_{q \in N} x_q(-c_B^T A_B^{-1}(A_N)_q + c_N^T e_q) \\ &= c^T x^* + \sum_{q \in N} x_q((c_N^T)_q - c_B^T A_B^{-1}(A_N)_q) \\ &= c^T x^* + \sum_{q \in N} x_q r_q \end{aligned}$$

where $r_q \leq 0$ for all $q \in N$, showing as desired that $c^T x \leq c^T x^*$. □

Example 4.14. In Example 4.12, where we want to maximize $x_1 + x_2$, we showed that we can start from the BFS $x = [1, 0, 2, 0]$, which for $B = \{1, 3\}$, is rewritten as $\begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix}$, and reach two new extreme points $\tilde{x}$ using

$$\tilde{x} = \begin{bmatrix} x_B \\ x_N \end{bmatrix} + \lambda \begin{bmatrix} -A_B^{-1}(A_N)_q \\ e_q \end{bmatrix}.$$

The corresponding objective function to maximize is thus

$$x_1 + x_2 = \underbrace{[1, 0, 1, 0]}_{c^T} \begin{bmatrix} x_1 \\ s_1 \\ x_2 \\ s_2 \end{bmatrix}.$$

We can compute the corresponding reduced cost $r_q = (c_N^T)_q - c_B^T A_B^{-1}(A_N)_q$. For $q = 1$ (x_2) and $\lambda = 4/3$:

$$\tilde{x} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} + \frac{4}{3} \begin{bmatrix} -\begin{bmatrix} 0 & 1/2 \\ 1 & 1/2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 0 \\ 4/3 \\ 0 \end{bmatrix}$$

thus

$$r_1 = (c_N^T)_1 - c_B^T A_B^{-1}(A_N)_1 = 1 - 1/2 = 1/2 > 0.$$

There is thus an improvement in the cost function by moving into this direction, given by λr_q and

$$c^T \tilde{x} = c^T x + \lambda r_q = 1 + \frac{4}{3}\frac{1}{2} = 5/3.$$

For $q = 2$ (s_2) and $\lambda = 2$:

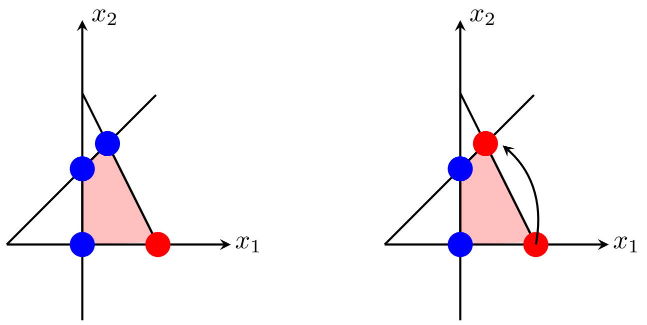
$$\tilde{x} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} + 2 \begin{bmatrix} -\begin{bmatrix} 0 & 1/2 \\ 1 & 1/2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 2 \end{bmatrix}$$

thus

$$r_2 = (c_N^T)_2 - c_B^T A_B^{-1}(A_N)_2 = 0 - 1/2 = -1/2 \leq 0,$$

and there is thus no improvement in the cost function by moving into this direction. In fact, the objective function decreases:

$$c^T \tilde{x} = c^T x + \lambda r_2 = 1 + 2\frac{-1}{2} = 0.$$



From the BFS $[1, 0, 2, 0]$, we thus go the BFS $[1/4, 4/3, 0, 0]$. We can check by computing the reduced costs (see Exercise 36) that this solution is optimal

Choice of λ. We saw that given a BFS x, we can compute a new feasible solution $\tilde{x}$ given by

$$\tilde{x} = \begin{bmatrix} x_B \\ x_N \end{bmatrix} + \lambda \begin{bmatrix} -A_B^{-1}(A_N)_q \\ e_q \end{bmatrix},$$

where we will denote by d_q the "direction" vector $\begin{bmatrix} -A_B^{-1}(A_N)_q \\ e_q \end{bmatrix}$ and we next determine $\lambda > 0$ (if the BFS is not degenerate which we assume, $\lambda > 0$ always exists). We choose q (among the possible non-basic variables) such that the reduced cost $r_q > 0$ to guarantee that the cost function increases.

The choice of λ tells us "how far" we will go away from the BFS x in the direction given by d_q.

If $d_q \geq 0$, then $\tilde{x} = x + \lambda d_q \geq 0$ for any choice of λ, so λ can be chosen arbitrarily big. Then

$$c^T\tilde{x} = c^T x + \lambda c^T d_q = c^T x + \lambda r_q$$

becomes arbitrarily big and the LP is unbounded.

Thus we may assume that d_q has at least one negative component $(d_q)_i$ (and i must be in B since indices in N correspond to e_q). For the ith component $\tilde{x}_i$ of $\tilde{x}$ which should be non-negative so $\tilde{x}$ remains feasible, we have

$$\tilde{x}_i = x_i + \lambda(d_q)_i \geq 0 \Rightarrow \lambda \leq \frac{x_i}{-(d_q)_i}.$$

and since there may be several negative components in d_q, we choose

$$\lambda = \min_{i\in B}\left\{\frac{x_i}{-(d_q)_i},\ (d_q)_i < 0\right\}.$$

Then for the index i that achieves the minimum, we get

$$\tilde{x}_i = x_i - \frac{x_i}{(d_q)_i}(d_q)_i$$

and the corresponding basic variable x_i is set to 0. Thus this choice of λ takes us from one BFS to another BFS. We emphasize again that if the BFS were degenerate, we could have $x_i = 0$, this would achieve the above minimum, λ would be set to zero, and we would remain at the same extreme point.

We can now revisit Algorithm 10.

Algorithm 11 Simplex Algorithm

Input: an LP in standard form $\max c^T x$, such that $Ax = b,\ x \geq 0$.

Output: a vector x^* that maximizes the objective function $c^T x$ (or that the LP is unbounded).

```
1: Start with an initial BFS $x$ with basis $B$ and $N = \{1 \dots n\} \backslash B$;
2: For $q \in N$, compute $r_q = c^T d_q = c_q - c_B^T A_B^{-1} (A_N)_q$.
3: while (there is a $q$ such that $r_q > 0$) do
4:     if ($d_q \geq 0$) then
5:         the LP is unbounded, stop.
6:     else
7:         Compute $\lambda = \min_{i \in B} \{ \frac{x_i}{-(d_q)_i},\ (d_q)_i < 0 \}$.
8:         $x \leftarrow x + \lambda d_q$.
9:         Update $B$ and $N$.
```

An Initial Basic Feasible Solution. The first step of the algorithm consists of finding one BFS. Remark that if the matrix A contains I_m as an $m \times m$ submatrix, and $b \geq 0$, then there is an obvious BFS x': choose $A_B = I_m$, then $x_B = A_B^{-1} b = b$ and

$$x' = \begin{bmatrix} 0_{n-m} \\ b \end{bmatrix} \Rightarrow Ax' = [A_N,\ I_m] \begin{bmatrix} 0_{n-m} \\ b \end{bmatrix} = b.$$

Now an $m \times m$ submatrix which is the identity will be present if the LP is such that we need m slack variables to transform the m inequalities (of the form $\leq$) defining the constraints into m equalities.

Tableau. The Simplex Algorithm can be computed by writing a linear program in the form of a *tableau.* Suppose we have an initial BFS given by setting the slack variables to be $s = x_B = b$, and $x = x_N = 0$, and create an array of the form:

$$\begin{array}{cc|c} A_N & I_m & b \\ \hline c^T & 0_m & -0 \end{array}$$

representing the linear system of equations

$$\begin{bmatrix} 0 & A_N & I_m \\ -1 & c^T & 0_m \end{bmatrix} \begin{bmatrix} f \\ x \\ s \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \iff s = b - A_N x, \ \ -f + c^T x = 0$$

so the first n columns of the tableau correspond to $x_1, \dots, x_n$, while the next $n + m$ columns correspond to $s_1, \dots, s_m$. When $x = 0$ and $s = b$, $c^T x = 0$, and $f = -0$, this is the value of the objective function in the BFS $x = 0, s = b$, shown in the right bottom of the tableau. The BFS can be read from the tableau, since the variables corresponding to the columns of A_N are 0, and we read $x_B = s = b$, $x_N = 0$.

The next step of the algorithm is to compute the reduced costs $r_q = c_q - c_B^T A_B^{-1}(A_N)_q$, where $A_B = I_m$, but since B contains the indices of the slack variables, $c_B = 0_m$, and $r_q = c_q$. The condition "there is a q such that $r_q > 0$" then simplifies to check whether we have $c_q > 0$ ($c_i = 0$ for all $i \in B$), so we pick a column q of the tableau (called *pivot column*) corresponding to a coefficient $c_q > 0$. We then compute

$$\begin{aligned} d_q &= \begin{bmatrix} -A_B^{-1}(A_N)_q \\ e_q \end{bmatrix} = \begin{bmatrix} -(A_N)_q \\ e_q \end{bmatrix}, \\ \lambda &= \min_{i \in B} \left\{ \frac{x_i}{-(d_q)_i},\ (d_q)_i < 0 \right\} = \min_{i \in B} \left\{ \frac{b_i}{a_i},\ a_i \in (A_N)_q \right\} \end{aligned}$$

since $A_B = I_m$, $x_B = b$. Note that if $d_q \geq 0$, the LP is unbounded. So there should be coefficients in $-(A_N)_q$ which are negative, that is, coefficients in $(A_N)_q$ which are positive. This means that given the choice of the column q, we look at the coefficients $a_{iq} > 0$, and choose i that minimizes b_i/a_{iq}, this gives the limit on how much we can increase x_q, and i is the *pivot row.* The last step is the update of x, B, N, and x is updated to $\tilde{x}$:

$$\tilde{x} = \begin{bmatrix} b \\ 0_{n-m} \end{bmatrix} + \frac{b_i}{a_{iq}} \begin{bmatrix} -(A_N)_q \\ e_q \end{bmatrix}$$

so the ith row of $\tilde{x}$ becomes 0 as it should be, the qth non-basic variable is increased by b_i/a_i, thus i goes form B to N, while q goes from N to B. We denote by $\tilde{B}$ the new basis, and by $\tilde{N}$ the other indices.

Next we wish to read the new basic variables $x_{\tilde{B}}$ from the tableau, and how the objective function changes as a function of $\tilde{B}$. The current tableau allows us to read $[A_N, I_m] \begin{bmatrix} x \\ s \end{bmatrix} = b$, so given the new $\tilde{B}$, we get a corresponding matrix $A_{\tilde{B}}$, we multiply the above equation by $A_{\tilde{B}}^{-1}$ to get

$$A_{\tilde{B}}^{-1}[A_N, I_m] \begin{bmatrix} x \\ s \end{bmatrix} = A_{\tilde{B}}^{-1} b.$$

Now some columns of $[A_N, I_m]$ correspond to the matrix $A_{\tilde{B}}$ and to the vector $x_{\tilde{B}}$. From them, we get $x_{\tilde{B}}$. The other columns of $[A_N, I_m]$ correspond to $A_{\tilde{N}}$ and to the vector $x_{\tilde{N}}$, from them, we get $A_{\tilde{B}}^{-1} A_{\tilde{N}} x_{\tilde{N}}$, that is we can rewrite the above equation as

$$x_{\tilde{B}} + A_{\tilde{B}}^{-1} A_{\tilde{N}} x_{\tilde{N}} = A_{\tilde{B}}^{-1} b$$

where setting $x_{\tilde{N}} = 0$ gives $x_{\tilde{B}} = A_{\tilde{B}}^{-1} b$ as the current BFS. The cost $c^T x$ for the current BFS is given by $c_{\tilde{B}}^T A_{\tilde{B}}^{-1} b$. More generally, we evaluate $c^T x$ in $[x_{\tilde{B}}, x_{\tilde{N}}]^T$ to get

$$c^T x = c_{\tilde{B}}^T x_{\tilde{B}} + c_{\tilde{N}}^T x_{\tilde{N}} = c_{\tilde{B}}^T A_{\tilde{B}}^{-1} b + (c_{\tilde{N}}^T - c_{\tilde{B}}^T A_{\tilde{B}}^{-1} A_{\tilde{N}}) x_{\tilde{N}}.$$

The linear system of equations is updated by:

$$\begin{bmatrix} A_{\tilde{B}}^{-1} & 0 \\ -c_{\tilde{B}}^T A_{\tilde{B}}^{-1} & 1 \end{bmatrix} \begin{bmatrix} 0 & A_N & I_m \\ -1 & c^T & 0_m \end{bmatrix} \begin{bmatrix} f \\ x \\ s \end{bmatrix} = \begin{bmatrix} A_{\tilde{B}}^{-1} & 0 \\ -c_{\tilde{B}}^T A_{\tilde{B}}^{-1} & 1 \end{bmatrix} \begin{bmatrix} b \\ 0 \end{bmatrix} = \begin{bmatrix} A_{\tilde{B}}^{-1} b \\ -c_{\tilde{B}}^T A_{\tilde{B}}^{-1} b \end{bmatrix}.$$

We just computed the upper part, for the lower part, the same principle applies: $c_{\tilde{B}}^T A_{\tilde{B}}^{-1}$ multiplies $[A_N, I_m]$ so the columns corresponding to $\tilde{B}$ and the corresponding coefficients $x_{\tilde{B}}$ of $[x, s]^T$ will give $c_{\tilde{B}}^T x_{\tilde{B}}$, while the other columns will give $c_{\tilde{B}}^T A_{\tilde{B}}^{-1} A_{\tilde{B}} x_{\tilde{B}}$, and $-f - c_{\tilde{B}}^T x_{\tilde{B}} - c_{\tilde{B}}^T A_{\tilde{B}}^{-1} A_{\tilde{N}} x_{\tilde{N}} + c_{\tilde{B}}^T x_{\tilde{B}} + c_{\tilde{N}}^T x_{\tilde{N}} = -c_{\tilde{B}}^T A_{\tilde{B}}^{-1} b$. So

$$s = b - A_N x, \;\; f = c^T x$$

got updated to

$$x_{\tilde{B}} = A_{\tilde{B}}^{-1} b - A_{\tilde{B}}^{-1} A_{\tilde{N}} x_{\tilde{N}}, \;\; f = c^T x = c_{\tilde{B}}^T A_{\tilde{B}}^{-1} b + (c_{\tilde{N}}^T - c_{\tilde{B}}^T A_{\tilde{B}}^{-1} A_{\tilde{N}}) x_{\tilde{N}}. \quad (4.1)$$

The multiplication by $A_{\tilde{B}}^{-1}$ is done by a Gaussian elimination, in such a way that the column corresponding to the new basic variable gets a 1 in row i and 0 elsewhere, elsewhere includes the objective function row:

- Multiply row i of A_N by $1/a_{iq}$, where a_{iq} is the coefficient in the ith row and qth column of A_N.
- For row $i' \neq i$ of A, add $-a_{i'q}/a_{iq}$ times row i to row i', where $a_{i'q}$ is the coefficient in the i'th row and qth column of A_N.
- Add $-c_q/a_{iq}$ times row i to the last (objective function) row.

For an explicit computation, the tableau from the initial stage gets updated to:

x_1 column		x_q column	s_i column		
$\frac{a_{i1}}{a_{iq}}$	$\dots$	$\frac{a_{iq}}{a_{iq}} = 1$	$\frac{1}{a_{iq}}$	$\frac{b_i}{a_{iq}}$	row i
$a_{i'1} - \frac{a_{i'q}}{a_{iq}} a_{i1}$	$\dots$	$a_{i'q} - \frac{a_{i'q}}{a_{iq}} a_{iq} = 0$	0	$b_{i'} - \frac{a_{i'q}}{a_{iq}} b_i$	row i'
$c_1 - a_{i1} \frac{c_q}{a_{iq}}$	$\dots$	$c_q - a_{iq} \frac{c_q}{a_{iq}} = 0$	$-\frac{c_q}{a_{iq}}$	$-0 - \frac{c_q}{a_{iq}} b_i$	

and the matrix $A_{\tilde{B}}$ at this stage is an identity matrix where the ith column got replaced by the column $[a_{1q}, \dots, a_{iq}, \dots, a_{mq}]^T$, so its inverse is also an identity matrix, where the ith column got replaced this time by the column $[-\frac{a_{1q}}{a_{iq}}, \dots, \frac{1}{a_{iq}}, \dots, -\frac{a_{mq}}{a_{iq}}]^T$.

The coefficients $c_{\tilde{N}}^T - c_{\tilde{B}}^T A_{\tilde{B}}^{-1} A_{\tilde{N}}$ appear in the last row of the tableau for the non-basic variables (we have zeroes for basic variables), $-c_{\tilde{B}}^T A_{\tilde{B}}^{-1} b$ in the last column of the last row, as a result of the multiplication $[-c_{\tilde{B}}^T A_{\tilde{B}}^{-1}, 1] \begin{bmatrix} A_N & I_m \\ c^T & 0_m \end{bmatrix} x = -c_{\tilde{B}}^T A_{\tilde{B}}^{-1} b$.

At this step of the tableau, the last column containing -0 gets updated to $-0 - \frac{b_i}{a_{iq}} c_q = -0 - \lambda r_q$, so we can read the updated value of the objective function by negating the coefficient contained in the last row last column. Also, c_q becomes 0 for the new basic variable x_q, and 0 becomes $-\frac{c_q}{a_{iq}}$ for the new non-basic variable s_i. The last row of the tableau thus contains $c_j - a_{ij}\frac{c_q}{a_{iq}}$ for $j = 1, \ldots, n$, and 0 after that but for the term that corresponds to s_i, for which it is $-\frac{c_q}{a_{iq}}$ and

$$\sum_j \left(c_j - a_{ij}\frac{c_q}{a_{iq}}\right) x_j - s_i\frac{c_q}{a_{iq}} = \sum_j c_j x_j - \frac{c_q}{a_{iq}}\left(\sum_j a_{ij}x_j + s_i\right) = c^T x - \frac{c_q}{a_{iq}} b_i$$

by looking at the ith row of $[A_N, I_m]\begin{bmatrix} x \\ s \end{bmatrix}$.

We can now iterate the process (call A_N, b, c the newly obtained matrix and vectors), which gives the simplex algorithm in tableau form.

Algorithm 12 Simplex Algorithm

```
    Input: an LP in standard form max $c^T x$, such that $Ax = b,\ x \geq 0$.
    Output: a vector $x^*$ that maximizes the objective function $c^T x$ (or that
the LP is unbounded).
 1: Start with an initial BFS $x$ with basis $B$ and $N = \{1 \ldots n\} \backslash B$;
 2: Create the corresponding tableau.
 3: while (there is a $q$ such that $c_q > 0$) do
 4:     Choose the pivot column $q$.
 5:     if ($a_{iq} \leq 0$ for all $i$) then
 6:         the LP is unbounded, stop.
 7:     Choose a pivot row $i$, that is among $a_{iq} > 0$, choose $i$ such that $b_i/a_{iq}$
    is minimized.
 8:     Multiply row $i$ by $1/a_{iq}$.
 9:     For $i' \neq i$ add $-a_{i'q}/a_{iq}$ times row $i$ to row $i'$.
10:     Add $-c_q/a_{iq}$ times row $i$ to the objective function row.
```

Example 4.15. Consider the following LP:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + 3x_2 \leq 9 \\ & 2x_1 + x_2 \leq 8 \\ & x_1, x_2 \geq 0 \end{aligned}$$

and introduce the slack variables s_1, s_2. A BFS is given by $[0, 0, 9, 8]$, and for this BFS, the objective function is taking the value 0. Using slack variables, we write the initial tableau

x_1	x_2	s_1	s_2	b
1	3	$\boxed{1}$	$\boxed{0}$	9
2	1	$\boxed{0}$	$\boxed{1}$	8
1	1	0	0	0

We are looking for a column q for which $c_q > 0$, pick $q = 1$. This corresponds to the non-basic variable x_1 which we want to increase. For choosing the row, compute $b_1/a_{11} = 9$ and $b_2/a_{21} = 8/2 = 4$, so the minimum is given by choosing the row $i = 2$.

You may want to keep in mind how this step of the algorithm relates to the constraints of the LP: since x_1 and x_2 are constrained by $x_1 + 3x_2 \leq 9$, $2x_1 + x_2 \leq 8$, the first equation says that x_1 could be at most 9 if $x_2 = 0$, the second equation says that $2x_1$ could be at most 8, that is x_1 could be at most $8/2 = 4$ if $x_2 = 0$. In order not to violate any constraint, we choose the smallest increment for x_1, which is $x_1 = 4$.

If the rows are called ρ_1, ρ_2, ρ_3, $\rho_2' = \rho_2/2$, $\rho_1' = \rho_1 - \rho_2'$, $\rho_3' = \rho_3 - \rho_2'$:

$$\begin{array}{cccc|c}
x_1 & x_2 & s_1 & s_2 & b \\
\boxed{0} & 5/2 & \boxed{1} & -1/2 & 5 \\
\boxed{1} & 1/2 & \boxed{0} & 1/2 & 4 \\
\hline
0 & 1/2 & 0 & -1/2 & -4
\end{array}$$

The objective function increased from 0 to 4. The new BFS is $[4, 0, 5, 0]$. Let us also illustrate (4.1) on this example. The last row of the tableau is

$$\begin{aligned}
0 \cdot x_1 + \tfrac{1}{2}x_2 - \tfrac{1}{2}s_2 &= (1-1)x_1 + (1 - \tfrac{1}{2})x_2 - \tfrac{1}{2}s_2 \\
&= x_1 + x_2 - \tfrac{1}{2}(2x_1 + x_2 + s_2) \\
&= x_1 + x_2 - 4
\end{aligned}$$

since $2x_1 + x_2 + s_2 = 8$. Thus we have that the objective function $c^T x = x_1 + x_2$ can be read from the last row of the tableau:

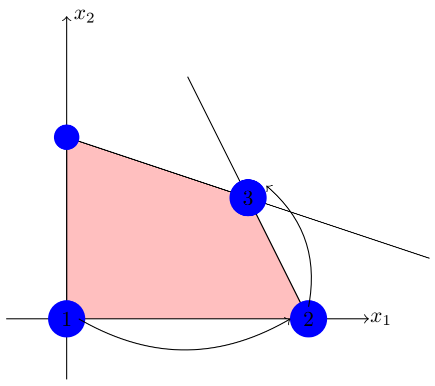
$$c^T x = \underbrace{(0 \cdot x_1 + \tfrac{1}{2}x_2 - \tfrac{1}{2}s_2)}_{\text{last row}} + \underbrace{4}_{\text{–last row, last column}} .$$

Next for the column pivot 2, we choose the pivot row to be 1, since $5/(5/2) = 2$ while $4/(1/2) = 8$. Then $\rho_1'' = (2/5)\rho_1'$, $\rho_2'' = \rho_2' - \rho_1''/2$, $\rho_3'' = \rho_3' - \rho_1'/2$:

$$\begin{array}{cccc|c}
x_1 & x_2 & s_1 & s_2 & b \\
\boxed{0} & \boxed{1} & 2/5 & -1/5 & 2 \\
\boxed{1} & \boxed{0} & -1/5 & 3/5 & 3 \\
\hline
0 & 0 & -1/5 & -2/5 & -5
\end{array}$$

We see that $c_q \leq 0$ for all q, thus the algorithm stops. The objective function has value 5. The optimal solution is given by $x_1 = 3$, $x_2 = 2$.

Geometrically, the algorithm goes from one BFS to another as shown below: it starts at $(0, 0)$, then goes to $(4, 0)$, and then to $(3, 2)$.



Artificial Variables. We argued above that if the matrix A contains I_m as an $m \times m$ submatrix, and $b \geq 0$, then we have an immediate BFS. That $b \geq$ is not a restriction, one can always multiply the corresponding equality in the LP standard form so that $-b_i \leq 0$ becomes $b_i \geq 0$. Finding an $m \times m$ submatrix inside A may not be easy, if we can cannot just take the columns corresponding to the slack variables. Of course, we can try some exhaustive search, pick m columns of A until we get a choice which is linearly independent, and then we inverse this $m \times m$ submatrix. Alternatively, we can introduce *artificial variables* $w_1, \ldots, w_m$ and solve the LP:

$$\begin{aligned} \min_{x,w} \quad & \textstyle\sum_{i=1}^m w_i \\ s.t. \quad & Ax + w = b \\ & x, w \geq 0. \end{aligned}$$

We can solve this LP using the Simplex Algorithm (and note the presence of the identity matrix in front of w).

- If the original problem $Ax = b$, $x \geq 0$, has a feasible solution, then the above LP has for optimal value 0 and $w = 0$. This optimal solution with $w = 0$ gives the desired BFS for $Ax = b$, $x \geq 0$.
- If the above LP has an optimal value which is strictly positive, then the original LP is not feasible.

This is called the *Two Phase Simplex Algorithm.* See Exercise 38 for an Example.

4.3 Duality

Definition 4.10. Given a *primal program*

$$\begin{aligned} P : \max \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

where $c, x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and A is an $m \times n$ matrix of rank m, $m \leq n$, the *dual* of P is defined by

$$\begin{aligned} D : \min \quad & y^T b \\ s.t. \quad & y^T A \geq c^T \\ & y \geq 0 \end{aligned}$$

where $c \in \mathbb{R}^n$, $b, y \in \mathbb{R}^m$ and A is an $m \times n$ matrix of rank m, $m \leq n$.

It is not hard to see that the dual of the dual is the primal (see Exercise 39).

Theorem 4.6. (Weak Duality Theorem) *If x is feasible for P and y is feasible for D, then $c^T x \leq b^T y$.*

Proof. Since y is feasible for D, $y^T A \geq c^T$, and since x is feasible for P, $x \geq 0$ so

$$c^T x \leq (y^T A)x.$$

We flip the same argument. Since x is feasible for P, $Ax \leq b$, and since y is feasible for D, $y \geq 0$ so

$$y^T Ax \leq y^T b.$$

Hence

$$c^T x \leq (y^T A)x \leq b^T y.$$

□

Corollary 4.7. *If y is a feasible solution for D, then P is bounded. Similarly, if x is a feasible solution for P, then D is bounded.*

Proof. If y is a feasible solution for D, then any feasible solution x for P satisfies $c^T x \leq b^T y$ so $b^T y$ is an upper bound for any x feasible.

If x is a feasible solution for P, then any feasible solution y for d satisfies $b^T y \geq c^T x$, so $c^T x$ is a lower bound for any y feasible. □

Corollary 4.8. *If x^* is feasible for P, and y^* is feasible for D and $c^T x^* = b^T y^*$, then x^* is optimal for P and y^* is optimal for D.*

Proof. For all x feasible for P, the Weak Duality theorem tells us that

$$c^T x \leq b^T y^* = c^T x^*.$$

This shows that x^* is optimal (maximal) for P.

Similarly, for all y feasible for D, the Weak Duality theorem tells us that

$$b^T y \geq c^T x^* = b^T y^*.$$

This shows that y^* is optimal (minimal) for D. □

Example 4.16. Consider the primal program

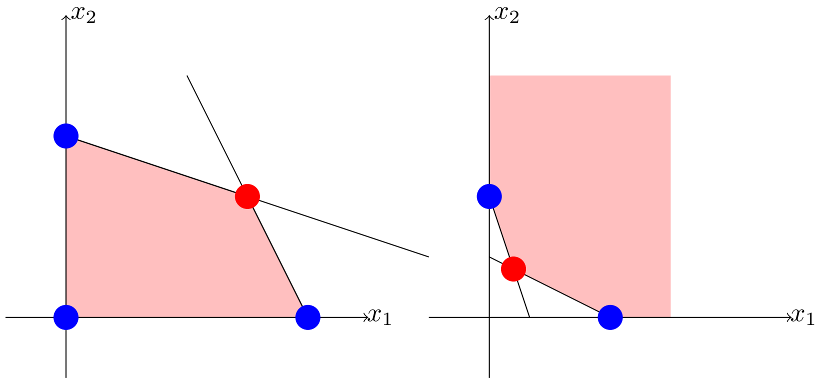
$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + 3x_2 \leq 9 \\ & 2x_1 + x_2 \leq 8 \\ & x_1, x_2 \geq 0 \end{aligned}$$

with $b^T = [9, 8]$ and $c^T = [1, 1]$.

We compute its dual:

$$\begin{aligned} \max \quad & 9y_1 + 8y_2 = b^T y \\ s.t. \quad & y_1 + 2y_2 \geq 1 \\ & 3y_1 + y_2 \geq 1 \\ & y_1, y_2 \geq 0. \end{aligned}$$

The points $x^* = (3, 2)$ and $y^* = (1/5, 2/5)$ are both feasible, $c^T x^* = 5 = b^T y$ so both are optimal.



Weak duality states than any feasible solution of the dual gives an upper bound on any solution of the primal (and vice-versa). There is a stronger version stated next, that says that values of the optimal solutions for the primal and dual match.

Theorem 4.9. (Strong Duality Theorem) *If P has an optimal solution x^*, then D has an optimal solution y^* such that $c^T x^* = b^T y^*$.*

Proof. Write the constraints of P as

$$Ax + s = b, \ x, s \geq 0,$$

where s is the vector of slack variables (an LP already in the form $Ax = b$ can be written with inequalities by noting that $Ax = b \iff Ax \leq b, \ -Ax \leq -b$).

Consider the bottom row in the final tableau of the Simplex Algorithm applied to P:

$$\begin{array}{c|c|c} x \text{ columns} & s \text{ columns} & b \text{ column} \\ & & \\ c_1^* \dots c_n^* & -y_1^* \dots - y_m^* & -f^* \end{array}$$

where for now $c_1^*, \dots, c_n^*, -y_1^* \dots - y_m^*$ are just some notations for the reduced costs that appear at this stage of the tableau computations. What we know is that f^* is by construction of the tableau the optimal value of the objective function $c^T x$, and that all the reduced costs are negative or zero, since we assumed that this is the final tableau, thus:

$$c_j^* \leq 0 \text{ for all } j, \quad -y_i^* \leq 0 \text{ for all } i.$$

Recall (see (4.1)) that the last row of the tableau can be read as $c^T x = (c^*)^T x - (y^*)^T s + f^*$ where $s = b - Ax$, thus

$$c^T x = f^* - b^T y^* + ((c^*)^T + (y^*)^T A)x$$

and using $x = 0$ in this expression gives $f^* = b^T y^*$. But if this is the case, then

$$c^T x = ((c^*)^T + (y^*)^T A)x \Rightarrow c = c^* + A^T y^*. \tag{4.2}$$

Since $c_j^* \leq 0$ for all j, we get

$$A^T y^* \geq c.$$

This combined with $y_j^* \geq 0$ shows that y^* is feasible for D, and $f^* = b^T y^*$ gives the objective function of D at y^*, namely it is the optimal value of P. Using the Weak Duality Theorem, y^* is optimal for D. $\square$

Example 4.17. Consider the primal problem

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t \quad & x_1 + 3x_2 + s_1 = 9 \\ & 2x_1 + x_2 + s_2 = 8 \\ & x_1, x_2, s_1, s_2 \geq 0 \end{aligned}$$

whose dual problem is

$$\begin{aligned} \min \quad & 9y_1 + 8y_2 \\ s.t \quad & y_1 + 2y_2 \geq 1 \\ & 3y_1 + y_2 \geq 1 \\ & y_1, y_2 \geq 0 \end{aligned}$$

The initial tableau for the primal is:

$$\begin{array}{cccc|c} x_1 & x_2 & s_1 & s_2 & b \\ 1 & 3 & \boxed{1} & \boxed{0} & 9 \\ 2 & 1 & \boxed{0} & \boxed{1} & 8 \\ \hline 1 & 1 & 0 & 0 & 0 \end{array}$$

corresponding to the initial BFS $x = [0, 0, 9, 8]$.

We already computed in Example 4.15 that the final tableau is

$$\begin{array}{cccc|c} x_1 & x_2 & s_1 & s_2 & b \\ \boxed{0} & \boxed{1} & 2/5 & -1/5 & 2 \\ \boxed{1} & \boxed{0} & -1/5 & 3/5 & 3 \\ \hline 0 & 0 & \underbrace{-1/5}_{-y_1^*} & \underbrace{-2/5}_{-y_2^*} & -5 \end{array}$$

corresponding to the BFS $x^* = [3, 2, 0, 0]$, with value 5 for the objective function. According to the proof of the Strong Duality Theorem, the coefficients y_1^*, y_2^* read in the last row of the tableau form an optimal solution for the dual. We can check that $9\frac{1}{5} + 8\frac{2}{5} = 5$, thus the primal has an optimal solution $x^* = [3, 2, 0, 0]$, and the dual has an optimal solution $y^* = [1/5, 2/5]$ such that both their objective functions take value 5 at their respective optimal solutions. This confirms that $y^* = [1/5, 2/5]$ is indeed an optimal solution for the dual.

We saw an LP may be feasible and bounded, feasible and unbounded, and infeasible. Thus for a primal and its dual, there are a priori 9 possibilities:

	P feasible bounded	P feasible unbounded	P infeasible
D feasible bounded	✓	×	×
D feasible unbounded	×	×	✓
D infeasible	×	✓	✓

- The Weak Duality Theorem says that if a primal and its dual are both feasible, then both are bounded feasible.
- The Strong Duality Theorem says if an LP has an optimal solution (thus feasible and bounded), then its dual cannot be infeasible.

The following theorem can be proven as a corollary of the Strong Duality Theorem.

Theorem 4.10 (The Equilibrium Theorem). *Let x^* and y^* be feasible solutions for a primal and its dual respectively. Then x^* and y^* are optimal if and only if*

(1) $\sum_{j=1}^n a_{ij}x_j^* < b_i$ (or $(Ax^*)_i < b_i$) $\Rightarrow y_i^* = 0$ for all i,

(2) $\sum_{i=1}^m a_{ij}y_i^* > c_j$ (or $(y^*A)_j > c_j$) $\Rightarrow x_j^* = 0$ for all j.

Proof. ($\Leftarrow$) (1) The sum $\sum_{i=1}^m y_i^* b_i$ contains terms for which $\sum_{j=1}^n a_{ij}x_j^* = b_i$, and terms for which $\sum_{j=1}^n a_{ij}x_j^* < b_i$ but in this case $y_i^* = 0$:

$$\sum_{i=1}^m y_i^* b_i = \sum_{i=1}^m y_i^* (\sum_{j=1}^n a_{ij}x_j^*) = \sum_{i=1}^m \sum_{j=1}^n a_{ij}x_j^* y_i^*.$$

(2) Repeating the above argument, namely that when $\sum_{i=1}^m a_{ij}y_i^* > c_j$, $x_j^* = 0$:

$$\sum_{i=1}^m \sum_{j=1}^n a_{ij}x_j^* y_i^* = \sum_{j=1}^n c_j x_j^*.$$

Putting both equalities together gives

$$\sum_{i=1}^m y_i^* b_i = \sum_{j=1}^n c_j x_j^* \iff b^T y^* = c^T x^*$$

and by Corollary 4.8, x^* and y^* are optimal.

($\Rightarrow$) We repeat the proof of the Weak Duality Theorem. Since y^* is feasible, $(y^*)^T A \geq c^T$, and since x^* is feasible, $x^* \geq 0$, thus

$$c^T x^* \leq (y^*)^T A x^*.$$

Since x^* is feasible, $Ax \leq b$, and since y^* is feasible, $y \geq 0$, so

$$(y^*)^T A x^* \leq (y^*)^T b \Rightarrow c^T x^* \leq (y^*)^T A x^* \leq b^T y^*,$$

or equivalently

$$\sum_{j=1}^n c_j x_j^* \leq \sum_{i=1}^m \sum_{j=1}^n y_i^* a_{ij} x_j^* \leq \sum_{i=1}^m y_i^* b_i.$$

But we know more, we know that x^* and y^* optimal, so we may invoke the Strong Duality Theorem to tell us that $c^T x^* = b^T y^*$ and in fact all the above inequalities and equalities.

The first inequality thus becomes

$$\sum_{j=1}^n \left(c_j - \sum_{i=1}^m \sum_{j=1}^n y_i^* a_{ij} \right) x_j^* = 0.$$

Since x^* is feasible, each x_j^* is nonnegative, and the constraints $y^T A \geq c^T$ on the dual forces $c_j - \sum_{i=1}^m \sum_{j=1}^n y_i^* a_{ij} \leq 0$, thus for this sum (composed of only non-positive terms) to be zero, for each j, either one of the terms needs to be zero. Hence if $\sum_{i=1}^m \sum_{j=1}^n y_i^* a_{ij} > c_j$, $x_j^* = 0$.

We repeat the same argument. The second inequality gives the equality

$$\sum_{i=1}^m \left(\sum_{j=1}^n a_{ij} x_j^* - b_i \right) y_i^* = 0.$$

Since y^* is feasible, each y_i^* is nonnegative, and the constraints $Ax \leq b$ on the primal forces $\sum_{j=1}^n a_{ij}x_j^* - b_i \leq 0$, thus for the sum to be zero, for each i, either one of the terms needs to be zero. Hence, if $\sum_{j=1}^n a_{ij}x_j^* < b_i$, then $y_i^* = 0$. $\square$

(1) and (2) are sometimes called the *complementary slackness conditions*. They require that a strict inequality (slackness) in a variable in a standard problem implies that the complementary constraint in the dual be satisfied with equality.

Example 4.18. Consider the primal problem

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t \quad & x_1 + 2x_2 \leq 4 \\ & 4x_1 + 2x_2 \leq 12 \\ & -x_1 + x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

whose dual problem is

$$\begin{aligned} \min \quad & 4y_1 + 12y_2 + y_3 \\ s.t \quad & y_1 + 4y_2 - y_3 \geq 1 \\ & 2y_1 + 2y_2 + y_3 \geq 1 \\ & y_1, y_2, y_3 \geq 0 \end{aligned}$$

We are given that $(x_1^*, x_2^*) = (8/3, 2/3)$, and $x_1^* + x_2^* = 10/3$. Since $x_1^* > 0$, $x_2^* > 0$, this means that constraints on y^* must be met with equality, namely

$$y_1^* + 4y_2^* - y_3^* = 1, \; 2y_1^* + 2y_2^* + y_3^* = 1.$$

Now using $(x_1^*, x_2^*) = (8/3, 2/3)$, we have

$$\begin{aligned} x_1^* + 2x_2^* &= 4 \leq 4 \\ 4x_1^* + 2x_2^* &= 12 \leq 12 \\ -x_1^* + x_2^* &= -2 < 1. \end{aligned}$$

Since we have two constraints with equality, and one with strict inequality, the one with strict inequality means $y_3^* = 0$. Thus the two equalities in y_1^*, y_2^* become

$$y_1^* + 4y_2^* = 1, \; 2y_1^* + 2y_2^* = 1.$$

We can solve for y_1^*, y_2^*: $(y_1^*, y_2^*) = (1/3, 1/6)$. Since this vector is feasible, the if part of the Equilibrium Theorem implies it is optimal. Furthermore $4(1/3) + 12(1/6) = 10/3$, which is the same as the optimal value for the primal, another check for optimality!

4.4 Two Person Zero Sum Games

We give an application of linear programming to game theory, for zero sum games with two players.

A two person zero sum game is a game with two players, where one player wins and the other player loses.

Example 4.19. Consider the game of "Odds and Evens" (also known as matching pennies). Suppose that Player 1 takes evens, and Player 2 takes odds. Each player simultaneously shows either one finger or two fingers, if the number of fingers matches, the result is even, Player 1 wins (say 2 dollars) otherwise it is odd, Player 2 wins (say 2 dollars). Each player has 2 strategies, show one finger or two fingers.

	Player 2, 1 finger	Player 2, 2 fingers
Player 1, 1 finger	2	-2
Player 1, 2 fingers	-2	2

This table is read from the view point of Player 1, that is a 2 is a gain of 2 for Player 1, and a -2 is a loss of 2 for Player 1. It is read the other way round for Player 2, a 2 is bad for Player 2, this is what he has to pay to Player 1, while a -2 is good, it means Player 1 owes him 2.

We get a *pay off matrix*

$$A = \begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix}$$

We consider games where players move simultaneously. For two players, we have a general *pay off matrix* $A = (a_{ij})$ which summarizes the gains/losses of both players. Suppose Player 1 indexes the rows of A, and Player 2 its columns, which means that there is one row of A for every move of Player 1, and one column of A for every move of Player 2. We say that Player 1 wins $a_{ij} > 0$ from Player 2, so $a_{ij} > 0$ is good for Player 1, and bad for Player 2.

Suppose we are given the pay off matrix

$$\begin{bmatrix} -5 & 3 & 1 & 20 \\ 5 & 5 & 4 & 6 \\ -4 & 6 & 0 & -5 \end{bmatrix}$$

For Player 1, each row consists of a move, so for the first move, the worst is -5, for the second move, the worst is 4, for the last move, the worst is -5. For Player 1, it makes sense to play move 2, because this moves ensures a win.

For Player 2, each column consists of a move, and the worst for him is the largest gain for Player 1, so for the first move, the worst is 5, for the second move, the worst is 6, for the 3rd move, the worst is 4, and for the last move, the worst is 20. So it makes sense for Player 2 to use move 3, this ensures the least loss.

It turns out that for this example $a_{23} = 4$ is both the smallest entry in row 2, and the largest entry in column 3. The game is thus considered "solved", or we say that the game has an *equilibrium* which has value 4 (4 is sometimes called a saddle point), because there is exists a strategy which is best for both players: at the same time, it maximizes Player 1's win, and minimizes Player 2's loss.

Example 4.20. The game "Scissors-Paper-Stone" has no saddle point. In this game, each player shows either scissors, paper or stone with a hand: scissors cut paper, paper covers stone, stone breaks scissors,

	Scissors	Paper	Stone
Scissors	0	1	-1
Paper	-1	0	1
Stone	1	-1	0

The worst for Player 1 is -1 on each row. The worst for Player 2 is 1 on each column, so there is no saddle point.

The game "Odds and Evens" has no saddle point either.

We speak of "mixed" strategy when each move of a player is probabilistic. In contrast, a "pure" strategy is when each move is deterministic. In the above solved game, there was a pure strategy leading to an equilibrium: Player 1 chooses move 2, and Player 2 chooses move 3.

Consider a "mixed" strategy. Player 1 has a set of moves, move i, $i = 1, \dots, m$, whose gains/losses are specified by the coefficients a_{ij} of the pay off matrix A. Each move is done with probability p_i. If Player 2 plays j, Player 1's expected pay off is

$$\sum_{i=1}^{m} a_{ij} p_i.$$

Player 1 wishes to maximize (over $p = (p_1, \dots, p_m)$) his minimal expected pay off:

$$\min_j \sum_{i=1}^{m} a_{ij} p_i.$$

Similarly, Player 2 has a set of moves, and move j, $j = 1, \dots, n$ is attached a probability q_j. Player 2 wishes to minimize over $q = (q_1, \dots, q_n)$ his maximal expected loss:

$$\max_i \sum_{j=1}^{n} a_{ij} q_j.$$

There may be other ways to define what both players are interested in doing, but we will see that this formulation is meaningful in that it will lead to a saddle point.

Let us write the Player's optimization problem as a linear program. Player 2 wants to minimize its maximal expected loss, that is

$$\min_q \left(\max_i \sum_{j=1}^{n} a_{ij} q_j \right)$$

such that

$$\sum_{j=1}^{n} q_j = 1, \; q \geq 0.$$

An equivalent formulation is

$$\begin{array}{rl} \min\limits_{q,v} & v \\ s.t. & \sum_{j=1}^{m} a_{ij}q_j \leq v, \ i = 1, \ldots, m \\ & \sum_{j=1}^{n} q_j = 1 \\ & q \geq 0. \end{array}$$

This is an equivalent formulation because the first constraint $\sum_{j=1}^{m} a_{ij}q_j \leq v, \ i = 1, \ldots, m$ ensures that $\sum_{j=1}^{m} a_{ij}q_j$ will take the highest possible value v, while v itself will be minimized. The second and third constraints tell us that q defines a probability distribution. Similarly, for Player 1, we have

$$\begin{array}{rl} \max\limits_{p,v} & v \\ s.t. & \sum_{i=1}^{m} a_{ij}p_i \geq v, \ j = 1, \ldots, n \\ & \sum_{i=1}^{m} p_i = 1 \\ & p \geq 0. \end{array}$$

Now we add a constant k to each a_{ij} so that $a_{ij} > 0$ for all i, j, this does not change the nature of the game (if the constraints $\sum_{j=1}^{m} a_{ij}q_j \leq v$ is replaced by the constraints $\sum_{j=1}^{m}(a_{ij} + k)q_j = \sum_{j=1}^{m} a_{ij}q_j + k \leq v \Rightarrow \sum_{j=1}^{m} a_{ij}q_j \leq v - k$, then just set $v' = v - k$ and the objection function becomes to minimize $v' + k$, this thus does not change the optimal solution, though the value of the objective function is of course shifted by k), but it guarantees $v > 0$. For example, if all $a_{ij} < 0$, $v = 0$ could be a candidate for an upper bound. So without loss of generality, assume $a_{ij} > 0$ for all i, j. Then do the following change of variables $x_j = q_j/v$ (now v cannot be zero, and x_j is not infinite) inside Player 2's LP:

$$\begin{array}{rl} \min\limits_{x,v} & v \\ s.t. & \sum_{j=1}^{m} a_{ij}x_j \leq 1, \ i = 1, \ldots, m \\ & \sum_{j=1}^{n} x_j = \frac{1}{v} \\ & x \geq 0 \end{array}$$

and since $v = 1/\sum_{j=1}^{n} x_j$ from the second constraint, we finally get

$$\begin{array}{rl} (P) : \max & \sum_{j=1}^{n} x_j \\ s.t. & Ax \leq 1 \\ & x \geq 0 \end{array}$$

which is a primal LP in the form we are familiar with. We do the same transformation for Player 1. Assume $a_{ij} > 0$ and set $y_i = p_i/v$:

$$\begin{array}{rl} \max\limits_{p,v} & v \\ s.t. & \sum_{i=1}^{m} a_{ij}y_i \geq v, \ j = 1, \ldots, n \\ & \sum_{i=1}^{m} y_i = \frac{1}{v} \\ & y \geq 0, \end{array}$$

which becomes

$$\begin{aligned}(D): \min_{y} \quad & \textstyle\sum_{i=1}^{m} y_i \\ s.t. \quad & A^T y \geq 1 \\ & y \geq 0,\end{aligned}$$

and we see that P and D are dual, hence they have the same optimal value (that is, assuming they are feasible and bounded), reached respectively for x^* and y^*.

Thus Player 1 can guarantee an expected gain of a least

$$v = 1/\sum_{i=1}^{m} y_i^*$$

using the strategy $p = vy^*$.

Player 2 can guarantee an expected loss of a most

$$v = 1/\sum_{j=1}^{n} x_j^*$$

using the strategy $q = vx^*$.

The game is thus solved, and has value v.

Example 4.21. Consider the game of "Odds and Evens", given by the pay off matrix:

	Player 2, 1 finger, q_1	Player 2, 2 fingers, q_2
Player 1, 1 finger, p_1	2	-2
Player 1, 2 fingers, p_2	-2	2

We saw there is no saddle point for a pure strategy, so we consider a mixed strategy where each move is assigned a probability.

If we look at Player 2, if Player 1 chooses move 1, his expected loss is $2q_1 - 2q_2$, if Player 1 chooses move 2, his expected loss is $-2q_1 + 2q_2$. So Player 2 will try to minimize the maximum loss between $2q_1 - 2q_2$ and $-2q_1 + 2q_2$. The corresponding LP is

$$\begin{aligned}\min_{q,v} \quad & v \\ s.t. \quad & 2q_1 - 2q_2 \leq v \\ & -2q_1 + 2q_2 \leq v \\ & q_1 + q_2 = 1 \\ & v, q \geq 0,\end{aligned}$$

and in order to have only pay off coefficients that are positive, we add a constant $k = 3$ to every coefficient, to get

$$\begin{array}{rc} \min\limits_{q,v} & v \\ s.t. & 5q_1 + q_2 \leq v \\ & q_1 + 5q_2 \leq v \\ & q_1 + q_2 = 1 \\ & v, q \geq 0. \end{array}$$

We do the change of variables $x_j = q_j/v$ to get

$$\begin{array}{rc} \max & x_1 + x_2 \\ s.t. & 5x_1 + x_2 \leq 1 \\ & x_1 + 5x_2 \leq 1 \\ & x \geq 0. \end{array}$$

We then solve this LP. The initial tableau is:

$$\begin{array}{cccc|c} 5 & 1 & 1 & 0 & 1 \\ 1 & 5 & 0 & 1 & 1 \\ \hline 1 & 1 & 0 & 0 & 0 \end{array}$$

for the BFS $[0, 0, 1, 1]$ whose objective function takes value 0. Pick $q = 1$ as column pivot, then the row pivot is $i = 1$:

$$\begin{array}{cccc|c} 1 & 1/5 & 1/5 & 0 & 1/5 \\ 0 & 24/5 & -1/5 & 1 & 4/5 \\ \hline 0 & 4/5 & -1/5 & 0 & -1/5 \end{array}$$

This is the BFS $[1/5, 0, 0, 4/5]$ whose objective function takes value $1/5$. Pick $q = 2$ as column pivot, then pick the row pivot $i = 2$:

$$\begin{array}{cccc|c} 1 & 0 & 1/5 + 1/120 & -1/24 & 1/5 - 1/30 \\ 0 & 1 & -1/24 & 5/24 & 1/6 \\ \hline 0 & 0 & -1/5 + 1/30 & -1/6 & -1/5 - 4/30 \end{array}$$

This is the BFS $[1/6, 1/6, 0, 0]$ whose objective function takes value $1/3 = 1/v$. To $x_1 = 1/6 = q_1/v$ corresponds the probability $q_1 = vx_1 = 3/6 = 1/2$. Thus Player 2's optimal strategy is to choose move 1 with probability $q_1 = 1/2$, and move 2 with probability $q_2 = 1/2$. A posteriori, this is the strategy that makes most sense, due to the symmetry of the pay off matrix.

4.5 Exercises

Exercise 35. Compute all the basic feasible solutions of the following LP:

$$\begin{aligned} \min \quad & x_1 + x_2 \\ s.t. \quad & x_1 + 2x_2 \geq 3 \\ & 2x_1 + x_2 \geq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Exercise 36. Consider the linear program:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & -x_1 + x_2 \leq 1 \\ & 2x_1 + x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Prove by computing reduced costs that $[1/3, 4/3, 0, 0]$ is an optimal solution for this LP.

Exercise 37. Solve the following LP using the simplex algorithm:

$$\begin{aligned} \max \quad & 6x_1 + x_2 + x_3 \\ s.t. \quad & 9x_1 + x_2 + x_3 \leq 18 \\ & 24x_1 + x_2 + 4x_3 \leq 42 \\ & 12x_1 + 3x_2 + 4x_3 \leq 96 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Exercise 38. Solve the following LP using the simplex algorithm:

$$\begin{aligned} \max \quad & 3x_1 + x_3 \\ s.t. \quad & x_1 + 2x_2 + x_3 = 30 \\ & x_1 - 2x_2 + 2x_3 = 18 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Exercise 39. Show that the dual of the dual is the primal.

Exercise 40. Consider the following LP:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + 2x_2 \leq 4 \\ & 4x_1 + 2x_2 \leq 12 \\ & -x_1 + x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Compute its dual.

Exercise 41. Consider the following LP:

$$\begin{aligned} \max \quad & 2x_1 + 4x_2 + x_3 + x_4 \\ s.t. \quad & x_1 + 3x_2 + x_4 \leq 4 \\ & 2x_1 + x_2 \leq 3 \\ & x_2 + 4x_3 + x_4 \leq 3 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

Compute its dual. Show that $x = [1, 1, 1/2, 0]$ and $y = [11/10, 9/20, 1/4]$ are optimal solutions for respectively this LP and its dual.

Exercise 42. In Exercise 40, we computed the dual of the LP:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + 2x_2 \leq 4 \\ & 4x_1 + 2x_2 \leq 12 \\ & -x_1 + x_2 \leq 1 \\ & x_1, x_2 \geq 0. \end{aligned}$$

Solve both this LP and its dual.

Exercise 43. Solve the "Scissors-Paper-Stone" whose pay off matrix is

$$\begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}.$$

Exercise 44. Solve the game whose pay off matrix is

$$\begin{bmatrix} 2 & -1 & 6 \\ 0 & 1 & -1 \\ -2 & 2 & 1 \end{bmatrix}.$$

Chapter 5

The Network Simplex Algorithm

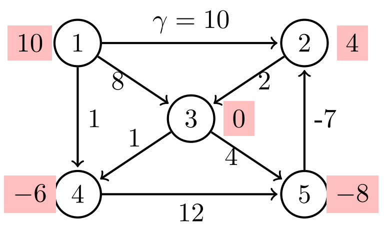
Recall the problem of min cost flow discussed in Chapter 3. We are given a network $G = (V, E)$, with $|V| = n$ nodes, $|E| = m$ edges. Each node v has a demand $d(v)$ which can be positive or negative, and a demand function d must satisfy $\sum_{v \in V} d(v) = 0$. Each edge e has a cost $\gamma(e)$, which can also be positive or negative. We recall from Definition 3.8 that a flow in such a network is a map $f : E \to \mathbb{R}$ with

1. $0 \leq f(e) \leq c(e)$ for all $e \in E$, where $c(e)$ represents the edge capacity,
2. $d(v) = \sum_{u \in O(v)} f(v,u) - \sum_{u \in I(v)} f(u,v)$ for all $v \in V$,

and that the min cost flow problem consists of finding a flow such that the cost $\gamma(f) = \sum_{e \in E} \gamma(e) f(e)$ is minimized.

For now, we assume that all capacities are infinite.

The examples and notes below are closely following the notes [2]. As an example, consider a network with $n = 5$ nodes and $m = 8$ edges. Nodes 4 and 5 are in demand ($d(v) < 0$ for $v = 4, 5$), node 3 is a transit node ($d(v) = 0$ for $v = 3$) and nodes 1 and 2 are in supply ($d(v) > 0$ for $v = 1, 2$). We can check that $\sum_{v \in V} d(v) = 10 + 4 - 8 - 6 = 0$. The cost $\gamma(e)$ is written next to each edge e.



With the notation $d_i = d(i)$, $\gamma_{ij} = \gamma((i,j))$ and $f_{ij} = f((i,j))$, write a demand vector d, a cost vector γ and the cost function $\gamma(f)$ as

$$d = \begin{bmatrix} 10 \\ 4 \\ 0 \\ -6 \\ -8 \end{bmatrix}, \ \gamma = \begin{bmatrix} \gamma_{12} \\ \gamma_{13} \\ \gamma_{14} \\ \gamma_{23} \\ \gamma_{34} \\ \gamma_{35} \\ \gamma_{45} \\ \gamma_{52} \end{bmatrix} = \begin{bmatrix} 10 \\ 8 \\ 1 \\ 2 \\ 1 \\ 4 \\ 12 \\ -7 \end{bmatrix}, \ \gamma(f) = \sum_{e \in E} \gamma(e) f(e) = \gamma^T f.$$

Thus solving the min cost flow problem in this network is equivalent to solve

$$\min \gamma^T f$$

subject to the constraints that define a flow, namely

1. $0 \leq f_{ij}$ since all edges capacities are infinite for now,
2. $d_i = \sum_j f_{ij} - \sum_j f_{ji}$ for all i.

The first constraint just says $f \geq 0$, so let us look at the second constraint. Take $i = 1$, and the demand d_1 which is 10 at node 1 must be satisfied, that is $10 = f_{12} + f_{13} + f_{14}$ must hold. Repeating this constraint for every node in the network gives:

$$\begin{array}{ll} \min & \gamma^T f \\ s.t & \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & -1 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} f_{12} \\ f_{13} \\ f_{14} \\ f_{23} \\ f_{34} \\ f_{35} \\ f_{45} \\ f_{52} \end{bmatrix} = \begin{bmatrix} 10 \\ 4 \\ 0 \\ -6 \\ -8 \end{bmatrix} \\ & f \geq 0, \end{array}$$

or in short,

$$\begin{array}{rl} \min & \gamma^T f \\ s.t & Af = d \\ & f \geq 0, \end{array}$$

where the matrix A is actually the *incidence matrix* of the graph G, that is A is an $n \times m$ matrix whose n rows and m columns correspond to the nodes and edges of G respectively, such that $a_{i,j} = -1$ if the edge that labels the jth column enters node i, 1 if it leaves i and 0 otherwise (the opposite sign convention is also used). In our example, row 2 corresponds to node 2, the columns are indexed by the edges:

$$\begin{array}{cccccccc} (1,2) & (1,3) & (1,4) & (2,3) & (3,4) & (3,5) & (4,5) & (5,2) \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{array}$$

and the edge $(1,2)$ enters 2, so the sign is -1, while $(2,3)$ leaves 2, so the sign is 1. Columnwise, the column for (i,j) will have a 1 in row i and a -1 in row j.

Notice that the rows of A are linearly dependent, if you sum them, you obtain the whole zero vector. The linear dependency is happening because of the constraint that $\sum_{v\in V} d(v) = 0$, and the positive demand must be equal to the negative one. This means that one of the equations can be canceled.

5.1 Uncapacitated Networks

In min cost flow networks, the flow f is the unknown to be found, so we will change the notation and use x to denote the flow, this makes the notation consistent with linear programming, the view point we will adopt next, without hopefully creating confusion. We have motivated above that the min cost flow network problem in the case where the edge capacities are unbounded (in which case the network is called uncapacitated) can be formulated as a linear program:

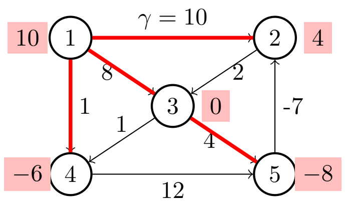
$$\begin{array}{rl} \min & \gamma^T x \\ s.t & Ax = d \\ & x \geq 0, \end{array}$$

where the $n \times m$ matrix A is the incidence matrix of the graph G. The rank of A is $n-1$ (one row equation is redundant, because positive demand must equal negative demand), thus A contains $n-1$ linearly independent columns.

Example 5.1. Consider again the network of the introduction with incidence matrix

$$\begin{array}{cccccccc} (1,2) & (1,3) & (1,4) & (2,3) & (3,4) & (3,5) & (4,5) & (5,2) \end{array}$$
$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & -1 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 \end{bmatrix}.$$

Then the columns corresponding to $(1,2)(1,3)(1,4)(3,5)$ are linearly independent, and they form a spanning tree (disregarding orientation):



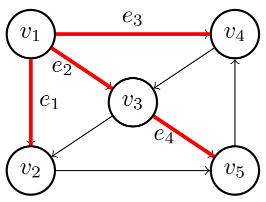
$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

We will show that this is no coincidence, that that for a connected network, $n-1$ linearly independent vectors correspond to a spanning tree (disregarding orientation), and vice-versa.

Lemma 5.1. *Let T be a set of columns of A such that the corresponding edges form a spanning tree. Then the columns in T are linearly independent.*

Proof. Every tree is made of n nodes $v_1, \dots, v_n$, and for every $i \geq 2$, there is exactly one edge with end point equal to v_i, and the other equal to one of the previously labeled $v_1, \dots, v_{i-1}$. So visit the tree and label the nodes by $v_1, \dots, v_n$ and the edges by $e_1, \dots, e_{n-1}$ in order of visit. Then order the $n-1$ columns of the spanning tree and the n rows of A following the same ordering. Discard the first row, to obtain an $(n-1) \times (n-1)$ matrix A' which is upper triangular and contains nonzero elements on its diagonal by construction. Its determinant is the product of the diagonal coefficients, thus $\det(A') \neq 0$ and A has rank $n-1$, showing that the columns in T are linearly independent. □

Example 5.2. We illustrate the algorithm given in this proof on our running example. To start with, if we visit the nodes in the order of their labels, that is, $v_i = i$, $i = 1, \dots, 5$, then the edges are visited in the order $e_1 = (1,2)$, $e_2 = (1,3)$, $e_3 = (1,4)$, $e_4 = (3,5)$, which is the current ordering for the columns of the rank $n-1$ submatrix which is already such that once the first row is removed, it is an upper diagonal matrix. But we can also change the ordering, e.g., choose $v_1 = 1$, $v_2 = 4$, $v_3 = 3$, $v_4 = 2$, $v_5 = 5$, then edges are visited in the order $e_1 = (v_1, v_2)$, $e_2 = (v_1, v_3)$, $e_3 = (v_1, v_4)$, $e_4 = (v_3, v_5)$.



$$\begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{bmatrix} 1 & 1 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

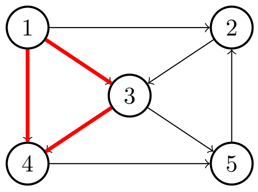
Conversely:

Lemma 5.2. *If a subset T of $n-1$ columns of A are linearly independent, then the corresponding edge set is a spanning tree.*

Proof. We do a proof by contradiction. Suppose that the corresponding edge set is not a spanning tree, then it must contain a cycle (we disregard the orientation of the edges for the cycle). It is sufficient to show that the columns of A associated with a cycle are linearly dependent.

Given a cycle, the coefficients of the linear combination are obtained as follows. Arbitrarily choose one direction for the cycle, and set the coefficient of the column of (i,j) in the cycle as 1 if the edge has the same orientation as the cycle, and -1 otherwise. This linear combination of the columns of A involved in the cycle yields the zero vector. □

Example 5.3. Consider the cycle $(1,3),(3,4),(4,1)$. Then the sign for the column of $(1,3)$ should be 1, the sign for the column of $(3,4)$ should be 1 as well, and the sign for the column of $(1,4)$ should be -1.



$$\begin{array}{ccc} (1,3) & (3,4) & (1,4) \\ \end{array}$$
$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & -1 \\ 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}$$

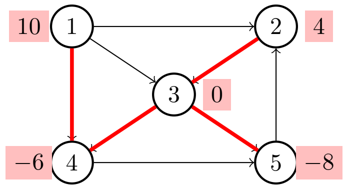
Both lemmas together are summarized in this theorem.

Theorem 5.3. *Given a connected flow network, with incidence matrix A, a submatrix A_T of size $(n-1)\times(n-1)$ has rank $n-1$ if and only if the edges associated with the set T of columns of A_T form a spanning tree.*

Note that this statement considers a square submatrix A_T, this is because once $n-1$ columns are chosen which correspond to a spanning tree, the incidence matrix A has n rows, thus we start by by considering an $n\times(n-1)$ matrix of rank $n-1$, out of which we get A_T by removing a row.

Basic Feasible Solutions. We now recall that a basic solution for a linear program (LP) is obtained by choosing $n-1$ linearly independent columns of the constraint matrix A (where $n-1$ is the rank of A). This means, a basic solution for the min cost flow problem is a spanning tree. As for any LP, when taking $n-1$ columns of A, we can get columns which are dependent, but when they are not (that is, we have a spanning tree), let A_T denote the corresponding matrix (with one row removed, again, one row is redundant), and we have either $x = A_T^{-1}d \geq 0$, that is x is a basic feasible solution (BFS), or $x = A_T^{-1}d < 0$ (this is a basic solution, but it is infeasible). Presence of a basic variable taking value 0 gives a degenerate solution (that is, we have an empty edge in the spanning tree, a *degenerate spanning tree*), as for standard LPs.

Example 5.4. As a first example, consider the spanning tree showed below and its corresponding columns of A.



$$\begin{array}{cccc} (1,4) & (3,4) & (2,3) & (3,5) \\ \end{array}$$
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & -1 & 1 \\ -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

We need to solve

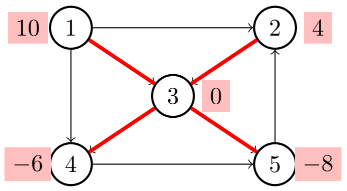
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & -1 & 1 \\ -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_{14} \\ x_{34} \\ x_{23} \\ x_{35} \end{bmatrix} = \begin{bmatrix} 10 \\ 4 \\ 0 \\ -6 \\ -8 \end{bmatrix}.$$

We could remove a row of this matrix to get A_T and invert it, but it is probably easier to just solve the system, since we immediately see that:

$$x_{14} = 10, x_{23} = 4, x_{35} = 8, -x_{14} - x_{34} = -6 \Rightarrow x_{34} = -4.$$

This solution is thus basic but not feasible since $x_{34} = -4$.

As a second example, consider the spanning tree showed below and its corresponding columns of A.



$$\begin{array}{cccc} (1,3) & (2,3) & (3,4) & (3,5) \\ \hline 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & -1 & 1 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{array}$$

We need to solve

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & -1 & 1 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_{13} \\ x_{23} \\ x_{34} \\ x_{35} \end{bmatrix} = \begin{bmatrix} 10 \\ 4 \\ 0 \\ -6 \\ -8 \end{bmatrix}.$$

We immediately see that:

$$x_{13} = 10, x_{23} = 4, x_{34} = 6, x_{35} = 8$$

and we get a BFS.

Checking for Optimality. Now that we know how to get basic feasible solutions (BFS) for our min cost flow problem, the next step is to see whether it is optimal. We will use duality theory for that.

The min cost flow problem is:

$$\begin{aligned} \min \quad & \gamma^T x \\ s.t \quad & Ax = d \\ & x \geq 0, \end{aligned}$$

which is equivalent to

$$\begin{aligned} \max \quad & -\gamma^T x \\ s.t \quad & Ax \leq d \\ & Ax \geq d \\ & x \geq 0, \end{aligned}$$

and we can alternatively write

$$\begin{aligned} (P) : \max \quad & -\gamma^T x \\ s.t \quad & \begin{bmatrix} A \\ -A \end{bmatrix} x \leq \begin{bmatrix} d \\ -d \end{bmatrix} \\ & x \geq 0, \end{aligned}$$

where the constraint matrix is of size $2n \times m$, and x is the vector containing the flow on all the m edges of the network.

The dual problem is thus given by:

$$\begin{aligned} \min \quad & [d^T, -d^T] \begin{bmatrix} s \\ t \end{bmatrix} \\ s.t \quad & \begin{bmatrix} s^T & t^T \end{bmatrix} \begin{bmatrix} A \\ -A \end{bmatrix} \geq -\gamma^T \\ & s, t \geq 0. \end{aligned}$$

Let us rewrite a bit the dual:

$$\begin{aligned} \min \quad & d^T s - d^T t \\ s.t \quad & s^T A - t^T A \geq -\gamma^T \\ & s, t \geq 0 \\ \Rightarrow \min \quad & d^T(s - t) \\ s.t \quad & A^T(s - t) \geq -\gamma \\ & s, t \geq 0 \\ \Rightarrow \max \quad & d^T u \\ s.t. \quad & -A^T u \geq -\gamma \end{aligned}$$

by setting $u := t - s$. Note that $t, s \geq 0$ means that u can be positive or negative. This gives our final dual:

$$\begin{aligned} (D) : \max \quad & d^T u \\ s.t \quad & A^T u \leq \gamma \end{aligned}$$

or equivalently, since we recall that in the transpose A^T of the incidence matrix A, every row contains exactly one 1 in column i and one -1 in column j, for the edge (i, j):

$$\begin{aligned} (D) : \max \quad & \textstyle\sum_{i=1}^n d_i u_i \\ s.t \quad & u_i - u_j \leq \gamma_{ij}, \text{for all } (i, j) \in E. \end{aligned}$$

Let us keep in mind that the primal has one redundant equation, which means that the dual has one redundant variable, which can be set to zero without altering the problem.

Consider a basic feasible solution for the primal, and partition the edges of the network into the set B of edges of the spanning tree, and the set N of other edges (since we know that the edges of the spanning tree corresponding to a basic solution, we use the LP notation B and N).

If an edge (i,j) in B is in the optimal solution (which is not degenerate), then $x_{ij} > 0$ and by complementary slackness from Theorem 4.10, the corresponding dual constraint must be satisfied with equality:

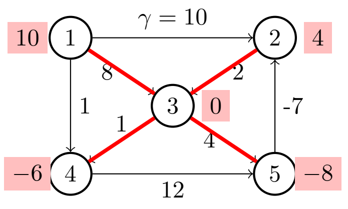
$$u_i - u_j = \gamma_{ij}, \text{ for all } (i,j) \in B$$

for u_i, u_j coefficients of an optimal solution u for the dual. There are $n-1$ such equations, one for each edge of the spanning tree, and we have n variables, one of them being zero (corresponding to the redundant equation in A). With a system of $n-1$ equations in $n-1$ variables, we can solve for u, and check that u is feasible (otherwise it cannot be optimal), namely check that

$$u_i - u_j \leq \gamma_{ij}, \text{ for all } (i,j) \in N.$$

If u satisfies the complementary slackness conditions, it is optimal by the Equilibrium Theorem (Theorem 4.10) (namely, both x and u are feasible, (2) is satisfied by construction and (1) is always satisfied since all constraints for the primal are equalities).

Example 5.5. Consider the spanning tree showed below



$$\begin{array}{c} (1,3)(2,3)(3,4)(3,5) \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & -1 & 1 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \end{array} \begin{bmatrix} x_{13} \\ x_{23} \\ x_{34} \\ x_{35} \end{bmatrix} = \begin{bmatrix} 10 \\ 4 \\ 0 \\ -6 \\ -8 \end{bmatrix}.$$

and its corresponding BFS:

$$x_{13} = 10, x_{23} = 4, x_{34} = 6, x_{35} = 8.$$

The corresponding dual equations are:

$$\begin{aligned} u_1 - u_3 &= \gamma_{13} = 8 \\ u_2 - u_3 &= \gamma_{23} = 2 \\ u_3 - u_4 &= \gamma_{34} = 1 \\ u_3 - u_5 &= \gamma_{35} = 4 \end{aligned}$$

We choose to set $u_3 = 0$ (any u_i could be chosen, but u_3 appears often, so it suggests that it will be easy to solve the system). Then

$$u_1 = 8,\ u_2 = 2,\ u_4 = -1,\ u_5 = -4.$$

Finally we check the feasibility of u by looking at $u_i - u_j \leq \gamma_{ij}$ for $(i,j) \in N$:

$$\begin{array}{lll} u_1 - u_2 & \leq \gamma_{12} = 10 \Rightarrow 8 - 2 \leq 10 & \checkmark \\ u_1 - u_4 & \leq \gamma_{14} = 1 \Rightarrow 8 + 1 = 9 \not\leq 1 & \times \\ u_4 - u_5 & \leq \gamma_{45} = 12 & \\ u_5 - u_2 & \leq \gamma_{52} = -7 & \end{array}$$

so $u = (8, 2, 0, -1, -4)$ is not optimal.

If we keep in mind the structure of the general simplex algorithm (see Algorithm 10) which we recall below for convenience, what we have done so far is start with an initial BFS (described in terms of spanning tree), then check for optimality (by complementary slackness).

Algorithm 10 General Simplex Algorithm

```
    Input: an LP in standard form max $c^T x$, such that $Ax = b,\ x \geq 0$.
    Output: a vector $x^*$ that maximizes the objective function $c^T x$.
1: Start with an initial BFS.
2: while (the current BFS is not optimal) do
3:        Move to an improved adjacent BFS.
4: return $x^*$ =BFS;
```

We are now left with moving to an improved adjacent BFS (or verify that the problem is unbounded).

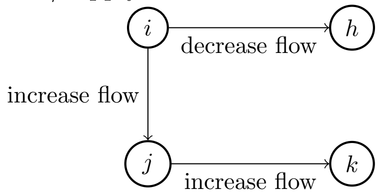
Pivoting. If a solution x does not verify the optimality conditions, there must exist an edge $(i,j) \in N$ such that

$$u_i - u_j > \gamma_{ij} \iff \gamma_{ij} - u_i + u_j = \gamma_{ij} - u^T A_{ij} < 0$$

where A_{ij} denotes the column of A associated to (i,j). We have seen in the proof of the Strong Duality Theorem (see (4.2)) that the reduced cost associated to x is given by $\gamma - A^T u$ thus $\gamma_{ij} - u_i + u_j$ is the reduced cost associated to the variable x_{ij}, and having a positive reduced cost is profitable when it comes to minimize the cost of the flow. We thus need to update u_i, u_j such that they satisfy this inequality instead of violating it, but then, this means that x_{ij} will change accordingly, and will go from being 0 to nonzero. This means activating the edge (i,j). We recognize the pivoting phase of the simplex algorithm, where a variable x_{ij} moves from N to B.

So add (i,j) to B. Then (i,j) forms a cycle within the spanning tree. In order to restore a BFS, we must get rid of this cycle, that is remove one edge of the cycle. Let us see which one.

To maintain the feasibility of the current solution, one must necessarily alter the value of the flow on all edges of the cycle, increasing the flow on edges that have the same orientation as (i, j) (called direct edges), and decreasing the flow of the edges having opposite direction (reverse edges), to keep the demand/supply satisfied.



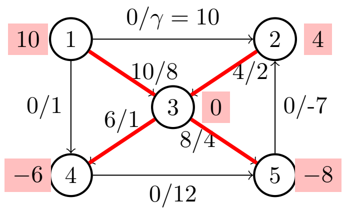
- If there is at least one reverse edge, as the flow on (i, j) is increased, the flow on reverse edges decreases, and there is at least one edge, say (u, v) such that its flow reaches zero before all the other edges, then (u, v) leaves B. The maximum feasible value ϑ of the flow in (i, j) is then

  $$\vartheta(i,j) = \min\{x_{hk},\ (h,k) \text{ reverse edge in the cycle}\}.$$

  The new BFS is thus obtained by simply increasing by ϑ the flow in direct edges and decreasing by the same amount the flow in reverse edges of the cycle.

- If all edges are direct, then we know that increasing the flow on (i, j) reduces the cost, so we increase the flow on the other edges of the cycle accordingly, and since we consider an uncapacitated network, there is no upper bound on how much we can increase the flow on (i, j) (or on any edge of the cycle) and the problem is then unbounded. Note that we are activating the edge (i, j) because it improves the objective function, so the only way the cost is getting minimized by increasing the flow on every edge is when $\sum_{(h,k) \text{ in cycle}} \gamma_{hk} < 0$ (this is a negative cycle of infinite capacity, see Definition 3.10 and the remarks below).

Example 5.6. We continue the example given by the spanning tree below, whose corresponding BFS is the flow shown:



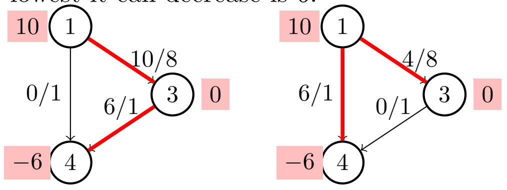
We check the feasibility of u by looking at $u_i - u_j \leq \gamma_{ij}$ for $(i, j) \in N$ and we already found in the previous example an inequality that is not satisfied:

$$\begin{aligned} u_1 - u_2 &= \gamma_{12} = 10 \Rightarrow 8 - 2 \leq 10 \ \checkmark \\ u_1 - u_4 &= \gamma_{14} = 1 \Rightarrow 8 + 1 = 9 \not\leq 1 \ \times \end{aligned}$$

We thus activate the edge (1,4), which creates a cycle $(1) \to (3) \to (4) \leftarrow (1)$, and

$$\vartheta(1,4) = \min\{x_{13}, x_{34}\} = \min\{6, 10\} = 6.$$

When we are pushing more flow on (1,4), it decreases on the reverse edges, and the lowest it can decrease is 6.



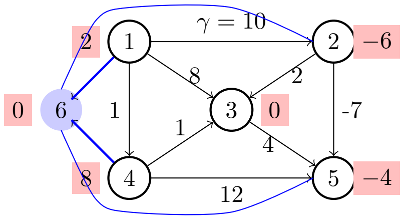
We thus update the BFS:

$$\begin{bmatrix} x_{12} \\ x_{13} \\ x_{14} \\ x_{23} \\ x_{34} \\ x_{35} \\ x_{45} \\ x_{52} \end{bmatrix} = \begin{bmatrix} 0 \\ 10 \\ 0 \\ 4 \\ 6 \\ 8 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} x_{12} \\ x_{13} \\ x_{14} \\ x_{23} \\ x_{34} \\ x_{35} \\ x_{45} \\ x_{52} \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ 6 \\ 4 \\ 0 \\ 8 \\ 0 \\ 0 \end{bmatrix}.$$

We recognize the pivot operation, where x_{14} goes out of B, and is replaced by x_{34}. See Exercise 45 for the end of this example.

Artificial variables. Both the network simplex algorithm and the simplex algorithm start with a basic feasible solution. In the regular simplex algorithm, we saw that we can use artificial variables to find a basic feasible solution that will serve as a starting point for the algorithm.

Let us mimic this for the network simplex algorithm, and introduce "artificial edges". We convey all the flows produced by nodes with positive demands into an artificial node via artificial edges, and redistribute such a flow via other artificial edges, connecting the new node to nodes with a negative demand.

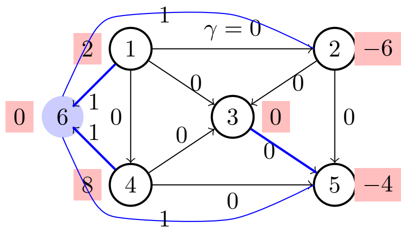


As for the simplex algorithm, the objective is to minimize the flow on artificial edges, so in the artificial problem, we set the cost of each artificial edge equal to 1, and the cost of all others to 0.

We can solve this new min cost flow problem using the network simplex algorithm, but for this to be helping with our original problem, we need to be able to find an immediate BFS. If there is no transit node, then artificial edges form an initial BFS. If there is a transit node, artificial edges are not enough to form a spanning tree of the artificial network (as illustrated above). To obtain an initial BFS, we can add an arbitrary edge of the original network for each transit node to connect it to the tree formed by the artificial edges, so that no loop is formed in the process.

If some artificial variables have a nonzero flow, the problem is infeasible. If the flow in all artificial variables is 0, these can be removed (together with artificial nodes) to get a BFS for the original network.

Example 5.7. In this network (close to our running example, but for the direction of the edges (3,4), (5,2) and demands), we created artificial edges, and added the edge (3,5) to form a spanning tree, our initial BFS. Edges in the original network have a cost of 0 (we can use them as much as we want), while artificial edges have a cost of 1, since we ideally would like not to use them.



We want to minimize $\gamma^T x$ where γ is a vector with zero everywhere, but for $\gamma_{46} = \gamma_{16} = \gamma_{62} = \gamma_{65} = 1$. We have an immediate BFS given by (note the last equation which describes that the artificial node 6 is a transit node)

$$\begin{array}{ccccc} (1,6) & (3,5) & (4,6) & (6,2) & (6,5) \\ \end{array}$$
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 \\ -1 & 0 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_{16} \\ x_{35} \\ x_{46} \\ x_{62} \\ x_{65} \end{bmatrix} = \begin{bmatrix} 2 \\ -6 \\ 0 \\ 8 \\ -4 \\ 0 \end{bmatrix}$$

from which we get

$$\begin{bmatrix} x_{16} \\ x_{35} \\ x_{46} \\ x_{62} \\ x_{65} \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 8 \\ 6 \\ 4 \end{bmatrix}.$$

In words, we are just setting the flow on the artificial edges so that it goes from the supply nodes to the demand nodes, based on their respective demands.

Using complementary slackness, we get:

$$\begin{aligned} u_1 - u_6 &= \gamma_{16} = 1 \\ u_4 - u_6 &= \gamma_{46} = 1 \\ u_6 - u_2 &= \gamma_{62} = 1 \\ u_6 - u_5 &= \gamma_{65} = 1 \\ u_3 - u_5 &= \gamma_{35} \leq 0 \end{aligned}$$

and setting $u_6 = 0$ gives $u_1 = u_4 = 1$, $u_2 = u_5 - 1$, $u_3 \leq -1$. Note that we have a degenerate solution since $x_{35} = 0$. We then check for feasibility:

$$\begin{aligned} u_1 - u_2 &= 2 \not\leq 0 \quad \times \\ u_4 - u_3 &\geq 2 \not\leq 0 \quad \times \end{aligned}$$

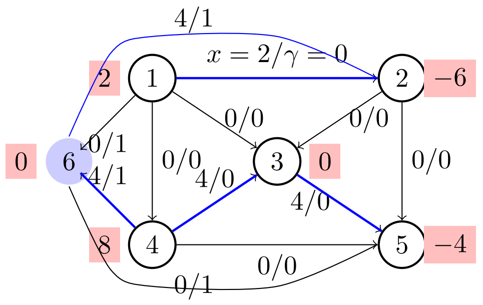
so $(1, 2)$ enters B, creating the cycle $(6) \leftarrow (1) \rightarrow (2) \leftarrow (6)$, where $x_{16} = 2 < x_{62} = 6$ so $(1, 6)$ leaves B. The BFS is updated, x_{16} is replaced by x_{12}, and x_{62} is decreased by 2:

$$\begin{bmatrix} x_{12} \\ x_{35} \\ x_{46} \\ x_{62} \\ x_{65} \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 8 \\ 4 \\ 4 \end{bmatrix}.$$

We notice that this cycle does not contain $(4, 3)$, so the second inequality is not changing, and $(4, 3)$ enters B, creating a cycle $(6) \leftarrow (4) \rightarrow (3) \rightarrow (5) \leftarrow (6)$, so $x_{65} = 4 < x_{46} = 8$ and $(6, 5)$ leaves B. The BFS is updated, x_{65} is replaced by x_{43}, x_{46} is decreased by 4, and x_{35} is increased by 4:

$$\begin{bmatrix} x_{12} \\ x_{35} \\ x_{46} \\ x_{62} \\ x_{43} \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 4 \\ 4 \\ 4 \end{bmatrix}.$$

At this point, the network flow is as follows:



We use complementary slackness once more:

$$\begin{aligned} u_1 - u_2 &= \gamma_{12} = 0 \\ u_4 - u_6 &= \gamma_{46} = 1 \\ u_6 - u_2 &= \gamma_{62} = 1 \\ u_4 - u_3 &= \gamma_{43} = 0 \\ u_3 - u_5 &= \gamma_{35} = 0 \end{aligned}$$

and setting $u_6 = 0$ gives $u_3 = u_4 = u_5 = 1$, $u_2 = u_1 = -1$. We then check for feasibility:

$$\begin{aligned} u_1 - u_6 &= -1 \\ u_1 - u_4 &= 0 \\ u_1 - u_3 &= 0 \\ u_4 - u_5 &= 0 \\ u_6 - u_5 &= -1 \\ u_2 - u_5 &= -2 \end{aligned}$$

and since γ_{ij} takes for value 0 and 1, the vector u is feasible. Thus we have an optimal solution, but two artificial edges are not set to 0, thus the original problem is infeasible.

In fact, this is something that can be easily checked from the original network: node 2 is unreachable from node 4, and node 1 is unable to satisfy the demand of node 2.

5.2 Capacitated Networks

Suppose now that the edges have a finite capacity, that is the for every edge $e \in E$, $f(e) \leq c(e)$. We need to modify the current LP program

$$\begin{aligned} \min \quad & \gamma^T x \\ s.t \quad & Ax = d \\ & x \geq 0 \end{aligned}$$

and add the capacity constraint $x \leq c$ where c is the vector containing the capacities. Introducing a slack vector s, this becomes

$$\begin{aligned} \min \quad & \gamma^T x \\ s.t \quad & \begin{bmatrix} A & 0_m \\ I_m & I_m \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} = \begin{bmatrix} d \\ c \end{bmatrix} \\ & x, s \geq 0. \end{aligned}$$

Basic Feasible Solutions. We saw that in the uncapacitated case, basic solutions correspond to spanning trees, so we could partition the edges into the set B of edges belonging to the spanning tree, and the set N of the other edges.

For capacitated networks, we partition the edges into three sets, the set $\mathcal{S}$ of saturated edges, that is edges where the flow is reaching its capacity, the set $\mathcal{N}$ of edges with zero flow , and the set $\mathcal{B}$ containing the other edges. We show next how this tripartition will help us to find basic feasible solutions.

Theorem 5.4. *Given a connected flow network, a basic feasible solution x and the corresponding tripartition $(\mathcal{N}, \mathcal{S}, \mathcal{B})$ of the edges, the edges in $\mathcal{B}$ do not form cycles.*

Proof. Suppose there is a cycle (disregarding direction) in $\mathcal{B}$, say (i_1, i_2),(i_2, i_3), (i_4, i_3), (i_4, i_5), (i_1, i_5), fix the cycle direction to be that of (i_1, i_2), thus (i_1, i_2), (i_2, i_3), (i_4, i_5) are in the same direction, and (i_4, i_3), (i_1, i_5) are not.

Let us look at the corresponding columns in the constraint matrix given by:

$$\begin{bmatrix} A & 0_m \\ I_m & I_m \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} = \begin{bmatrix} d \\ c \end{bmatrix}$$

Since the cycle is formed by edges in $\mathcal{B}$, this means that none of these edges are saturated, and so to each of them correspond a non-zero slack variable. The columns (and rows) of the constraint matrix involving the cycle are thus:

$$\begin{array}{c|cccccccccc|}
 & x_{i_1,i_2} & x_{i_2,i_3} & x_{i_4,i_3} & x_{i_4,i_5} & x_{i_1,i_5} & s_{i_1,i_2} & s_{i_2,i_3} & s_{i_4,i_3} & s_{i_4,i_5} & s_{i_1,i_5} \\
i_1 & 1 & 0 & 0 & 0 & 1 & & & & & \\
i_2 & -1 & 1 & 0 & 0 & 0 & & & & & \\
i_3 & 0 & -1 & -1 & 0 & 0 & & & & & \\
i_4 & 0 & 0 & 1 & 1 & 0 & & & & & \\
i_5 & 0 & 0 & 0 & -1 & -1 & & & & & \\
(i_1,i_2) & 1 & & & & & 1 & & & & \\
(i_2,i_3) & & 1 & & & & & 1 & & & \\
(i_4,i_3) & & & 1 & & & & & 1 & & \\
(i_4,i_5) & & & & 1 & & & & & 1 & \\
(i_1,i_5) & & & & & 1 & & & & & 1
\end{array}$$

The upper part can be interpreted as usual, in terms of edges. The lower part represents slack variables equations, (i_1, i_2) means that s_{i_1,i_2} is the slack variable corresponding to x_{i_1,i_2}.

Since x is a BFS, and the columns of $x_{i_1,i_2}, x_{i_2,i_3}, x_{i_4,i_5}$ are linearly independent, $x_{i_1,i_2}, x_{i_2,i_3}, x_{i_4,i_5}$ are part of the basic variables. So are the slack variables $s_{i_1,i_2}, s_{i_2,i_3}, s_{i_4,i_5}$, since their columns are linearly independent as well.

Now perform the following linear combination of the columns: use $(+1)$ for the columns of $x_{i,j}$ with direct edges and of $s_{i,j}$ with reverse edges, and use (-1) for the columns of $x_{i,j}$ with reverse edges, and $s_{i,j}$ with direct edges.

The choice of the coefficients ± 1 for $x_{i,j}$ makes explicit the presence of a cycle (and thus a linear combination), as done in the proof of Lemma 5.2.

$$\begin{array}{c} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ (i_1,i_2) \\ (i_2,i_3) \\ (i_4,i_3) \\ (i_4,i_5) \\ (i_1,i_5) \end{array} \begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & +1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & +1 \\ 1 & & & & \\ & 1 & & & \\ & & -1 & & \\ & & & 1 & \\ & & & & -1 \end{bmatrix}$$

Thus when summing these columns, the upper part becomes a zero vector. The choices of ±1 for $s_{i,j}$ ensures that the lower part also becomes a zero vector once the remaining columns are added. This shows that we obtain a zero vector, the columns are thus linearly dependent, and therefore, the corresponding 10 variables cannot all be basic variables, so some of the $x_{i_4,i_3}, x_{i_1,i_5}, s_{i_4,i_3}, s_{i_1,i_5}$ are not and must be zero. If either x_{i_4,i_3} or x_{i_1,i_5} is 0, then its edge belongs to $\mathcal{N}$ and not to $\mathcal{B}$, and if either s_{i_4,i_3} or s_{i_1,i_5} is 0, then its edge belongs to $\mathcal{S}$ and not to $\mathcal{B}$, a contradiction in all cases, and thus there cannot be a cycle in $\mathcal{B}$. $\square$

We have a network of n nodes and the largest set of edges we can have with no cycle is a spanning tree, which involves n nodes and $n-1$ edges.

As a consequence of the above theorem:

- In a basic solution, at most $n-1$ edges can belong to $\mathcal{B}$ (because adding one more edge necessarily gives a cycle).

Thus a set $\mathcal{B}$ with $n-1$ edges correspond to a spanning tree. However, unlike in the uncapacitated case, where the spanning tree gives immediately a basic feasible solution, in this case, we need to know $\mathcal{S}$ and $\mathcal{N}$, based on which we get a system of linear equations from which we can compute the BFS (so the BFS does not only depend on the spanning tree, but also on $\mathcal{S}$ and $\mathcal{N}$).

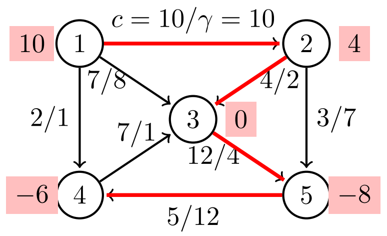
Note that:

- A choice of $\mathcal{S}$ and $\mathcal{N}$ means that we are setting flows with edges in $\mathcal{N}$ to be zero (non-basic variables), and flows with edges in $\mathcal{S}$ to be equal to their capacities (that is the corresponding slack variables become 0, thus are non-basic variables). So this sets basic and non-basic variables.

- We are then left with exactly $n-1$ basic variables to be computed, and they correspond to the edges of the spanning tree. We use demands at the nodes to obtain $n-1$ equations, so we can solve the system. We obtain a basic solution, but we need to check whether it is a basic feasible solution.

If there are saturated or empty edges in $\mathcal{B}$, then we have a degenerate basic solution (and a degenerate spanning tree).

Example 5.8. Consider the network below, where every edge e is given a capacity $c(e)$ and a cost $\gamma(e)$ (the lower capacity is assumed to be 0).



We can write a demand vector d, a cost vector γ and a capacity vector c as

$$d = \begin{bmatrix} 10 \\ 4 \\ 0 \\ -6 \\ -8 \end{bmatrix}, \ \gamma = \begin{bmatrix} \gamma_{12} \\ \gamma_{13} \\ \gamma_{14} \\ \gamma_{23} \\ \gamma_{34} \\ \gamma_{35} \\ \gamma_{45} \\ \gamma_{52} \end{bmatrix} = \begin{bmatrix} 10 \\ 8 \\ 1 \\ 2 \\ 7 \\ 4 \\ 1 \\ 12 \end{bmatrix}, \ c = \begin{bmatrix} c_{12} \\ c_{13} \\ c_{14} \\ c_{23} \\ c_{34} \\ c_{35} \\ c_{45} \\ c_{52} \end{bmatrix} = \begin{bmatrix} 10 \\ 7 \\ 2 \\ 4 \\ 3 \\ 12 \\ 7 \\ 5 \end{bmatrix}.$$

Consider the spanning tree given by $\mathcal{B} = \{(1,2),(2,3),(3,5),(5,4)\}$. Suppose that $\mathcal{N} = \{(4,3)\}$, $\mathcal{S} = \{(1,3),(1,4),(2,5)\}$.

The edge $(4,3) \in \mathcal{N}$ which means its flow is 0, so $x_{43} = 0$. For the edges in $\mathcal{S}$, they are saturated, that means:

$$x_{13} = c_{13} = 7, \ x_{14} = c_{14} = 2, \ x_{25} = c_{25} = 3.$$

So we have $n - 1 = 4$ flow variables to be computed for edges in $\mathcal{B}$, and we already know the values of the 4 flow variables given by $\mathcal{N}$ and $\mathcal{S}$, so we use demands at the nodes to solve:

$$\begin{aligned} x_{12} &= d_1 - x_{14} - x_{13} = 10 - 2 - 7 = 1 \\ x_{23} &= d_2 + x_{12} - x_{25} = 4 + 1 - 3 = 2 \\ x_{35} &= d_3 + x_{13} + x_{23} = 7 + 2 = 9 \\ x_{54} &= d_4 + x_{35} + x_{25} = -8 + 3 + 9 = 4, \end{aligned}$$

and we get a basic solution which is feasible.

Optimality. To figure out whether a BFS is optimal, we use duality, as for the uncapacitated case.

To the primal program:

$$\begin{aligned} (P) : \min \quad & \gamma^T x \\ s.t \quad & \begin{bmatrix} A & 0_m \\ I_m & I_m \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} = \begin{bmatrix} d \\ c \end{bmatrix} \\ & x, s \geq 0 \end{aligned}$$

corresponds the dual program (see Exercise 46 for the details of the computation):

$$\begin{aligned}\max \quad & [d^T, c^T]\begin{bmatrix}u\\ v\end{bmatrix}\\ s.t \quad & \begin{bmatrix}A^T & I_m\\ 0_m & I_m\end{bmatrix}\begin{bmatrix}u\\ v\end{bmatrix} \le \begin{bmatrix}\gamma\\ 0_m\end{bmatrix}.\end{aligned}$$

We then invoke complementary slackness. The primal involves the variables x and s. If $(i,j) \in \mathcal{B}$, then $x_{ij} > 0$ and $s_{ij} > 0$ (since the edge is not saturated), and complementary slackness for x_{ij} gives $u_i - u_j + v_{ij} = \gamma_{ij}$, while complementary slackness for s_{ij} gives $v_{ij} = 0$, and $u_i - u_j = \gamma_{ij}$:

$$\gamma_{ij} - u_i + u_j = 0,\ (i,j) \in \mathcal{B}.$$

As for the uncapacitated case, we get a system of $n-1$ equations in n variables, one of which can be fixed to 0. The values for the coefficients of u that are computed must satisfy the other dual constraints.

For an empty edge $(i,j) \in \mathcal{N}$, complementary slackness for $s_{ij} > 0$ gives $v_{ij} = 0$, so with $u_i - u_j + v_{ij} \le \gamma_{ij}$:

$$u_i - u_j \le \gamma_{ij},\ (i,j) \in \mathcal{N}.$$

For a saturated edge, $x_{ij} = c_{ij} > 0$ (an edge with 0 capacity can just be ignored in the network), complementary slackness for x_{ij} gives $u_i - u_j = \gamma_{ij} - v_{ij}$, with $v_{ij} \le 0$, and we get:

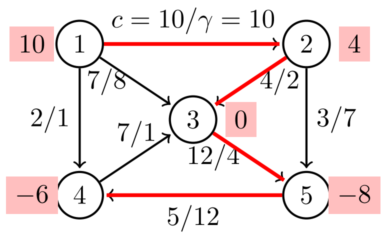
$$u_i - u_j \ge \gamma_{ij},\ (i,j) \in \mathcal{S}.$$

If all equations

$$\gamma_{ij} - u_i + u_j = 0,\ (i,j) \in \mathcal{B},\ u_i - u_j \le \gamma_{ij},\ (i,j) \in \mathcal{N},\ u_i - u_j \ge \gamma_{ij},\ (i,j) \in \mathcal{S},$$

are satisfied, then the flow is optimal.

Example 5.9. We continue Example 5.8, with spanning tree given by $\mathcal{B} = \{(1,2),(2,3),(3,5),(5,4)\}$, $\mathcal{N} = \{(4,3)\}$, and $\mathcal{S} = \{(1,3),(1,4),(2,5)\}$, corresponding to the BFS $x_{12} = 1$, $x_{23} = 2$, $x_{35} = 9$, $x_{54} = 4$ with $x_{13} = c_{13} = 7$, $x_{14} = c_{14} = 2$, $x_{25} = c_{25} = 3$ and $x_{43} = 0$.



For all $(i,j) \in \mathcal{B}$, $\gamma_{ij} - u_i + u_j = 0$:

$$\begin{aligned} \gamma_{12} - u_1 + u_2 = 0 &\quad \Rightarrow 10 - u_1 + u_2 = 0 \\ \gamma_{23} - u_2 + u_3 = 0 &\quad \Rightarrow 2 - u_2 + u_3 = 0 \\ \gamma_{35} - u_3 + u_5 = 0 &\quad \Rightarrow 4 - u_3 + u_5 = 0 \\ \gamma_{54} - u_5 + u_4 = 0 &\quad \Rightarrow 12 - u_5 + u_4 = 0. \end{aligned}$$

Set $u_4 = 0$. Then $u_5 = 12$, $u_3 = 16$, $u_2 = 18$, $u_1 = 28$. The vector u is not feasible, since $(2,5) \in \mathcal{S}$, but $u_2 - u_5 \geq \gamma_{25} = 7$ should be satisfied, while $18 - 12 = 4 \leq 7$.

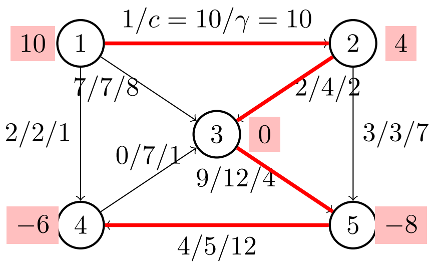
Pivoting. If the vector u is not feasible, and thus the corresponding flow x is not optimal, an edge (empty or full) that does not satisfy its constraint may be chosen to join $\mathcal{B}$, creating a cycle C, then accordingly one edge must leave.

Thus one must find the maximum flow ϑ circulating in the cycle C generated by the addition of (i,j). The value of ϑ may be limited by the capacity of the direct edges which saturate, as well as by reverse edges that are getting empty:

$$\vartheta = \min\{x_{pq},\ (p,q) \text{ direct edge in } C,\ c_{pq} - x_{pq},\ (p,q) \text{ reverse edge in } C\}.$$

The edge that determines ϑ is not basic anymore, and is replaced by (i,j).

Example 5.10. We continue our example

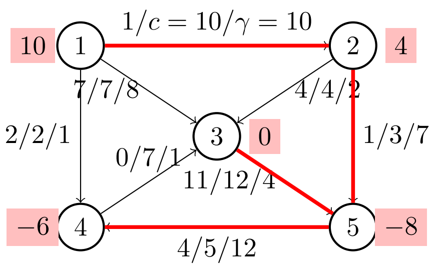


whose BFS is $x_{12} = 1$, $x_{23} = 2$, $x_{35} = 9$, $x_{54} = 4$ with $x_{13} = c_{13} = 7$, $x_{14} = c_{14} = 2$, $x_{25} = c_{25} = 3$ and $x_{43} = 0$.

The edge (2,5) which is currently saturated is activated, it creates the cycle C given by $(5) \leftarrow (2) \rightarrow (3) \rightarrow (5)$. Since (2,5) is saturated, we need to decrease it, which means increasing on the rest of the cycle whose edges are reversed. Then $x_{23} = 2, c_{23} = 4$ and $x_{35} = 9, c_{35} = 12$, which means that on edge (2,3), we could increase the flow by 2, while on the edge (3,5), we could increase the flow by 3. So (2,3) leaves, and (2,5) gets decreased by 2. So $x_{23} = 4$, $x_{25} = 1$ and $x_{35} = 11$.

The sets $\mathcal{B}, \mathcal{S}, \mathcal{N}$ get updated as follows:

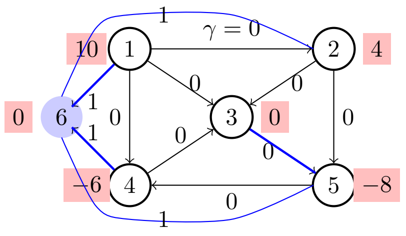
$$\mathcal{B} = \{(1,2),(3,5),(5,4),(2,5)\}, \mathcal{N} = \{(4,3)\}, \mathcal{S} = \{(1,3),(1,4),(2,3)\}.$$



This example has been used to illustrate the different steps of the capacitated simplex network algorithm. See Exercise 47 for a complete solution of this example.

Artificial variables. An initial solution can be used by introducing an artificial node and artificial edges (with as high a capacity as needed), mimicking the uncapacitated case.

Example 5.11. Consider our running example, and suppose we do not know any BFS. We introduce an artificial node, and corresponding artificial edges as shown below. The artificial edges together with the edge $(3, 5)$ form a spanning tree. Since it is a capacitated network, we need to assume that the artificial edges have each a capacity which is finite, but we do not specify the actual value of these capacities, e.g., $(1, 6)$ has capacity c_{12} and c_{12} is some value higher than whatever flow we will push through $(1, 6)$ ($c_{12} \geq 11$ will do here) so the edge is never saturated.



We want to minimize $\gamma^T x$ where γ is a vector with zero everywhere, but for $\gamma_{64} = \gamma_{16} = \gamma_{26} = \gamma_{65} = 1$. We have an immediate BFS given by (note the last equation which describes that the artificial node 6 is a transit node)

$$\begin{array}{ccccc} (1,6) & (3,5) & (6,4) & (2,6) & (6,5) \end{array}$$
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 \\ -1 & 0 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_{16} \\ x_{35} \\ x_{64} \\ x_{26} \\ x_{65} \end{bmatrix} = \begin{bmatrix} 10 \\ 4 \\ 0 \\ -6 \\ -8 \\ 0 \end{bmatrix}$$

from which we get

$$\begin{bmatrix} x_{16} \\ x_{35} \\ x_{64} \\ x_{26} \\ x_{65} \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \\ 6 \\ 4 \\ 8 \end{bmatrix}.$$

There is no difference with what we did for the uncapacitated case, we just get the flow to go from the supply nodes to the demand nodes via the artificial edges, the only technical difference is to suppose there are capacities such that we do not saturate edges.

Using complementary slackness, we get:

$$\begin{aligned} u_1 - u_6 &= \gamma_{16} = 1 \\ u_3 - u_5 &\leq \gamma_{35} = 0 \\ u_6 - u_4 &= \gamma_{64} = 1 \\ u_2 - u_6 &= \gamma_{26} = 1 \\ u_6 - u_5 &= \gamma_{65} = 1 \end{aligned}$$

and setting $u_6 = 0$ gives $u_1 = u_2 = 1$, $u_4 = u_5 = -1$ and $u_3 \leq -1$. We then check for feasibility:

$$\begin{aligned} u_1 - u_3 &\geq 2 \not\leq 0 \quad \times \\ u_1 - u_4 &= 2 \not\leq 0 \quad \times \\ u_2 - u_3 &\geq 2 \not\leq 0 \quad \times \\ u_2 - u_5 &= 2 \not\leq 0 \quad \times \end{aligned}$$

so $(1,3)$ is activated, creating the cycle $(1) \to (3) \to (5) \leftarrow (6) \leftarrow (1)$, then $x_{13} = 7$ is saturated and:

$$\begin{bmatrix} x_{16} \\ x_{35} \\ x_{64} \\ x_{26} \\ x_{65} \end{bmatrix} = \begin{bmatrix} 3 \\ 7 \\ 6 \\ 4 \\ 1 \end{bmatrix}.$$

This removes the degeneracy in x_{35}, but does not change the edges in the spanning tree. We then activate $(1,4)$, this creates the cycle $(1) \to (4) \leftarrow (6) \leftarrow (1)$, $x_{14} = 2$ saturates, and

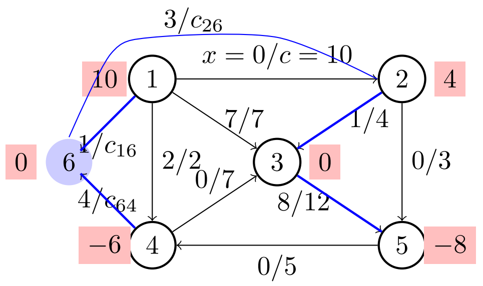
$$\begin{bmatrix} x_{16} \\ x_{35} \\ x_{64} \\ x_{26} \\ x_{65} \end{bmatrix} = \begin{bmatrix} 1 \\ 7 \\ 4 \\ 4 \\ 1 \end{bmatrix}.$$

The spanning tree still has not changed. We activate $(2,3)$, $x_{23} = 1$, this time

$(2, 3)$ enters $\mathcal{B}$ and $(6, 5)$ leaves:

$$\begin{bmatrix} x_{16} \\ x_{35} \\ x_{64} \\ x_{26} \\ x_{23} \end{bmatrix} = \begin{bmatrix} 1 \\ 8 \\ 4 \\ 3 \\ 1 \end{bmatrix}.$$

At this point, the network flow is as follows:



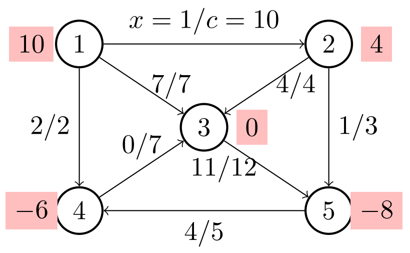
We use complementary slackness once more:

$$\begin{aligned} u_1 - u_6 &= \gamma_{16} = 1 \\ u_3 - u_5 &= \gamma_{35} = 0 \\ u_6 - u_4 &= \gamma_{64} = 1 \\ u_2 - u_6 &= \gamma_{26} = 1 \\ u_2 - u_3 &= \gamma_{23} = 0 \end{aligned}$$

and setting $u_6 = 0$ gives $u_1 = u_2 = u_5 = u_3 = 1$, $u_4 = -1$. We then check for feasibility and $u_5 - u_4 = 2 \not\leq 0$. We activate $(5, 4)$, $x_{54} = 3$, then $x_{23} = 4$ saturates, and:

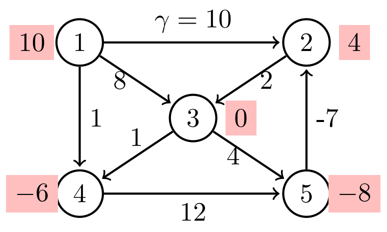
$$\begin{bmatrix} x_{16} \\ x_{35} \\ x_{64} \\ x_{23} \\ x_{54} \end{bmatrix} = \begin{bmatrix} 1 \\ 11 \\ 1 \\ 4 \\ 3 \end{bmatrix}.$$

Thus we are back to a degenerate basis. But now we are left with only 2 artificial edges each with a flow of 1, namely $(1, 6)$ and $(6, 4)$, so just need to activate $(1, 2)$, to get $x_{12} = 1$, $x_{25} = 1$, and :



5.3 Exercises

Exercise 45. Consider the following min cost flow network, where the demands are written at the nodes, and the costs are written next to each edge. Each edge capacity is infinite, and lower bound is 0.

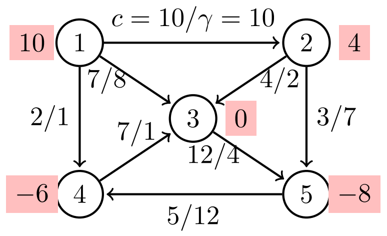


Compute a minimal cost flow for this network, using the BFS $(x_{13}, x_{23}, x_{34}, x_{35}) = (10, 4, 6, 8)$.

Exercise 46. Compute the dual of:

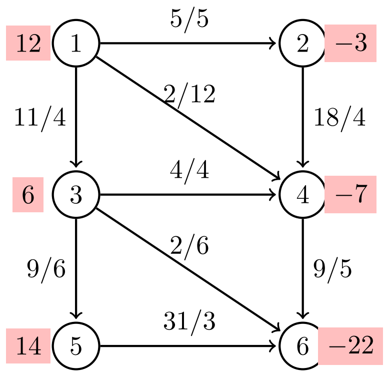
$$\begin{array}{ll} \min & \gamma^T x \\ s.t & \begin{bmatrix} A & 0_m \\ I_m & I_m \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} = \begin{bmatrix} d \\ c \end{bmatrix} \\ & x, s \geq 0. \end{array}$$

Exercise 47. Consider the following min cost flow problem, where every edge e is given a capacity $c(e)$ and a cost $\gamma(e)$.



1. Compute an optimal solution using the network simplex algorithm.
2. Compute an optimal solution using the negative cycle canceling algorithm.

Exercise 48. Consider the following min cost flow problem, where every edge e is given a capacity $c(e)$ and a cost $\gamma(e)$.



Compute an optimal solution using the network simplex algorithm. Compare the algorithm and the solution with Exercise 33.

Chapter 6

Semidefinite Programming

Semidefinite programming is a form of convex optimization that generalizes linear programming, and also provides a unified framework for several standard problems, including quadratic programming. The notes below follow closely [6].

Definition 6.1. An $n \times n$ real matrix X is *positive semidefinite* if

$$v^T X v \geq 0, \ \forall v \in \mathbb{R}^n.$$

We write $X \succeq 0$. We say that X is positive definite if

$$v^T X v > 0, \ \forall v \in \mathbb{R}^n.$$

We write $X \succ 0$.

We denote by S^n the set of $n \times n$ real symmetric matrices, that is

$$S^n = \{X \in M_n(\mathbb{R}), \ X^T = X\}.$$

Then

$$\begin{aligned} S^n_+ &= \{X \in S^n, \ X \succeq 0\}, \\ S^n_{++} &= \{X \in S^n, \ X \succ 0\}. \end{aligned}$$

Note that $X \succeq Y$ means $X - Y \succeq 0$.

Lemma 6.1. *The set*

$$S^n_+ = \{X \in S^n, \ X \succeq 0\}$$

is convex.

Proof. Pick $\lambda \in]0,1[$, and $X, W \in S^n_+$, we want to see that $\lambda X + (1-\lambda) W \in S^n_+$. So take any $v \in \mathbb{R}^n$, then

$$v^T(\lambda X + (1-\lambda)W)v = \lambda v^T X v + (1-\lambda) v^T W v \geq 0$$

as needed. □

We recall a few facts about symmetric matrices.

- If $X \in S^n$, we have a decomposition of X into $X = QDQ^T$ for Q orthonormal (that is $Q^{-1} = Q^T$) and D diagonal. The columns of Q form a set of n orthonormal eigenvectors of X, whose eigenvalues are the corresponding diagonal entries of D.
- If $X \in S^n$ and $X \succeq 0$, then $v^T QDQ^T v^T = (v^T Q)D(v^T Q)^T \geq 0$ for all v, so pick v^T to be in turn each of the row of $Q^{-1} = Q^T$, then $v^T Q$ will range through all the unit vectors, and $(v^T Q)D(v^T Q)^T$ will range through all the eigenvalues of X, so all of them are are non-negative.
- If $X \in S^n$, $X \succeq 0$ and if $x_{ii} = 0$, then $x_{ij} = x_{ji} = 0$ for all $j = 1, \dots, n$ (see Exercise 49).
- Consider the matrix
$$M = \begin{bmatrix} P & v \\ v^T & d \end{bmatrix}$$
for $P \succ 0$, $P \in S^n$, v a vector, d scalar. Then $M \succ 0 \iff d - v^T P^{-1} v > 0$. This is saying that a symmetric matrix is positive definite if and only if its Schur complement is.

Definition 6.2. A *semidefinite program (SDP)* is an optimization problem of the form

$$\begin{array}{rc} \min & C \bullet X \\ s.t. & A_i \bullet X = b_i,\ i = 1, \dots, m \\ & X \succeq 0 \end{array}$$

where we will assume that X, C and $A_i, i = 1, \dots, m$, are symmetric, and we use the notation:

$$C \bullet X = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} = \text{trace}(C^T X).$$

Recall that by definition, $\text{trace}(M) = \sum_{j=1}^{n} m_{jj}$.

Example 6.1. Take

$$A_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 3 & 7 \\ 1 & 7 & 5 \end{bmatrix},\ A_2 = \begin{bmatrix} 0 & 2 & 8 \\ 2 & 6 & 0 \\ 8 & 0 & 4 \end{bmatrix},\ C = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 9 & 0 \\ 3 & 0 & 7 \end{bmatrix},\ b_1 = 11,\ b_2 = 19.$$

Then we have

$$\begin{array}{rc} \min & C \bullet X = x_{11} + 4x_{12} + 6x_{13} + 9x_{22} + 7x_{33} \\ s.t. & x_{11} + 2x_{13} + 3x_{22} + 14x_{23} + 5x_{33} = 11 \\ & 4x_{12} + 16x_{13} + 6x_{22} + 4x_{33} = 19 \\ & X \succeq 0. \end{array}$$

The above example almost looks like an LP, but for the constraint $X \succeq 0$. We will show next that in fact, every LP can be written as an SDP (the above example serves as a counter-example that the converse is not true).

Suppose that we have the LP:

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

(constraints of the form $Ax \leq b$ can be rewritten with equality using slack variables).

Then define

$$A_i = \mathrm{diag}(a_{i1}, \ldots, a_{in}),\ i = 1, \ldots, m,\ C = \mathrm{diag}(c_1, \ldots, c_n),\ X = \mathrm{diag}(x_1, \ldots, x_n)$$

and the above LP can be written as the following SDP:

$$\begin{aligned} \min \quad & C \bullet X = c^T x \\ s.t. \quad & A_i \bullet X = b_i,\ i = 1, \ldots, m \\ & X \succeq 0 \end{aligned}$$

where the eigenvalues of X are $x_1, \ldots, x_n$ so $x_1, \ldots, x_n \geq 0$ can be expressed in terms of $X \succeq 0$.

6.1 Duality

Given an SDP:

$$\begin{aligned} (SDP) : \min \quad & C \bullet X \\ s.t. \quad & A_i \bullet X = b_i,\ i = 1, \ldots, m \\ & X \succeq 0, \end{aligned}$$

define its dual as

$$\begin{aligned} (SDD) : \max \quad & \textstyle\sum_{i=1}^m y_i b_i \\ s.t. \quad & \textstyle\sum_{i=1}^m y_i A_i + S = C \\ & S \succeq 0. \end{aligned}$$

In particular, we get from the dual that $C - \sum_{i=1}^m y_i A_i \succeq 0$. Note that S has to be symmetric since C and A_i are.

Example 6.2. For our previous example, we get

$$\begin{aligned} (SDD) \max \quad & 11y_1 + 19y_2 \\ s.t. \quad & y_1 \begin{bmatrix} 1 & 0 & 1 \\ 0 & 3 & 7 \\ 1 & 7 & 5 \end{bmatrix} + y_2 \begin{bmatrix} 0 & 2 & 8 \\ 2 & 6 & 0 \\ 8 & 0 & 4 \end{bmatrix} + S = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 9 & 0 \\ 3 & 0 & 7 \end{bmatrix} \\ & S \succeq 0. \end{aligned}$$

Alternatively:

$$\begin{array}{rc} (SDD)\max & 11y_1+19y_2 \\ s.t. & \begin{bmatrix} 1-y_1 & 2-2y_2 & 3-y_1-8y_2 \\ 2-2y_2 & 9-3y_1-6y_2 & -7y_1 \\ 3-y_1-8y_2 & -7y_1 & 7-5y_1-4y_2 \end{bmatrix} \succeq 0. \end{array}$$

The above SDP-SDD formulation is the equivalent of the primal-dual formulation for LP:

$$\begin{array}{rc} (P):\min & c^T x \\ s.t. & A_i x = b_i,\ i=1,\dots,m \\ & x \geq 0, \end{array}$$

where A_i are the rows of A, and

$$\begin{array}{rc} (D):\max & \sum_{i=1}^m y_i b_i \\ s.t. & \sum_{i=1}^m y_i A_i^T + s = c \\ & s \geq 0. \end{array}$$

We express the duality here using equalities instead of inequalities, but as we saw in the chapter on linear programming, one can always move from one formulation to another, and express the dual programs accordingly.

Let us start by recalling what we know about weak and strong duality for LP, then we shall see what can be generalized to SDP.

Given a feasible solution x of (P), and a feasible solution (y,s) of its dual (D), we have, using that $A_i x = b_i$

$$c^T x - \sum_{i=1}^m y_i b_i = \left(c^T - \sum_{i=1}^m y_i A_i\right) x = s^T x \geq 0$$

since $x, s \geq 0$. This is the Weak Duality Theorem, which says that if x is feasible for (P) and y is feasible for (D), then $c^T x \leq b^T y$.

The difference $c^T x - \sum_{i=1}^m y_i b_i$ is called the *duality gap*, it is the gap between the two objective functions. We know from the Strong Duality Theorem that as long as the primal LP is feasible and bounded, then the primal and the dual both attain their optima with no duality gap. That is, there exist x^* and (y^*, s^*) feasible for the primal and the dual respectively, such that

$$c^T x^* - \sum_{i=1}^m y_i^* b_i = (s^*)^T x^* = 0.$$

It turns out that weak duality holds for SDP:

Theorem 6.2. *Given a feasible solution X of SDP, a feasible solution (Y,S) of SDD, the duality gap is*

$$C \bullet X - \sum_{i=1}^m y_i b_i = S \bullet X \geq 0.$$

If $C \bullet X - \sum_{i=1}^m y_i b_i = 0$, then X and (Y, S) are each optimal solutions to the SDP and SDD respectively, and furthermore $SX = 0$.

Proof. To start with, we want to prove that $S \bullet X \geq 0$. Since S and X are symmetric and positive definite, write

$$S = PDP^T, \ X = QEQ^T$$

for P, Q orthornomal and D, E diagonal matrices whose diagonal entries are non-negative. Then since S is symmetric:

$$\begin{aligned} S \bullet X &= \text{trace}(S^T X) = \text{trace}(SX) \\ &= \text{trace}(P(DP^T QEQ^T)) = \text{trace}(DP^T QEQ^T P) \end{aligned}$$

since $\text{trace}(MN) = \text{trace}(NM)$. Now multiplying $P^T QEQ^T P$ by $D = \text{diag}(d_1, \ldots, d_n)$ means that row j of $P^T QEQ^T P$ is multiplied by d_j, and

$$\text{trace}(DP^T QEQ^T P) = \sum_{j=1}^n d_j (P^T QEQ^T P)_{jj} \geq 0$$

because $d_j \geq 0$ (S is positive semidefinite) and $(P^T QEQ^T P)_{jj}$ are the diagonal coefficients of the matrix $(P^T QEQ^T P)$ which is positive semidefinite (write $v^T (P^T Q) E (P^T Q)^T v = w^T E w$ with $w = v^T (P^T Q)$), thus they are non-negative (see Exercise 50). This completes the proof that $S \bullet X \geq 0$.

Next we want to prove that if $C \bullet X - \sum_{i=1}^m y_i b_i = 0$, then X and (Y, S) are optimal and $SX = 0$. Since $C \bullet X - \sum_{i=1}^m y_i b_i = S \bullet X$, we have that $S \bullet X = 0$. Optimality is clear from this, because $C \bullet X \geq \sum_{i=1}^m y_i b_i$, for all X and (Y, S), so from the view point of X, it is true for every X that $C \bullet X \geq \sum_{i=1}^m y_i^* b_i$ where y^* gives the largest value, and thus since we want to minimize $C \bullet X$, optimality for X is reached with equality. From the view point of Y, for every Y, $C \bullet X^* \geq \sum_{i=1}^m y_i b_i$ for X^* which minimizes $C \bullet X$, and so optimality for Y is reached with equality.

We are left to show that $SX = 0$. We just showed above that $S \bullet X = \sum_{j=1}^n d_j (P^T QEQ^T P)_{jj}$ where every term of the sum is non-negative, so

$$\sum_{j=1}^n d_j (P^T QEQ^T P)_{jj} = 0$$

and for each j, either $d_j = 0$ or $(P^T QEQ^T P)_{jj} = 0$.

- If $d_j = 0$, then the jth row and column of D are zero, so S and SX have their jth row and column zero as well.
- If $(P^T QEQ^T P)_{jj} = 0$, then $(P^T QEQ^T P)_{ij} = (P^T QEQ^T P)_{ji} = 0$ (see one of the facts recalled above about symmetric positive definite matrices), and so the jth row and column of $(P^T QEQ^T P)$ are zero. Multiplying this matrix by PD will give a new matrix whose jth row and columns are still zero, which is SX.

□

Strong duality on the other hand does not hold. Here is a classical example given by Lovász.

Example 6.3.

$$\begin{array}{ll} \min & y_1 \\ s.t. & \begin{bmatrix} 0 & y_1 & 0 \\ y_1 & y_2 & 0 \\ 0 & 0 & y_1+1 \end{bmatrix} \succeq 0. \end{array}$$

We remember from the facts about symmetric positive semidefinite matrices listed earlier that a zero on the diagonal means the corresponding row and column are zero. Thus $y_1 = 0$ in any feasible solution for this SDP. Once $y_1 = 0$, we must have $y_2 \geq 0$, and so the minimum for this SDP is 0.

One can compute (see Exercise 51) that the dual of this SDP is:

$$\begin{array}{ll} \max & -x_{33} \\ s.t. & x_{12} + x_{21} + x_{33} = 1 \\ & x_{22} = 0 \\ & X \succeq 0. \end{array}$$

Since $x_{22} = 0$ and $X \succeq 0$, again, a zero on the diagonal means the corresponding row and column are zero, so $x_{12} = x_{21} = 0$ and $x_{23} = x_{32} = 0$. But $x_{12} + x_{21} + x_{33} = 1$, so $x_{33} = 1$ and the optimum is -1. So strong duality does not hold.

For SDP, strong duality holds under the so-called Slater conditions.

Theorem 6.3. *Let z_P^* and z_D^* denote the optimal values of the objective functions for the SDP and its dual respectively. Suppose there exists a feasible solution X^* of the SDP such that $X^* \succ 0$, and there exists a feasible solution (Y^*, S^*) of its dual SDD such that $S^* \succ 0$. Then both the SDP and the SDD attain their optimal value, and $z_P^* = z_D^*$.*

See [8] for a proof.

Weak duality extends from LP to SDP. Strong duality, under some stronger conditions for SDP than for LDP, extends as well. However, there is no direct analog of a basic feasible solution for SDP. There is no finite algorithm for solving SDP, but while this sounds discouraging at first, SDP is an actually very powerful optimization framework, and there are extremely efficient algorithms to solve semidefinite programs.

SDP has wide applications in combinatorial optimization.

- A number of NP-hard combinatorial optimization problems have convex relaxations that are semidefinite programs (we will give an example of such a relaxation in the next section). SDP relaxations are often tight in practice, in certains cases, the optimal solution fo the SDP relaxation can be converted to a feasible solution for the original problem with provably good objective value.

- SDP can be used to model constraints that include linear inequalities, convex quadratic inequalities, lower bounds on matrix norms, lower bounds on determinants of symmetric positive semidefinite matrices.
- SDP can be used to solve linear programs, to optimize a convex quadratic form under convex quadratic inequality constraints among many other applications (see Exercise 52 for an eigenvalue optimization example).

6.2 An SDP Relaxation of the Max Cut Problem

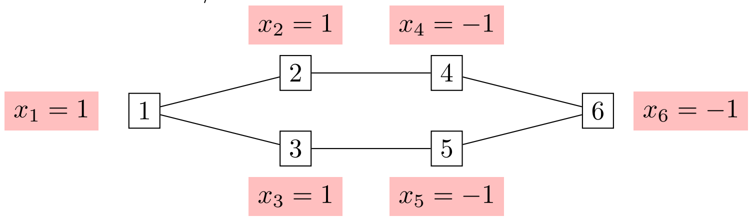
Let $G = (V, E)$ be an undirected graph. Let $w_{ij} = w_{ji} \geq 0$ be the weight on the edge $(i, j) \in E$.

Problem 6. The *max cut* problem consists of determining a subset S of nodes for which the sum of weights of the edges that cross from S to its complement $\bar{S} = V \backslash S$ is maximized.

Set $x_j = 1$, $j \in S$, and $x_j = -1$ for $j \in \bar{S}$. Then we can formulate our max cut problem as:

$$\begin{array}{rl} \textbf{maxcut}: \max & \frac{1}{4}\sum_{i,j=1}^{n} w_{ij}(1 - x_i x_j) \\ s.t. & x_j \in \{1, -1\},\ j = 1, \ldots, n. \end{array}$$

Every term in this sum is of the form $1 - x_i x_j$, which is equal to 0 if both x_i, x_j have the same sign, and equal to 2 if x_i, x_j have reverse signs. Having the same sign means that both nodes i, j are either in S or $\bar{S}$, so they should not contribute to the cut, and indeed $1 - x_i x_j = 0$ in this case. When x_i, x_j have reverse signs, then it means that one node i or j is in S and the other is in $\bar{S}$. However, since the sum is over i, j, we will encounter in the sum once the term $w_{ij}(1 - x_i x_j)$ and once the term $w_{ji}(1 - x_j x_i)$, accounting for $4w_{ij}$, which explains the factor $1/4$.



Then let

$$Y = xx^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} (x_1, \ldots, x_n)$$

so that $y_{ij} = x_i x_j$. Set $W = (w_{ij})$ to be the matrix containing all the weights. The coefficients x_i, x_j must be ± 1 by construction, we can capture this by asking $y_{jj} = 1$. Indeed $y_{jj} = x_j^2 = 1$ implies that $x_j = \pm 1$ for all j. Then we can reformulate the max cut problem as follows:

$$\begin{aligned} \textbf{maxcut} : \max \quad & \tfrac{1}{4}\textstyle\sum_{i,j=1}^{n} w_{ij} - W \bullet Y \\ s.t. \quad & y_{jj} = 1,\ j = 1, \dots, n \\ & Y = xx^T. \end{aligned}$$

We note that Y is a rank 1 positive semidefinite matrix. We "relax" this condition by removing the rank 1 restriction. This gives

$$\begin{aligned} \textbf{relaxmaxcut} : \max \quad & \tfrac{1}{4}\textstyle\sum_{i,j=1}^{n} w_{ij} - W \bullet Y \\ s.t. \quad & y_{jj} = 1,\ j = 1, \dots, n \\ & Y \succeq 0. \end{aligned}$$

Now we have that the optimal solution for the relax-max-cut problem is greater or equal to that for the max-cut problem, we write MAXCUT $\leq$ RELAX. However, it was proven by Goemans and Williamson (1995) that in fact 0.87856·RELAX $\leq$ MAXCUT $\leq$ RELAX, that is, the optimal value of the SDP relaxation is guaranteed to be no more than 12% higher than the optimal value for the NP-hard max cut problem.

6.3 An SDP Relaxation of the Independent Set Problem

Let $G = (V, E)$ be an undirected graph with $|V| = n$ vertices.

Problem 7. The *stable/independent set* problem consists of determining a subset S of nodes, no two of which are connected by an edge. The size $\alpha(G)$ of the largest stable set is called the *stability number of a graph.*

A natural integer programming formulation of $\alpha(G)$ is

$$\begin{aligned} \alpha(G) = \max \quad & \textstyle\sum_{i=1}^{n} x_i \\ s.t. \quad & x_i + x_j \leq 1,\ \{i, j\} \in E \\ & x_i \in \{0, 1\},\ i = 1, \dots, n. \end{aligned}$$

Label every node by 0 or 1, the first condition tells that if there is an edge between i and j, then both labels cannot be 1, it is either 0 and 1 or 0 and 0, capturing the property that nodes with label 1 form a stable set.

We can give an LP relaxation of this problem by letting x_i range from 0 to 1:

$$\begin{aligned} LP = \max \quad & \textstyle\sum_{i=1}^{n} x_i \\ s.t. \quad & x_i + x_j \leq 1,\ \{i, j\} \in E \\ & 0 \leq x_i \leq 1,\ i = 1, \dots, n. \end{aligned}$$

We thus have $\alpha(G) \leq LP$.

There is also an SDP relaxation due to Lovász:

$$\begin{aligned} \vartheta(G) : \max \quad & J \bullet X \\ s.t. \quad & I \bullet X = 1 \\ & X_{ij} = 0, \ \{i,j\} \in E \\ & X \succeq 0, \end{aligned}$$

where I is the identity matrix and J the whole one matrix.

Indeed, let S be a stable set, and define the vector $x = \frac{1}{\sqrt{|S|}} x'$ where $x'_i = 1$ if and only if $i \in S$ and it is 0 otherwise. let $X = xx^T$, so that $X_{ij} = x_i x_j$. Then

$$X_{ij} = 0, \ \{i,j\} \in E$$

since S is stable, and $X \succeq 0$ (we can check directly that $v^T xx^T v \geq 0$). Finally

$$\text{trace}(X) = \frac{1}{|S|}\text{trace}(xx'^T) = \frac{1}{|S|}\text{trace}(x'^T x) = 1.$$

This shows that S is feasible for the SDP relaxation. We are left to look at the objective function:

$$\text{trace}(JX) = \text{trace}(11^T X) = \frac{1}{|S|}\text{trace}(1^T x' x'^T 1) = \frac{1}{|S|}(x'^T 1)^2 = |S|.$$

Therefore the optimal value of the SDP can only be larger than the objective value at this one particular feasible solution, so it proves that

$$\alpha(G) \leq \vartheta(G).$$

In fact, it can also be proven that $\vartheta(G) \leq LP$.

6.4 Exercises

Exercise 49. If $X \succeq 0$ and if $x_{ii} = 0$, then $x_{ij} = x_{ji} = 0$ for all $j = 1, \ldots, n$.

Exercise 50. Show that if A is positive semidefinite, then all its diagonal coefficients are non-negative.

Exercise 51. Compute the dual of the following SDP:

$$\begin{aligned} \min \quad & y_1 \\ s.t. \quad & \begin{bmatrix} 0 & y_1 & 0 \\ y_1 & y_2 & 0 \\ 0 & 0 & y_1 + 1 \end{bmatrix} \succeq 0. \end{aligned}$$

Exercise 52. Given symmetric matrices B and A_i, $i = 1, \ldots, k$, consider the problem of minimizing the difference between the largest and the smallest eigenvalue of $S = B - \sum_{i=1}^k w_i A_i$. Show that this problem can be formulated as a SDP.

Chapter 7

Solutions to Exercises

Exercises marked by (*) were given in midterms or exams.

7.1 Pólya's Enumeration Theorem

Exercise 1. Compute the number of $(6,3)$-necklaces, that is the number of necklaces with 6 beads and 3 colours.

Solution 1. Let g denote a rotation by $2\pi/6$. The group acting here is the group of rotations given by g^i, $i=1,\dots,6$. Writing rotations in terms of permutations, we have

$$\begin{array}{rcl}
g = & (123456) & |\mathrm{Fix}(g)| = 3 \\
g^2 = & (135)(246) & |\mathrm{Fix}(g^2)| = 3^2 \\
g^3 = & (14)(25)(36) & |\mathrm{Fix}(g^3)| = 3^3 \\
g^4 = & (153)(264) & |\mathrm{Fix}(g^4)| = 3^2 \\
g^5 = & (165432) & |\mathrm{Fix}(g^5)| = 3 \\
g^6 = & (1)(2)(3)(4)(5)(6) & |\mathrm{Fix}(g^6)| = 3^6
\end{array}$$

Using Burnside Lemma, the number of $(6,3)$-necklaces is

$$\frac{1}{6}(3+3^2+3^3+3^2+3+3^6) = 130.$$

Exercise 2. Prove that

$$\frac{1}{n}\sum_{d|n}\phi(d)k^{n/d}$$

counts the number of (n,k)-necklaces, where ϕ is Euler totient function.

Solution 2. We know from Theorem 1.3 that the numer of (n,k)-necklaces is

$$\frac{1}{n}\sum_{i=1}^{n} k^{\gcd(n,i)}.$$

Now given a fixed value a taken by $\gcd(n,i)$, the question is, how many i are there from 1 to n such that $\gcd(n,i)=a$. To satisfy $\gcd(n,i)=a$, we need $n=an'$, $i=ai'$ for some n',i', such that $\gcd(n',i')=1$. Now we know how to count the number of i' coprime to n', this is by definition the Euler totient function $\phi(n')=\phi(n/a)$. Set $d=n/a$ to obtain the desired result.

Exercise 3. Compute the cycle index $P_{C_n}(X_1,\dots,X_n)$.

Solution 3. To compute the cycle index, we need to know the number $c(g^m)$ of cycles, which for C_n is $\gcd(m,n)$ by Theorem 1.3. Thus

$$\begin{aligned} P_{C_n}(X_1,\dots,X_n) &= \frac{1}{|G|}\sum_{i=0}^{n-1} X_1^{c_1(g^i)}\cdots X_n^{c_n(g^i)} \\ &= \frac{1}{|G|}\sum_{i=0}^{n-1} X_1^{\gcd(i,n)}\cdots X_n^{\gcd(i,n)} \\ &= \frac{1}{|G|}\sum_{d|n}\phi(d)X_d^{n/d} \end{aligned}$$

using Exercise 2.

Exercise 4. Let D,C be two finite sets, let G be a group acting on D, and let C^D be the set of functions $f: D\to C$. Show that

$$(g*f)(d)=f(g^{-1}*d)$$

is indeed a group action on C^D.

Solution 4. We check the compatibility property.

$$(g*\underbrace{(h*f)}_{f'})(d)=\underbrace{(h*f)}_{f'}(g^{-1}*d)=f(h^{-1}*(g^{-1}*d))=f((gh)^{-1}*d)=(gh*f)(d),$$

so $(g*(h*f))=(gh*f)$ as desired.

Remark. Note that $(g*f)(d)=f(g*d)$ is usually not an action of G on C^D. The compatibility property says that for $g,h\in G$

$$(g*(h*f))(d)=(h*f)(g*d)=f(h*(g*d))=f(hg*d)=(hg*f)(d).$$

This means that $(g*(h*f))=hg*f$ and we do not have a group action in general.

For $D = \{1, \dots, n\}$ and G a group that permutes the elements of D, write a vector $(c_1, \dots, c_n) = (f(1), \dots, f(n))$, which completely determines the function f. Then $g * f$ sends f to some function f', and thus to another such vector. The definition $(g * f)(i) = f(g^{-1}i)$ says that $g * (c_1, \dots, c_n) = (c_{g^{-1}(1)}, \dots, c_{g^{-1}(n)})$.

Exercise 5. Show that

$$\{1, r, r^2, r^3, m, rm, r^2m, r^3m\}$$

form a group with respect to composition of maps, where r is a rotation by 90 degrees (clockwise) and m is a reflection through the horizonal axis.

Solution 5. We need to check the definition of a group.

- We need to check closure, namely, composition of two elements in the set gives an element in the set. Composition of two rotations gives a rotation. So we are concerned with terms of the form $r^i m$, $i = 0, \dots, 3$. Note that $r^3 m = mr$. Then

$$r^i m r^j m = r^i (mr) r^{j-1} m = r^i r^3 m r^{j-1} m = r^{i+3} (mr) r^{j-2} m$$

  and iterate the process j times, so that r^{j-k} reaches r^0 once $k = j$.

- We have an identity element 1.

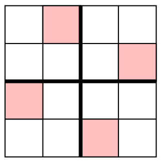
- We need to show that every element is invertible. For a rotation, $r^{-i} = r^{4-i}$ is the inverse which belongs to the set. Then m is its own inverse. Finally mr^{-i} is the inverse of $r^i m$, and $mr^{-i} = mr^{4-i} = mr^j$ for $j = 0, \dots, 3$. Now use that $mr^j = (mr)r^{j-1} = r^3 m r^{j-1}$ and iterate j times, so that $r^{j-k} = 1$, which happens when $k = j$ to show that the inverse belongs to the set.

Exercise 6. Consider an $n \times n$ chessboard, $n \geq 2$, where every square is either colored by blue (B) or red (R). How many different colorings are there, if different means that one cannot obtain a coloring from another by either a rotation (by either 90, 180 or 270 degrees) or a reflection (along the vertical and horizontal axes, and the 2 diagonals)?

Solution 6. Suppose that n is even, then the chessboard contains n^2 squares, and it makes sense to cut the chessboard into 4 equal parts of size $n^2/4$ ($n^2/4$ makes sense when n is even):

$n^2/4$	$n^2/4$
$n^2/4$	$n^2/4$

We compute Fix(g) for every g in the dihedral group. The identity fixes all of them. For the rotations r and r^3, note that when one of the 4 parts of size $n^2/4$ is given a colour, call this part P, then the 3 other parts must be of the same colour. So there are $2^{n^2/4}$ choices of colours in P.



If you want to think in terms of numbers of cycles, each of the $n^2/4$ squares in P corresponds to one cycle (of length 4).

For the rotation r^2, each element in P is sent back to itself after applying r^2 twice, so if we only fix squares in P, we still have degrees of freedom. We can then fix the upper half of the chessboard, this gives us $2^{n^2/2}$ choices, there is again one cycle (of length 2) for each square in the upper half.

Similarly, we get $2^{n^2/2}$ for both the horizontal and vertical mirror reflections.

For the two diagonal reflections, cut the square across each diagonal, to obtain two triangles, of size $n(n+1)/2$, then remove the diagonal from each trianlgle. This gives two triangles of size $n(n-1)/2$, and the diagonal of size n. So we obtain $2^{n(n-1)/2}$ colourings for the squares in each triangle (corresponding to $n(n-1)/2$ cycles of length 2), and n squares on the diagonal gives 2^n colourings (n cycles of length 1).

We have in summary:

g	$\lvert\text{Fix}(g)\rvert$
1	2^{n^2}
r	$2^{n^2/4}$
r^2	$2^{n^2/2}$
r^3	$2^{n^2/4}$
m	$2^{n^2/2}$
r^2m	$2^{n^2/2}$
rm	$2^{n(n+1)/2}$
r^3m	$2^{n(n+1)/2}$

Note that cycles have length at most 4. Thus the cycle index only records cycles of length 1 to 4, and the variables are X_1, X_2, X_3, X_4.

$$\begin{aligned} P_G(X_1,X_2,X_3,X_4) &= \frac{1}{8}\sum_{g\in G} X_1^{c_1(g)}X_2^{c_2(g)}X_3^{c_3(g)}X_4^{c_4(g)} \\ &= \frac{1}{8}(X_1^{n^2} + 2X_4^{n^2/4} + 3X_2^{n^2/2} + 2X_1^nX_2^{n(n-1)/2}) \end{aligned}$$

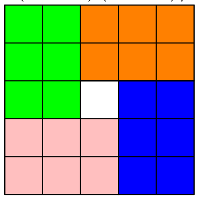
To count the number of chessboards with 2 colourings, we evaluate the cycle index in $X_1 = X_2 = X_3 = X_4 = 2$:

$$P_G(2,2,2,2) = \frac{1}{8}(2^{n^2} + 2\cdot 2^{n^2/4} + 3\cdot 2^{n^2/2} + 2\cdot 2^n 2^{n(n-1)/2})$$

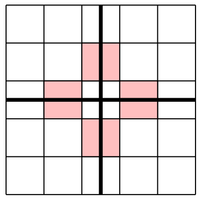
which for the case $n = 2$ simplifies to:

$$P_G(2,2,2,2) = \frac{1}{8}(16+4+12+16) = 6.$$

Consider now the case where n is odd, say $n = 2n' + 1$. Then the chessboard contains $n^2 = (2n' + 1)^2 = 4(n')^2 + 4n' + 1$ squares. Remove the square which is in the middle of the chessboard, this gives $n^2 - 1 = (n + 1)(n - 1)$ squares, and this number is divisible by 4. So divide the chessboard into 4 rectangles of size $(n + 1)(n - 1)/4$.



As usual, the identity map fixes everything. If the rotation is r or r^3, once one of the 4 rectangles is coloured, then there is no choice for the rest of the chessboard colouring (and we get a cycle of length 4 for each square in the first rectangle). If it is r^2, then we need to fix the colouring of two rectangles (and we get a cycle of length 2 for each square in these two rectangles). In both cases, the colouring of the middle square is free (this is a cycle of length 1).



For the the vertical and horizontal mirror reflections, once we remove the vertical (resp. horizontal) middle line of the chessboard, we get $n^2 - n = 4(n')^2 + 2n'$ squares, once half of them are coloured, the others are fixed. We thus have $(n^2 - n)/2$ squares whose colour can be chosen (corresponding to $(n^2 - n)/2$ cycles of length 2), plus the middle line (corresponding to n cycles of length 1), that is a total of $(n^2 - n)/2 + n = (n^2 + n)/2$ degrees of freedom. For the diagonal reflections, similarly, we remove the corresponding diagonal, after which it is enough to choose the colour of the triangle that is half the chessboard, that is $n(n - 1)/2$ squares, to have all the colours decided on the other triangle (so $2(n - 1)/2$ cycles of length 2), after which the diagonal can be coloured, which gives n cycles of length 1, one per square on the diagonal.

We have in summary:

g	$\lvert\mathrm{Fix}(g)\rvert$
1	2^{n^2}
r	$2^{(n^2-1)/4} \cdot 2$
r^2	$2^{(n^2-1)/2} \cdot 2$
r^3	$2^{(n^2-1)/4} \cdot 2$
m	$2^{n(n+1)/2}$
r^2m	$2^{n(n+1)/2}$
rm	$2^{n(n+1)/2}$
r^3m	$2^{n(n+1)/2}$

Note that cycles have length at most 4. Thus the cycle index only records cycles of length 1 to 4, and the variables are X_1, X_2, X_3, X_4.

$$\begin{aligned} P_G(X_1,X_2,X_3,X_4) &= \frac{1}{8}\sum_{g\in G} X_1^{c_1(g)}X_2^{c_2(g)}X_3^{c_3(g)}X_4^{c_4(g)} \\ &= \frac{1}{8}(X_1^{n^2}+2X_4^{(n^2-1)/4}X_1+X_2^{(n^2-1)/2}X_1+4X_1^nX_2^{n(n-1)/2}). \end{aligned}$$

To count the number of chessboards with 2 colourings, we evaluate the cycle index in $X_1 = X_2 = X_3 = X_4 = 2$:

$$P_G(2,2,2,2) = \frac{1}{8}(2^{n^2} + 4\cdot 2^{(n^2-1)/4} + 2\cdot 2^{(n^2-1)/2} + 4\cdot 2^n 2^{n(n-1)/2}).$$

which for the case $n = 3$ simplifies to:

$$P_G(2,2,2,2) = \frac{1}{8}(2^9 + 4\cdot 2^2 + 2\cdot 2^4 + 4\cdot 2^3 2^3) = 102.$$

Exercise 7. Consider $(4,3)$-necklaces, that is the number of necklaces with 4 beads and 3 colours, say blue (B), green (G) and red (R) . Use Polya Enumeration Theorem to list the different necklaces involving at least two blue beads.

Solution 7. The set of interest is formed by necklaces with 4 beads, and by definition of necklaces, rotations of necklaces are not considered as giving new necklaces. Therefore the group acting here is the group of rotations by $(2\pi/4)k$, $k = 0,1,2,3$. For each rotation, we write the corresponding rotation in its cycle decomposition, and its contribution to the cycle index:

(1234)	X_4
$(13)(24)$	X_2^2
(1432)	X_4
$(1)(2)(3)(4)$	X_1^4

thus we get

$$P(X_1,X_2,X_4) = \frac{1}{4}(2X_4 + X_2^2 + X_1^4).$$

To list the different necklaces with 2 blue beads, we need to compute

$$\frac{1}{4}(2(G^4+B^4+R^4) + (G^2+B^2+R^2)^2 + (G+B+R)^4).$$

Computing this polynomial by hand is tedious (it is quite easy with a computer though, say using maxima or mathematica), but the question asks only for at least two blue beads. This means that in fact, we can discard terms which do not contributed to at least two blue beads. In $2(G^4+B^4+R^4)$, we only need $2B^4$. Then

$$(G^2+B^2+R^2)^2 = G^2B^2 + B^2(G^2+B^2+R^2) + \ldots$$

where ... includes the terms with no contribution. Finally,

$$(G+B+R)^4 = (G^2+B^2+R^2+2GR+2BR+2BG)^2$$

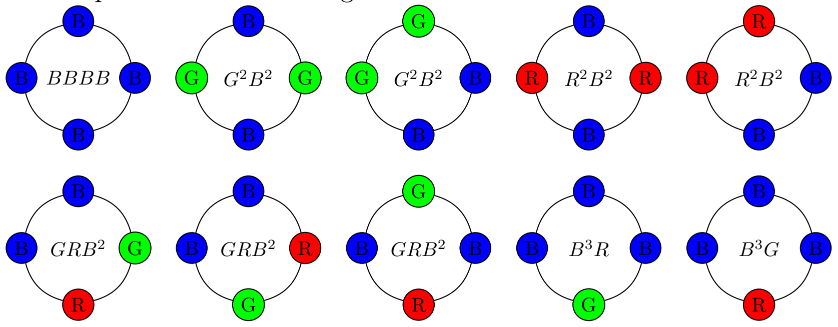
gives the following contributions:

$$G^2(B^2),\ R^2(B^2), B^2(G^2+B^2+R^2+2GR+2BR+2BG),\ 2GR(B^2),$$
$$2BR(B^2+2BR+2BG),\ 2BG(B^2+2BR+2BG).$$

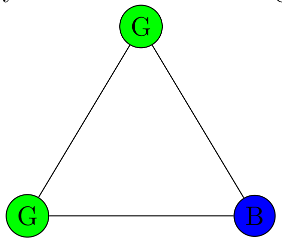
We now collect terms by powers of B which we divide by:

$$B^4,\ 2G^2B^2,\ 2B^2R^2,\ 3GRB^2,\ B^3R,\ B^3G.$$

This corresponds to the following necklaces:



Exercise 8. (*) Consider an equilateral triangle whose vertices are coloured, they can be either blue or green. Here is an example of colouring:



Two colourings of the vertices are considered equivalent if one can be obtained from another via a rotation or reflection of the triangle.

1. List all the rotation(s) and reflection(s) of the equilateral triangle and argue they form a group.
2. Compute the cycle index polynomial for the group of rotations and reflections of the equilateral triangle.
3. Use Pólya's Enumeration Theorem to list the different colourings using two colours of the equilateral triangle.

Solution 8. 1. We have the identity, the reflection with respect to the vertical line, and with respect to the two diagonals, then we have 2 rotations, one by $2\pi/3$ and one by $4\pi/3$. There are (at least) 3 valid ways to prove this is group: (a) argue that we have all the permutations on 3 elements, therefore this is the symmetric group S_3, (b) recognize the dihedral group D_3, and (c) check the axioms of group explicitly (closure, identity, inverse).

2. The corresponding permutations in terms of cycles are:

$$(123),(132),(1)(23),(13)(2),(12)(3),(1)(2)(3).$$

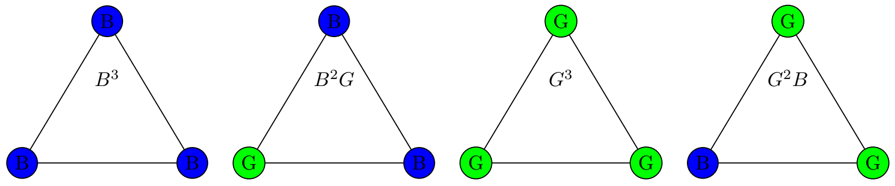
The cycle index is

$$P(X_1,X_2,X_3)=\frac{1}{6}(X_1^3+3X_1X_2+2X_3).$$

3. We evaluate

$$P(B+G,B^2+G^2,B^3+G^3)=B^3+B^2G+BG^2+G^3.$$

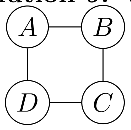
This gives:



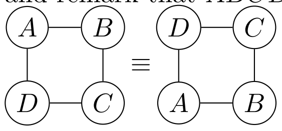
Exercise 9. (*) We know that a (n,k)-necklace is an equivalence class of words of length n over an alphabet size k, under rotation. Consider now an (n,k)-bracelet, that is an equivalence class of words of length n over an alphabet of size k, under both rotation (as necklaces), but also under reversal, which means for example that the bracelet $ABCD$ is equivalent to the bracelet $DCBA$: $ABCD \equiv DCBA$.

1. For (4,2)-necklaces, they are orbits under the action of the group of rotations by $(2\pi/4)k, k=0,1,2,3$. For (4,2)-bracelets, they are also orbits under the action of some group G. What is this group G? List its elements.
2. Compute the cycle index polynomial for the group G.
3. Use Polya's Enumeration Theorem to list the different (4,2)-bracelets.

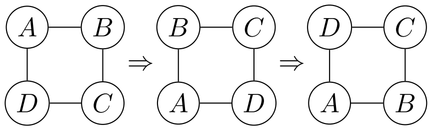
Solution 9. Write a necklace $ABCD$ as



and remark that $ABCD \equiv DCBA$ means



and $DCBA$ can be obtained from $ABCD$ by



that is a composition of a rotation with a mirror reflection. Thus the group involved is the dihedral group of rotations and reflections of the square, for which we have:

g		$\lvert\text{Fix}(g)\rvert$
1	$(1)(2)(3)(4)$	2^4
r	(1234)	2
r^2	$(13)(24)$	2^2
r^3	(1432)	2
m	$(13)(24)$	2^2
r^2m	$(12)(34)$	2^2
rm	$(24)(1)(3)$	2^3
r^3m	$(13)(2)(4)$	2^3

Thus the cycle index is

$$P_G(X_1, X_2, X_4) = \frac{1}{8}(X_1^4 + 3X_2^2 + 2X_4^1 + 2X_2^1X_1^2)$$

and to enumerate colourings with 2 colours using Polya's Enumeration Theorem, we compute:

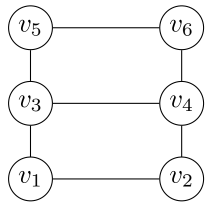
$$P_G((B+G),(B^2+G^2),B^4+G^4) = B^4 + G^4 + 2B^2G^2 + B^3G + BG^3.$$

7.2 Basic Graph Theory

Exercise 10. Prove that the set of all automorphisms of a graph forms a group.

Solution 10. By definition, an automorphism of $G = (V, E)$ is a bijection $\alpha : V \to V$ such that $\{u, v\} \in E \iff \{\alpha(u), \alpha(v)\} \in E$ for all $\{u, v\} \in E$. To show that the set of automorphisms forms a group, we first need to show that the composition of two automorphisms still is an automorphism. Let α_1, α_2 be two automorphisms. The composition of a bijection is a bijection. Now we need to check that $\{u, v\} \in E \iff \{\alpha_2\alpha_1(u), \alpha_2\alpha_1(v)\} \in E$ for all $\{u, v\} \in E$. Because α_1 is an automorphism, we know that $\{\alpha_1(u), \alpha_1(v)\} \in E$. But then, because α_2 is an automorphismi it must be that $\{\alpha_2\alpha_1(u), \alpha_2\alpha_1(v)\} \in E$. The identity element is the identity map which sends every vertex to itself. The inverse of every automorphism α is α^{-1} and α^{-1} satisfies that $\{\alpha^{-1}(u), \alpha^{-1}(v)\} \in E$ whenever $\{u, v\} \in E$: apply α on $\{\alpha^{-1}(u), \alpha^{-1}(v)\}$ to get $\{u, v\} \in E$.

Exercise 11. Compute the automorphism group of the following graph.



Solution 11. There are four automorphisms, given by the following permutations of the vertices: the identity map, (v_2, v_1), (v_4, v_3), (v_6, v_5), then (v_5, v_1), (v_2, v_6), (v_3), (v_4), and the composition of both: (v_5, v_2), (v_1, v_6), (v_4, v_3). To show that the list is complete, notice that the orbit of say v_5 is at most 4, because a node of degree 2 cannot be sent to a node of degree 3. Now the orbit of v_5 is already of size 4, and we look at the stabilizer of v_5. But if v_5 is fixed, then we have exactly 2 possibilities: swap v_3 and v_6, or keep them fixed too. But they cannot be swapped, they have different degrees. Then they are fixed too, but then so is v_4 which is connected to both, and so the size of the stabilizer is 1, containing only the identity map. This shows that the automorphism group has size 4.

Exercise 12. Compute the automorphism group of the n-cycle graph, the graph given by n vertices $1, \dots, n$, and n edges given by $\{i, i+1\}$ where $i, i+1$ are understood modulo n.

Solution 12. To give the explanation, we visualize the cycle graph in the plane. For that, consider a circle centered around the origin, and place the n vertices on the circle, so that they are equidistant, with one vertex on the y-axis, say v_1. Then label the other vertices from v_2 to v_n, clockwise.

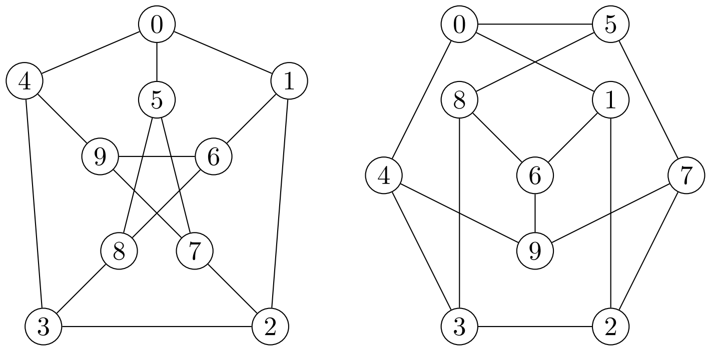
We claim that permutations of the vertices given by a rotation $(2\pi/n)m$, $m = 1, \dots, n$, are part of the group of automorphisms. Indeed, such a rotation (clockwise), sends v_1 to v_{1+m}, and in fact, any v_i to v_{i+m}, with subscripts understood modulo n. Now because there is exactly one edge between each v_i and v_{i+1}, after permutation, we have an edge between v_{i+m} and v_{i+1+m}, a valid edge. This gives us n elements in the automorphism group.

Now one more element is given by the vertical reflection. Indeed, v_1 is on the axis, v_2 is $2\pi/n$ away clockwise, while v_n is $2\pi/n$ away counter-clockwise, etc, so vertices are positioned symmetrically with respect to the y-axis.

Because automorphisms form a group, the composition of the above gives us $2n$ automorphisms.

Next we prove that there is no other automorphism. For that, use that $|\mathrm{Aut}(G)| = |\mathrm{Stab}(v_1)||\mathrm{Orb}(v_1)|$, and we know that there are n elements in the orbit of v_1 (since we cannot have more, and we already have that many). Now we evaluate $|\mathrm{Stab}(v_1)|$, by repeating the same formula, $|\mathrm{Stab}(v_1)| = |\mathrm{Stab}(v_2)||\mathrm{Orb}(v_2)|$ where we look only at automorphisms fixing v_1. The orbit size is 2, since v_1 has only two neighbours. But once v_1 and v_2 are fixed, all the other vertices are, so the stabilizer (of any other vertex) has size 1 (the identity map) and we have established the automorphism group (which is called a dihedral group).

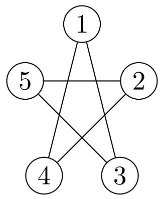
Exercise 13. Show that the following two graphs are isomorphic:



Solution 13. The given labeling already defines the isomorphism. Formally this isomorphism α maps the vertices $v_0, \ldots, v_9$ of the first graph to those $w_0, \ldots w_9$ of the second graph. For example, in the first graph, say 0 is labeling v_0, and 1 is labeling v_1, while in the second graph, 0 is labeling w_0, 5 is labeling w_1, 8 is labeling w_2 and 1 is labeling w_3. Then α maps v_0 to w_0 and v_1 to w_3. In the first graph, there is an edge between v_0 and v_1, and when α is applied, we get $\{\alpha(v_0), \alpha(v_1)\} = \{w_1, w_4\}$ which is an edge in the second graph. To prove the isomorphism, every edge should be checked to be preserved by α.

Exercise 14. (*)

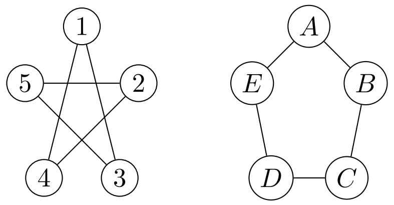
1. Compute the automorphism group of the following graph.



2. Give an example of a connected graph with at least 2 vertices whose automorphism group is of size 1.

3. Suppose two connected graphs G and G' have the same automorphism group, that is $\text{Aut}(G) \cong \text{Aut}(G')$. Does it imply that G is isomorphic to G'? Justify your answer.

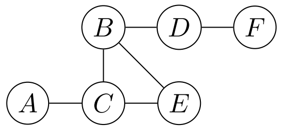
Solution 14. 1. There are (at least) two ways to compute the automorphism group of the above graph. The first one is to show that the graph is isomorphic to the cycle graph



The isomorphism α is given by $1 \mapsto A, 3 \mapsto B, 5 \mapsto C, 2 \mapsto D, 4 \mapsto E$. The edges are $13, 14, 24, 25, 35$, and $\alpha(1)\alpha(3), \alpha(1)\alpha(4), \alpha(2)\alpha(4), \alpha(2)\alpha(5), \alpha(3)\alpha(5)$ give the edges AB, AE, DE, DC, BC which are the edges of the cycle graph. Then we know the automorphism group of this graph by Exercise 12. It is the dihedral group with 10 elements, 5 rotations, 1 vertical mirror reflection and the identity, and combinations of them.

Alternatively we can compute $\mathrm{Aut}(G)$ directly. We argue that rotations by $2\pi/5j$, $j = 0, 1, 2, 3, 4$ are in $\mathrm{Aut}(G)$, and so is the vertical mirror reflection. Then since we have a group, we know that we have all the compositions, so 10 elements. Next we need to show that we cannot have more automorphisms. For that, take the vertex 1, then $|\mathrm{Aut}(G)| = |\mathrm{Orb}(1)||\mathrm{Stab}(1)|$ and we look at $|\mathrm{Stab}(1)|$. But once 1 is fixed, an automorphism can either swap $4, 3$, or fix them, and once this is decided, all the rest is fixed, so $|\mathrm{Aut}(G)| = 5 \cdot 2 = 10$ and the group is complete.

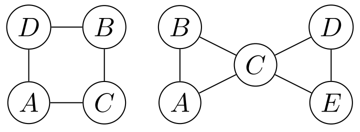
2. Consider the graph



Note that $\deg(C) = 3 = \deg(B)$, so an automorphism of this graph is forced to send a degree 3 node to a degree 3 node, thus it would have to either do nothing on B, C, or swap them (the orbit of B contains either 1 or 2 elements). Suppose the automorphism sends B to itself, and C to itself. Then A which is adjacent to C is fixed and so is E (we cannot switch a degree 1 node with a degree 2 node). Then for the same reason, D and F are fixed.

Suppose then that the automorphism swaps B and C. Then the edges BD, DF, BE, BC, CE, AC are sent to CD, DF, CE, BC, BE, AB This is not a valid automorphism, because CD for example is not a valid edge, so the automorphism will have to also permute other vertices. Since CD is not a valid edge, we need to send D to A, B, E. But sending D to A is not possible, D has degree 2 and A has degree 1, sending D to B is not possible because B is used already in this permutation, and if we swap D and E, then we get that the edges are CE, EF, CD, BC, BD, AB, and we still have CD which is not a valid edge, thus the only valid automorphism is the identity.

3. No, here is a counter-example. The two following graphs have the same automorphism group, and obviously they cannot be isomorphic (they have different numbers of vertices):



The cycle graph has an automorphism group made of 4 rotations and 4 reflections. The other graph, the butterfly graph has the same automorphism group. This is because the 4 rotations and the 4 reflections are all automorphisms of this graph, and there cannot be anymore automorphism by the usual argument: if we compute the orbit of B, it contains 4 elements (it cannot be sent to C), then we compute its stabilizer. Its stabilizer contains 2 elements (D, E can be swapped, or fixed).

Exercise 15. (*) For all $n \geq 2$, give a graph whose automorphism group is the symmetric group S_n, that is, the group of permutations of n elements.

Solution 15. The complete graph K_n has for automorphism group the symmetric group S_n. Indeed, it is true for $n = 1$, $n = 2$. Suppose it is true for K_{n-1} and consider K_n. Fix a vertex v of K_n. An automorphism τ of K_n either fixes v, or sends v to another vertex. If v is fixed, the automorphism τ can permute the neighbours of v, which comprise all other $n-1$ vertices, and these vertices form the graph K_{n-1}, so we know that S_{n-1} is its automorphism group, and thus τ can be any element of S_{n-1}. The same argument applies if we have an automorphism τ' that sends v to another vertex say u. Since v is connected to every edge in the graph, so is u for any choice of u (including $u = v$ which is the case where v is fixed), now the automorphism will send u to a vertex w which cannot be u, so removing u, we get the complete graph K_{n-1} (which includes v and w), and the automorphism τ' restricted to K_{n-1} can be any element of S_{n-1}. We have thus shown that an automorphism can map one vertex to any other of the n vertices, and then any of the remaining $n-1$ vertices among each other, so we get the whole of S_n.

Exercise 16. Count, using Pólya's Enumeration Theorem, the number of isomorphism classes of graphs with 4 vertices, and count how many there are for each possible number of edges.

Solution 16. We list all 4! permutations:

1234 (1)(2)(3)(4)	2134 (12)	3124 (132)	4123 (1432)
1324 (23)	2314 (123)	3214 (13)	4231 (14)
1423 (243)	2413 (1234)	3421 (1324)	4321 (14)(23)
1243 (34)	2143 (12)(34)	3142 (1342)	4132 (142)
1342 (234)	2341 (1234)	3241 (134)	4213 (143)
1432 (24)	2431 (124)	3412 (13)(24)	4312 (1423)

The cycle index, if we wanted to compute all the permutations of the vertices, would be given by

$$P_{S_4}(X_1, X_2, X_3, X_4) = \frac{1}{4!}(X_1^4 + 6X_1^2X_2 + 8X_1X_3 + 6X_4 + 3X_2^2).$$

However we are interested in the permutations induced on the edges. Let us call our 4 nodes a, b, c, d.

The term $6X_1^2X_2$ means that two vertices are fixed, say a, b, while c and d are switched. In terms of edges, this means that

$$(ab) \mapsto (ab), (ac) \mapsto (ad), (ad) \mapsto (ac), (bc) \mapsto (bd), (bd) \mapsto (bc), (cd) \mapsto (cd).$$

Thus the term in the cycle index corresponding to this edge permutation is $6X_1^2X_2^2$.

The term $3X_2^2$ means that two pairs of vertices are switched, say a, b are switched, and c, d are switched. In terms of edges, this means that

$$(ab) \mapsto (ba), (ac) \mapsto (bd), (ad) \mapsto (bc), (bc) \mapsto (ad), (bd) \mapsto (ac), (cd) \mapsto (dc).$$

Thus the term in the cycle index corresponding to this edge permutation is $3X_1^2X_2^2$.

The term $8X_1X_3$ means that one vertex is fixed, say a, while b, c, d are shifted: (bcd). In terms of edges, this means that

$$(ab) \mapsto (ac), (ac) \mapsto (ad), (ad) \mapsto (ab), (bc) \mapsto (cd), (bd) \mapsto (cb), (cd) \mapsto (db).$$

Thus the term in the cycle index corresponding to the edge permutation is $8X_3^2$.

The term $6X_4$ means that a, b, c, d are shifted: $(abcd)$. In terms of edges, this means that

$$(ab) \mapsto (bc), (ac) \mapsto (bd), (ad) \mapsto (ba), (bc) \mapsto (cd), (bd) \mapsto (ca), (cd) \mapsto (da).$$

Thus the term in the cycle index corresponding to the edge permutation is $6X_4X_2$.

The term X_1^4 is the identity, so it yields X_1^6.

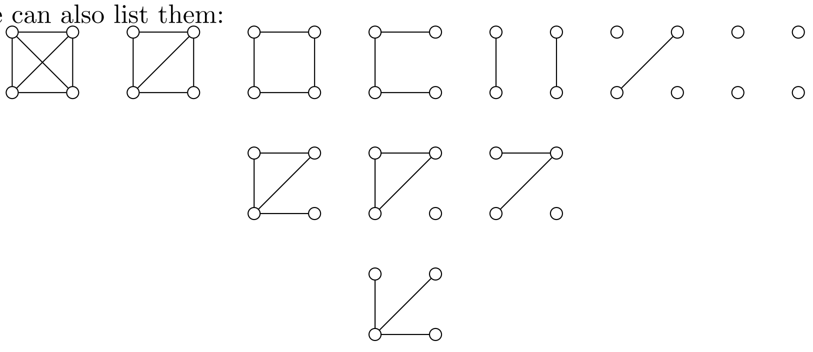
The cycle index is then:

$$P(X_1, X_2, X_3, X_4) = \frac{1}{4!}(X_1^6 + 9X_1^2X_2^2 + 8X_3^2 + 6X_4X_2).$$

We next evaluate the cycle index, to give (after a bunch of computations that I did not reproduce here):

$$P(1+E, 1+E^2, 1+E^3, 1+E^4) = E^6 + E^5 + 2E^4 + 3E^3 + 2E^2 + E + 1.$$

The cycle index tells us the number of isomorphism classes per number of edges, so 1 isomorphism class for 6 edges, 5 edges, 1 edge, and no edge, then 2 isomorphism classes for 4 edges and 2 edges, and 3 isomorphism classes for 3 edges. We can also list them:



Exercise 17. Prove (by induction on the length l of the walk) that every walk from u to v in a graph G contains a path between u and v.

Solution 17. We provide a proof by induction on the length l of the walk $u_0u_1 \dots u_l$ ($u = u_0$, $v = u_l$). If $l = 1$, there is only one edge, and thus the walk is a path. Suppose true for less than l, that is, for walks of length $\leq l-1$, there exists a path inside the walk.

Consider a walk of length l. If this walk is not a path, then there is $u_i = u_j$ with $i < j$ (if a walk is not a path, either an edge or a vertex is repeated or both, but either way, at least one vertex is repeated). But if $u_i = u_j$, this means that the walk goes from u_0 to u_i, then at u_i it loops until it reach $u_j = u_i$ (the loop is $u_i, u_{i+1}, \dots, u_{j-1}, u_j$) and then it continues from u_j to u_l. But then, one way to go from u_0 to u_l is just to skip this loop, and do directly $u_0u_1 \dots u_i = u_j \dots u_l$. Now this walk has length less than l, so by induction hypothesis, it contains a path from u to v as desired.

Exercise 18. Prove that adding one edge to a tree creates exactly one cycle.

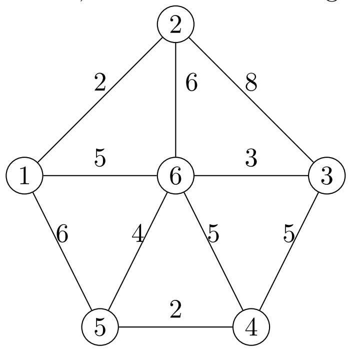
Solution 18. First we need to show that at least one cycle is created. Take any 2 distinct vertices u, v, and add one edge $\{u, v\}$ between the two. Now take a distinct 3rd vertex w. By definition of tree, there exists a path from u to w, and a path from w to v. Now concatenation of both paths gives a walk from u to v. It may not be a path because some parts of the two paths could intersect, but we know from Exercise 17 that we can find a path inside. Now this path together with $\{u, v\}$ creates a cycle.

Next we need to show that there cannot be more than one cycle. Suppose that adding $\{u, v\}$ creates two or more cycles. Let these 2 cycles be $(u, v, \dots, u_1, u_2, \dots, u)$ and $(u, v, \dots, v_1, v_2, \dots, u)$. But this means that we can go from v to u through $\dots, u_1, u_2, \dots$ or through $\dots, v_1, v_2, \dots$, a contradiction to the definition of tree, which says there should be a unique path from one vertex to another.

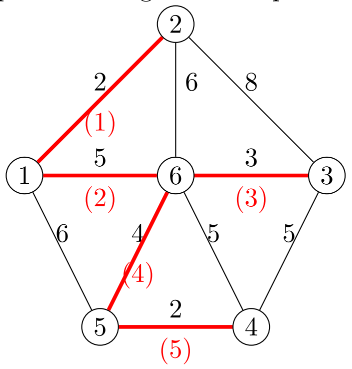
Exercise 19. Prove or disprove the following claim: if G is a graph with exactly one spanning tree, then G is a tree.

Solution 19. We have to prove that the claim is true, namely, if G is a graph with exactly one spanning tree T, then G must be a tree. Suppose that it is not true, that G is not a tree, then G must contain at least one edge $\{u, v\}$ which is not in its spanning tree T. Then since u, v both belong to the spanning tree T, there is a path in T from u to v, thus once we add the edge $\{u, v\}$ to T, we create a cycle (see Exercise 18). Now using this cycle, we will create another spanning tree T': take for T' the graph $T \cup \{u, v\} \backslash e$ where e is any edge in the path from u to v. Let us check that T' is indeed a tree. It surely is connected, because we know that removing an edge from a cycle does not disconnect a graph. It contains no cycle, because adding $\{u, v\}$ created exactly one cycle, and this cycle was removed when e was removed. This gives a contradiction, so G must be a tree.

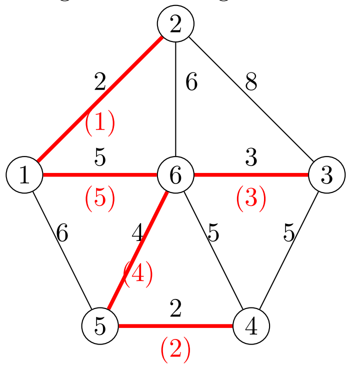
Exercise 20. Find a minimum spanning tree for this graph, using once Prim algorithm, and once Kruskal algorithm:



Solution 20. Using Prim's algorithm, we get the following tree, the numbers in parentheses give the steps of the algorithm.



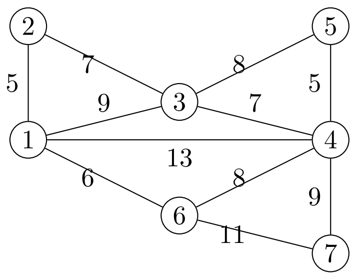
Using Kruskal's algorithm we get:



Note that steps 1 and 2 could be done in the reverse order.

Exercise 21. (*)

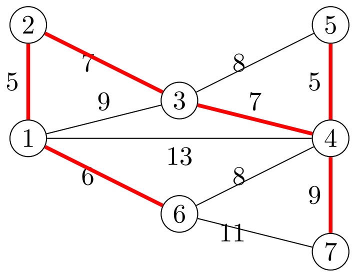
1. Compute a minimum spanning tree in the following graph with the method of your choice. Describe the steps of the algorithm.



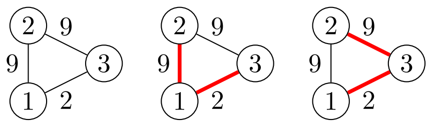
2. Construct, if possible, a connected weighted graph G with two minimum spanning trees.

3. Let G be a connected weighted graph, with m edges with respective weights $e_1 \leq e_2 \ldots \leq e_{i-1} < e_i = e_{i+1} < e_{i+2} \leq \ldots \leq e_m$, and let T_k be a minimum spanning tree found by Kruskal algorithm. Suppose that there exists a different minimum spanning tree T such that T and T_k share the same edges $e_1, \ldots e_{i-1}$, for some $i \geq 2$, but T_k contains e_i and not e_{i+1}, while T contains e_{i+1} but not e_i. Show that T can be found by an instance of Kruskal algorithm.

Solution 21. 1. Whether you use Prim's or Kruskal's algorithm, the following MST is found:



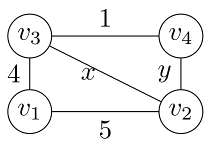
2. The following graph has two MSTs:



3. Kruskal will pick $e_1, \ldots, e_{i-1}$, then it can choose e_i or e_{i+1}. To choose e_{i+1}, it must be that adding e_{i+1} to $e_1, \ldots, e_{i-1}$ creates no cycle, which is the case since $e_1, \ldots, e_{i-1}, e_{i+1}$ are inside a MST. Then we argue that e_i cannot be present in T. If e_i were included in T, then in the first tree T_k, Kruskal would have taken e_i then e_{i+1}, however after e_i, it moved to e_{i+2}. This means that e_i and e_{i+1} create a cycle. So with e_{i+1} in T, e_i cannot be present.

We remark that this step is a part of proving that Kruskal can actually find all MST in a given graph.

Exercise 22. 1. Consider the following weighted undirected graph, where x, y are unknown integers.



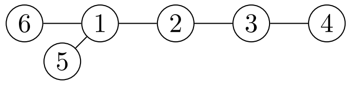
Give, if possible, values for x, y such that the graph contains (a) a single minimum spanning tree, (b) at least two minimum spanning trees.

2. Given a weighted undirected graph $G = (V, E)$, such that every edge e in E has a distinct weight. Prove or disprove the following: G contains a unique minimum spanning tree.

Solution 22. 1. For (a), choose for example $x = 2$ and $y = 3$, then there is a single minimum spanning tree with weight 7. It is found by Kruskal's algorithm. For (b), choose for example $x = y = 1$. Then there are at least two spanning trees of weight 6, given by (v_3, v_4), (v_3, v_2), (v_3, v_1), and by (v_3, v_4), (v_4, v_2), (v_3, v_1).

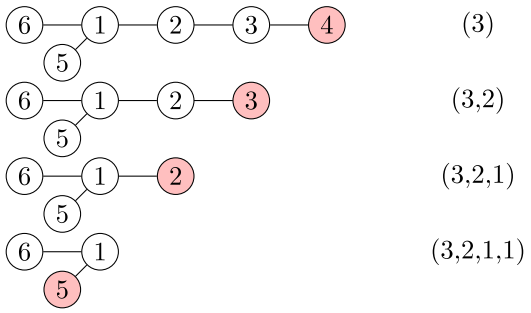
2. The minimum spanning tree is unique. To prove this is the case, assume by contradiction there are two minimum spanning trees, say T_1 and T_2. They must differ on at least one edge, among these edges, we choose the one of smallest weight, let us call it e (it is unique since weights are distinct) and suppose it belongs to T_1 but not to T_2. Then we can add it to T_2 which creates a cycle, cycle which is not included in the tree T_1, so there is an edge e' of this cycle not belonging to T_1. The weight of e' must be more than that of e by minimality of e. Now replacing e' with e in T_2 reduces the weight, therefore contradicting the hypothesis that T_2 is a minimum spanning tree.

Exercise 23. Compute the Prüfer code of the following tree:



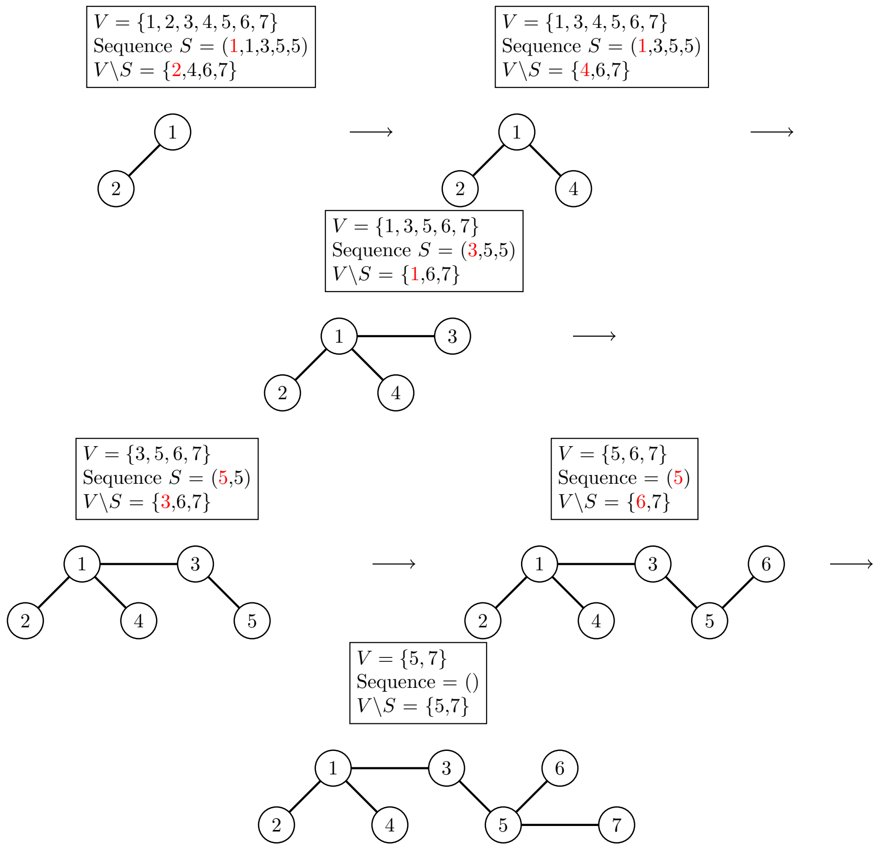
Construct the tree corresponding to the Prüfer code (1,1,3,5,5).

Solution 23. To compute a Prüfer code for a tree, find the leaf whose label is the smallest, put in the list the unique neighbour of this list, then remove the leaf and the edge that connects it to its unique neighbour, and repeat the procedure. This is illustrated below for the current graph. For example, in the first step, the leaf with the smallest label is 4, its neighbour is 3, put 3 in the list, remove 4 together with its edge.



To construct the tree corresponding to the Prüfer code $(1, 1, 3, 5, 5)$, start by noting that this sequence has 5 elements, therefore, the tree has 7 vertices. Then identify the first leaf, it must have a label in $\{1, \dots, 7\}$, but not in 1,1,3,5,5, and among these values 2,4,6,7, it has the smallest label, namely 2, and we know from $(1, 1, 3, 5, 5)$ that the unique neighbour of 2 is 1.

Now we iterate the procedure on $(1, 3, 5, 5)$. To identify the leaf attached to 1, we look for a label in $\{1, \dots, 7\}$, but not in 1,3,5,5, and this label cannot be 2 since it has already been allocated. The choices are 4,6,7, and the smallest is 4. Thus 4 is a leaf, with unique neighbour 1. Iterate the process as shown below.



$V = \{1, 2, 3, 4, 5, 6, 7\}$
Sequence S = (1,1,3,5,5)
$V\backslash S$ = {2,4,6,7}

$V = \{1, 3, 4, 5, 6, 7\}$
Sequence S = (1,3,5,5)
$V\backslash S$ = {4,6,7}

$V = \{1, 3, 5, 6, 7\}$
Sequence S = (3,5,5)
$V\backslash S$ = {1,6,7}

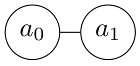
$V = \{3, 5, 6, 7\}$
Sequence S = (5,5)
$V\backslash S$ = {3,6,7}

$V = \{5, 6, 7\}$
Sequence = (5)
$V\backslash S$ = {6,7}

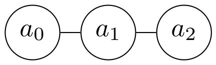
$V = \{5, 7\}$
Sequence = ()
$V\backslash S$ = {5,7}

Exercise 24. (*) Determine which trees have Prüfer codes that have distinct values in all positions.

Solution 24. Consider a tree with n vertices. If $n = 2$, the only Prüfer code is () and we have the tree



If $n = 3$, we have the Prüfer code (a_1) for the tree



We first prove by induction on n, the number of vertices in the tree, that the degree $\deg_T(v)$ of a vertex v in a labeled tree T is one more than the number of times the label appears in the Prüfer code.

This claim is true for $n = 2$ (both labels do not appear at all, and they have degree 1) and $n = 3$ (a_1 appears once and has degree 2). Suppose this is true for trees with less than n vertices, $n \geq 2$, given $(a_1, \ldots, a_{n-2})$, we want to prove that this is still true for the corresponding tree T which has n vertices.

We know that a_1 is the unique neighbour of the leaf a_0 of smallest label. So a_0 never appears in the Prüfer code. Then its unique neighbour appears in the first position of the Prüfer code. Looking at the Prüfer code from its second position $(\ldots, a_{n-2})$, we get a Prüfer code for a tree T' where, by induction hypothesis, $\deg_{T'}(v) - 1$ is the number of times the label v appears in the Prüfer code. So for all vertices which are neither a_0 nor a_1, they will have the same degree in T' as in T, thus they appear the same number of times in the corresponding Prüfer codes. Since a_0 does not belong to the vertices of T', the node a_1 has a degree $\deg_{T'}(a_1)$ in T' which is one less than its degree in T, so $\deg_{T'}(a_1) = \deg_T(a_1) - 1$, so the label a_1 appears $\deg_{T'}(a_1) - 1 = \deg_T(a_1) - 2$ in T', but it also appears as the first coefficient of the Prüfer code for T. So in T, it appears $\deg_T(a_1) - 1$. Since a_0 is a leaf, it is not present in the Prüfer code for T and so it does appear $\deg_T(a_0) - 1 = 0$.

Now T is a tree where all nodes have degree ≤ 2. So let P be the unique path in T from one leaf of T to another. Suppose there exists a vertex of T that is not in P, this would change the degree of at least one vertex of P, which is not possible (suppose $\deg(v)$ is increased by 1, if v has degree 2, this contradicts that all nodes have degree ≤ 2, otherwise v cannot be a leaf). So P is all of T, and T must be a path.

7.3 Network Flows

Exercise 25. Show that

$$|f| = \sum_{u \in I(t)} f(u,t),$$

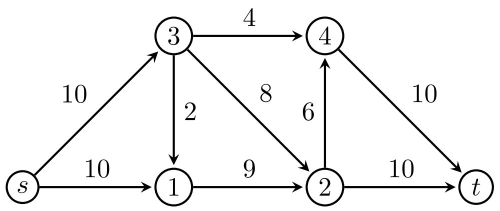
that is, the strength of the flow is the sum of the values of f on edges entering the sink.

Solution 25. By definition, $|f| = \sum_{u \in O(s)} f(s,u)$, that is, the strength of the flow is the sum of the values of a flow f on the edges leaving the source. Now use the flow conservation, which is true for every vertex $v \in V$, $v \neq s, t$, to say that

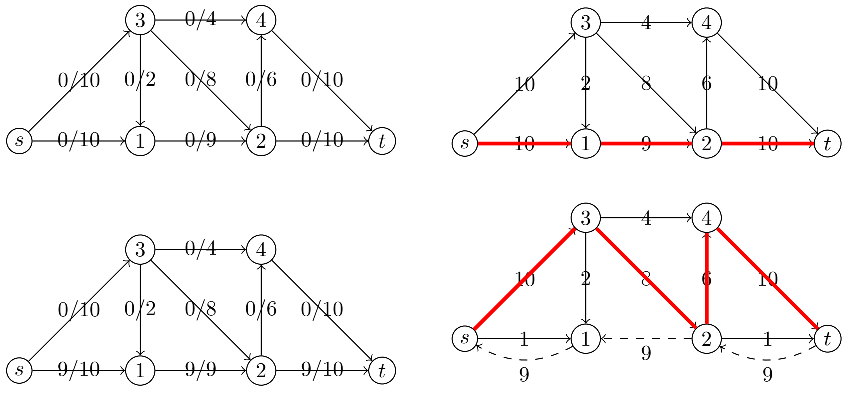
$$\begin{aligned} |f| &= \sum_{u \in O(s)} f(s,u) \\ &= \sum_{u \in O(s)} f(s,u) + \sum_{v \in V \setminus \{s,t\}} \left[\sum_{u \in O(v)} f(v,u) - \sum_{u \in I(v)} f(u,v) \right] \\ &= \sum_{u \in I(t)} f(u,t) \end{aligned}$$

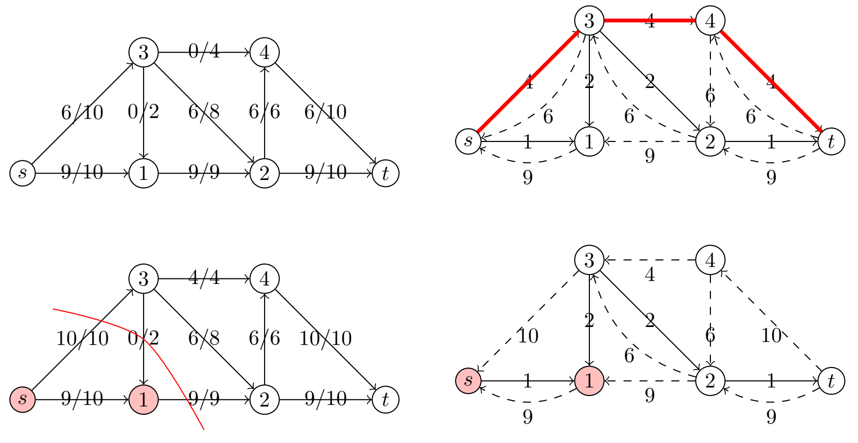
as desired. Indeed, each edge e appears twice in $\sum_{u \in O(v)} f(v,u) - \sum_{u \in I(v)} f(u,v)$ (e.g. the edge (v_1, v_2) will appear once as going out of v_1, and once as coming in v_2), once as an outoing edge (with positive sign) and once as an incoming edge (with negative sign) - except for edges entering t which appear exactly once, with positive sign, and edges leaving s which appear exactly once, with negative sign, thus canceling out the first sum $\sum_{u \in O(s)} f(s,u)$.

Exercise 26. Use Ford-Fulkerson algorithm to find a maximum flow in the following network:



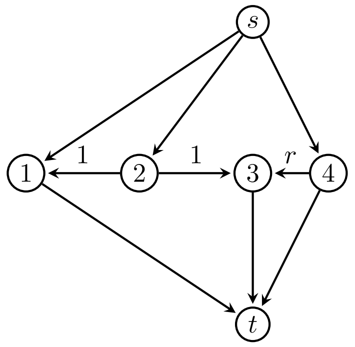
Solution 26. We write the graph on the left, and the residual graph on the right.





There is no path from s to t anymore, so the algorithm stops with a max flow of 19. We can also observe what the proof of Ford-Fulkerson algorithm tells us, namely that when the algorithm stops, the min-cut can (S,T) be read by putting in S the source s and nodes that can be reached from s, in this case $S=\{s,1\}$.

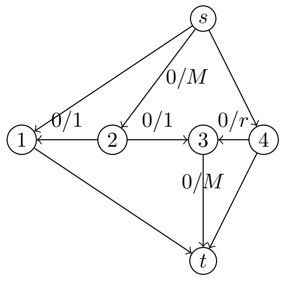
Exercise 27. Here is a famous example of network (found on wikipedia and in many other places) where Ford-Fulkerson may not terminate. The edges capacities are 1 for $(2,1)$, $r=(\sqrt{5}-1)/2$ for $(4,3)$, 1 for $(2,3)$, and M for all other edges, where $M \geq 2$ is any integer.

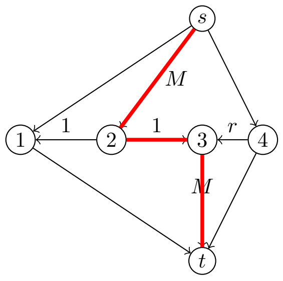


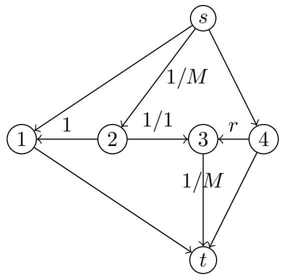
1. Explain why Ford-Fulkerson algorithm may not be able to terminate.

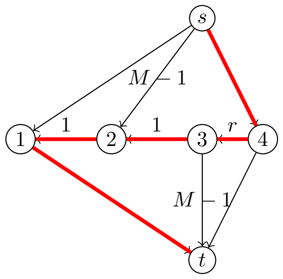
2. Apply Edmonds-Karp algorithm to find a maximum flow in this network.

Solution 27. We start with an initial flow of $f=0$. At this step (step 1), choose for path P the vertices $s,2,3,t$, where the flow 1 can be sent, since 1 is the minimum among $M,1,M$.

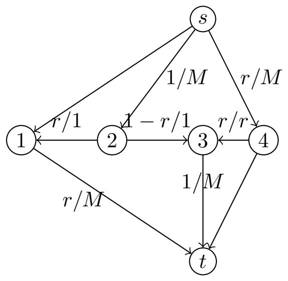


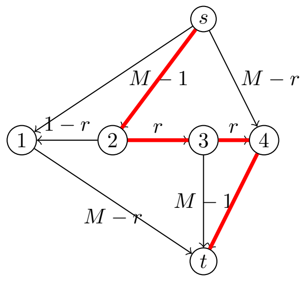




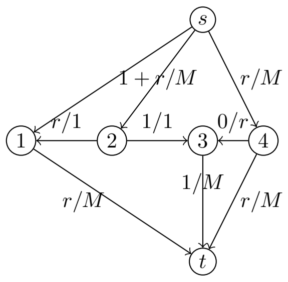


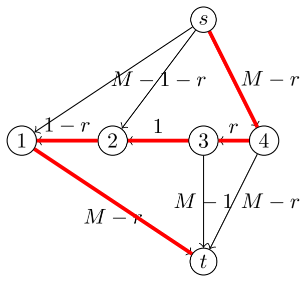
At step 2, choose for path P the vertices $s, 4, 3, 2, 1, t$. Since $r < 1$, the flow sent is r.

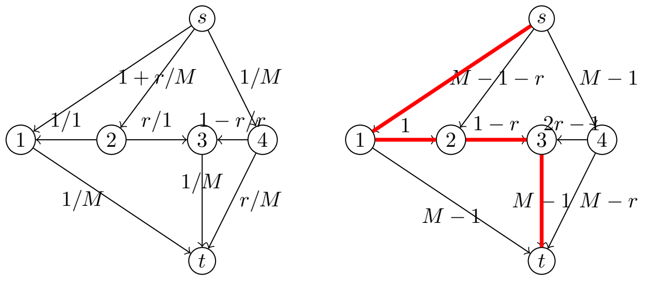




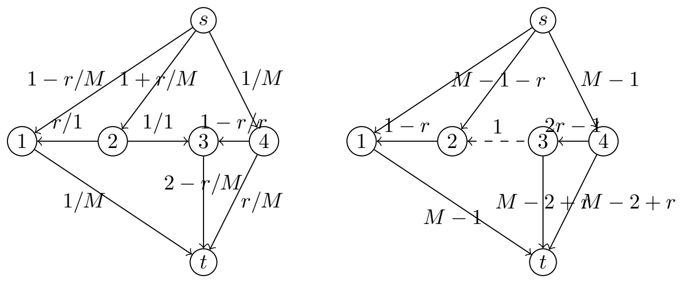
Note that the residual capacity for $(2, 1)$ is $1 - r$, but it turns out that $1 - r = r^2$. At this step (step 3), pick the path $s, 2, 3, 4, t$. Since $r < 1$, we send r again. At this step (step 4), pick the path $s, 4, 3, 2, 1, t$. Since $r \approx 0.618$ and $1 - r < r$, we send $1 - r$.





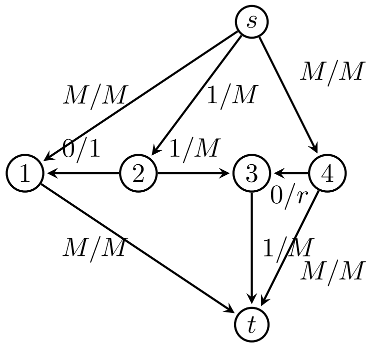


We finally pick the path $s, 1, 2, 3, t$, which carries $1 - r$.



At step 2, the residual capacities for $(2,1), (2,3), (4,3)$ were $1, 0, r$, while at step 6, they become $1 - r = r^2, 0, 2r - 1 = r^3$ (since $r^2 = 1 - r$, $r^3 = r - r^2 = r - (1 - r)$). This means that we can reuse the same paths used in steps 2,3,4,5 again in the same order, and do it repeatedly as many times as we wish, and the residual capacities will remain in the same form (namely $r^n, 0, r^{n+1}$), and the algorithm does not terminate.

Using Edmonds-Karp algorithm, a shortest path is used first. Since there are 2 paths of length 2, each of them will be used one after the other, each contributing a flow of M. At this point, there will be a path of length 3 left, $s, 2, 3, t$ which gives a contribution of 1 to the flow, after which the algorithm terminates. The max flow is thus $2M + 1$, as shown below.



Exercise 28. Menger's Theorem states the following. Let G be a directed graph, let u, v be distinct vertices in G. Then the maximum number of pairwise edge-disjoint paths from u to v equals the minimum number of edges whose

removal from G destroys all directed paths from u to v. Prove Menger's Theorem using the Max Flow- Min Cut Theorem.

Solution 28. We make G into a network by assigning each edge a capacity of 1, and letting $u = s$ be the source, and $v = t$ be the sink. Let $(S, \bar{S})$ be a min cut, with max low f. Since every edge has capacity 1, $C(S, \bar{S})$ is the number of edges in the cut, thus removing $C(S, \bar{S})$ edges destroys all s, t directed paths, and the minimum number of edges whose removal from G destroys all directed paths from s to t is upper bounded by

$$C(S, \bar{S}) = |f|$$

since the min cut is the max flow. Now we look at $|f|$. Every edge going out of s carries a flow of either 0 or 1. Then no matter which path these flows are following, they will be edge disjoint because for two paths not to be edge disjoint, they need to meet at one vertex, and then continue together on the same edge, which is impossible, because what goes in (2 units of low) comes out (1 unit of flow), by the property of flow. So every edge out of the source with flow 1 will go through a path, all of them, $|f|$ of them, will give $|f|$ edge disjoint paths from s to t. A priori, we have no guarantee that we get all edge disjoint paths like that, so $|f|$ is upper bounded by the maximum number of edge disjoint s, t paths. We have thus showed $\leq$, that is, the minimum number of edges whose removal from G destroys all directed paths from u to v is less or equal to the maximum number of pairwise edge-disjoint paths from u to v.

The reverse inequality is clear, because we must remove at least one edge from each of the edge disjoint s, t paths to destroy a path (and no edge removal can destroy two edge disjoint s, t paths).

Exercise 29. (*) Let $G = (V, E)$ be an undirected graph that remains connected after removing any $k - 1$ edges. Let s, t be any two nodes in V.

1. Construct a network G' with the same vertices as G, but for each edge $\{u, v\}$ in G, create in G' two directed edges (u, v), (v, u) both with capacity 1. Take s for the source and t for the sink of the network G'. Show that if there are k edge-disjoint directed paths from s to t in G', then there are k edge-disjoint paths from s to t in G.

2. Use the max-flow min-cut theorem to show that there are k edge-disjoint paths from s to t in G'.

Solution 29.

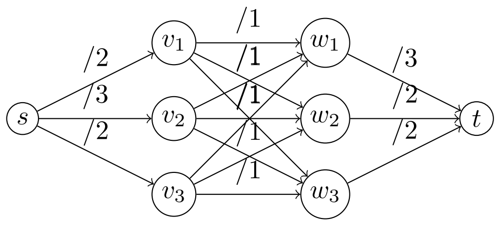
1. We have to show that if there are k edge-disjoint directed paths from s to t in G', then there are k edge-disjoint paths from s to t in G. If we have two edge-disjoint paths P_1, P_2 in G', as long as they do not use both directions (u, v) and (v, u) of an undirected edge $\{u, v\}$, they will correspondto edge-disjoint paths in G. So the only case to worry about is if they do use both (u, v) and (v, u). So suppose P_1 uses (u, v) and P_2 uses (v, u), that is, P_1 is of the form $s \to u \to v \to t$, while P_2 is of the form $s \to v \to u \to t$. Then consider the paths P_1' that follows P_1 from

s to u, and then P_2 from u to t, and P_2' that follows P_2 from s to v, and then P_1 from v to t. So now P_1' and P_2' do not have edges in common, and the corresponding paths will be edge-disjoint in G.

2. To find k edge-disjoint paths from s to t in G', we invoke the max-flow min-cut theorem. Since G remains connected after removing any $k-1$ edges, this implies that the minimum cut in G' has value at least k. To this mininimum cut corresponds an integral maximum flow in which the flow on every edge in G' is either 0 or 1 (since the capacities are 1 for all edges). So following the paths given by the edges carrying a flow of 1, we find k edge-disjoint paths from s to t in G'.

So the second part tells us there are k edge-disjoint paths in G', and the first part tells us that these give us k edge-disjoint paths in G, so we just showed that if a graph remains connected after removing any $k-1$ edges, then there are at least k edge-disjoint paths between every pair of vertices in G. Since the converse is true, this gives us a characterization of k-edge connectivity, and a graph G is k-edge connected if and only G remains connected after removing any $k-1$ edges if and only if there are at least k edge-disjoint paths between every pair of vertices in G.

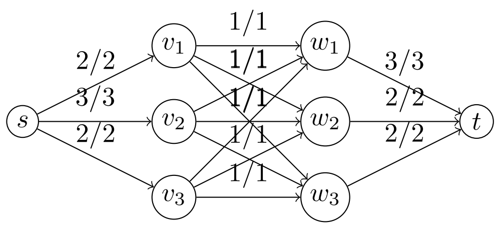
Exercise 30. 1. Compute a maximal flow in the following network, where each edge (v_i, w_j) has a capacity of 1, for $i, j = 1, 2, 3$: **Solution.** we give a flow of 1 over the edges (v_1, w_1), (v_1, w_2), (v_2, w_1), (v_2, w_2),(v_2, w_3), (v_3, w_1), (v_3, w_3).



2. Interpret the above as a placement of a number of balls of different colours into bins of different capacities, such that no two balls with the same colour belong to the same bin. More generally, describe in terms of flow over a network the problem of placing b_i balls of a given colour, for $i = 1, \ldots, m$ colours, into n bins of different capacities c_j, $j = 1, \ldots, n$, such that no two balls with the same colour belong to the same bin (b_i, c_j are all positive integers).

3. Give a necessary and sufficient condition on the max flow of the above graph to ensure that the ball placement problem has a solution (that is a condition (C) such that (C) holds if and only if the ball placement problem has a solution).

Solution 30. 1. We give a flow of 1 over the edges (v_1, w_1), (v_1, w_2), (v_2, w_1), (v_2, w_2),(v_2, w_3), (v_3, w_1), (v_3, w_3)

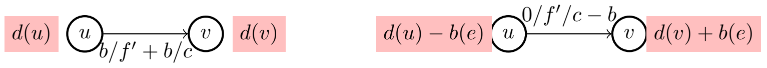


2. We can interpret the flow as having 2 red balls, 3 green balls, and 2 blue balls, the 2 red balls are put in bins w_1 and w_2, one green ball goes into each of the bins, and 2 blue balls are put in bin w_1 and w_3. In other words, bin 2 has one green and one red ball, bin 1 has one bin of each colour, and bin 3 has one blue and one green ball. More generally, if we have b_i balls for $i = 1, \dots, m$ colours, we create v_i, $i = 1, \dots, m$ vertices, and the edge (s, v_i) has a capacity of b_i. Then we create w_j, $j = 1, \dots, n$ bins, and there are nm edges connecting each v_i with each w_j, finally there is an edge (w_j, t) of capacity c_j for $j = 1, \dots, n$.

3. Suppose the max flow for the above graph saturates the outgoing edges of s (capacities are integers thus so is the max flow), then the flow from s to v_i represents b_i balls of each of the m colours, then the condition on having edges of capacity 1 means that no two balls of the same colour can be allocated to the same bin, and since we have a max flow, all the $\sum_i b_i$ balls will be allocated to the bins, finally since we have a max flow, the number of balls in each bin will not exceed the bin capacity. We have just shown that a max flow of $\sum_i b_i$ provides a solution to the ball placement problem. Conversely, suppose that we have a solution the ball placement, we need to show this provides a max flow for our graph. Since we have a solution to the ball placement, we know that the b_i balls of the same colour go to b_i different bins, this means b_i edges where the flow between v_i and w_j is 1, this holds for all colours, so the flow across v_i and w_j is exactly $\sum_i b_i$. Then since the flow going out of every v_i is b_i, we need to have an incoming flow of b_i by definition of flow, which is possible since the capacity of the edge (s, v_i) is b_i. Finally, since we have a solution to the ball placement problem, the number of balls per bin cannot exceed the bin capacity, so whatever flow enters w_j, the same flow can go out on the edge (w_j, t). We thus have a well defined flow, which has value $\sum_i b_i$, and since this quantity is equal to the cut between s and the rest of the graph, this is a max flow. In summary, we have shown that the ball placement problem has a solution if and only if the corresponding network has a max flow of value $\sum_i b_i$.

Exercise 31. Show that one can always consider min-cost-flow networks were lower capacities are zero, that is, if a network has lower capacites which are not zero, the network can be replaced by an equivalent network were all lower capacities are zero.

Solution 31. It is often useful to allow edges with a lower bound $b(e)$ which is not zero. However, to describe proofs or algorithms, it is also often simpler to assume that $b(e) = 0$ for all edges e. To have both cases to cohabit, one way is to describe proofs and algorithms with $b(e) = 0$, and show that one can always replace a network G with $b(e)$ not zero with another network G' where $b'(e) = 0$.



To do so, consider another min cost flow network G' with the same nodes and edges as G. In G, every edge has a lower capacity $b(e)$ which can be positive or negative, and an upper capacity $c(e)$. In G', every node has lower capacity $b'(e) = 0$ and upper capacity $c'(e) = c(e) - b(e)$. This is just shifting the range of values taken by the flow so that it starts at 0 instead of $b(e)$. Also for $e = (u, v)$, the demands $d(u)$ and $d(v)$ in G become the demands $d'(u)$, $d'(v)$ in G' with $d'(u) = d(u) - b(e)$, $d'(v) = d(v) + b(e)$, which means that if several edges are adjacent to u, then $d'(u) = d(u) + \sum_e \text{ incoming } b(e) - \sum_e \text{ outgoing } b(e)$. Then work in G' and for a flow $f'(e)$ in the new network G', replace the flow $f(e)$ in the original problem with$f'(e) + b(e)$.

We start by showing that $f' + b$ in G is indeed a flow in G. The flow bound constraint $b(e) \leq f(e) = f'(e) + b(e) \leq c(e)$ in G then becomes $0 \leq f'(e) \leq c(e) - b(e) = c'(e)$, which is satisfied in G'. Then for flow conservation, we have for every node $v \in G$:

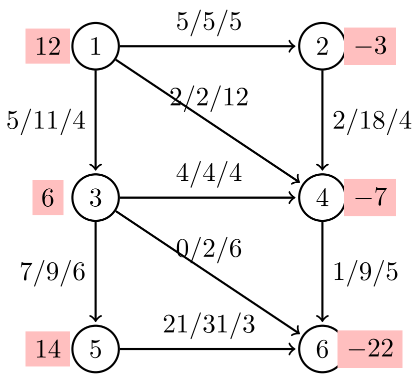
$$\begin{aligned}
& \sum_u f(v,u) - \sum_u f(u,v) \\
= \; & \sum_u (f'(v,u) + b(v,u)) - \sum_u (f'(u,v) + b(u,v)) \\
= \; & d'(v) + \sum_u b(v,u) - \sum_u b(u,v) \\
= \; & (d(v) + \sum_u b(u,v) - \sum_u b(v,u)) + \sum_u b(v,u) - \sum_u b(u,v) \\
= \; & d(v)
\end{aligned}$$

using flow conservation in G' for the second equality, and the definition of d' for the next one. Next we argue that doing the cost minimization in G' results in the cost minimization in G. The minimization of cost in G becomes minimizing

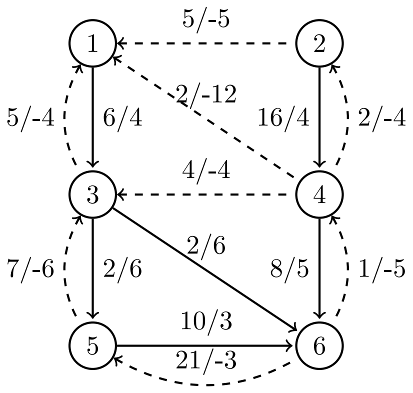
$$\sum \gamma(e) f(e) = \sum \gamma(e) f'(e) + \sum \gamma(e) b(e)$$

and since $\sum \gamma(e) b(e)$ is a constant, it is equivalent to do the minimization over $f(e)$ or over $f'(e)$ (of course the actual values of the optimal cost are different).

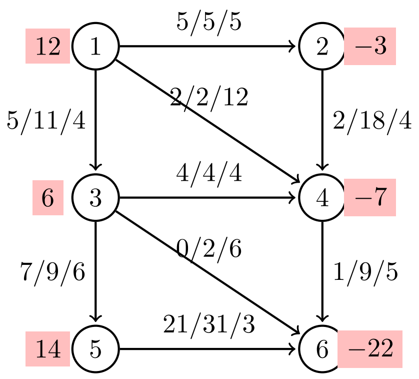
Exercise 32. Compute the residual graph G_f of the following graph G with respect to the flow given. Each edge e is labeled with $(f(e), c(e), \gamma(e))$.



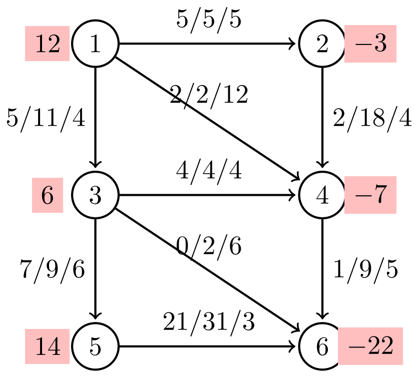
Solution 32. The residual graph is obtained by putting on edges the capacity minus the flow, assuming this quantity is strictly positive (otherwise, the edge is removed), and a reverse edge with the flow on it (if the flow is strictly positive). This is the same procedure as for Ford-Fulkerson's algorithm. Then the cost $\gamma(e)$ remains the same on edges with the original direction, and it becomes $\gamma(e)$ for new edges in the reverse direction. Demands are set to 0:

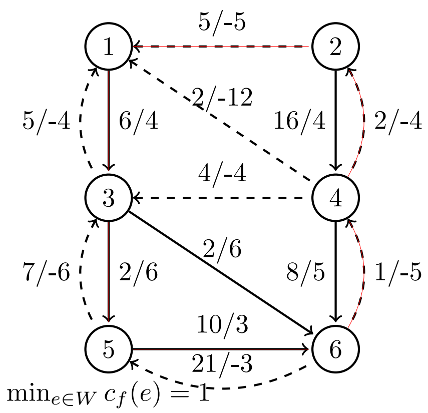


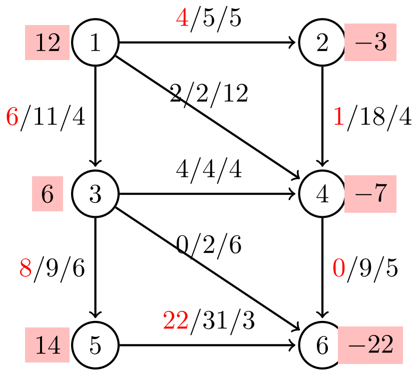
Exercise 33. Consider the following graph G. Each edge e is labeled with $(f(e), c(e), \gamma(e))$. Solve the min-cost-low problem for this graph, using the flow given as initial feasible flow.

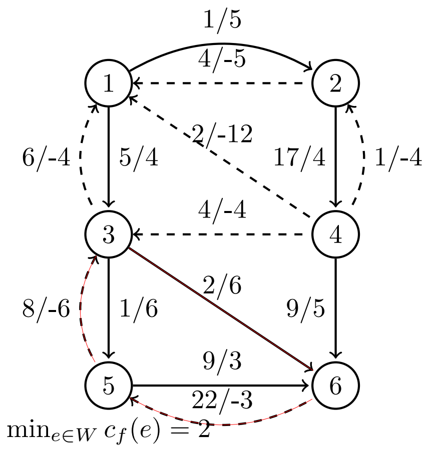


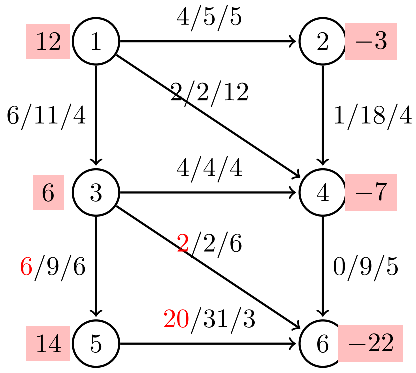
Solution 33. We use the cancelling cycle algorithm. We compute the residual graph, find a negative cost cycle, update the flow according in the original graph, and repeat.

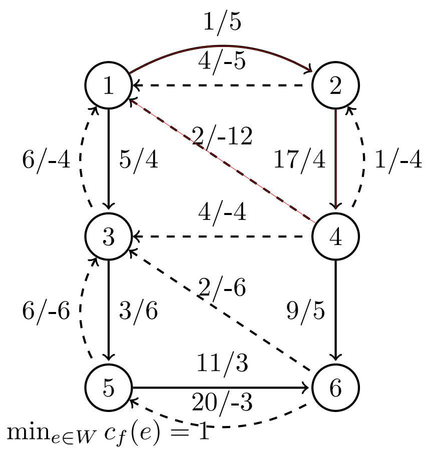




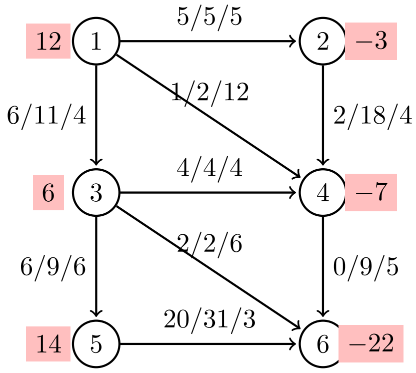




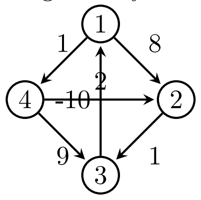




We do one more iteration of updating the original network, which gives a residual graph with no negative cost cycle, and thus we found an optimal flow:



Exercise 34. Use Floyed-Warshall algorithm to detect the presence of at least one negative cycle in the graph below.



Solution 34. The matrix D at $k=0$ is given by

$$D = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ -10 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix}.$$

At the first iteration, $k = 1$, we have

$$D = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ -10 & -2 & 0 & -9 \\ \infty & 2 & 9 & 0 \end{bmatrix}.$$

At the second iteration, $k = 2$, we have

$$D = \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ -10 & -2 & -1 & -9 \\ \infty & 2 & 3 & 0 \end{bmatrix}.$$

When $k = 3$, we have

$$D = \begin{bmatrix} -1 & 7 & 8 & 9 \\ -9 & -1 & 0 & -8 \\ -11 & -3 & -2 & -10 \\ -7 & 1 & 2 & -6 \end{bmatrix}.$$

Finally for $k = 4$, we have

$$D = \begin{bmatrix} -1 & 7 & 8 & 9 \\ -9 & -1 & 0 & -8 \\ -17 & -9 & -8 & -16 \\ -7 & 1 & 2 & -6 \end{bmatrix}.$$

Indeed, all the nodes are involved in a negative cycle.

7.4 Linear Programming

Exercise 35. Compute all the basic feasible solutions of the following LP:

$$\begin{aligned} \min \quad & x_1 + x_2 \\ s.t. \quad & x_1 + 2x_2 \geq 3 \\ & 2x_1 + x_2 \geq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Solution 35. We introduce slack variables:

$$\begin{aligned} \min \quad & x_1 + x_2 \\ s.t. \quad & x_1 + 2x_2 - s_1 = 3 \\ & 2x_1 + x_2 - s_2 = 2 \\ & x_1, x_2, s_1, s_2 \geq 0. \end{aligned}$$

To look for basic solutions, we need 2 columns of A which are linearly independent, where

$$A = \begin{bmatrix} 1 & 2 & -1 & 0 \\ 2 & 1 & 0 & -1 \end{bmatrix}, \text{ and } \begin{bmatrix} 1 & 2 & -1 & 0 \\ 2 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

If we pick the first 2 columns:

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1/3 & 2/3 \\ 2/3 & -1/3 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 1/3 \\ 4/3 \end{bmatrix}.$$

This gives the basic feasible solution $(1/3, 4/3, 0, 0)$. Choosing the columns 1 and 3 gives:

$$\begin{bmatrix} 1 & -1 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ s_1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

thus $s_1 = -1$ and this basic solution is not feasible. Choosing the columns 1 and 4 gives:

$$\begin{bmatrix} 1 & 0 \\ 2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

and $(3, 0, 0, 4)$ gives a basic feasible solution. Choosing the columns 2 and 3 gives:

$$\begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_2 \\ s_1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

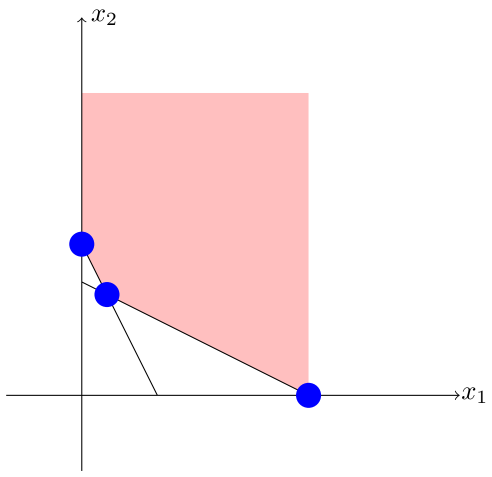
for the basic feasible solution $(0, 2, 1, 0)$. Choosing the columns 2 and 4 gives:

$$\begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_2 \\ s_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

and $s_2 = -1/2$, which is not feasible. Choosing the columns 3 and 4 gives:

$$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

and $s_1 = -3$, which is not feasible. We thus found three basic feasible solutions, corresponding to $(1/3, 4/3), (3, 0)$ and $(0, 2)$.



Exercise 36. Consider the linear program:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & -x_1 + x_2 \leq 1 \\ & 2x_1 + x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Prove by computing reduced costs that $[1/3, 4/3, 0, 0]$ is an optimal solution for this LP.

Solution 36. The constraint matrix A taking into account slack variables is:

$$A = \begin{bmatrix} -1 & 1 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{bmatrix}, \text{ with } b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

and we want to show that the BFS $[1/3, 4/3, 0, 0]$ corresponding to $B = \{1, 2\}$ is optimal. We compute the reduced cost $r_q = (c_N^T)_q - c_B^T A_B^{-1} (A_N)_q$ for $q = 1, 2$, corresponding to the non-basic variables s_1, s_2. We have

$$A_B^{-1} = \frac{1}{3} \begin{bmatrix} -1 & 1 \\ 2 & 1 \end{bmatrix}, \ A_N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

So for $q = 1$, which is s_1, we have

$$r_1 = (c_N^T)_1 - c_B^T A_B^{-1} (A_N)_1 = 0 - [1, 1] \begin{bmatrix} -1 & 2 \end{bmatrix} = -1.$$

For $q = 2$, which is s_2, we have

$$r_2 = (c_N^T)_2 - c_B^T A_B^{-1} (A_N)_2 = 0 - [1, 1] \begin{bmatrix} 1 & 1 \end{bmatrix} = -2.$$

This shows that the BFS is optimal.

Exercise 37. Solve the following LP using the simplex algorithm:

$$\begin{aligned} \max \quad & 6x_1 + x_2 + x_3 \\ s.t. \quad & 9x_1 + x_2 + x_3 \leq 18 \\ & 24x_1 + x_2 + 4x_3 \leq 42 \\ & 12x_1 + 3x_2 + 4x_3 \leq 96 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Solution 37. This LP is of the form $Ax \leq b$, we add slack variables s to obtain an LP of the form $Ax + s = b$, $s, b \geq 0$. This also means that we have an immediate basic feasible solution givey by setting $x = 0$ and $z = b$, that is

$$(0, 0, 0, 18, 42, 96).$$

This corresponds to a simplex tableau of the form

x_1	x_2	x_3	s_1	s_2	s_3	
9	1	1	1	0	0	18
24	1	4	0	1	0	42
12	3	4	0	0	1	96
6	1	1	0	0	0	0

We decide a first column pivot, corresponding to the variable we want to increase, say x_2, $j = 2$. The reason for choosing $j = 2$ is simply because computations seem easier in this column. Then the first row gives the strongest constraint on x_2, since $18/1 < 96/3 = 32$. Note that pushing x_2 to 18 means $x_1 = x_3 = 0$ from the 1st constraint, it also means $4x_3 \leq 24$, that is x_3 can be pushed to 8 from the 2nd constraint, and $4x_3 \leq 96 - 3 \cdot 18 = 42$, from the 3rd constraint is actually a less strong constraint on x_3. These can be read from the simplex tableau, by expressing the 2nd and 3rd constraints as a function of the first one, which is done by asking that $a_{12} = 1$, and the other coefficients in this column are 0. By computing (row 2)−(row 1), and (row 3)− (3· row 1), we get

x_1	x_2	x_3	s_1	s_2	s_3	
9	1	1	1	0	0	18
15	0	3	−1	1	0	24
−15	0	1	−3	0	1	42
−3	0	0	−1	0	0	−18

where we also updated the last row by computing (row 4)−(row 1). This simplex tableau also gives us the reduced costs in the last row, and all coefficients being non-positive means that was have already obtained the optimal. Then $x_1 = s_1 = 0$ and the optimal solution is $(x_1, x_2, x_3) = (0, 18, 0)$, the objective function has a maximum of 18.

Exercise 38. Solve the following LP using the simplex algorithm:

$$\begin{aligned} \max \quad & 3x_1 + x_3 \\ s.t. \quad & x_1 + 2x_2 + x_3 = 30 \\ & x_1 - 2x_2 + 2x_3 = 18 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Solution 38. Since this LP is in the form $Ax = b$, we use artificial variables to solve it.

We introduce the variables w_1, w_2, and first solve the following LP:

$$\begin{aligned} \min \quad & w_1 + w_2 \\ s.t. \quad & x_1 + 2x_2 + x_3 + w_1 = 30 \\ & x_1 - 2x_2 + 2x_3 + w_2 = 18 \\ & x_1, x_2, x_3, w_1, w_2 \geq 0. \end{aligned}$$

To minimize $w_1 + w_2$ is equivalent to maximize $-w_1 - w_2$, and it seems that we are close to have a basic feasible solution, wince we could set $x_1 = x_2 = x_3$ to 0 from which we could get initial values for w_1, w_2, except that the objective function is not expressed in terms of x_1, x_2, x_3. We thus remedy to this, and ot the computations using the simplex tableau:

x_1	x_2	x_3	w_1	w_2	
1	2	1	1	0	30
1	$-s$	2	0	1	18
0	0	0	-1	-1	0

To express the objective function in terms of x_1, x_2, x_3, we need to have 0 in the last row, in columns 3 and 4. We add row 1 and row 2 to row 3, to get

x_1	x_2	x_3	w_1	w_2	
1	2	1	1	0	30
1	-2	2	0	1	18
2	0	3	0	0	48

We now can read a basic feasible solution. Set $x_1 = x_2 = x_3$ to 0, and $(w_1, w_2) = (30, 18)$ to get $(0, 0, 0, 30, 18)$.

Then if we choose for pivot the first column, we get

x_1	x_2	x_3	w_1	w_2	
0	4	-1	1	-1	12
1	-2	2	0	1	18
0	4	-1	0	-2	12

and then using for pivot the second column:

x_1	x_2	x_3	w_1	w_2	
0	1	$-1/4$	$1/4$	$-1/4$	3
1	0	$3/2$	$1/2$	$1/2$	24
0	0	0	-1	-1	0

This corresponds to the basic feasible solution $(24, 3, 0, 0, 0)$ which corresponds to the basic feasible solution $(24, 3, 0)$ of the original LP. (Note that choosing as pivot the 3rd column first, and then the second column, gives $(0, 7, 16)$.) Deleting the w_1, w_2 columns, and using the original objective function gives

x_1	x_2	x_3	
0	1	$-1/4$	3
1	0	$3/2$	24
3	0	1	0

If we use the first column as pivot 1, we need to have a 0 instead of a 3, thus we get

x_1	x_2	x_3	
0	1	$-1/4$	3
1	0	$3/2$	24
0	0	$1-9/2$	-72

This is telling us that the basic feasible solution that we found is actually optimal.

Exercise 39. Show that the dual of the dual is the primal.

Solution 39. For a primal P of the form

$$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

the dual D is

$$\begin{aligned} \min \quad & b^T y \\ s.t. \quad & A^T y \geq c \\ & y \geq 0. \end{aligned}$$

It is equivalent to

$$\begin{aligned} \max \quad & -b^T y \\ s.t. \quad & -A^T y \leq -c \\ & y \geq 0 \end{aligned}$$

which is of the form

$$\begin{aligned} \max \quad & (c')^T y \\ s.t. \quad & A' y \leq b' \\ & y \geq 0 \end{aligned}$$

with $c' = -b$, $b' = -c$ and $A' = -A^T$. Now this LP is of the same for as a primal, and its dual is

$$\begin{aligned} \min \quad & (b')^T x \\ s.t. \quad & (A')^T x \geq c' \\ & x \geq 0 \end{aligned}$$

and replacing c', b', A' gives

$$\begin{aligned} \min \quad & -c^T x \\ s.t. \quad & -Ax \geq -b \\ & x \geq 0, \end{aligned}$$

which is

$$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

as desired.

Exercise 40. Consider the following LP:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + 2x_2 \leq 4 \\ & 4x_1 + 2x_2 \leq 12 \\ & -x_1 + x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Compute its dual.

Solution 40. The LP is of the form

$$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

with $c = (1, 1)$, $b^T = (4, 12, 1)$ and

$$A = \begin{bmatrix} 1 & 2 \\ 4 & 2 \\ -1 & 1 \end{bmatrix}$$

so its dual is

$$\begin{aligned} \min \quad & b^T y \\ s.t. \quad & A^T y \geq c \\ & y \geq 0, \end{aligned}$$

namely

$$\begin{aligned} \min \quad & 4y_1 + 12y_2 + y_3 \\ s.t. \quad & y_1 + 4y_2 - y_3 \geq 1 \\ & 2y_1 + 2y_2 + y_3 \geq 1 \\ & y \geq 0. \end{aligned}$$

Exercise 41. Consider the following LP:

$$\begin{aligned} \max \quad & 2x_1 + 4x_2 + x_3 + x_4 \\ s.t. \quad & x_1 + 3x_2 + x_4 \leq 4 \\ & 2x_1 + x_2 \leq 3 \\ & x_2 + 4x_3 + x_4 \leq 3 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

Compute its dual. Show that $x = [1, 1, 1/2, 0]$ and $y = [11/10, 9/20, 1/4]$ are optimal solutions for respectively this LP and its dual.

Solution 41. The dual is given by

$$\begin{aligned} \min \quad & 4y_1 + 3y_2 + 3y_3 \\ s.t. \quad & 3y_1 + y_2 + y_3 \geq 4 \\ & 4y_3 \geq 1 \\ & y_1 + y_3 \geq 1 \\ & y \geq 0. \end{aligned}$$

We just need to check that $x = (1, 1, 1/2, 0)$ is feasible for the primal, and $y = (11/10, 9/20, 1/4)$ is feasible for the dual, the compute the objective function for each case, to see that one obtains $13/2$ twice. Therefore, they are both optimal for their respective problems.

Exercise 42. In Exercise 40, we computed the dual of the LP:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & x_1 + 2x_2 \leq 4 \\ & 4x_1 + 2x_2 \leq 12 \\ & -x_1 + x_2 \leq 1 \\ & x_1, x_2 \geq 0. \end{aligned}$$

Solve both this LP and its dual.

Solution 42. Write the simplex tableau, by choosing the first column as first pivot (with 2nd row), and then by choosing the second column as second pivot (with 1rst row) gives

x_1	x_2	s_1	s_2	s_3	
0	1	$2/3$	$-1/6$	0	$2/3$
1	0	$-1/3$	$1/12$	0	$8/3$
0	0	$-1/3$	$-1/6$	0	$-10/3$

from which we can read that an optimal solution for the primal is $(8/3, 2/3)$ for which the objective function is $10/30$, and an optimal solution for the dual is $(1/3, 1/6, 0)$, which has also the same optimal objective value $10/30$ (as it should be).

Exercise 43. Solve the "Scissors-Paper-Stone" whose pay off matrix is

$$\begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}.$$

Solution 43. Player 2 is trying to

$$\begin{array}{ll} \min & v \\ s.t. & \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \leq \begin{bmatrix} v \\ v \\ v \end{bmatrix} \\ & q_1 + q_2 + q_3 = 1 \\ & q \geq 0. \end{array}$$

We add $k = 1$ to each coefficient of the pay off matrix to make them non-negative (enough in this case to guarantee that $v > 0$). Once we know $v > 0$, we do the change of variables $x_j = q_j/v$, which implies that $v = 1/\sum x_j$ and the linear program becomes

$$\begin{array}{ll} \max & x_1 + x_2 + x_3 \\ s.t. & \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\ & x \geq 0. \end{array}$$

Using a simplex tableau, we have

x_1	x_2	x_3	s_1	s_2	s_3	
1	2	0	1	0	0	1
0	1	2	0	1	0	1
2	0	1	0	0	1	1
1	1	1	0	0	0	0

Using the second row and the first column as pivot, and then the second column and first row as second pivot, gives

0	1	$-1/4$	$1/2$	0	$-1/4$	$1/4$
0	0	$9/4$	$-1/2$	1	$1/4$	$3/4$
1	0	$1/2$	0	0	$1/2$	$1/2$
0	0	$3/4$	$-1/2$	0	$-1/4$	$-1/2 - 1/4$

and a last pivot operation gives

$$x_1 = 1/3,\ x_2 = 1/3,\ x_3 = 1/3,$$

and $1/v = 1$ thus $v = 1$. Then

$$q = (1/3, 1/3, 1/3) = p.$$

Considering the symmetry of the pay off matrix, a posteriori, it should look clear that choosing one of the 3 moves uniformly at random is the best strategy.

Exercise 44. Solve the game whose pay off matrix is

$$\begin{bmatrix} 2 & -1 & 6 \\ 0 & 1 & -1 \\ -2 & 2 & 1 \end{bmatrix}.$$

Solution 44. Player 2 is trying to

$$\begin{array}{ll} \min & v \\ s.t. & \begin{bmatrix} 2 & -1 & 6 \\ 0 & 1 & -1 \\ -2 & 2 & 1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \leq \begin{bmatrix} v \\ v \\ v \end{bmatrix} \\ & q_1 + q_2 + q_3 = 1 \\ & q \geq 0. \end{array}$$

We add $k = 2$ to each coefficient of the pay off matrix to make them non-negative (enough in this case to guarantee that $v > 0$). Once we know $v > 0$, we do the change of variables $x_j = q_j/v$, which implies that $v = 1/\sum x_j$ and the linear program becomes

$$\begin{array}{ll} \max & x_1 + x_2 + x_3 \\ s.t. & \begin{bmatrix} 4 & 1 & 8 \\ 2 & 3 & 1 \\ 0 & 4 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\ & x \geq 0. \end{array}$$

Using a simplex tableau, we have

$$\begin{array}{cccccc|c} 4 & 1 & 8 & 1 & 0 & 0 & 1 \\ 2 & 3 & 1 & 0 & 1 & 0 & 1 \\ 0 & 4 & 3 & 0 & 0 & 1 & 1 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{array}$$

Using the first row and the first column as pivot, and then the second column and second row as second pivot, gives

$$\begin{array}{cccccc|c} 1 & 0 & 2+6/20 & 1/4+1/20 & -2/20 & 0 & 4/20 \\ 0 & 1 & -6/5 & 0 & -1/5 & 2/5 & 1/5 \\ 0 & 0 & 24/5+3 & 4/5 & -8/5 & 1 & -4/5+1 \\ \hline 0 & 0 & -1/10 & -1/10 & -3/10 & 0 & -2/5 \end{array}$$

from which we get that $1/v = 2/5$ thus $v = 5/2$. Then

$$x = (1/5, 1/5, 0) \Rightarrow q = (5/2)x = (1/2, 1/2, 0)$$

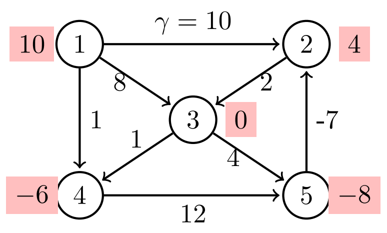
and

$$y = (1/10, 3/10, 0) \Rightarrow p = (5/2)y = (1/4, 3/4, 0).$$

Finally note that the value of the game for the original pay off matrix is $v - 2 = 1/2$.

7.5 The Network Simplex Algorithm

Exercise 45. Consider the following min cost flow network, where the demands are written at the nodes, and the costs are written next to each edge. Each edge capacity is infinite, and lower bound is 0.



Compute a minimal cost flow for this network, using the BFS $(x_{13}, x_{23}, x_{34}, x_{35}) = (10, 4, 6, 8)$.

Solution 45. Using complementary slackness, we have

$$\begin{aligned} u_1 - u_3 &= 8 \\ u_2 - u_3 &= 2 \\ u_3 - u_4 &= 1 \\ u_3 - u_5 &= 4 \end{aligned}$$

and letting $u_3 = 0$ gives $u_1 = 8, u_2 = 2, u_4 = -1$ and $u_5 = -4$. Then we check that the other dual conditions are satisfied. We have

$$u_5 - u_2 = -4 - 2 = -6$$

which is not less than -7. Thus (5,2) enters and creates the cycle $(5) \rightarrow (2) \rightarrow (3) \rightarrow (5)$. We notice that all the edges of the cycle have the same orientation. We currently have a flow of $x_{52} = 0$, $x_{23} = 4$, $x_{35} = 8$. Suppose we increase the flow by a constant k, this means we can push k units of flow on $x_{52} = k$, then $x_{23} = 4 + k$ and $x_{35} = 8 + k$. This creates for every node in the cycle an increase of k units which go in, but also k units which go out, which maintains the demand/supply constraints. Then the cost over this cycle is

$$c_{52}k + c_{23}(4+k) + c_{35}(8+k) = c_{23}4 + c_{34}8 + k(c_{52} + c_{23} + c_{35})$$

and the last term is $k(2+4-7) = -k$, in other terms, k can be increased arbitrarily, this will decrease the cost, and therefore the problem is unbounded. There is no optimal and the cost can be made arbitrarily small.

Exercise 46. Compute the dual of:

$$\begin{array}{ll} \min & \gamma^T x \\ s.t & \begin{bmatrix} A & 0_m \\ I_m & I_m \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} = \begin{bmatrix} d \\ c \end{bmatrix} \\ & x, s \geq 0. \end{array}$$

Solution 46. We first rewrite the primal program as:

$$\begin{array}{ll} \max & -\gamma^T x + 0_m^T \cdot s \\ s.t & \begin{bmatrix} A & 0_m \\ I_m & I_m \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} \leq \begin{bmatrix} d \\ c \end{bmatrix} \\ & \begin{bmatrix} -A & 0_m \\ -I_m & -I_m \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} \geq \begin{bmatrix} -d \\ -c \end{bmatrix} \\ & x, s \geq 0, \end{array}$$

or, using a single constraint matrix:

$$\begin{array}{ll} \max & [-\gamma^T,\ 0_m] \begin{bmatrix} x \\ s \end{bmatrix} \\ s.t & \begin{bmatrix} A & 0_m \\ I_m & I_m \\ -A & 0_m \\ -I_m & -I_m \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} = \begin{bmatrix} d \\ c \\ -d \\ -c \end{bmatrix} \\ & x, s \geq 0. \end{array}$$

Then the dual is

$$\begin{array}{ll} \min & [d^T,\ c^T,\ -d^T,\ -c^T] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \\ s.t & [w_1^T,\ w_2^T,\ w_3^T,\ w_4^T] \begin{bmatrix} A & 0_m \\ I_m & I_m \\ -A & 0_m \\ -I_m & -I_m \end{bmatrix} \geq [-\gamma^T \quad 0_m^T] \\ & w_1, w_2, w_3, w_4 \geq 0, \end{array}$$

where $w_1, w_2, w_3, w_4 \in \mathbb{R}^m$. This can be rewritten as

$$\begin{aligned} \min \quad & d^T(w_1 - w_3) + c^T(w_2 - w_4) \\ s.t. \quad & A^T(w_1 - w_3) + (w_2 - w_4) \geq -\gamma \\ & w_2 - w_4 \geq 0 \\ & w_1, w_2, w_3, w_4 \geq 0. \end{aligned}$$

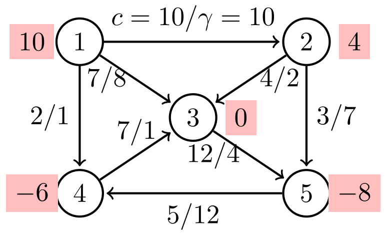
Set $v := w_4 - w_2$ and $u := w_3 - w_1$. We get

$$\begin{aligned} \min \quad & -d^T u - c^T v \\ s.t. \quad & -A^T u - v \geq -\gamma \\ & -v \geq 0, \end{aligned}$$

which finally gives

$$\begin{aligned} (D) : \max \quad & d^T u + c^T v \\ s.t. \quad & A^T u + v \leq \gamma \\ & v \leq 0. \end{aligned}$$

Exercise 47. Consider the following min cost flow problem, where every edge e is given a capacity $c(e)$ and a cost $\gamma(e)$.



1. Compute an optimal solution using the network simplex algorithm.
2. Compute an optimal solution using the negative cycle canceling algorithm.

Solution 47. We solve the above min cost flow problem wtih both methods.

Network Simplex Algorithm. We start by looking for a basic feasible solution, which can be described by a tripartion $(\mathcal{N}, \mathcal{S}, \mathcal{B})$ of the edges. We could use artificial variables, but we instead rely on our intuition since the graph is small enough. Since node 1 has a demand of 10, we need to send 10 units of flow through its outgoing edges, so we try to saturate $(1,3), (1,4)$, then $x_{12} = 1$. Then node 2 has a demand of 4, $x_{12} = 1$ brings one more unit of flow, so we need to have 5 units of flow going out, we try to saturate $(2,5)$, then $x_{23} = 2$. Then at node 3, we try to send $x_{35} = 9$, then $x_{43} = 0$. Then we are left with $x_{54} = 4$.

In summary,

$$\mathcal{B} = \{(1,2), (2,3), (3,5), (5,4)\}, \mathcal{N} = \{(4,3)\}, \mathcal{S} = \{(1,3), (1,4), (2,5)\},$$

corresponding to the BFS $x_{12} = 1$, $x_{23} = 2$, $x_{35} = 9$, $x_{54} = 4$ with $x_{13} = c_{13} = 7$, $x_{14} = c_{14} = 2$, $x_{25} = c_{25} = 3$ and $x_{43} = 0$. We check for optimality.

For all $(i, j) \in \mathcal{B}$, $\gamma_{ij} - u_i + u_j = 0$:

$$\begin{aligned}
\gamma_{12} - u_1 + u_2 = 0 &\quad \Rightarrow 10 - u_1 + u_2 = 0\\
\gamma_{23} - u_2 + u_3 = 0 &\quad \Rightarrow 2 - u_2 + u_3 = 0\\
\gamma_{35} - u_3 + u_5 = 0 &\quad \Rightarrow 4 - u_3 + u_5 = 0\\
\gamma_{54} - u_5 + u_4 = 0 &\quad \Rightarrow 12 - u_5 + u_4 = 0.
\end{aligned}$$

Set $u_4 = 0$. Then $u_5 = 12$, $u_3 = 16$, $u_2 = 18$, $u_1 = 28$. The vector u is not feasible, since $(2, 5) \in \mathcal{S}$, but $u_2 - u_5 \geq \gamma_{25} = 7$ should be satisfied, while $18 - 12 = 4 \leq 7$. Thus this solution is not optimal, and we try to improve it.

The edge (2,5) which is currently saturated is activated, it creates the cycle C given by $(5) \leftarrow (2) \rightarrow (3) \rightarrow (5)$. Since (2,5) is saturated, we need to decrease it, which means increasing on the rest of the cycle whose edges are reversed. Then $x_{23} = 2, c_{23} = 4$ and $x_{35} = 9, c_{35} = 12$, which means that on edge (2,3), we could increase the flow by 2, while on the edge (3,5), we could increase the flow by 3. So (2,3) leaves, and (2,5) gets decreased by 2. So $x_{23} = 4$, $x_{25} = 1$ and $x_{35} = 11$.

The sets $\mathcal{B}, \mathcal{S}, \mathcal{N}$ get updated as follows:

$$\mathcal{B} = \{(1, 2), (3, 5), (5, 4), (2, 5)\}, \mathcal{N} = \{(4, 3)\}, \mathcal{S} = \{(1, 3), (1, 4), (2, 3)\}.$$

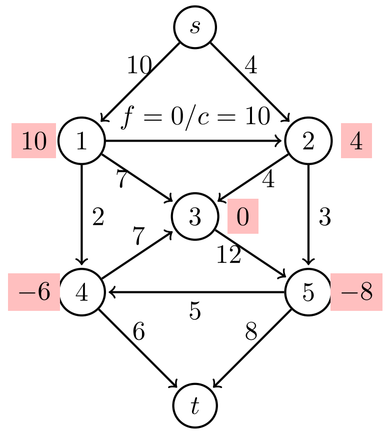
We check again for optimality. For all $(i, j) \in \mathcal{B}$, $\gamma_{ij} - u_i + u_j = 0$:

$$\begin{aligned}
\gamma_{12} - u_1 + u_2 = 0 &\quad \Rightarrow 10 - u_1 + u_2 = 0\\
\gamma_{25} - u_2 + u_5 = 0 &\quad \Rightarrow 7 - u_2 + u_5 = 0\\
\gamma_{35} - u_3 + u_5 = 0 &\quad \Rightarrow 4 - u_3 + u_5 = 0\\
\gamma_{54} - u_5 + u_4 = 0 &\quad \Rightarrow 12 - u_5 + u_4 = 0.
\end{aligned}$$

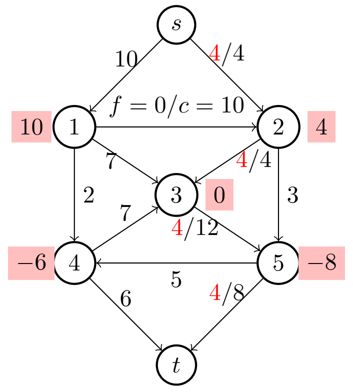
Setting $u_5 = 0$, we get $u_4 = -12$, $u_3 = 4$, $u_2 = 7$, $u_1 = 17$. Then

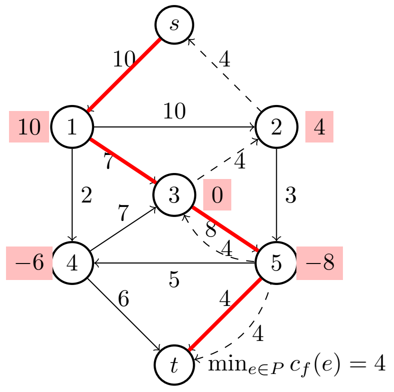
$$\begin{aligned}
u_4 - u_3 &\leq \gamma_{43} &\Rightarrow -12 - 4 \leq 1 \checkmark\\
u_1 - u_3 &\geq \gamma_{13} &\Rightarrow 17 - 4 \geq 8 \checkmark\\
u_1 - u_4 &\geq \gamma_{14} &\Rightarrow 17 + 12 \geq 1 \checkmark\\
u_2 - u_3 &\geq \gamma_{23} &\Rightarrow 7 - 4 \geq 2 \checkmark.
\end{aligned}$$

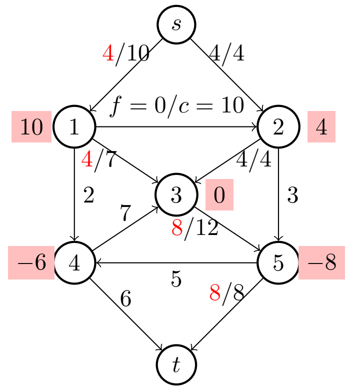
Cycle Canceling Algorithm. We compute a solution using the negative cycle canceling algorithm. For that, we need an initial feasible flow, which we compute by introducing an artificial source and sink and artificial with capacities the demands of the corresponding nodes, and by solving a max-flow min-cut problem.

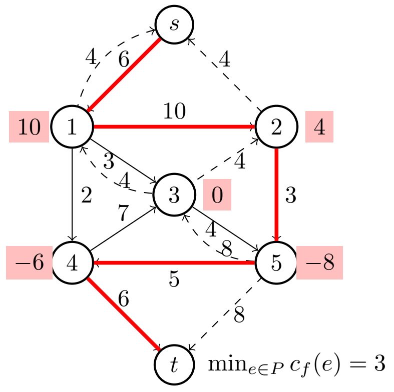


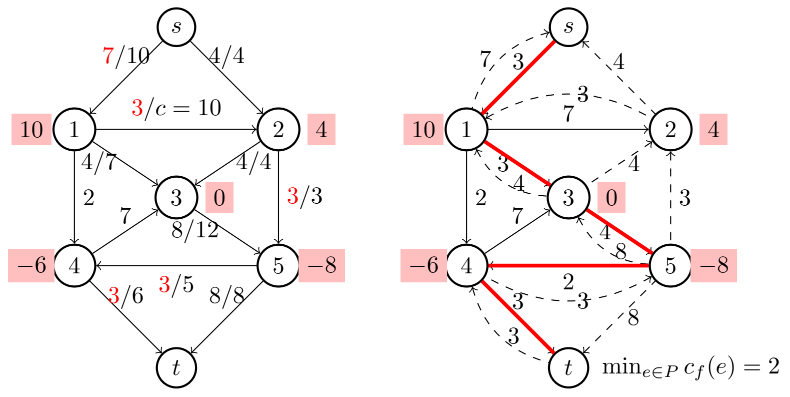
We start by sending a flow of 4 in the residual graph of the above graph, we update the original graph accordingly, and compute its residual graph:

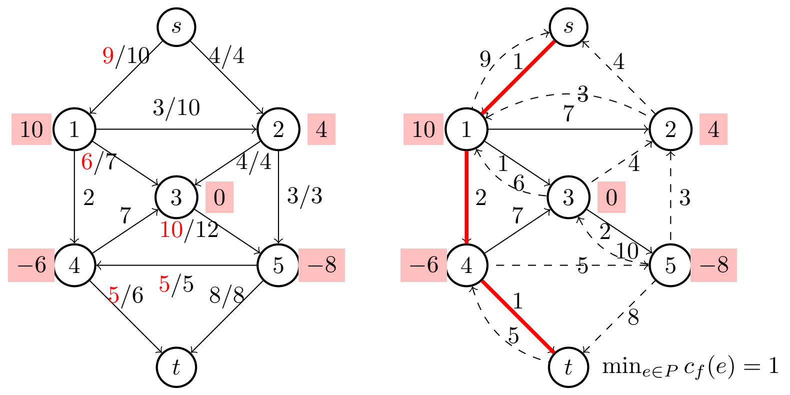




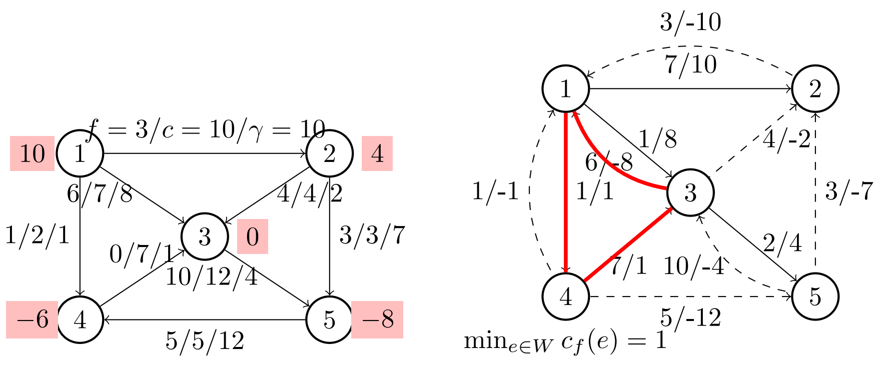


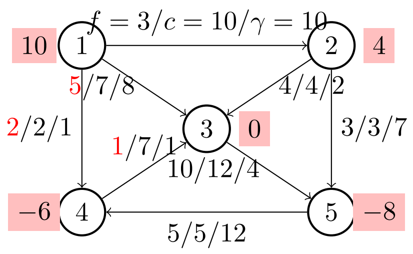


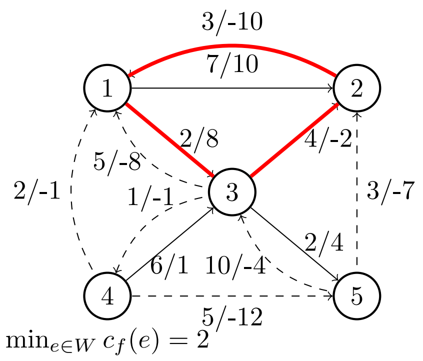


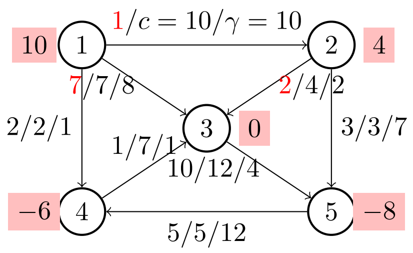


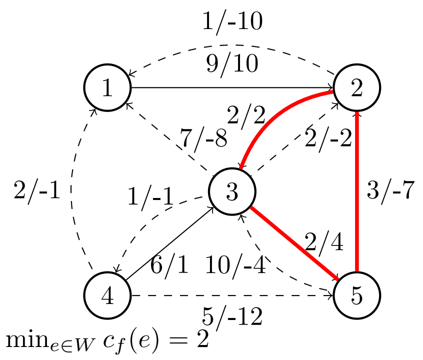
Updating the flow by 1 in the original graph gives an optimal solution to the max flow min cut, which saturates the source edges. Therefore we get a feasible solution for the min cost flow problem, which is displayed below, together with its residual graph, in which we look for a negative cost cycle W.

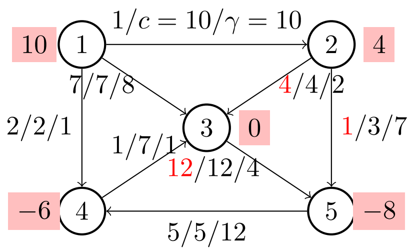


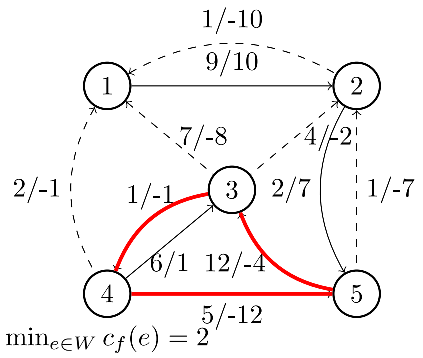




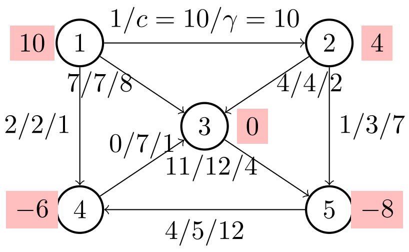








We update the flow once more to get



whose residual network has no negative cycle left, therefore its flow is optimal, for a cost of $10 + 56 + 2 + 8 + 0 + 48 + 44 + 7 = 168$.

In order to check that the solution is indeed optimal, it is also possible to use the LP view point. The current solution has one edge with zero flow, $\mathcal{N} = \{(4,3)\}$, three edges that saturate $\mathcal{S} = \{(1,3),(2,3),(1,4)\}$ and $\mathcal{B} = \{(1,2),(2,5),(3,5),(5,4)\}$. From

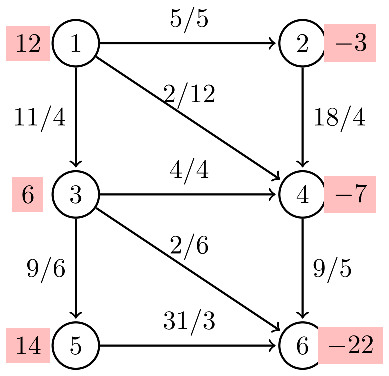
$$\begin{aligned} 10 - u_1 + u_2 &= 0\\ 7 - u_2 + u_5 &= 0\\ 4 - u_3 + u_5 &= 0\\ 12 - u_5 + u_4 &= 0 \end{aligned}$$

with setting $u_5 = 0$, we get $u_4 = -12,\ u_3 = 4,\ u_2 = 7,\ u_1 = 17$. Then

$$\begin{aligned} u_4 - u_3 &\le 1 && \Rightarrow -12 - 4 \le 1 \checkmark\\ u_1 - u_3 &\ge 8 && \Rightarrow 17 - 4 \ge 8 \checkmark\\ u_2 - u_3 &\ge 2 && \Rightarrow 7 - 4 \ge 2 \checkmark\\ u_1 - u_4 &\ge 1 && \Rightarrow 17 + 12 \ge 1 \checkmark \end{aligned}$$

which proves optimality of the flow found. We may observe that both methods give the same solution.

Exercise 48. Consider the following min cost flow problem, where every edge e is given a capacity $c(e)$ and a cost $\gamma(e)$.



Compute an optimal solution using the network simplex algorithm. Compare the algorithm and the solution with Exercise 33.

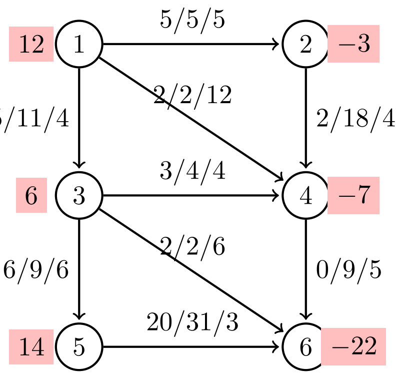
Solution 48. To find an initial solution, we partition the edges into

$$\mathcal{N} = \{(4,6)\},\ \mathcal{S} = \{(1,2),(1,4),(3,6)\},\ \mathcal{B} = \{(1,3),(2,4),(3,4),(3,5),(5,6)\}$$

with the following flows:

$$x_{12} = 5, x_{13} = 5, x_{14} = 2, x_{24} = 2, x_{34} = 3, x_{35} = 6, x_{36} = 2, x_{46} = 0, x_{56} = 20.$$

This flow was obtained intuitively, that is, by starting at node 1, and see possible flows outgoing this edge, and iterating this process. It could also be solved algorithmically, using artificial variables, but for small examples solved by hand, it is typically less effort to rely on intuition.

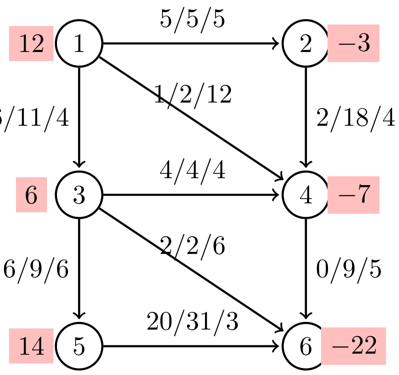


We check for optimality. We have $\gamma_{ij} - u_i + u_j = 0$ for $(i, j) \in \mathcal{B}$.

$$\begin{aligned}
4 - u_1 + u_3 &= 0\\
4 - u_3 + u_4 &= 0\\
6 - u_3 + u_5 &= 0\\
3 - u_5 + u_6 &= 0\\
4 - u_2 + u_4 &= 0
\end{aligned}$$

with setting $u_3 = 0$, we get $u_1 = 4,\ u_4 = -4,\ u_5 = -6,\ u_6 = -9,\ u_2 = 0$. Then $u_1 - u_4 \geq 12$ does not hold, since $8 \not\geq 12$. We thus need to update the edge $(1, 4)$. It is saturated, so we can decrease it by either 1 or 2. Then doing so will result in an increase by 1 or 2 in the cycle $(1) \to (4) \leftarrow (3) \leftarrow (1)$, but on $(3, 4)$ we can only increase by 1. So we update x_{34} from 3 to 4. Then $x_{14} = 1$, $x_{13} = 6$ and

$$\mathcal{N} = \{(4, 6)\},\ \mathcal{S} = \{(1, 2), (3, 4), (3, 6)\},\ \mathcal{B} = \{(1, 3), (2, 4), (1, 4), (3, 5), (5, 6)\}.$$



We check for optimality. We have $\gamma_{ij} - u_i + u_j = 0$ for $(i,j) \in \mathcal{B}$.

$$\begin{aligned} 4 - u_1 + u_3 &= 0 \\ 12 - u_1 + u_4 &= 0 \\ 6 - u_3 + u_5 &= 0 \\ 3 - u_5 + u_6 &= 0 \\ 4 - u_2 + u_4 &= 0 \end{aligned}$$

with setting $u_3 = 0$, we get $u_1 = 4,\ u_4 = -8,\ u_5 = -6,\ u_6 = -9,\ u_2 = -4$. Then

$$\begin{aligned} u_4 - u_6 &\leq 5 && \Rightarrow -8 + 9 \leq 5 \checkmark \\ u_1 - u_2 &\geq 5 && \Rightarrow 4 + 4 \geq 5 \checkmark \\ u_3 - u_4 &\geq 4 && \Rightarrow 8 \geq 2 \checkmark \\ u_3 - u_6 &\geq 6 && \Rightarrow 9 \geq 1 \checkmark \end{aligned}$$

which proves optimality of the flow found.

7.6 Semi-definite Programming

Exercise 49. If $X \succeq 0$ and if $x_{ii} = 0$, then $x_{ij} = x_{ji} = 0$ for all $j = 1, \ldots, n$.

Solution 49. We have $X = QDQ^T$. Then let Q_i be the ith row of Q. We have $Q_i D Q_i^T = x_{ii}$, and $Q_i \text{diag}(d_1, \ldots, d_n) Q_i^T = (q_{i1}d_1, q_{i2}d_2, \ldots, q_{in}d_n) Q_i^T = \sum_{j=1}^n q_{ij}^2 d_j$. So for $\sum_{j=1}^n q_{ij}^2 d_j$ to be zero, knowing that $X \succeq 0$ and so $d_i \geq 0$ for all i, we need $q_{ij}^2 d_j = 0$ for all j, and thus either $d_j = 0$ or $q_{ij} = 0$. So the vector $(q_{i1}d_1, q_{i2}d_2, \ldots, q_{in}d_n)$ is in fact the whole zero vector, and we have $x_{ij} = Q_i D Q_j^T = (q_{i1}d_1, q_{i2}d_2, \ldots, q_{in}d_n) Q_j^T = 0$.

Exercise 50. Show that if A is positive semi-definite, then all its diagonal coefficients are non-negative.

Solution 50. Suppose by contradiction that A has a diagonal coefficient which is negative, that is $a_{ii} < 0$. Then take the standard vector e_i with zero everywhere but a 1 in the ith position. Then $e_i^T A e_i = a_{ii} < 0$, a contradiction to A being positive semi-definite.

Exercise 51. Compute the dual of the following SDP:

$$\begin{array}{ll} \min & y_1 \\ s.t. & \begin{bmatrix} 0 & y_1 & 0 \\ y_1 & y_2 & 0 \\ 0 & 0 & y_1 + 1 \end{bmatrix} \succeq 0. \end{array}$$

Solution 51. We have

$$\begin{bmatrix} 0 & y_1 & 0 \\ y_1 & y_2 & 0 \\ 0 & 0 & y_1 + 1 \end{bmatrix} = y_1 \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} + y_2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \succeq \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

with $b_1 = 1, b_2 = 0$, so the dual is

$$\max \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \bullet X$$

with constraints given by

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet X = 1, \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \bullet X = 0$$

and $X \succeq 0$. This simplifies to

$$\begin{aligned} \max \quad & -x_{33} \\ s.t. \quad & x_{12} + x_{21} + x_{33} = 1 \\ & x_{22} = 0 \\ & X \succeq 0. \end{aligned}$$

Exercise 52. Given symmetric matrices B and A_i, $i = 1, \dots, k$, consider the problem of minimizing the difference between the largest and the smallest eigenvalue of $S = B - \sum_{i=1}^{k} w_i A_i$. Show that this problem can be formulated as a SDP.

Solution 52. Denote by $\lambda_{\max}(S), \lambda_{\min(S)}$ the largest and smallest eigenvalues of S. Then the problem considered is

$$\begin{aligned} \min_{w,S} \quad & \lambda_{\max}(S) - \lambda_{\min}(S) \\ & S = B - B - \textstyle\sum_{i=1}^{k} w_i A_i. \end{aligned}$$

Since S is symmetric, it can be written as $S = QDQ^T$ where Q is orthonormal and D is diagonal, containing the eigenvalues of S. The conditions

$$\lambda I \preceq S \preceq \mu I$$

can be written

$$Q(\lambda I)Q^T \preceq QDQ^T \preceq Q(\mu I)Q^T$$

by noting that $Q\lambda IQ^T = \lambda QQ^T = \lambda I$ since λI is a scalar matrix, and Q is orthonormal. Then we multiply the above by Q^T (on the left) and Q (on the right) to obtain

$$\lambda I \preceq D \preceq \mu I.$$

This means $D - \lambda I \succeq 0$ and $\mu I - D \succeq 0$, where all the matrices involved are diagonal, with positive coefficients. Thus

$$\lambda_{\min}(S) - \lambda \geq 0, \ \mu - \lambda_{\max}(S) \geq 0.$$

Therefore our minimization problem can be rewritten as

$$\begin{array}{cc} \min\limits_{w,S,\mu,\lambda} & \mu - \lambda \\ & S = B - B - \sum_{i=1}^{k} w_i A_i \\ & \lambda I \preceq S \preceq \mu I \end{array}$$

which is a SDP.

Index

Bibliography

[1] CMSC 251. Lecture 24: Floyd-warshall algorithm. http://www.cs.umd.edu/~meesh/351/mount/lectures/lect24-floyd-warshall.pdf.

[2] A. Agnetis. Min-cost ow problems and network simplex algorithm. http://www.dii.unisi.it/~agnetis/simpretENG.pdf.

[3] Algorithms and COS 226 Data Structures. Min cost flow. https://www.cs.princeton.edu/courses/archive/spring03/cs226/lectures/mincost.4up.pdf.

[4] C. Burfield. Floyd-warshall algorithm. http://math.mit.edu/~rothvoss/18.304.1PM/Presentations/1-Chandler-18.304lecture1.pdf.

[5] T. S. Ferguson. Linear programming. https://www.math.ucla.edu/~tom/LP.pdf.

[6] R. M. Freund. Introduction to semidefinite programming. https://pdfs.semanticscholar.org/b6dd/dbdadc5f8a0dffe31b28367892c4622492cf.pdf.

[7] The Fuzzy-Neural Group. Lecture 4: Simplex method. https://www.ise.ncsu.edu/fuzzy-neural/wp-content/uploads/sites/9/2017/08/505Lecture-4.pdf.

[8] A. Gupta and R. O'Donnell. Semidefinite duality. https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/notes/lecture12.pdf.

[9] CS 4820 Introduction to Algorithms. The edmonds-karp max-flow algorithm. http://www.cs.cornell.edu/courses/cs4820/2011sp/handouts/edmondskarp.pdf.

[10] Amaury Pouly. Min cost flows. http://perso.ens-lyon.fr/eric.thierry/Graphes2010/amaury-pouly.pdf.

[11] AM 00121: Operations research: Deterministic Model. Simplex method for linear programming. `http://www.dam.brown.edu/people/huiwang/classes/am121/Archive/simplex_121_c.pdf`.

[12] G. Royle. Pólya counting i. `http://teaching.csse.uwa.edu.au/units/CITS7209/polya.pdf`.

[13] Y. Ye. Dual interpretations and duality applications. `https://web.stanford.edu/class/msande310/lecture05.pdf`.