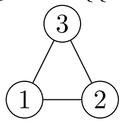
Chapter 2

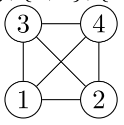
Basic Graph Theory

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 $\{u, v\}$.

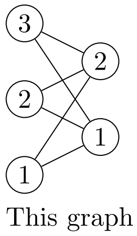
Example 2.1. The graph $K_n = (\{1, \dots, n\}, \{\{i, j\},\ 1 \leq i < j \leq n\})$ is called a *complete graph*. The term "complete" captures the fact that every vertex is connected to every other vertex. For $n = 3$, K_3 has $\{1, 2, 3\}$ for vertices, and $\{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$ for edges.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

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

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 $V = \{1, \dots, n\}$. 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.

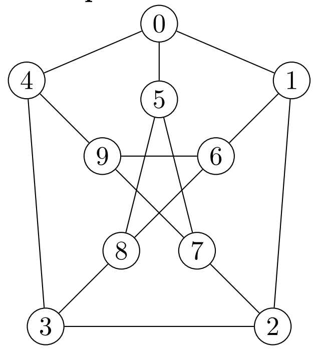


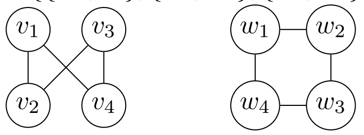
Figure 2.1: Julius Petersen (1839-1910), a Danish mathematician whose contributions to mathematics led to the development of graph theory.

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:

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

which gives the 4 edges of E' as desired.

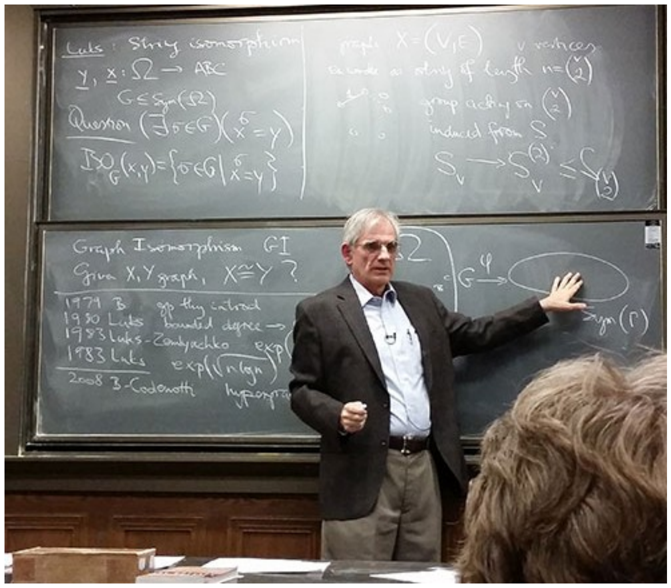
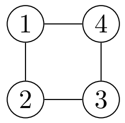


Figure 2.2: László Babai (born in 1950), a Hungarian professor of computer science and mathematics at the University of Chicago, claimed on January 9 2017 to have a quasi-polynomial algorithm with running time $2^{O((\log n)^c)}$, for n the number of vertices and $c > 0$ a fixed constant.

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.

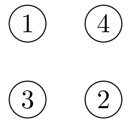
Example 2.6. Compute the automorphism group $\text{Aut}(G)$ of the graph G given by:



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.

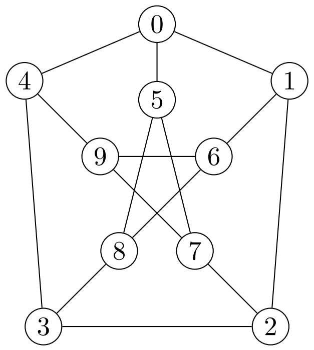
Now that we get a grip on what is going on, let us look at the second column. If the vertices 2 and 1 are switched, then we have no choice than switching 3 and 4, this is 2143. Then when 2 goes in position 1, apart switching 1 and 2, the other scenario that can happen is that 1 goes to 4, in which case,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

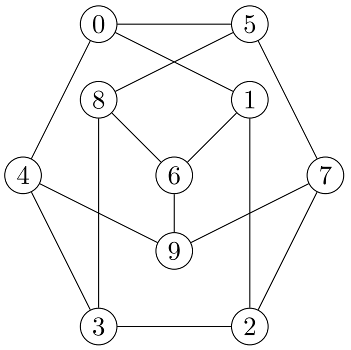
Applying the same reasoning to the other two columns gives the following list of group automorphisms: $1234, 1432, 2143, 2341, 3214, 3412, 4123, 4321$.

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 $\text{Aut}(G)$ 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.

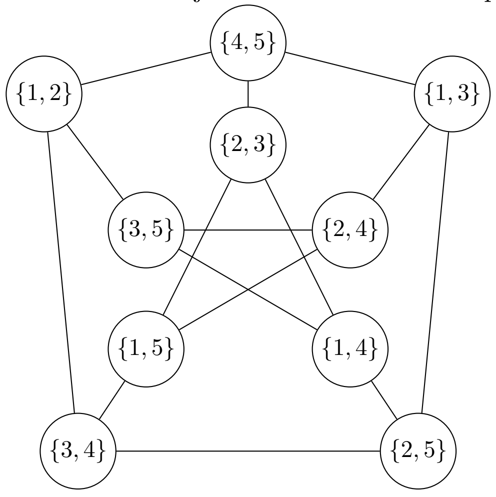
Next, let us compute the automorphism group $\mathrm{Aut}(G)$ for G the Petersen graph. To do so, let us construct a labelling of each of its $n = 10$ vertices, where each label is a pair $\{x, y\}$ with $x, y \in \{1, 2, 3, 4, 5\}$. Note that if we choose any 2 distinct numbers out of $\{1, 2, 3, 4,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

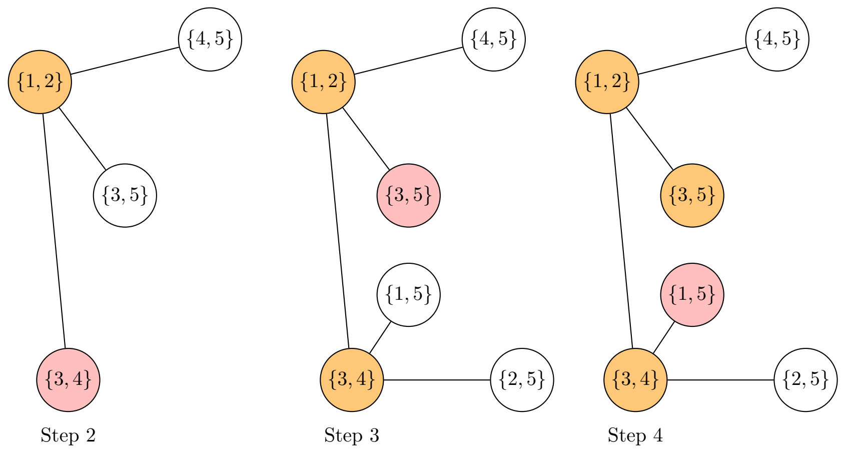
Remark. We can also repeat the Orbit-Stabilizer Theorem computation when S_5 acts on v_1. In that case, the orbit is the same and $|\mathrm{Orb}(v_1)| = 10$. For computing $\mathrm{Stab}_{S_5}(v_1)$, we need to count the permutations that are sending v_1 to itself, and since v_1 is connected to $\{4,5\}$,$\{3,4\}$ 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

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

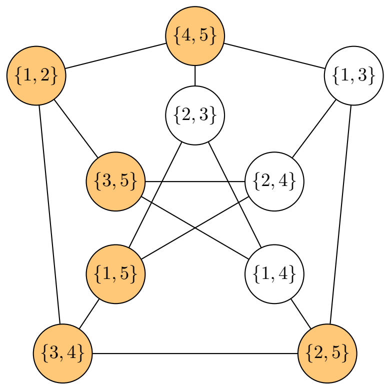


Step 3. We repeat the process once more. To compute $|\text{Stab}_{\text{Stab}_{\text{Aut}(G)}(v_1)}(v_2)|$, we take a 3rd vertex $v_3 = \{3, 5\}$, and let the group $\text{Stab}_{\text{Stab}_{\text{Aut}(G)}(v_1)}(v_2)$ act on it. The notation is not very friendly, but it just says that among the automorphisms 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.

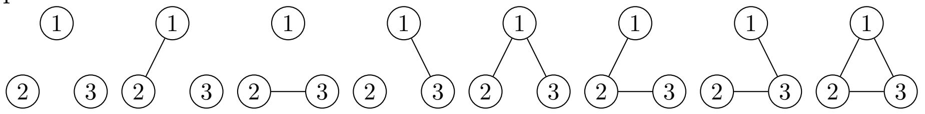
Step 4. We repeat the process one last time with $v_4 = \{1, 5\}$. Once v_1, v_2, v_3 are fixed, the orbit of v_4 contains only 2 elements, v_4 and $\{2, 5\}$, which give

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

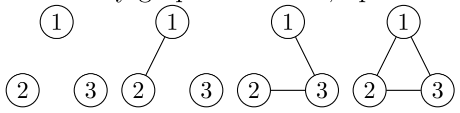
The process is finished, because: by now, $\{1, 2\}$, $\{3, 5\}$, $\{3, 4\}$ are fixed. Then $\{4, 5\}$ is fixed because it is connected to $\{1, 2\}$ whose 2 other adjacent nodes are fixed. Then we should be looking at automorphisms further fixing $\{1, 5\}$, which means that $\{2, 5\}$ is fixed, since 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:

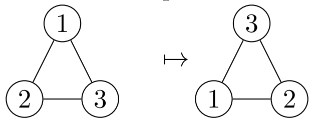


How many graphs are there, up to isomorphism? We have 4 of them:



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:

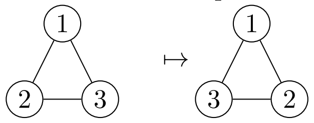


This permutation of vertices induces a permutation on the edges, because

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

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.

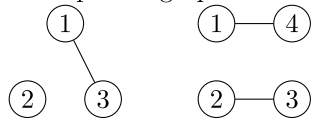
Definition 2.8. A *cycle* is a trail $(v_1, \ldots, v_n)$ such that $v_1 = v_n$.

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 $d(u, v) := \infty$.

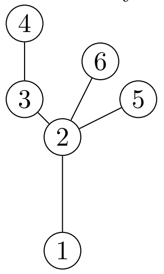
Definition 2.11. A graph is *connected* if $d(u, v) < \infty$ for all $u, v \in V$.

Examples of graphs which are not connected are:

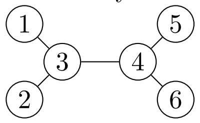


Definition 2.12. A *tree* is a connected graph that contains no cycle.

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

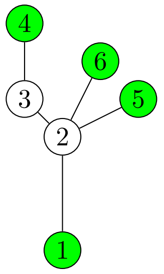


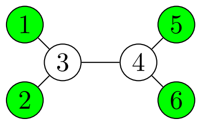
but it may also look different:



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.*

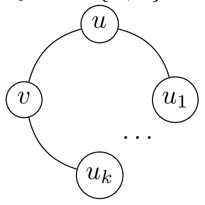
Proof. Let v be a leaf and $G' = G\backslash\{v\}$. We want to show that G' is connected and has no cycle.

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 $\{u, u_1, u_2, \dots, u_k, v\}$:



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,

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

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. □

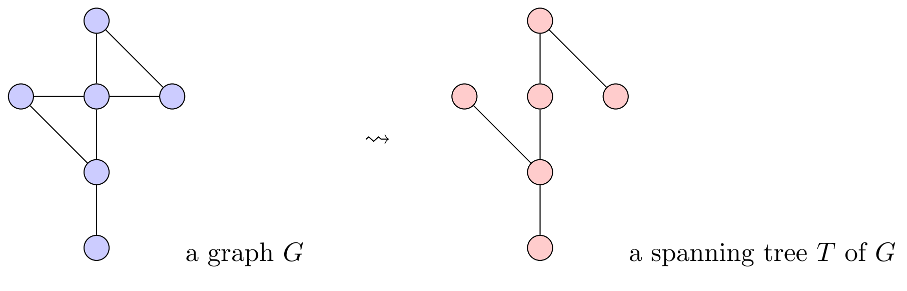
2.4 Minimum Spanning Trees

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, $V = \{1, 2, \ldots, n\}$
        $w : E \to \mathbb{R}_{\geq 0}$ 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, $w(e) \le w(e')$.

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 $T \cup \{e\}$. □

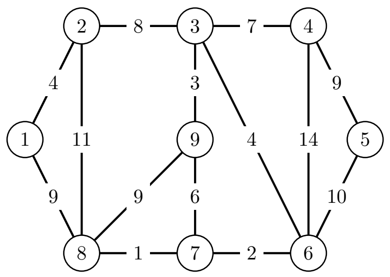
Another algorithm to compute the MST of a graph is Kruskal's algorithm.

Algorithm 2 Kruskal's algorithm

```
    Input:  $G = (V, E)$ a connected graph, $V = \{1, 2, \ldots, n\}$
            $w : E \to \mathbb{R}_{\ge 0}$ its weight function
    Output: edge set $T$ of a minimum spanning tree
 1: Sort all edges such that $w(e_1) \le \cdots \le w(e_m)$.
 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 $T$
```

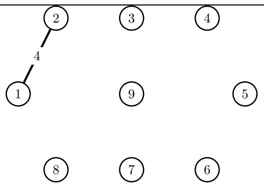
This algorithm can actually be seen as a particular case of the MinTree algorithm. In the MinTree algorithm, Kruskal's algorithm always chooses i such that there is an edge $e = \{u, v\}$, $u \in V_i$, $v \notin V_i$ such that e has minimum weight among all edges which do not have both vertices inside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

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

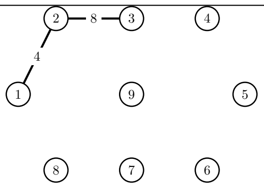


Using Prim's algorithm, the progression of T is as follows.

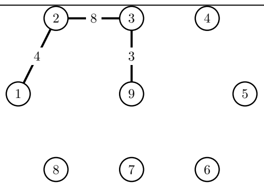
1. Choose $V_1 = \{1\}$, and $e = \{u, v\}$ with $u \in V_1$ and v outside, with minimal weight. Between weights 4 and 9, choose 4.



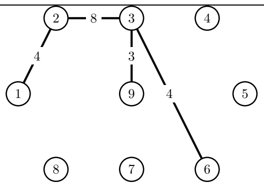
2. Then $V_1 = \{1, 2\}$ and $T = \{\{1, 2\}\}$. To pick the next edge, the choices for the weights are $9, 11, 8$, so we pick 8.



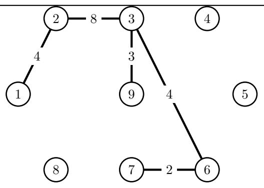
3. Then $V_1 = \{1, 2, 3\}$, $T = \{\{1, 2\}, \{2, 3\}\}$. For the next edge, we have weights 9, 11, 3, 4, and 7. So we choose 3.



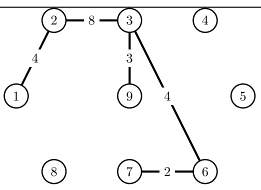
4. Then $V_1 = \{1, 2, 3, 9\}$, $T = \{\{1, 2\}, \{2, 3\}, \{3, 9\}\}$ and for the next edge, we can choose among weights 9, 11, 4, 7, and 6 so we pick 4.



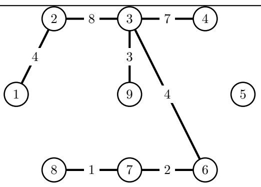
5. Then $V_1 = \{1, 2, 3, 9, 6\}$, $T = \{\{1, 2\}, \{2, 3\}, \{3, 9\}, \{3, 6\}\}$. The weights for the next edge are 9, 11, 7, 6, 2, 14 and 10, so we pick 2.



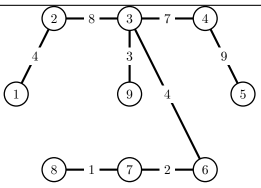
6. Then $V_1 = \{1, 2, 3, 9, 6, 7\}$, $T = \{\{1, 2\}, \{2, 3\}, \{3, 9\}, \{3, 6\}, \{6, 7\}\}$. The weights are now 9, 11, 7, 6, 14, 10, and 1. So we pick 1.



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.

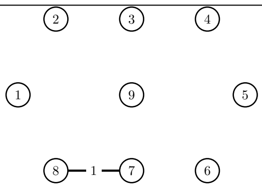


8. Then $V_1 = \{1, 2, 3, 9, 6, 7, 8, 4\}$, $T = \{\{1, 2\}, \{2, 3\}, \{3, 9\}, \{6, 9\}, \{6, 7\}, \{7, 8\}, \{3, 4\}\}$. The weights are now 9, 11, 14, 10, so we pick 9. This adds $\{4, 5\}$ to T and the algorithm stops since we have 8 edges.

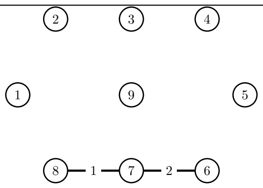


To use Kruskal's algorithm, we first sort the edges by weight: $w(\{7, 8\}) = 1$, $w(\{6, 7\}) = 2$, $w(\{3, 9\}) = 3$, $w(\{1, 2\}) = 4$, $w(\{3, 6\}) = 4$, $w(\{7, 9\}) = 6$, $w(\{3, 4\}) = 7$, $w(\{2, 3\}) = 8$, $w(\{1, 8\}) = 9$, $w(\{4, 5\}) = 9$, $w(\{8, 9\}) = 9$, $w(\{2, 8\}) = 11$, $w(\{4, 6\})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

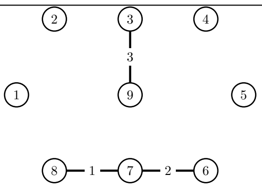
1.



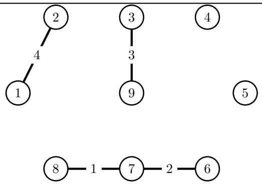
2.



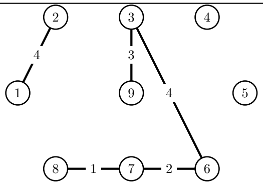
3.



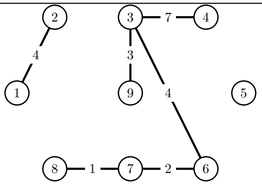
4.



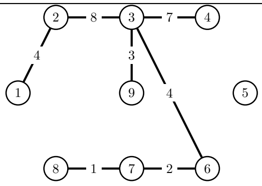
5.



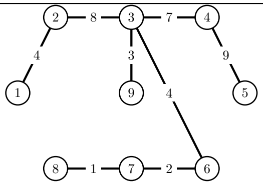
6.

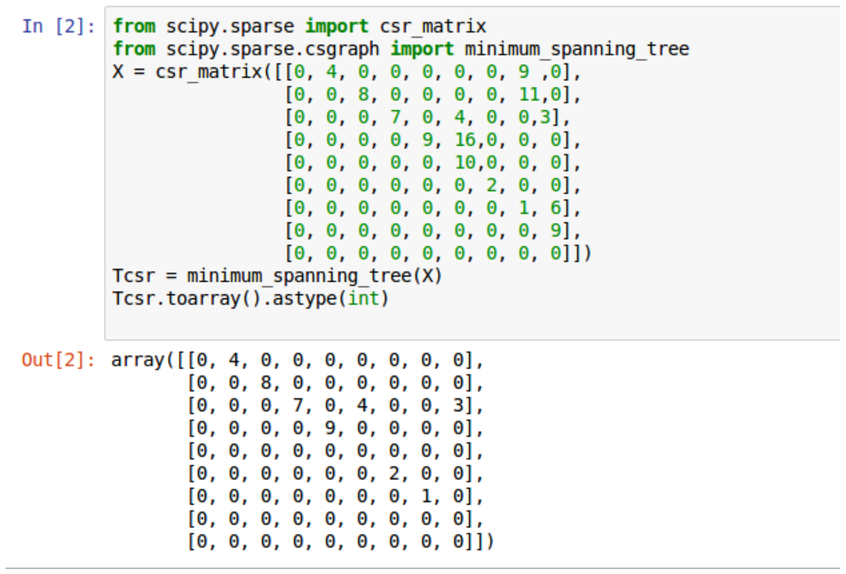


7.



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.

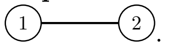
Minimum spanning tree algorithms can be used as part of more complicated algorithms, but they have also applications of their own. Here is an example of application to clustering. Suppose that you have n items, and you know the distance between each pair of items. You would like to cluster them into 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.

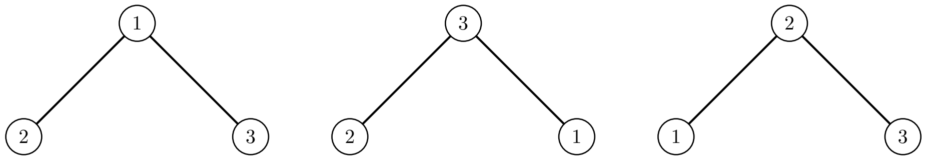
2.5 Labeled Trees

We next study trees whose n vertices are labeled from 1 to n. Formally, let $l : V \to \{1, \dots, n\}$, $v \mapsto l(v)$ 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.

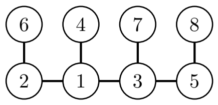
To be able to count the number of labeled trees (up to isomorphism), we will use a tree labeling called Prüfer's code, or Prüfer'sequence.

Algorithm 3 Algorithm for Prüfer's Code

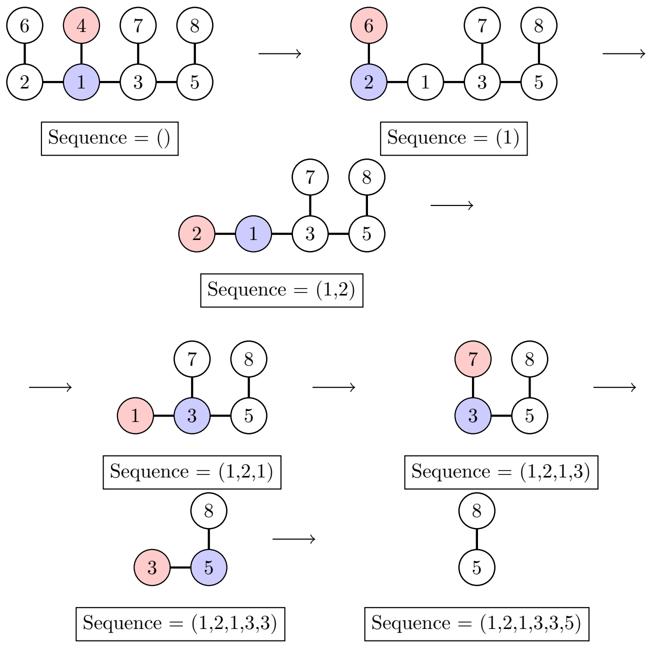
```
1: Input: A tree $T$ with vertex set $V$, $|V| = n \geq 2$.
2: Output: A sequence $(a_1, \cdots, a_{n-2}) \in V^{n-2}$.
3: Set $T_1 = T$.
4: for $i = 1, 2, \dots, n-2$ do
5:     Let $v$ be the leaf of $T_i$ with the smallest label.
6:     Set $a_i$ to be the unique neighbour of $v$ in $T_i$.
7:     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}.*

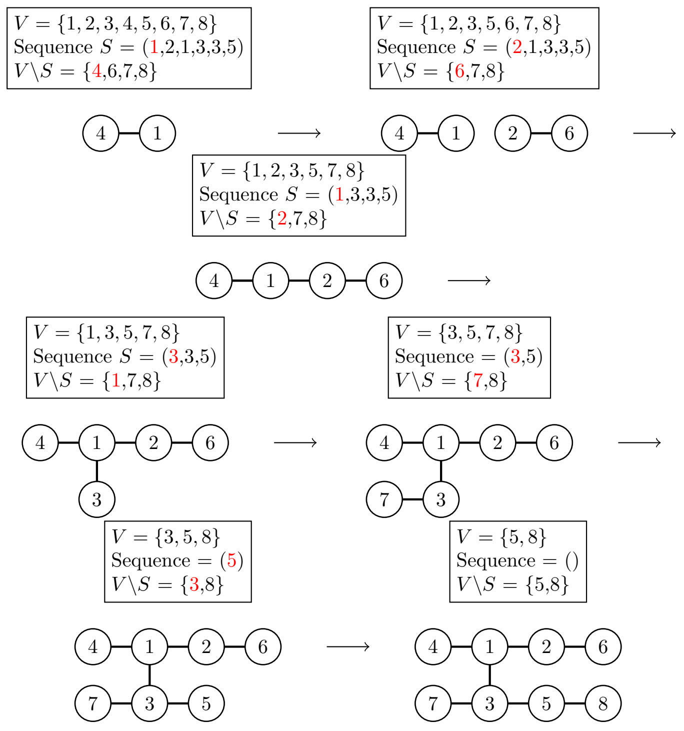
Proof. Let $|V| = n$. The preceding algorithm gives a function f which takes a tree T with n vertices and outputs $f(T) = (a_1, \cdots, a_{n-2})$. To show that f is a bijection, we need to show that every sequence $(a_1, \cdots, a_{n-2})$ defines uniquely a tree.

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 $f(T) = (a_1, \cdots, a_{n-2})$.

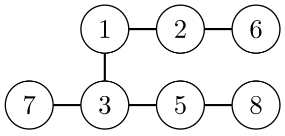
None of the a_i is a leaf in T since when a vertex is set to be a_i, it is adjacent to a leaf (when the graph is left with only leaves, then it has two vertices and the algorithm terminates) and $V \backslash \{a_1, \ldots, a_{n-2}\}$ contains only nodes that are leaves.

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. □

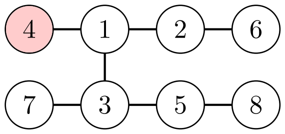
Example 2.12. Compute the tree corresponding to the Prüfer's code (1,2,1,3,3,5). We computed this Prüfer's code in the previous example, so if all works as it should, we should get back the same labeled tree. We observe that since the code is of length 6, we are looking for a tree with $n = 8$ vertices.



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



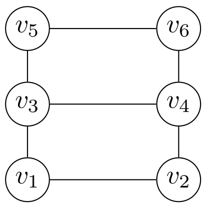
So when considering the sequence $(a_1, a_2, \ldots, a_{n-2}) = (1, 2, 1, 3, 3, 5)$, we know that $a_1 = 1$ has for neighbour a leaf v with label 4, thus we append the edge $\{v, a_1\}$ to T' to get the tree T:



2.6 Exercises

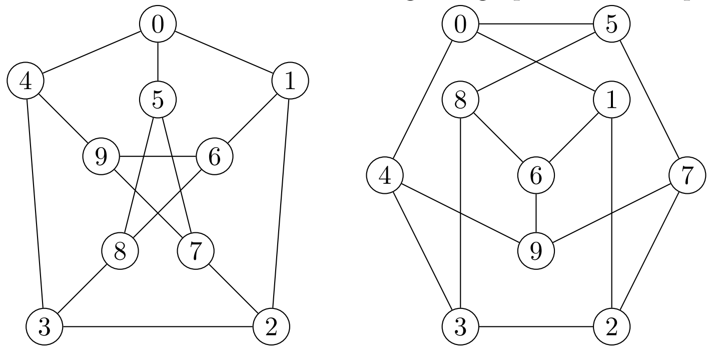
Exercise 10. Prove that the set of all automorphisms of a graph forms a group.

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

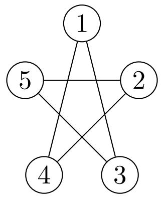


Exercise 12. Compute the automorphism group of the n-cycle graph, the graph given by n vertices $1, \ldots, n$, and n edges given by $\{i, i+1\}$ where $i, i+1$ are understood modulo n.

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.

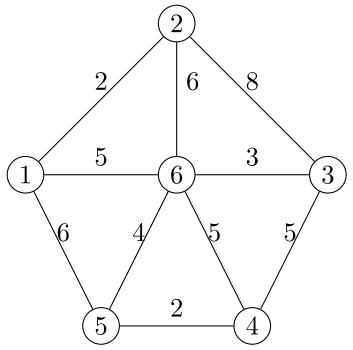
Exercise 16. Count, using Pólya's Enumeration Theorem, the number of isomorphism classes of graphs with 4 vertices, and count how many there are for each possible number of edges.

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.

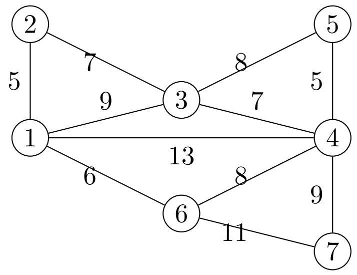
Exercise 18. Prove that adding one edge to a tree creates exactly one cycle.

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.

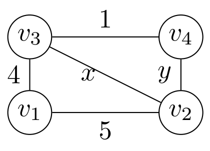


2. Construct, if possible, a connected weighted graph G with two minimum spanning trees.

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. (*)

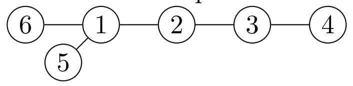
1. Consider the following weighted undirected graph, where x, y are unknown integers.



   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.