6. Graphs

6.1. Graphs

As we noticed before, a directed graph with vertex set A and edge set R is nothing else than a binary relation R on a set A. An ordinary graph with vertex set A has as edges subsets of size two from A. A symmetric and irreflexive relation R is often identified with the ordinary graph with edges $\{a, b\}$ where $(a, b) \in R$.

Many questions about relations can be phrased in terms of graphs. In this section we will discuss some of them. Although many of the definitions and result can also be stated for directed graphs, we will restrict our attention to ordinary graphs. We start with introducing some graph theoretical notation.

Let $\Gamma = (V, E)$ be an ordinary graph with vertex set V and edge set E. A *walk* in Γ is a sequence $(v_1, \dots, v_n)$ in V such that $\{v_i, v_{i+1}\} \in E$. A *path* in Γ is a sequence $(v_1, \dots, v_n)$ in V such that $\{v_i, v_{i+1}\} \in E$ and $v_i \neq v_{i+2}$. The *length* of the path $(v_1, \dots, v_n)$ equals $n - 1$. The point v_1 is the *starting point* of the path and v_n is the *end point.* A *shortest path* from a to b is a path from a to b of minimal length. The *distance* between two points equals the length of a shortest path between them. If there is no such path, the distance is set to infinity. The graph Γ is called *connected* whenever for every two vertices a and b of Γ there is a path with starting point a and end point b. A *cycle* is a path $(v_1, \dots, v_n)$ with the same starting and end point so, $v_1 = v_n$.

The degree $\deg(v)$ of a vertex $v \in V$ equals the cardinality of the set $\Gamma_v = \{w \in V \mid \{v, w\} \in E\}$. A graph is called *regular* if all its vertices have the same degree, it is called k-regular if all its vertices have degree k.

A connected graph without cycles is called a *tree.* A tree contains vertices of degree 1. These special vertices are called the *leafs* of the tree.

If $\Gamma = (V, E)$ is a graph the the complement $\overline{\Gamma}$ of Γ is the graph with the same vertex set as Γ but with two vertices adjacent if and only if they are not adjacent in Γ. So, if $\binom{V}{2}$ denotes sthe set of all subsets of V of size 2, then $\overline{\Gamma} = (V, \binom{V}{2} \setminus E)$.

6.2. Some special graphs

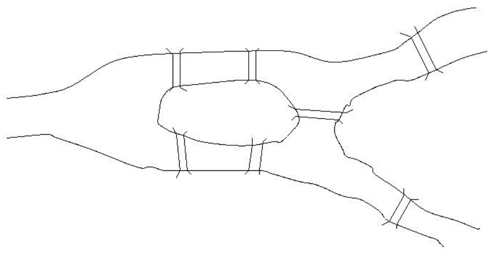
A *complete* graph is a graph in which any two vertices are adjacent. By K_n we denote the complete graph with n vertices.

A graph $\Gamma = (V, E)$ is called *completely bipartite* if its vertex set V can be partitioned into two subsets V_1 and V_2 such that any edge $e \in E$ meets both V_1 and V_2 in a vertex. The complete bipartite graph with $|V_1| = m$ and $|V_2| = n$ will be denoted by $K_{m,n}$.

If $\Gamma = (V, E)$ is a graph, then its *line graph* is the graph whose vertices are the edges of Γ and two of such edges are adjacent if and only if they meet in a vertex of Γ.

6.3. Euler and Hamilton Cycles

Example 6.3.1 One of most famous problems in graph theory is the problem of the bridges of Königsberg (now called Kaliningrad). The Preusian city of Königsberg was divided into 4 parts by a river, one of these parts being an island in the river. The 4 regions of the city were connected by 6 bridges. On sunny Sundays the people of Königsberg used to go walking along the river and over the bridges. The question is, is there a walk using all the bridges once, that brings the pedestrian back to its starting point. The problem was solved by Leonard Euler, who showed that such a walk is not possible.



A cycle in a graph is called an *Euler cycle* if it contains all edges of the graph once.

The following result is due to Euler.

Theorem 6.3.2 *A finite connected graph contains an Euler cycle if and only if all vertices have even degree.*

Proof. Suppose Γ is a graph admitting an Euler cycle E. Suppose v is a vertex of the graph. Then inside the Euler cycle E, we see v just as often as an end point of an edge from E as a starting point. Hence the number of edges on v is even.

Now suppose all vertices of the finite graph Γ have even degree. We start a path of Γ in the vertex v. Each time we arrive in a vertex u, there are only an odd number of edges on u in the path. Except when we arrive back in v. If the path P_1 constructed in this way is not an Euler cycle yet, there is at least one edge, e say, in Γ not visited yet. We may even assume, by connectedness of Γ, that this edge contains a vertex w from P_1. Now we make a path P_2 starting in w containing e. As soon as, in the process of constructing this path, we hit on a vertex of P_1, then, as we only have met an odd number of edges on this vertex, there has to be an edge not in P_1 and not yet part of P_2. So, the path P_2 may be constructed in such a way that it has no edges in common with P_1. The paths P_1 and P_2 can be combined to a path P (with more edges than P_1) in the following way. Start in v, follow the path P_1 until one reaches w, then follow the path P_2 completely, and finally continue from w over the path P_1 to end in v. Repeating this process will eventually yield an Euler cycle in Γ. $\square$

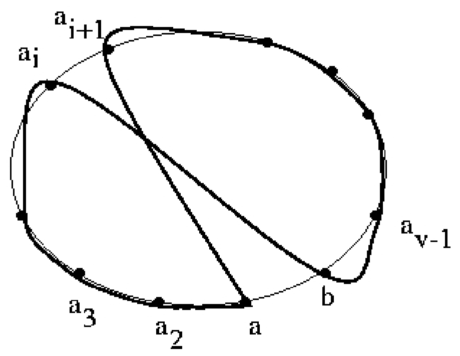
Notice that the above not only proves the theorem, but also describes an algorithm to find an Euler cycle.

A *Hamilton cycle* in a graph Γ is a cycle in the graph containing every vertex exactly ones. It does not seem to be possible to given a characterization of graphs admitting Hamilton cycles, similar to Theorem 6.6.2. However, the following result due to Ore, gives a partial result in this direction.

Theorem 6.3.3 *If in the finite graph Γ on v vertices for every two nonadjacent vertices a and b we have $\deg(a) + \deg(b) \geq v$, then Γ contains a Hamilton cycle.*

Proof. Suppose Γ is a finite graph on v vertices and for any two nonadjacent vertices a and b we have $\deg(a) + \deg(b) \geq v$. Suppose Γ contains no Hamilton cycle. We add edges to Γ as long as possible, but we avoid producing a Hamilton cycle. Since the complete graph on v vertices admits a Hamilton cycle, we end up with a graph Γ_0 in which there is no Hamilton cycle, but addition of any new edges produces one. Notice that, also in Γ_0, we still have that for any two nonadjacent vertices a and b the sum $\deg(a) + \deg(b)$ is greater or equal than v.

Now suppose a, b are nonadjacent vertices. Then after adding the edge $\{a, b\}$ to Γ_0 we obtain a Hamilton cycle $a = a_0, a_1, \ldots, a_v = b$ in Γ_0. Since $\deg(a) + \deg(b) \geq v$, there are two vertices a_i and a_{i+1} with b adjacent to a_i and a adjacent to a_{i+1}. However, then consider the path $a = a_1, \ldots, a_i, b = a_v, a_{v-1}, \ldots, a_{i+1}, a$ in Γ_0. This is a Hamilton cycle and contradicts our assumption on Γ_0. Thus Γ has to have a Hamilton cycle.$\square$



6.4. Spanning Trees and Search Algorithms

Example 6.4.1 A search engine at the university wants to search all the web servers on campus. These servers are connected by cables within an intranet. To do its search the search engine does not want to put too much load on the connections between the different web servers. In fact, it wants to use as as few connections as possible. If we represent the computer network as a graph with the web servers (and search engine) as vertices, two vertices connected, if and only if there is a cable connecting them, then we can phrase the problem as follows: find a connected subgraph on all the vertices with as few edges as possible.

Trees are graphs with the least number of vertices as follows from the following theorem.

Theorem 6.4.2 *Let Γ be a finite connected graph on v vertices. Then Γ contains at least $v-1$ edges, with equality if and only if Γ is a tree.*

Proof. Notice that in a connected graph each vertex is on at least one edge. We prove this result with induction to the number v of vertices in the graph.

Clearly, for $v = 1$ the result is true. Now suppose $v > 1$ and suppose Γ is a tree. Then removing a leaf of Γ, together with the unique edge on this leaf, yields a tree Γ' with $v-1$ vertices. By induction Γ' contains $v-2$ edges. Thus Γ contains $v-1$ edges.

Now suppose Γ contains $\leq v-1$ edges. Since $|E| \cdot 2 = 2 \cdot (v-1) = \Sigma_{v \in V} \deg(v)$, we find at least one vertex, x say, of degree 1. Removing this vertex and the unique edge on it from Γ yields a connected graph Γ' with $v-1$ vertices and $\leq v-2$ edges. By induction, Γ' is a tree with $v-2$ edges. But then, since the vertex x has degree 1 in Γ, we also find Γ to be a tree with $v-1$ edges. □

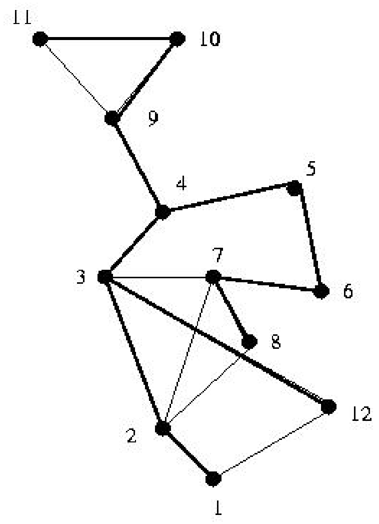
A *spanning tree* of a graph Γ is a subgraph of Γ which is a tree, containing all vertices and some edges of Γ. So the problem of Example 6.4.1 is equivalent to finding a spanning tree in a graph. Now we describe two search algorithms that in fact construct spanning trees.

Algorithm 6.4.3 [Depth First Search and Breadth First Search] Consider a finite connected graph Γ. Fix a vertex v of Γ and label it by $\mathsf{Label} := 1$. The vertex v will be the root of a spanning tree in Γ. We construct this tree now as follows.

While there is a labeled vertex that has a neighbor not in the tree, find the vertex, say w with the highest label that has neighbors not in the tree. Add the edge on w and one of its neighbors not yet in the tree to the tree, and label this neighbor by $\mathsf{Label} := \mathsf{Label} + 1$.

This algorithm is called "Depth First Search". "Breadth First Search" is a similar algorithm. However, here the while loop reads a little bit differently:

While there is a labeled vertex that has an unlabeled neighbor, find the vertex, say w with the smallest label that has neighbors not in the tree. Add the edge on w and one of its neighbors not yet in the tree to the tree, and label this neighbor by $\mathsf{Label} := \mathsf{Label} + 1$.



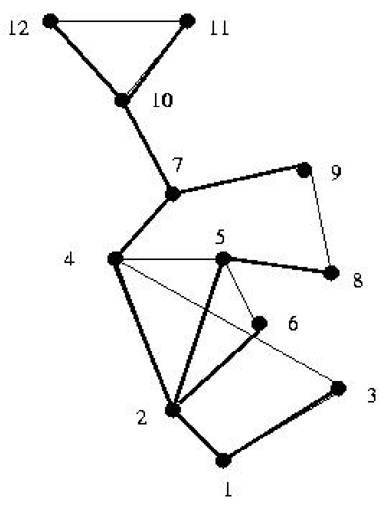
Clearly both algorithms stop after at most $|\Gamma|$ steps and do indeed yield a tree. Actually they both construct a spanning tree. To see this we have to show that every vertex of the graph gets a label. Suppose not, then by connectivity of the graph, there is a path starting with a labeled vertex w and ending with an unlabeled vertex u. walking through this path we will find a labeled vertex adjacent to an unlabeled one. However, this means the while-loop is not finished yet.

Actually the above algorithms imply the following theorem.

Theorem 6.4.4 *Let Γ be a finite connected graph, then Γ contains a spanning tree.*

6.5. Networks

Example 6.5.1 Consider the set A of (big) cities in the Netherlands. As we have seen in 1.5.5, the relation "there is a direct train connection between a and b" defines a binary relation R on this set. The transitive closure of this relation tells us whether we can travel by train from one city to another. It does not tell us what the best or fastest way to travel is. Therefore we need some extra information on the relation. For example, we do not only want to know that there is a train connection from Eindhoven to Tilburg, but we also want to know how long it takes the train to get form Eindhoven to Tilburg. So, for each $(a, b) \in R$ we want to know the time it costs to travel from a to b. Such extra information can be encode by a so-called "cost function". Having this information it is natural to ask for the fastest connection between two cities, i.e., to find the shortest path from a to b.



A graph $\Gamma = (V, E)$ together with a *cost function* cost on the set E of edges, i.e., a map from E to (usually)$\mathbb{R}$ (or any other set) is called a *network*. In such a network the length of a path $(v_1, \ldots, v_n)$ equals to sum $\Sigma_{i=1}^{n-1}\mathsf{cost}(\{v_i, v_{i+1}\})$. The following algorithm due to Dijkstra describes a way to find a shortest path in a network.

Algorithm 6.5.2 [Dijkstra's shortest path algorithm] Suppose $\Gamma = (V, E)$ is a finite connected graph with cost function $\mathsf{cost} : E \to \mathbb{R}^+$ a positive valued cost function. Given two elements s and t of V, a shortest (i.e., of minimal cost) path from s to t can be found as in the following algorithm.

In the process of the algorithm we will keep track of the sets Done and Remainder and some partial maps DefiniteDistance, EstimatedDistance and Predecessor.

Initiate the algorithm by setting Done $:= \{s\}$, Remainder $:= V \setminus$ Done. The distances are set by DefiniteDistance$(s) := 0$ and EstimatedDistance$(r) := \infty$ for all $r \in$ Remainder.

While Remainder contains the element t, determine for all elements r in Remainder the EstimatedDistance(r), being the minimum of DefiniteDistance(d) + cost(d, r), where d runs through the set of elements $n \in$ Done with $\{n, r\} \in E$. Moreover, set Predecessor(r) to be one of the elements d for which this minimum is attained.

For those elements $r \in$ Remainder for which EstimatedDistance(r) is minimal, we can decide that their distance to s is actually equal to EstimatedDistance(r). So, for those r we set DefiniteDistance(r) = EstimatedDistance(r), and we remove these points from Remainder and add them to Done.

Since in each step of the While-loop at least one element is added to Done, the algorithm will terminate and we will find the minimal length DefiniteDistance(t) of a path from s to t. Moreover, with the help of $\mathsf{Predecessor}$ we even can find a path realizing this minimal length.

Sometimes one is not interested in finding a shortest path in a network, but a longest. Notice that this question only makes sense if we are dealing with a network without cycles, or more in particular, when we are dealing with a *directed network* without cycles. The following variation on Dijkstra's shortest path algorithm yields a solution to that problem.

Algorithm 6.5.3 [Dijkstra's longest path algorithm] First we notice that we may assume that there are no cycles in the network. For otherwise the problem does not make sense.

Again we set $\mathsf{Done} := \{s\}$, $\mathsf{Remainder} := V \setminus \mathsf{Done}$. The distances, however, are set by $\mathsf{DefiniteDistance}(s) := 0$ and $\mathsf{EstimatedDistance}(r) := 0$ for all $r \in \mathsf{Remainder}$.

While $\mathsf{Remainder}$ contains the element t, determine for all elements r in $\mathsf{Remainder}$ which can only be reached by an edge starting in Done the $\mathsf{DefiniteDistance}(r)$, being the maximum $\mathsf{DefiniteDistance}(d)+\mathsf{cost}(d,r)$, where d runs through the set of elements $n \in \mathsf{Done}$ with $\{n,r\} \in E$. Moreover, set $\mathsf{Predecessor}(r)$ to be one of the elements d for which this maximum is attained.

We remove these points from $\mathsf{Remainder}$ and add them to Done.

Since in each step of the While-loop at least one element is added to Done, then algorithm will terminate, and we will find the maximal length $\mathsf{DefiniteDistance}(t)$ of a path from s to t. Moreover, with the help of $\mathsf{Predecessor}$ we even can find a path realizing this maximal length.

Algorithm 6.5.4 [Kruskal's Greedy Algorithm] Suppose Γ is a network. To find a minimal spanning tree, we first sort the edges with respect to their weight (or cost). We start with the minimal edge and each time we add an edge of minimal weight to the graph which does not close a cycle. The resulting subgraph will be a spanning tree of minimal weight.

Algorithm 6.5.5 [Prim's Algorithm] In Kruskal's Greedy Algorithm 6.5.4 we first have to sort the edges. The following algorithm avoid this. Start with a vertex v and put it into the set T. At this vertex choose a neighbor outside T on an edge with minimal weight. Add the neighbor and edge to the subgraph T. Now choose a minimal edge among the edges with a vertex in T and one outside T a vertex outside T and add it, together with its end points to T. Repeat this procedure till we have all vertices in T.

Proposition 6.5.6 *Kruskal's Greedy Algorithm and Prim's Algorithm provide us with a minimal spanning tree.*

Proof. Let T_m be the tree obtained after m steps in Kruskal's or Prim's algorithm. With induction on m we prove that there exists a minimal spanning tree T containing T_m. In particular, taking m to be the number of vertices in Γ, we find that the algorithms indeed yield a minimal spanning tree.

For $m = 1$ the graph T_m is just one vertex and hence contained in any minimal spanning tree. Suppose $T_m \subseteq T$ for some minimal spanning tree T. Then in the next step of the algorithm we add an edge $\{v, w\}$ to T_m to obtain T_{m+1}. If $\{v, w\}$ is also an edge of T, then clearly $T_{m+1} \subseteq T$. Thus assume T does not contain $\{v, w\}$. This implies that $\{v, w\}$ with some vertices of T forms a cycle C. Inside that cycle there is an edge $\{v', w'\} \neq \{v, w\}$ with $v' \in T_m$ but $w' \notin T_m$.

By the choice of $\{v, w\}$ in the algorithm, we have $\mathsf{cost}(\{v, w\}) \leq \mathsf{cost}(\{v', w'\}$. Hence, replacing the edge $\{v', w'\}$ in T by $\{v, w\}$ we obtain again a spanning tree, T' say, with $\mathsf{cost}(T) \geq \mathsf{cost}(T')$. In particular, T' is also a minimal spanning tree containing T_{m+1}.

□

6.6. Planar Graphs

A graph is called *planar* if one can draw the graph in the plane (on paper) such that no two edges cross. If a graph is planar, then it is possible to draw it in many different ways. Such a drawing is called a *map* of the graph.

Suppose Γ is a finite planar graph mapped into the plane. Then the following two constructions lead to a new graph also mapped into the plane.

- Add a new vertex and connect it to a vertex of Γ via an edge not crossing any edge of the map.
- Connect two non adjacent vertices of Γ by an edge not crossing any edge of the map.

Actually it is not hard to see that any planar map of a finite graph can be obtained by the appplying the above procedure $|E|$ times, (where E is the edges set of the graph) starting from a graph with a single vertex.

A map of a graph divides the plane in various different *regions*. The degree of a region is the number of edges at its border. As each edge of the graph borders exactly two regions we find the following formula for a finite graph $\Gamma = (V, E)$.

Lemma 6.6.1 *Let $\Gamma = (V, E)$ be a finite planar graph mapped into the plane. Then*

$$\sum_{\mathrm{r} \in \mathrm{R}} \mathrm{degree}(R) = 2|E|,$$

where R is the set of regions of the map of Γ in the plane.

The famous mathematician Euler proved the following formula.

Theorem 6.6.2 [Euler's Formula] $|V| - |E| + |R| = 2$.

Proof. We will prove the theorem using structural induction.

For the graph with just one vertex and no edges the results holds true.

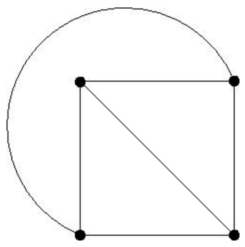
Now any planar map can be obtained from this graph with a single vertex by applying step 1 or 2 described above.

In step 1, no new region is createdm but both V and E get bigger by 1. So $|V| - |E| + |R|$ does not alter.

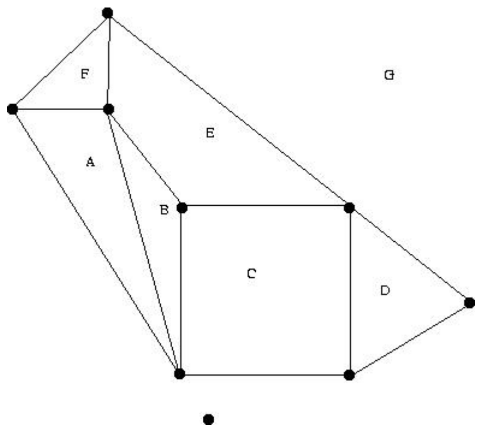
In step 2, $|V|$ remains the same, $|E|$ increases by 1 but also the number of regions increases by 1. Again $|V| - |E| + |R|$ does not alter.

Hence by structural induction, we have proved the theorem. □

Example 6.6.3 A map of the complete graph K_4 in the plane.



A planar graph with 8 vertices, 13 edges and 7 regions.



A consequence of Euler's formula is the following.

Proposition 6.6.4 *Suppose $\Gamma = (V, E)$ is a planar graph with at least 3 vertices. If every region of the graph is bounded by at least e edges, then $2 \leq |V| - (1 - 2/e)|E|$.*

In particular, if Γ is connected and contains at 3 vertices, then $|E| \leq 3|V| - 6$.

Proof. Let R the set of regions of a map of Γ into the plane. Then by 6.6.2 we have

$$|V| - |E| + |R| = 2.$$

As each region is bounded by at least e edges, 6.6.1 we find

$$2|E| \geq e|R|.$$

But then we find

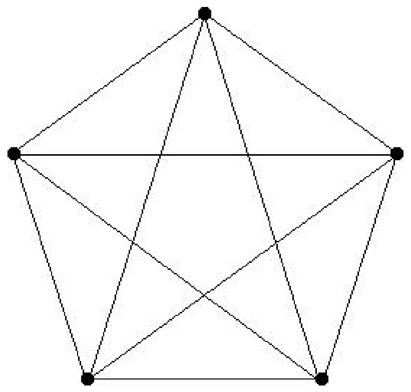
$$2 = |V| - |E| + |R| \leq |V| - (1 - 2/e)|E|,$$

which implies

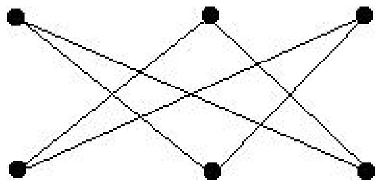
$$|E| + 2 \leq 3|V| - 6.$$

If Γ is connected and contains at 3 vertices, then every region is bounded by at least 3 edges, so we obtain $|E| \leq 3|V| - 6$. □

Example 6.6.5 The graph K_5 is not planar. Indeed, K_5 has 5 vertices and 10 edges, so $10 = |E| > 3|V| - 6 = 9$.



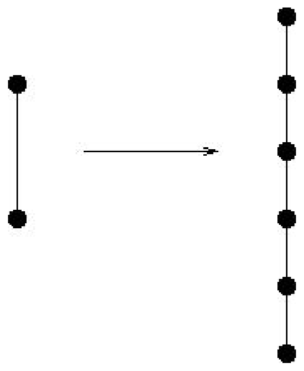
Consider the graph $K_{3,3}$. This graph has 6 vertices and 9 edges, so it does satisfy the conditions as given in the above proposition. However, $K_{3,3}$ is not planar. We will prove this by contradiction. So suppose $K_{3,3}$ is planar. Then by Euler's formula any map of $K_{3,3}$ has 5 regions. Since $K_{3,3}$ does not contain triangles, any region is bounded by at least 4 edges. So, we can apply the above proposition to obtain a contradiction.



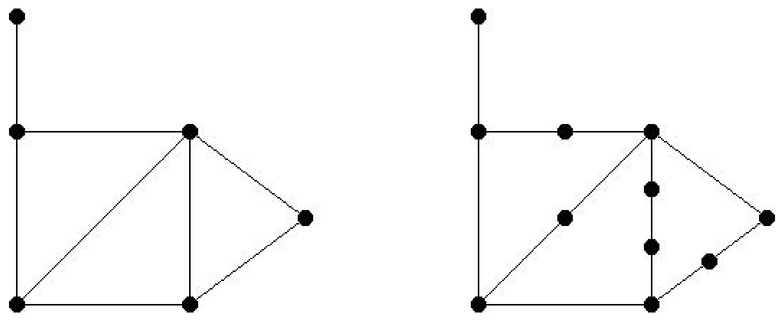
In 1930, the polish mathematician Kuratowski obtained the following beautiful characterization of planar graphs.

Theorem 6.6.6 [Kuratowski's Theorem] *A finite graph* Γ *is planar if and only if it contains no subgraph homeomorphic to* K_5 *or* $K_{3,3}$.

Two graphs are called *homeomorphic* if one can obtain one graph from the other, by replacing edges by strings.



Example 6.6.7 Below you see two homeomorphic graphs.



6.7. Graph Colorings

Consider a graph $\Gamma = (V, E)$. A coloring of Γ is an assigment of colors to the vertices of Γ such that two adjacent vertices have different colors. The minimal number of colors needed to color Γ is called the *chromatic number* of Γ.

One of the most famous results in graph theory is the following theorem due to Appel and Haken.

Theorem 6.7.1 [Four Color Theorem] *Any finite planar graph can be colored with at most* 4 *colors.*

6.8. Exercises

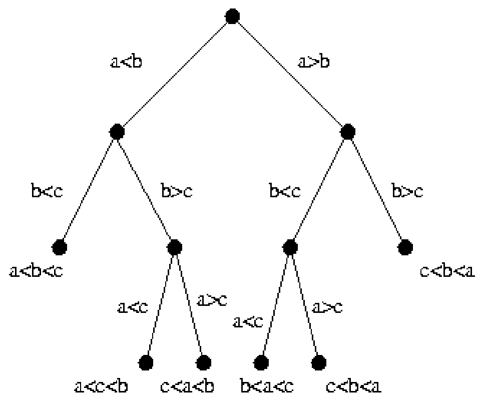
Exercise 6.8.1 A graph Γ is called regular of degree d if all its vertices have degree d. Show that in a finite regular graph $\Gamma = (V, E)$ of degree d we have

$$|V| \cdot d = 2 \cdot |E|.$$

Exercise 6.8.2 A tree Γ is called binary if all its vertices either have degree ≤ 3 or are leafs and one vertex, called the root and denoted by r, is a vertex of degree 2.

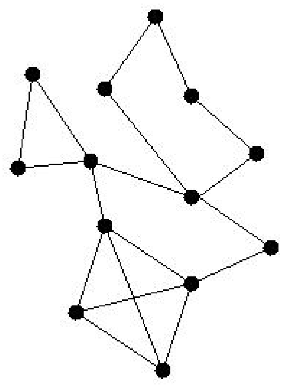
Let Γ be a finite binary tree. Fix a vertex r (called root) of Γ. By l we denote the number of leafs of Γ and by d the maximal distance of a vertex v in Γ to r. Prove that $l \leq 2^d$.

For sorting algorithms based on comparison of two elements we can make a decision tree. For example, for sorting the three elements $\{a, b, c\}$ the decision tree looks as follows:



How many leaves does such a decision tree have when sorting n elements? Suppose r is the starting point of the decision tree. Can you give a lower bound on d? What does this mean for the complexity of sorting algorithms based on comparison of two elements?

Exercise 6.8.3 Find spanning trees in the following graph using both "Depth first search" and "Breadth first search".



Exercise 6.8.4 Suppose T is a spanning tree in a finite connected graph Γ. If e is an edge of Γ which is not in T, then there is an edge d of T, such that replacing d by e in T yields again a spanning tree of Γ. Prove this.

Exercise 6.8.5 How can one use Theorem 6.6.2 to solve the problem of the Königsberger bridges, see 6.3.1?

Exercise 6.8.6 An *Euler path* is a path from a vertex a to a vertex b in a graph containing all edges of the graph just ones. Show that a finite connected graph Γ contains an Euler path from a to b if and only if a and b are the only vertices of odd degree in Γ.

Exercise 6.8.7 How does Theorem 6.6.2 generalize to directed graphs?

Exercise 6.8.8 Can one find an Euler cycle in the cube? And a Hamilton cycle?

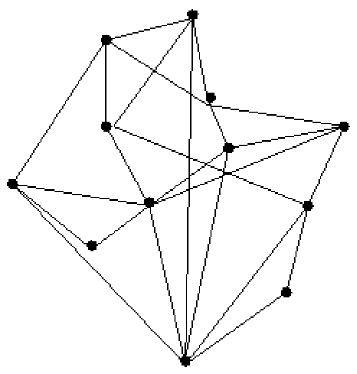
Exercise 6.8.9 Can one find an Euler cycle in the following graph? And a Hamilton cycle?

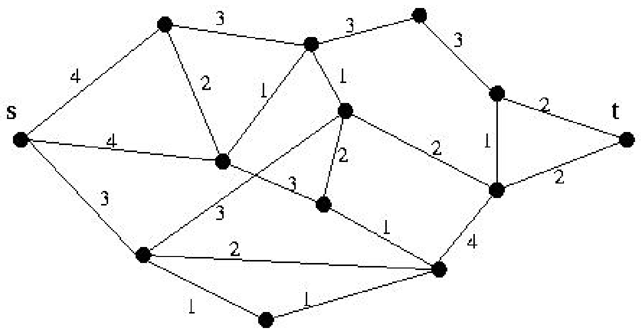
Exercise 6.8.10 Use Dijkstra's algorithm to find both the shortest path from s to t.

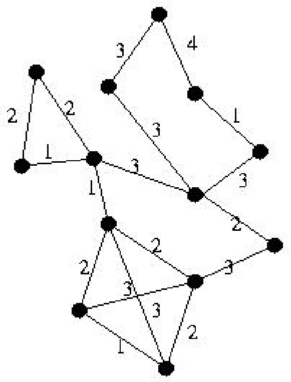
If we assume that one can only walk through the network from left to right, then what is the longest path from s to t?

Exercise 6.8.11 Apply both Kruskal's Greedy Algorithm and Prim's Algorithm to find a minimal spanning tree in the following network.

Exercise 6.8.12 The diameter of a graph is the maximal distance between any of its vertices. Describe an algorithm that determines the diameter of a finite connected graph.




s
t
4
3
3
3
2
1
1
4
2
1
2
3
3
3
2
1
2
4
2
1
1


3
4
2
2
1
1
3
1
3
3
2
2
2
3
3
3
1
2