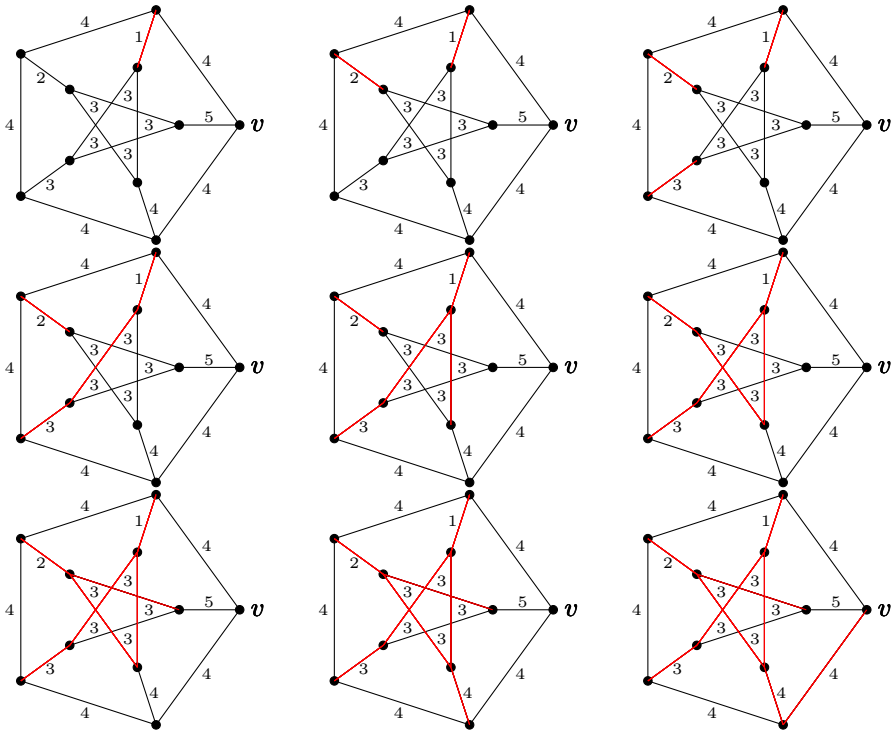
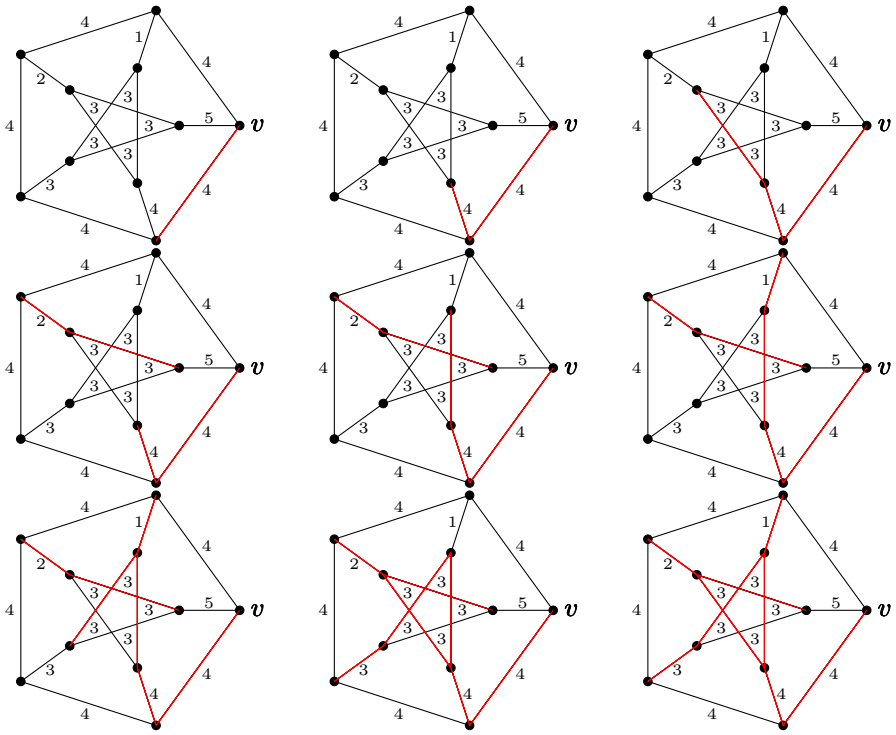
Solutions to Problem Sheet Graphs & Networks II

Exercise 1.

(a) A possible example execution is the following. The edges of our spanning tree will be marked in red. Note that the MST produced is **not** necessarily unique.



(b) One example application of Prim's algorithm that leads to the MST obtained in (a) is the following (it does not have to be the case in general):



Exercise 2. To a contradiction, assume that there exist two spanning trees of $G = (V, E)$ of minimum weigth, namely

$$T_1 = (V, E_1) \text{ and } T_2 = (V, E_2), \text{ with } E_1 \neq E_2 \text{ and } \sum_{e \in E_1} w(e) = \sum_{e \in E_2} w(e).$$

Consider now the *symmetric difference* of E_1 and E_2, denoted by

$$E_1 \triangle E_2 = (E_1 \setminus E_2) \cup (E_2 \setminus E_1).$$

Since all edge weights are unique in E, there is a unique edge $f \in E_1 \triangle E_2$ of minimal weight. Let us assume without loss of generality that $f \in E_1 \setminus E_2$, and consider the graph $T_2' = (V, E_2 \cup \{f\})$. Since T_2 is a tree, T_2' must contain a cycle involving the edge f. Since T_1 is also a tree and $f \in E_1$, this cycle of T_2' must contain at least one edge $f' \in E_2 \setminus E_1 \subset E_1 \triangle E_2$ such that $T_2'' = (V, E_2 \cup \{f\} \setminus \{f'\})$ is a spanning tree of G. Because f has minimal weigth in $E_1 \triangle E_2$, we have $w(f') > w(f)$. But this means that

$$\sum_{e \in E_2 \cup \{f\} \setminus \{f'\}} w(e) < \sum_{e \in E_2} w(e).$$

Thus T_2'' is a spanning tree of weight smaller than T_1 or T_2: contradiction. This implies that G contains a unique MST.

Exercise 3.

(a) We first prove the implication ($\Rightarrow$), or instead we will show that its *contrapositive*[1] is true. That is, we assume that there exists a proper subset $X \subset V$ such that for any pair of vertices $a \in X$ and $b \in V \setminus X$ there is no edge between a and b in G, and we will show that this implies that G is not connected. As X is a proper subset, both X and $V \setminus X$ are not empty. Thus we can select a pair of vertices $c \in X$ and $d \in V \setminus X$. Remark then that there can be no path in G connecting them, as any one of those paths would need to use an edge between X and $V \setminus X$.

We now prove ($\Leftarrow$) using the same method. Suppose that G is not connected. Then it contains a pair of vertices $u, v \in V$ without any path connecting them. Let us now define the subset $X_u \subset V$ as the set of vertices $w \in V$ connected to u by a path in G (X_u is in fact the connected component containing u). Notice that $X \neq \emptyset$ as $u \in X_u$, and that $X_u \neq V$ as $v \notin V$. This implies that X_u is a proper subset of V. Remark finally, that G cannot contain an edge between any pair of vertices $a \in X_u$ and $b \in V \setminus X_u$, or else, by concatenating a path between u and a, which exists because $a \in X$, with the edge between a and b, one obtains a path connecting u and b, thus contradicting $b \notin X$.

(b) We will explicit an algorithm that takes a connected graph and one of its vertices as argument and returns one of its spanning trees.

We will now discuss the correctness of this algorithm. Let us first remark that by (a), it is always possible to execute the subroutine "`Find edge` $\{a, b\} \in E$ `such that` $a \in V'$ `and` $b \in V \setminus V'$" of the `While` loop, given that V' is a proper subset of V. Furthermore, as G has n vertices and V' gains a vertex at each iteration of the loop, the loop will last $n - 1$ iterations. We now fix the following notations: $V_0 = \{r\}$, $E_0 = \emptyset$, and for $i \in \{1, \ldots, n\}$, if we are at the end of step i of the `While` loop then denote V' by V_i, E' by E_i and set $T_i = (V_i, E_i)$.

[1] See the Wikipedia entry https://en.wikipedia.org/wiki/Contraposition

procedure SPANNINGTREE(connected graph $G = (V, E)$, vertex $r \in V$)
  $V' \leftarrow \{r\}$
  $E' \leftarrow \emptyset$
  while $V' \neq V$ **do**
    Find edge $\{a, b\} \in E$ such that $a \in V'$ and $b \in V \setminus V'$
    $V' \leftarrow V' \cup \{b\}$
    $E' \leftarrow E' \cup \{\{a, b\}\}$
  end while
  return $T = (V', E')$
end procedure

So it is enough to prove that T_i is a spanning tree of $G[V_i]$, the subgraph of G induced by V_i. Remark that it is spanning by definiton as T_i and $G[V_i]$ have the same vertex set. We will then prove that it is also connected and acyclic, each time by an induction on i.

The *base case ($i = 0$)* can be the same in both proofs as $T_0 = (\{r\}, \emptyset)$ is considered a tree by convention.

- *Induction hypothesis*: For every $0 \leq i \leq n - 2$, T_i is connected.
  Let $a \in V_i$ and $b \in V \setminus V_i$ be the two vertices found by the subroutine such that $e = \{a, b\} \in E \setminus E_i$. Then $V_{i+1} = V_i \cup \{b\}$ and $E_{i+1} = E_i \cup \{e\}$. Let $u, v \in V_{i+1}$. If both u and v are in V_i (and are thus different from b), then by induction hypothesis there is a path in T_i from u to v. Else, consider $w \in V_i$ and we will build a path in T_{i+1} from w to b. If $w = a$, then the edge $e \in E_{i+1}$ forms such a path in T_{i+1}. If now $w \neq a$, then as both $w, a \in V_i$ we have a path $(e_1, e_2, \ldots, e_m)$ in T_i from w to a. Then the path $(e_1, e_2, \ldots, e_m, e)$ is a path in T_{i+1} from w to b.
- *Induction hypothesis*: For every $0 \leq i \leq n - 2$, T_i is acyclic.
  Let $a \in V_i$ and $b \in V \setminus V_i$ be the two vertices found by the subroutine such that $e = \{a, b\} \in E \setminus E_i$. And assume to a contradiction that T_{i+1} has a cycle C. By induction hypothesis T_i is acyclic, so C contains the vertex b and two edges $e_1 = \{b, a_1\}$ and $e_2 = \{b, a_2\}$ both incident to b. In particular, one of those two edges must be e. Suppose without loss of generality that $e_1 = e \in E_{i+1}$, i.e. $a_1 = a$. But this means that the edge $e_2 \in E_i$, and thus $b \in V_i$ which is impossible.

(c) The direction ($\Rightarrow$) follows from the correctness of the algorithm presented in (b).

To prove ($\Leftarrow$), consider a spanning tree T of G and let $u, v \in V$ be a pair of vertices of G. Because T is spanning, both $u, v \in V(T)$. And because T is a tree, it is connected. Therefore, there exists a unique path between u and v that uses edges in $E(T)$. But because T is a subgraph of G, we have $E(T) \subset E(G)$. Thus this path is also a path in G.

Exercise 4.

(a) Let $S \subset V(G)$ be a proper subset of the vertices of G. Since G is strongly connected, there is a directed path from any vertex in S to any vertex in $V(G) \setminus S$. Since S is a proper subset of $V(G)$, at least one such path exists. And there must be a directed edge (x, y) with $x \in S$ and $y \in V(G) \setminus S$ belonging to that path.

Using this observation, we can now build an arborescence $T = (S, E)$ of a strongly connected directed graph G iteratively, starting from a given vertex r, as follows.

procedure ARBORESCENCE(strongly connected directed graph G, vertex $r \in V(G)$)
  $S \leftarrow \{r\}$
  $E \leftarrow \emptyset$
  while $S \neq V(G)$ **do**
    Find edge $(a, b) \in E(G)$ such that $a \in S$ and $b \in V(G) \setminus S$
    $S \leftarrow S \cup \{b\}$
    $E \leftarrow E \cup \{(a, b)\}$
  end while
  return $T = (S, E)$
end procedure

In a manner analogue to the solution of Exercise 5(b) of Problems Sheet Graphs and Networks I, it can be proven that the above algorithm is correct, i.e. that the output is indeed an arborescence of G rooted at r.

(b) Note that an analogue of Prim's algorithm is essentially what we presented to prove part (a), except that instead of picking an arbitrary edge (x, y) with $x \in S$ and $y \in V(G) \setminus S$ at each step, we pick the one of lowest weight. Next, we exhibit a counter-example where this algorithm does not output an arborescence of minimum weight.

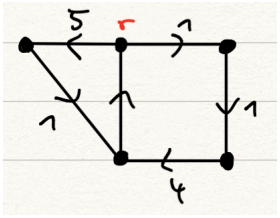


Figure 1: In this example, the algorithm outputs an arborescence from the vertex r of weight 11, while the optimal one has weight 8.

Exercise 5. Let $T = (V, E')$ be the MST of the edge-weighted complete graph $K_n = (V, E)$ build by Prim's algorithm. Since the depth-first search (DFS) will list every vertex $(v_1, v_2, \ldots, v_n)$ of T exactly once, and thus every vertex of K_n, the output of the algorithm H is a spanning subgraph of K_n. Edges of H will then be added incrementaly following the list $(v_1, v_2, \ldots, v_n)$, and will only contain edges of the form $\{v_i, v_{i+1}\}$ (where $i \leq 9$) picked first from T if it exists in E' then from $E \setminus E'$ otherwise. The edge $\{v_n, v_1\}$ is finally added to H so that it indeed is the cycle $(v_1, v_2, \ldots, v_n)$.

Next, we prove that the weight of $H = (V, E'')$ is at most twice the optimum. Suppose that at some step, the algorithm building H adds and edge $\{v_i, v_{i+1}\} \in E'' \setminus E'$. This means that during its execution, the DFS stopped at the vertex v_i, which is a leaf of T, and then backtracked to the vertex v_j (with $j < i$) such that $\{v_j, v_{j+1}\}, \{v_{j+1}, v_{j+2}\}, \ldots, \{v_{i-1}, v_i\} \in E'$, and also $\{v_j, v_{i+1}\} \in E'$. Now, because the weigthing w of the edges of K_n satisfies the triangular inequality we have

$$w(\{v_i, v_{i+1}\}) \leq w(\{v_j, v_{j+1}\}) + w(\{v_{j+1}, v_{j+2}\}) + \cdots + w(\{v_{i-1}, v_i\}).$$

Once the vertex v_{i+1} has been covered by H, the edges of T above cannot be added as further edges to H. To estimate the weigth of last edge $\{v_n, v_1\}$ of H, remark first that as it is a tree T admits a unique path P from v_1 to v_n. Second, the edges of P cannot have been used yet to

estimate the weight of the other edges of H as else this would contradict the fact that v_n is the last vertex of the DFS. All together, this means that when summing the weights of the edges in H, the edges of T are counted at most twice: once if they are in E'' and once to estimate a further edge in $E'' \setminus E'$. So that

$$w(H) \leq 2w(T).$$

Note finally that a MST of K_n is also a connected subgraph of K_n with minimum weight. In particular the weight of T will be smaller than that of any Hamiltonian cycle. Thus

$$2w(T) \leq 2\min\{w(C) : C \text{ is a Hamiltonian cycle of } K_n\}.$$

Exercise 6.

(a) 1. Suppose you have a subset $S \subseteq V(K_n)$ of at most $n-2$ vertices. Then at least two vertices $x, y \in V(K_n)$ are not contained in S, and the edge $\{x, y\} \in E(K_n)$ is not *covered* by S, i.e. incident with any vertex in S. This means that a vertex cover of K_n has size at least $n-1$.

   On the other hand, any subset $S \subseteq V(K_n)$ of size $n-1$ forms a vertex cover. Indeed, for any edge $e = \{x, y\} \in E(K_n)$ we must have either $x \in S$ or $y \in S$ since $|S| = n-1$. Thus e is covered by S, and the minimum vertex cover of K_n has size at most $n-1$.

2. Let $V(K_{n,m})$ be the union $L \cup R$ of two disjoint sets of vertices with $|L| = n$ and $|R| = m$. Notice that both L and R are vertex covers, since every edge contains exactly one vertex from L and one from R. Thus a minimum vertex cover of $K_{n,m}$ has size at most $\min\{n, m\}$.

   Suppose now that there exists a set of vertices $S \subseteq L \cup R$, such that $|S| < \min\{n, m\}$. Then there must be a pair of vertices $(a, b) \in L \times R$ such that neither a nor b is contained in S. But then the edge $\{a, b\} \in E(K_{n,m})$ will not covered by S. This means that the the minimum vertex cover of $K_{n,m}$ has size at least $\min\{n, m\}$.

3. Order the vertices of P_n following a walk from one leaf to the other one: $(v_1, v_2, \ldots, v_n)$. Then the sets $\{v_i : i \text{ is even}\} \subseteq V(P_n)$ and $\{v_i : i \text{ is odd}\} \subseteq V(P_n)$ both form a vertex cover of P_n, because every edge in $E(P_n)$ is of the form $\{v_i, v_{i+1}\}$, where $1 \leq i \leq n-1$. Thus a minimum vertex cover of P_n has size at most $\lfloor n/2 \rfloor$.

   To show that this is optimal, note that the maximum degree of P_n is 2, and hence $\lfloor n/2 \rfloor - 1$ vertices can cover at most

$$2(\lfloor n/2 \rfloor - 1) = \begin{cases} n-3, & \text{if } n \text{ is odd} \\ n-2, & \text{if } n \text{ is even} \end{cases}$$

   edges. But P_n has $n-1$ edges, so there must be uncovered ones.

(b) Take any edge $e \in E(G)$. Since the algorithm only stops when $E(G_i)$ is empty and we started with $E(G_0) = E(G)$, there must have been an iteration step i in which e was removed. But the only edges that are removed are those incident to x_i, the vertex added to our cover at that step, and hence $x_i \in e$, so the edge is covered.

For a counter-example, consider the path P_n where $n-1$ is divisible by 4. In particular, n is odd, so there is a center vertex with degree 2, say $v \in V(P_n)$. If v is chosen in the first step, then we are left with two disjoint paths, each of even length $(n-1)/2$. From $(a.3)$, we know that any vertex cover of $P_{(n-1)/2}$ has size at least $(n-1)/4$. Thus a minimum vertex cover of P_n in this case would require $2 \cdot (n-1)/4$ vertices to cover the edges of the two disjoint paths, plus the vertex v to cover its two incident edges. So the output of the algorithn will have size at least $1 + (n-1)/2$, which is one more than the optimal.