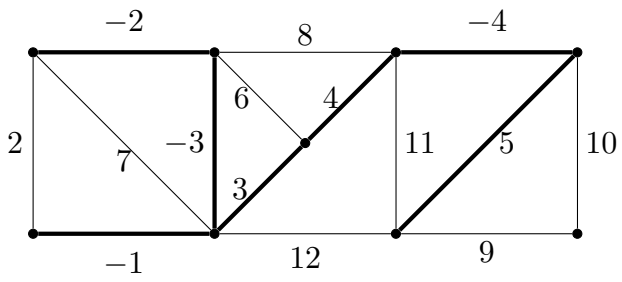
Graph theory - solutions to problem set 13

Exercises and problems

1. Find a minimum spanning tree in the following graph, once using the Tree-Growing Algorithm and once using the Forest-Growing Algorithm.

   Solution: Both algorithms give the same solution (because of the next problem):



2. Prove that if a weighted graph has all weights distinct, then it has a unique minimum spanning tree.

   Solution: Suppose w is injective and there are two distinct MSTs T and T^*. Take the minimum edge that is in exactly one of the two trees, say $e \in T \backslash T^*$, and consider $T^* + e$. It must contain a cycle, which must contain an edge $f \in T^* \backslash T$. Now $w(e) \neq w(f)$, and by the assumption that e is minimum we have $w(e) < w(f)$. Then $T^* + e - f$ is a spanning tree with $w(T^* + e - f) < w(T^*)$, contradiction.

3. Prove that the Forest-Growing Algorithm from the lecture correctly returns a minimum spanning tree.

   Solution: Call a forest *good* if it is contained in an MST. The empty graph that the algorithm starts with is clearly good. We will show inductively that all the forests occuring in the algorithm are good, hence also the last forest. Because the last forest satisfies $|E(F)| = |V(G)| - 1$, it must be a spanning tree, and then it is minimum because it is good.
   Let F be a forest in step 2, and suppose it is good, so contained in an MST T^*. We want to show that $F + e$ is good, if $w(e)$ is minimum in S and $F + e$ does not have a cycle. If $e \in T^*$, then clearly $F + e$ is good, so assume $e \notin T^*$. Then adding e to T^* must create a cycle, which must contain an edge f not in F, such that its endpoints are not in the same component of F (since e does not create a cycle in F, its endpoints are in different components, so the vertices of the cycle cannot all be in the same component).
   This implies that f is still in S: if not, then the algorithm must have previously considered f, but rejected it because it would have created a cycle in a predecessor of F. But this contradicts the fact that its endpoints are in different components of F. Since $f \in S$ the algorithm must have chosen e over f, so $w(f) \geq w(e)$. Then $T^{**} = T^* - f + e$ is a minimum spanning tree containing F and e, which means $F + e$ is good. This proves that the final forest is good, hence an MST.

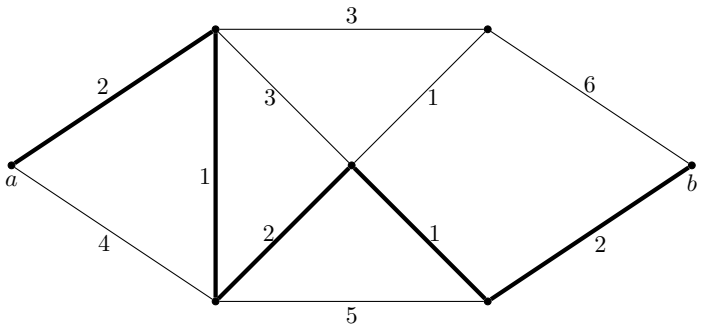
4. Give a "pruning" algorithm that finds a minimum spanning tree by starting with the whole graph and removing edges while preserving connectedness. You do not have to prove that it works correctly, just convince yourself that it does.

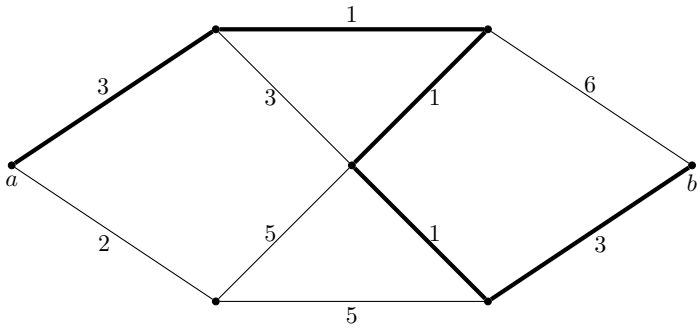
   Solution:

> **Pruning Algorithm**
>
> (a) Start with $H = G$ and $S = \emptyset$;
>
> (b) Repeat the following until $E(H) \backslash S = \emptyset$:
>
> i. Pick $e \in E(H) \backslash S$ with maximum $w(e)$;
>
> ii. If $H - e$ is connected, then remove e from H;
>
> iii. Add e to S.
>
> (c) Return H.

5. Use Dijkstra's algorithm to find a shortest path from a to b in the following two graphs.

   Solution:





6. Give polynomial-time algorithms for the following problems on a connected weighted undirected graph with positive weights. (You do not have to prove correctness or that it is really polynomial-time.)

   (a) Find the shortest cycle, if it exists.

   Solution: For each edge ab, remove it and use Dijkstra to find a shortest path of length ℓ_{ab} from a to b, then write down $\ell_{ab} + w(ab)$. Compare the resulting number for all edges ab, the smallest one is a shortest cycle.

   (b) Find the shortest path from a to b that has the fewest edges among all shortest ab-paths.

   Solution: Take a very small number ε and add it to all the edges. Then Dijkstra will find the shortest path with the fewest edges.

   (c) Find the widest path from a to b, i.e., the path for which the minimum weight of an edge is the largest.

   Solution: Modify Dijkstra's algorithm. Start with $d(v) = \infty$ for all vertices, including a, but still start with a. For the vertex v to add in each step, choose the one with the largest $d(v)$ among the ones not yet visited. When adding that vertex v, instead of making $d(v) \leq d(u) + w(uv)$ for every already-visited neighbor u, make $d(v) \geq \min\{d(u), w(uv)\}$.