Part V
Spanning Trees

1.1 On Spanning Trees, Kruskal's and Prim's Algorithm (p. 58)

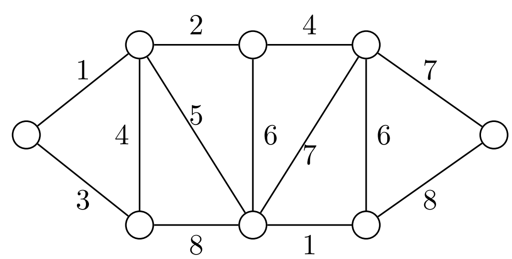
1. Show that for any edge $e \in E(G)$, with G connected, there exists at least one spanning tree T that contains e.

Solution: Let T_1 be an arbitrary spanning tree. If $e \in T_1$, at least T_1 contains e (what was to be shown).

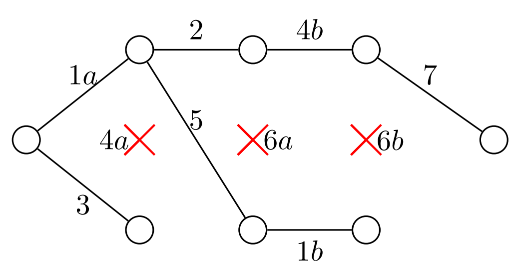
If $e \notin T_1$, then $T_1 + e$ contains a closed path c. Choose an edge $e' \in c$ arbitrarily, with $e \neq e'$. Now $T = T_1 + e - e'$ is a spanning tree with $e \in T$.

2. Apply Kruskal's and Prim's Algorithm to the graph G depicted in Figure 15.

Solution: The original graph G^α:

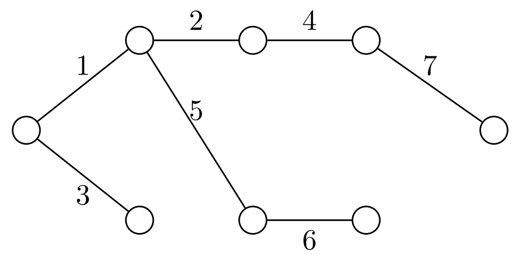


Output of Kruskal. The numbers are the order in which the edges are considered (in this case same as weights). A cross means that adding that edge would create a closed path.
Edges labeled ia and ib (with i an integer) are considered for inclusion in the same step of the algorithm: the letters a and b denote the relative order in which they appear to the algorithm.



Note that the edges added do not have to be connected to each other during the iterations of the algorithm.

Output of Prim. The starting node is arbitrary, but we choose the leftmost vertex. Again, the numbers are the order in which the edges are added.

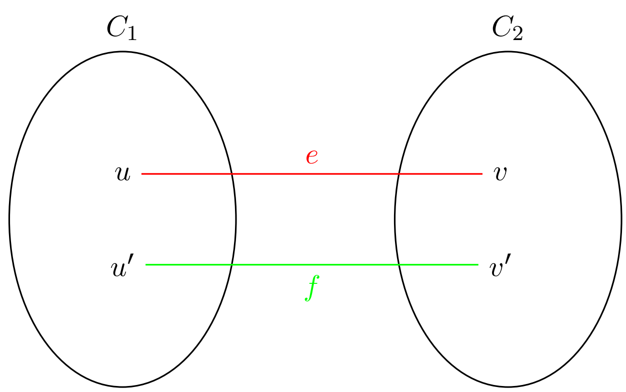


Note that this algorithm "grows" a tree out of one starting vertex. It stays a tree during the whole algorithm and keeps growing until it spans the graph G^α.

3. Let T_1 and T_2 be two spanning trees of minimal weight for G^α . Denote $\alpha^*(T)$ as the largest weight of an edge e belonging to T , i.e., $\alpha^*(T) = \max_{e \in E(T)} \alpha(e)$. Does $\alpha^*(T_1)$ equal $\alpha^*(T_2)$ in general?

Solution: Since we cannot find a counterexample, we try to prove this by contradiction. So, we assume $\alpha^*(T_1) > \alpha^*(T_2)$.

Let $e \in E(T_1)$ such that $\alpha(e) = \alpha^*(T_1)$. Then $e \notin E(T_2)$. Removing this e would disconnect the graph into 2 components, C_1 and C_2.



Now, T_2 is also a spanning tree, so $\exists u' \in C_1, v' \in C_2$ such that $f = (u', v') \in T_2$. This implies that $\alpha(f) < \alpha(e)$ as $\alpha(f) \leq \alpha^*(T_2) < \alpha^*(T_1) = \alpha(e)$.

Construct $T = T_1 - e + f$, which is also a (spanning) tree, with $\alpha(T) < \alpha(T_1)$ (because $\alpha(f) < \alpha(e)$). This would mean T_1 is not a minimal spanning tree, which contradicts the given.

Extra: Another proof: In exercise 4 the alternate proof shows that for any weight w, all minimum spanning trees of G^α must have the same number of edges of weight w. This immediately implies $\alpha^*(T_1) = \alpha^*(T_2)$.

4. Assume that G^α does not have a unique minimum spanning tree. Argue that any of these trees can be returned as output of Kruskal's algorithm.

Solution: Let T be any minimum spanning tree. Divide the edges of G into n sets where set i contains all edges e with $\alpha(e) = w_i$, and $w_1 < w_2 < \ldots < w_n$. Now define a new weight function α' as follows, for some ε where $0 < \varepsilon < \min_{\substack{e_1,e_2 \\ \alpha(e_1)\neq\alpha(e_2)}} |\alpha(e_1) - \alpha(e_2)| = \min_{i=1}^{n-1} w_{i+1} - w_i$. In other words, ε is a positive number bounded by the smallest non-zero difference in weights.

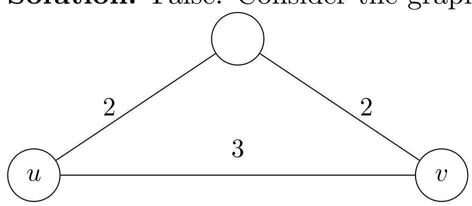
$$\alpha'(e) = \begin{cases} \alpha(e) & e \in T \\ \alpha(e) + \varepsilon & e \notin T \end{cases}$$

The goal of this new weight function α' is to make T the unique MST in $G^{\alpha'}$. The weight of T does not change, while any other (minimum) spanning tree sees its weight increase by at least ε, as these other trees must include at least one edge $e \notin T$. Because Kruskal's algorithm is correct, for $G^{\alpha'}$ it has to return T as it is the only MST. The order in which edges are sorted is the same as for G^α, except that each of the n sets described above are now split in two. The edges in T come first as their weight remains w_i, and they are followed by the edges in the i-th set that were not in T as their weight is now $w_i + \varepsilon$. Kruskal's algorithm does not care about the exact weights of edges, only the order in which they are sorted. As the previously described order is still correctly sorted for weight function α, it is possible for Kruskal's algorithm to return T when using the original weight function α if the edges are sorted as described above.

Extra: Sketch of another proof: Let T be a result of Kruskal's algorithm and let T_1 be any (other) minimal weight spanning tree. One can show that it is possible to transform the tree T_1 into T by adding and removing an edge of equal weight in every step. As edges added and removed in each step are of equal weight, T_1 and T must have the same number of edges of a given weight, implying any MST must have the same number of edges for a given weight w.
This property can then be used to prove that Kruskal's algorithm can return any MST T when edges are sorted in the same order as described in the previous part (for each weight w, place the edges in T first, followed by the edges that are not in T). Kruskal's algorithm will first encounter the edges e with $\alpha(e) = w_1$ and $e \in T$. As T is a tree, these edges can be added as they don't introduce a cycle. Then Kruskal considers the edges of weight w_1 that are not in T. We know that none of these can be added, as this would cause Kruskal to return a tree with more edges of weight w_1 than T, which is not possible due to the property. Similarly Kruskal now adds all edges of weight w_2 that are in T as they don't introduce a cycle, and cannot add any other edges with this weight. This process continues until Kruskal produces the tree T.

8. True or false: Assume p is a path of minimum weight between u and v, then there exists a minimum spanning tree T that contains p. ☆

Solution: False. Consider the graph



Then the path of minimum weight between u and v is just the edge (u, v), however it is not a part of the (only) minimum spanning tree consisting of the weight 2 edges.

11. Let T be a minimum spanning tree for G. Design an $O(|E|)$ algorithm that constructs a minimum spanning tree from T if (a) the weight of an edge e is decreased, (b) the weight of an edge e is increased. ☆

Solution:

(a) If $e \in E(T)$ then T is still an MST. If $e \notin E(T)$ then add e to T. This creates a cycle C. Run through C and remove the edge with the highest weight. Finding the cycle and its edges can be done in $O(|V|) \leq O(|E|)$ through DFS.

(b) If $e \notin E(T)$ then T is still an MST. If $e \in E(T)$ then remove e from T. This creates two trees T_1 and T_2. Now add an edge between T_1 and T_2 with the lowest weight.
To be able to execute the last step in $O(|E|)$ time we first run DFS on $T - e$ to find which vertices are in T_1 and which in T_2. This takes $O(|V| + |E(T)| - 1) = O(|V|) \leq O(|E|)$ time.

12. Develop a fast algorithm to compute a maximum weight spanning tree. ☆

Solution: Define $\alpha'(e) = -\alpha(e)$ for all edges e. This inverts the weight of all trees T. Clearly trees that have the highest weight are those that have the lowest inverted weight. Therefore any MST for $G^{\alpha'}$ is a maximum spanning tree for G^{α}. We can then use any (fast) algorithm to find minimum spanning trees in $G^{\alpha'}$.

Inverting the weights of all edges reverses their order when sorted, meaning this is equivalent to running a modified version of Kruskal where we sort the edges in decreasing (instead of increasing) weight.

14. Let (u, v) be an edge with a weight strictly smaller than the weight of any other edge that is connected to u. Prove that (u, v) must be part of any minimum spanning tree. ☆

Solution: Suppose $(u, v) \notin E(T)$, where T is a minimum spanning tree. Then adding (u, v) to T creates a cycle C. In C there exist two edges with u as one of the nodes, (u, v) and another edge which we denote by e. We can now delete e from T to get a spanning tree with lower weight, which is a contradiction.