1 Minimum spanning trees - Review

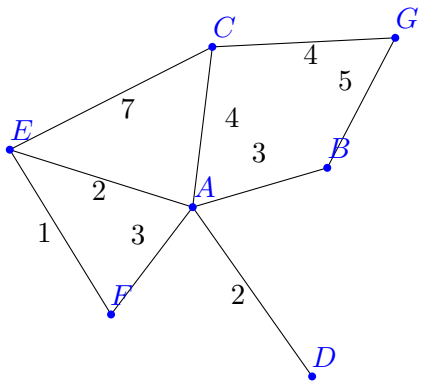
1. **Basics.** A tree is an undirected graph $T = (V, E)$ satisfying all of the following conditions:
   (a) T is connected,
   (b) T is acyclic,
   (c) $|E| = |V| - 1$.

   However, any two conditions imply the third.

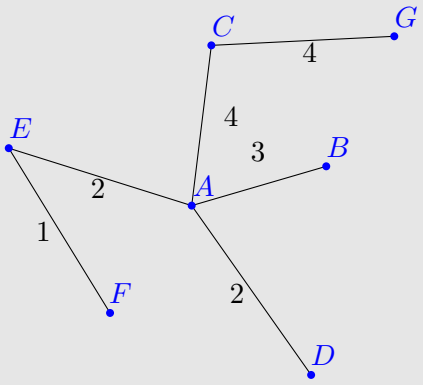
   A **spanning tree** of an undirected graph $G = (V, E)$ is a subgraph which is a tree and which connects all the vertices. (If G is not connected, G has no spanning trees.)

   A **minimum spanning tree** is a spanning tree whose total sum of edge costs is minimum.

   Exercise. *Compute a minimum spanning tree of the following graph:*



Solution.



2. **Cut property.** Let X be a subgraph of $G = (V, E)$ such that X is contained in some MST of G. Let $S \subset V$ be a cut such that no edge of X crosses the cut. Suppose e has minimum weight among the edges crossing the cut: then $X \cup \{e\}$ is contained in some MST of G.

   Exercise. *Prove the cut property.*

   Solution.
   The proof is by contradiction. Assume that no MST contains the edge e. Take any MST (without e), and add e to the MST. This results in a cycle in the tree. Then find an edge e' on this cycle which is contained in the cut S. (Such an edge must exist because a cut cannot cross a cycle only once.) Then $w(e) \leq w(e')$, so we can take out e' from the graph and leave a spanning tree with total weight no larger than before, which therefore must be an MST. Contradiction.

   Because of this property, we will use **greedy algorithms** to construct MST's: Start with $X = \varnothing$, and inductively do the following: Choose a cut $S \subset V$ with no edge of X crossing the cut, find the lightest edge e crossing the cut, set $X \stackrel{\text{def}}{=} X \cup \{e\}$. Repeat until X spans the graph.

3. **Two efficient algorithms.**

   (a) **Prim's algorithm.** X is a tree, and we repeatedly set S to be the edges between the vertices of X and the vertices not in X, adding edges until X spans the graph.

   Thus we are always adding the "closest" vertex to the tree. The implementation of this is almost identical to that of Dijkstra's algorithm.

   (b) **Kruskal's algorithm.** Sort the edges. Repeatedly add the lightest edge that doesn't create a cycle, until a spanning tree is found.

   Notice that in Prim's we explicitly set S based on X, whereas here we implicitly "choose" the cut S after we find the lightest edge that doesn't create a cycle. In other words, we don't actually find a cut corresponding to the edge e to be added, but we know that one must exist — if not, adding e would create a cycle.

4. **Exchange property.** Let T, T' be spanning trees in $G(V, E)$. Given any edge $e' \in T' - T$, there

exists an edge $e \in T - T'$ such that $(T - \{e\}) \cup \{e'\}$ is also a spanning tree.

In other words, we can repeatedly swap edges from one spanning tree to eventually reach any other.

Thus given any spanning tree T and any MST T', we can use the exchange property to "walk" from T to T': at each step, let e' be the lightest edge in $T' - T$.

2 Disjoint-set data structure.

1. Disjoint-set data structure enable us to efficiently perform operations such as placing elements into sets, querying whether two elements are in the same set, and merging two sets together. (Thus they're very useful for simulating graph connectivity.) Must implement the following operations:

   (i) MAKESET(x) — create a new set containing the single element x.

   (ii) UNION(x, y) — replace sets containing x and y by their union.

   (iii) FIND(x) — return name of set containing x.

   We add for convenience the function LINK(x, y) where x, y are roots: LINK changes the parent pointer of one of the roots to be the other root. In particular, UNION(x, y) = LINK(FIND(x), FIND(y)), so the main problem is to make the FIND operations efficient.

2. We have two main methods of optimization for disjoint-set data structures:

   (a) **Union by rank.** When performing a UNION operation, we prefer to merge the shallower tree into the deeper tree.

   (b) **Path compression.** After performing a FIND operation, we can simply attach all the nodes touched directly onto the root of the tree.

3. UNION BY RANK heuristic and PATH COMPRESSION:

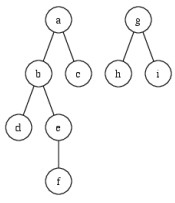


Figure 1: Perform UNION BY RANK

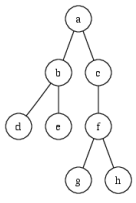


Figure 2: Perform FIND(h)

Exercise. *When using the union by rank optimization only, what is the asymptotic runtime of* FIND*?*

Solution.
$\log n$. It can be shown by induction that when using union by rank, a tree of height h has at least 2^{h-1} vertices. Since FIND has runtime $O(h)$, this implies an upper bound of $\log n$.

3 Additional Exercises

Exercise (CLRS 21.3-6)**.** *Show that a graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique light edge (one of minimum weight) crossing the cut.*

Solution.
Suppose we had two MST's T and T' of a graph $G = (V, E)$ that satisfied the above property. Consider any edge $t = (u, v) \in T$. Removing this edge from T separates the tree into 2 disjoint sets of vertices, S which contains u and $V - S$ which contains v. S and $V = S$ represent a particular cut in the graph, so we know that any MST of G must contain the lightest edge crossing this cut. But from our assumption, we know that t is indeed the lightest edge crossing this cut, so t is in all minimum spanning trees of G. In particular, $t \in T'$. Since t was arbitrarily chosen, we know that each edge in T is also an edge in T'. Since $|T| = |T'|$, we know that $T = T'$ and the MST for G is unique.

Exercise. *Show that if the edge weights of a graph are all unique, then the graph has a unique MST.*

Solution.
Using the previous exercise, since all the edge weights are distinct, any cut will have a unique light edge.

Exercise. (CLRS 24.1-8) *Let T be a MST of a graph G, and let L be the (sorted) list of edge weights of T. Show that for any other MST T' of G, L is also the sorted list of edge weights of T'.*

Solution.
It suffices to show that we can walk from T to T' while preserving the MST property, since it is then clear that the lists of edge weights must match. For this it is enough to show that we can walk them both to an MST T'' while preserving the MST property. To do this, keep selecting the lightest edge e in $(T - T') \cup (T' - T)$. If $e \in T - T'$, make the exchange on T': Adding e to T' creates a cycle, and some edge e' of this cycle must be in $T' - T$; we remove e'. This makes T' lighter so it must again be an MST (which means e' and e had equal weight), and T, T' now have more edges in common than before. This process terminates when both T, T' reach the same tree T''.

Exercise. *In a city there are N houses, each of which is in need of a water supply. It costs W_i dollars to build a well at house i, and it costs C_{ij} to build a pipe in between houses i and j. A house can receive water if either there is a well built there or there is some path of pipes to a house with a well. Give an algorithm to find the minimum amount of money needed to supply every house with water.*

Solution.
Create a graph with N nodes representing the houses and edges between every pair of nodes representing the potential pipes. Then add an additional "source" node which connects to house i with cost W_i, so that the graph now has $N + 1$ nodes. Find the MST of this new graph.