Minimum Spanning Tree.

input: a connected, undirected, edge-weighted graph $G = (V, E)$

output: a minimum-weight *spanning tree* of G

A spanning tree is a tree T in G that connects all the vertices. We represent T as the set of its edges. The weight of T is the sum of its edges' weights, so $\mathsf{wt}(T) = \sum_{e \in T} \mathsf{wt}(e)$. Consider some examples to get intuition.

Exercise: Give an example where a shortest-path tree has much larger total weight than a minimum-spanning tree. Give an example where the path between two vertices in a minimum-spanning tree is much longer than their shortest-path distance.

Greedy design pattern: the first step. Our goal is a greedy algorithm that builds a tree edge by edge. As usual, we start by considering candidate rules for easily determining just the first step, in this case, rules for choosing an edge e_1 that must be in some minimum spanning tree:

Rule (i). (Kruskal's rule.) Let e_1 be any edge of minimum weight.

Rule (ii). (Prim's rule.) Let u_1 be any vertex. Among edges out of u_1, let e_1 be one of min. weight.

Rule (iii). (Generic rule.) Let $(S, V \setminus S)$ be any *cut* in G, (that is, a partition of the vertices into two non-empty sets). Among the edges crossing the cut (that is, $\{(u, w) \in E : u \in S, w \notin S\}$), let e_1 be one of minimum weight.

Rule (i) is a special case of Rule (ii). That is, if we can pick an edge e_1 using Rule (i), then we could also pick it using Rule (ii). In turn, Rule (ii) is a special case of Rule (iii).

Our next goal is to determine whether any of these rules is correct. That is, do they guarantee that e_1 is in some optimal solution (some minimum spanning tree)? To try to prove that they do, as usual, we will use an exchange argument, of the following form:

1. Let T be any minimum spanning tree.
2. If e_1 is in T, we are done.
3. Otherwise, obtain tree T' from T by adding e_1 and (somehow) removing another edge.
4. Somehow show that T' is also a minimum spanning tree.

To make an argument like this, we need to study what kinds of edge exchanges we can do with spanning trees. The next observation captures the kind of edge exchange we use.

Observation 1. *Let T be any spanning tree of G. Let e' be any edge not in T, and let e be any edge on the path in T that connects the endpoints of e'. Then adding e' to T and removing e gives a spanning tree $T' = \{e'\} \cup T \setminus \{e\}$.*

Here's a different way to look at the exchange. Recall that a *cut* is a partition $(S, V \setminus S)$ of the vertices of G into two non-empty sets. An edge $(u, w) \in E$ *crosses* the cut if $u \in S$ and $w \in V \setminus S$.

Observation 2. *Let T be any spanning tree of G. Let e be any edge in T. Consider the cut formed by removing e from T (removing e leaves two subtrees T_1 and T_2, giving the cut $(S, V \setminus S)$ where S contains the vertices in T_1). Let e' be any edge crossing this cut. Then removing e from T and adding e' gives a spanning tree $T' = \{e'\} \cup T \setminus \{e\}$.*

Observation 1 is equivalent to Observation 2, in that e is on the path in T between the endpoints of e' if and only if e' is on the cut $(S, V \setminus S)$ defined by removing e from T. In this note we use only Observation 1. We use it to show that Rules (i)-(iii) are correct.

Lemma 1. *Rule (i) is correct. That is, given any edge-weighted graph $G = (V, E)$ and any edge e_1 of G chosen by Rule (i), there is a minimum spanning tree of G that contains e_1.*

Proof (short form). Let T be any minimum spanning tree in G. If e_1 is in T, we are done. Otherwise, obtain a spanning tree T' from T by using Observation 1 to add e_1. That is, add e_1 to T and remove any edge e on the path in T that connects the endpoints of e_1, to obtain spanning tree $T' = \{e_1\} \cup T \setminus \{e\}$.

Then spanning tree T' contains e_1, and the weight of T' is at most the weight of T, because $\mathsf{wt}(T') = \mathsf{wt}(T) + \mathsf{wt}(e_1) - \mathsf{wt}(e)$, and (by Rule (i)) e_1 has minimum weight among all edges in G, so $\mathsf{wt}(e_1) \leq \mathsf{wt}(e)$. So T' is also a minimum-weight spanning tree. □

Lemma 2. *Rule (ii) is correct. That is, given any edge-weighted graph $G = (V, E)$ and any edge e_1 of G chosen by Rule (ii), there is a minimum spanning tree of G that contains e_1.*

Proof (short form). Let T be any minimum spanning tree in G. If e_1 is in T, we are done. Otherwise, obtain a spanning tree T' from T using Observation 1 to add e_1, selecting the edge e to remove carefully, as follows. By Rule (ii), the edge e_1 has minimum weight among edges that are incident to one of its endpoints u_1. Let P be the path in T connecting u_1 to the other endpoint of e_1. *Let e be the first edge (incident to u_1) on that path P.* Then take $T' = \{e_1\} \cup T \setminus \{e\}$.

By Observation 1, T' is a spanning tree, and it contains e_1. The weight of T' is at most the weight of T, because $\mathsf{wt}(T') = \mathsf{wt}(T) + \mathsf{wt}(e_1) - \mathsf{wt}(e)$, and e_1 *has minimum weight among all edges leaving u_1*, so $\mathsf{wt}(e_1) \leq \mathsf{wt}(e)$. So T' is also a minimum-weight spanning tree. □

Lemma 3. *Rule (iii) is correct. That is, given any edge-weighted graph $G = (V, E)$ and any edge e_1 of G chosen by Rule (iii), there is a minimum spanning tree of G that contains e_1.*

Proof (short form). Let T be any minimum spanning tree in G. If e_1 is in T, we are done. Otherwise, obtain a spanning tree T' from T by using Observation 1 to add e_1, and selecting the edge e to remove as follows.

By Rule (iii), the edge e_1 crosses some cut $(S, V \setminus S)$ and has minimum weight among edges crossing that cut. Let u' and w' be the endpoints of edge e_1. Let P be the path between u' and w' in T. *Let e be an edge in P that crosses the cut $(S, V \setminus S)$.* (It must exist, as u' is on one side of the cut, and w' is on the other.)

Take $T' = \{e_1\} \cup T \setminus \{e\}$. By Observation 1, T' is a spanning tree containing e_1. Then $\mathsf{wt}(T')$ is at most $\mathsf{wt}(T)$, because $\mathsf{wt}(T') = \mathsf{wt}(T) + \mathsf{wt}(e_1) - \mathsf{wt}(e)$, and e_1 *has minimum weight among all edges crossing the cut*, so $\mathsf{wt}(e_1) \leq \mathsf{wt}(e)$. So T' is a minimum-weight spanning tree. □

Extending beyond the first step by contracting e_1. We now have a rule (in fact several) for choosing one edge in some minimum spanning tree. Our next goal is to extend the rule to find an entire minimum spanning tree. We first find a recursive algorithm based on optimal substructure, then convert it into an iterative algorithm that maintains the greedy invariant.

What kind of optimal substructure property can we show? Once we have committed to using some first edge $e_1 = (u', w')$ that we know is in some minimum spanning tree, how can we characterize the problem that remains — of choosing a set S of the remaining edges so that $S \cup \{e_1\}$ is a minimum spanning tree?

We use *edge contraction*. Define $G' = (V', E')$ to be the graph obtained from G by *contracting* edge $e_1 = (u', w')$. Specifically, contracting e_1 replaces e_1's two endpoints u' and w' by a single new vertex, say v', and replaces each edge e into u' or w' (other than (u', w')) by a corresponding edge of the same weight into v'. Strictly speaking, G' is a multigraph — G' may have more than one edge between a given pair of vertices. This doesn't matter though — our problem definitions and rules extend naturally to multi-graphs.

The main idea is that spanning trees in G that contain e_1 correspond exactly to spanning trees in G'. So, to find a minimum-spanning tree in G that contains e_1, we can just find a minimum-spanning tree in G'. Next we make this intuition precise.

Given any edge $e \in E \setminus \{e_1\}$ from the original graph G, let $\phi(e)$ denote the corresponding edge in the contracted graph G'. (If e is not incident to u' or w', then $\phi(e) = e$. Otherwise, $\phi(e)$ is obtained by replacing u' or w' in e by v'.) And, given any set $S \subseteq E \setminus \{e_1\}$ of edges in G, let $\phi(S) = \{\phi(e) : e \in S\}$ denote the set of corresponding edges in G'. The function ϕ is an invertible bijection between the spanning trees of G that contain e_1, and the spanning trees of G':

Observation 3. *Let $S \subseteq E \setminus \{e_1\}$ be any subset of edges (other than e_1) in G. Then $S \cup \{e_1\}$ is a spanning tree of G if and only if $\phi(S)$ is a spanning tree of G'. Also, $\mathsf{wt}(\phi(S))$ equals $\mathsf{wt}(S)$.*

Now we can state the optimal-substructure property that formalizes our intuition.

Lemma 4 (opt'l substructure)**.** *Let e_1 be an edge that is in any minimum spanning tree of G. Let G' be obtained from G by contracting e_1. Let R be any minimum spanning tree of G'. Let S be the edge set in G corresponding to R (so $R = \phi(S)$). Then $S \cup \{e_1\}$ is a minimum spanning tree of G.*

Proof (short form). Let e_1, G', T', and S be as in the lemma. Let T^* be any minimum spanning tree of G containing e_1. Let $S^* = T^* \setminus \{e_1\}$. Let $R^* = \phi(S^*)$ be the corresponding spanning tree of G' (Observation 3.) Because R is a minimum spanning tree of G', $\mathsf{wt}(R) \le \mathsf{wt}(R^*)$. So

$$\mathsf{wt}(S \cup \{e_1\}) = \mathsf{wt}(S) + \mathsf{wt}(e_1) = \mathsf{wt}(R) + \mathsf{wt}(e_1) \le \mathsf{wt}(R^*) + \mathsf{wt}(e_1) = \mathsf{wt}(S^*) + \mathsf{wt}(e_1) = \mathsf{wt}(T^*).$$

That is, $\mathsf{wt}(S \cup \{e_1\})$ is at most $\mathsf{wt}(T^*)$ (the minimum-spanning-tree weight in G). By Observation 3, $S \cup \{e_1\}$ is a spanning tree of G. It follows that $S \cup \{e_1\}$ is also a minimum spanning tree. □

Exercise: Show the converse of the lemma — if $S \cup \{e_1\}$ is a minimum spanning tree for G, then $\phi(S)$ is a minimum spanning tree for G'.

Recursive algorithm. This gives us the following recursive algorithm:

```
rMST($G = (V, E)$):                    — recursive MINIMUM SPANNING TREE algorithm —
1. if $|V| = 1$: return $\emptyset$                  (note: $\emptyset$ denotes the empty set)
2. let $e_1$ be an edge in selected by Rule (i), (ii), or (iii) (so $e_1$ is in some MST)
3. let $G'$ be obtained from $G$ by contracting edge $e_1$ (as described previously)
4. recursively compute minimum spanning tree $R$ = rMST($G'$) for $G'$
5. let $S = \phi^{-1}(R)$ be the corresponding set of edges in $G$ (as described previously)
6. return $\{e_1\} \cup S$
```

Theorem 1. *The recursive algorithm is correct. Given any connected edge-weighted graph G, it returns a minimum spanning tree of G.*

Proof sketch. The proof is by induction on the number of vertices in G, using Lemmas 1–3 (correctness of Rules (i)–(iii)) and Lemma 3 (optimal substructure) to prove the inductive step. We leave the details as an exercise. □

Kruskal's algorithm. What the algorithm does depends on which rule we apply in each step, and how we apply the rule — how we choose the vertex u_1 for Rule (ii), or the cut $(S, V \setminus S)$ for Rule (iii). To get some intuition, let's first consider the algorithm that applies just Rule (i).

We'll give three descriptions of this algorithm. To ease notation, when we contract some edge, we *identify* each edge in the contracted graph with the corresponding edge in the original graph (that is, we consider the two edges to be the same). Here is the first description:

```
iMST1($G = (V, E)$):                — iterative MST algorithm using Rule (i) —
1. initialize $F \leftarrow \emptyset$, $G' = G$, and $n = |V|$
2. for $t = 1, 2, \ldots, n-1$:
3.1.   let $e_t$ be any edge of minimum weight among those remaining in $G'$
3.2.   add $e_t$ to $F$
3.3.   modify $G'$ by contracting edge $e_t$ in $G'$
4. return $F$
```

At any point during the execution of the algorithm, the edges in F form a forest (a set of subtrees) in the original graph G. Each vertex v' in the contracted graph G' corresponds to a subtree t' in that forest (where v' is obtained by contracting the edges in t'). The edges remaining in G' are those that connect different components. So, we can also describe the algorithm as follows:

```
iMST1($G = (V, E)$):                — iterative MST algorithm using Rule (i) —
1. initialize $F \leftarrow \emptyset$ and $n = |V|$
2. for $t = 1, 2, \ldots, n-1$:
3.1.   let $e_t$ be any edge of minimum weight among those connecting different subtrees in $F$
3.2.   add $e_t$ to $F$
4. return $F$
```

Now, it is not too hard to see that we can also describe the algorithm as follows:

```
Kruskal($G = (V, E)$):                                  — Kruskal's algorithm —
1. initialize $F \leftarrow \emptyset$
2. let $e_1, e_2, \ldots, e_m$ be the edges in $E$, ordered by increasing weight
3. for $t = 1, 2, \ldots, m$:
4.1.   if $e_t$ connects different subtrees in $F$: add $e_t$ to $F$
5. return $F$
```

This is known as Kruskal's algorithm. We didn't say how to determine whether e_t connects different subtrees in F. We will discuss that later in the course. One naive way is to use what is called *depth-first search*, which takes linear time, leading to a quadratic-time algorithm. A better way uses a *disjoint-set* data structure. If we use that data structure, the total time is dominated by the time to sort the edges. So the total time $O(m \log n)$, where $m = |E|$ and $n = |V|$.

Prim's algorithm. Next let's consider the algorithm that applies Rule (ii), choosing "the same" vertex u_1 each time. Here is the first description:

```
iMST2($G = (V, E)$):                    — iterative MST algorithm using Rule (ii) —
1. initialize $F \leftarrow \emptyset$, $G' = G$, and $n = |V|$
2. fix any vertex $u_1$ in $V$
3. for $t = 1, 2, \ldots, n-1$:
4.1. let $e_t$ be any edge having minimum weight among those leaving $u_1$ in $G'$
4.2. add $e_t$ to $F$
4.3. modify $G'$ by contracting edge $e_t$ in $G'$ (and update $u_1$ to be the contracted vertex)
5. return $F$
```

Each iteration contracts an edge out of the previously contracted vertex. So the edges in F always form a single subtree in the original graph G. Instead of doing the contraction, we can keep track of that subtree directly:

```
Prim($G = (V, E)$):                    — Prim's algorithm —
1. fix any vertex $u_1$ in $V$
2. initialize $F$ to be the subtree containing just $u_1$
3. for $t = 1, 2, \ldots, n-1$, where $n = |V|$:
4.1. let $e_t$ be any edge of minimum weight among edges leaving subtree $F$
4.2. add $e_t$ to $F$
5. return $F$
```

This is commonly called Prim's algorithm, although Prim was not the first to discover it. (See https://en.wikipedia.org/wiki/Prim's_algorithm.) It is very similar to Dijkstra's algorithm for SINGLE-SOURCE SHORTEST PATHS, and, like that algorithm, can be implemented to run in time $O(m \log n)$ by using a heap containing the edges that leave F. (But for this algorithm, the edges are keyed just by their individual weight.)

Generic algorithm. If we allow the full generality of Rule (iii), we get the following generic algorithm, which generalizes both Prim's algorithm and Kruskal's algorithm. As the algorithm proceeds, it applies Rule (iii) to the graph obtained by contracting all edges in F. (In the contracted graph, each vertex corresponds to a subtree of F.)

To define the algorithm, say that a cut $(S, V \setminus S)$ *respects* F if none of the edges in F cross the cut. Equivalently, each subtree in F is either entirely within S, or entirely within $V \setminus S$.

```
gMST($G = (V, E)$):                    — generic MST algorithm —
1. initialize $F \leftarrow \emptyset$
2. while there exists a cut that respects $F$ (i.e., no edges in $F$ cross the cut):
3.1. choose any cut $(S, V \setminus S)$ that respects $F$
3.2. let $e_t$ be any edge having minimum weight among edges crossing the cut
     (at least one edge crosses the cut, as $G$ is connected)
3.3. add $e_t$ to $F$
4. return $F$
```

For completeness, here is a proof that the generic algorithm is correct.

Lemma 5. *The generic algorithm maintains the following invariant: the current set F of edges is contained in some minimum spanning tree.*

Proof (long form).

1. Consider any execution of the algorithm on some input $G = (V, E)$.
2. The invariant holds before the first iteration, when $F = \emptyset$.
3.1. Consider any iteration t such that the invariant holds before the iteration.
3.2. Let F denote F at the start of the iteration.
3.3. Let cut $(S, V \setminus S)$ and $e_t = (u, w)$ be as chosen in the iteration.
3.4. So the cut respects F, and e_t has minimum weight among edges crossing the cut.
3.5. Let T be a minimum spanning tree such that $F \subseteq T$ (per the invariant).
3.6.1. Case 1. First consider the case that $e_t \in T$.
3.6.2. Then $F \cup \{e_t\} \subseteq T$, so the invariant holds at the end of the iteration.
3.7.1. Case 2. Otherwise $e_t \notin T$.
3.7.2. The edge e_t crosses the cut, so the path in T between e_t's endpoints crosses the cut.
3.7.3. So some edge on that path crosses the cut. Let e be such an edge.
3.7.4. So e_t is not in T, and e is on the path in T between the endpoints of e_t.
3.7.5. By Observation 1, adding e_t to T and removing e gives a spanning tree $T' = \{e_t\} \cup T \setminus \{e\}$.
3.7.6. Edge e_t has minimum weight among edges crossing the cut (including e) so $\mathsf{wt}(e_t) \le \mathsf{wt}(e)$.
3.7.7. So $\mathsf{wt}(T') = \mathsf{wt}(T) + \mathsf{wt}(e_t) - \mathsf{wt}(e) \le \mathsf{wt}(T)$.
3.7.8. So T' is also a minimum-weight spanning tree of G.
3.7.9. Edge e crosses the cut, and the cut respects F, so $e \notin F$.
3.7.10. So $\{e_t\} \cup F \subseteq \{e_t\} \cup T \setminus \{e\} = T'$.
3.7.11. So, at the end of the iteration, the partial solution $F \cup \{e_t\}$ is a subset of the MST T'.
3.7.12. So the invariant holds at the end of the iteration.
3.8. By Blocks 3.6 and 3.7, the invariant holds at the end of the iteration.
4. By Block 3, each iteration that starts with the invariant true ends with it true.
5. And, as observed above, the invariant holds initially. So it holds throughout. □

Theorem 2. *The generic algorithm is correct. Given any connected edge-weighted graph G, it returns a minimum spanning tree for G.*

Proof (short form). Consider any execution of the algorithm on any input G. By Lemma 5, the invariant holds after the last iteration of the algorithm, so the final set F is a subset of some minimum spanning tree T of G. By the termination condition of the loop, F must connect all vertices in G (otherwise some cut would respect F). There cannot be any edge in $(u, w) \in T \setminus F$ (otherwise (u, w) and the path in F from u to w would form a cycle in T, contradicting that T is a tree). So $F = T$. That is, F is a minimum spanning tree. □

Reverse greedy algorithms. Here's another kind of greedy algorithm to consider. Start with all edges in the graph, and remove edges one by one, until the remaining edges form a spanning tree. Can you come up with an algorithm of this form? It should maintain the following invariant: *some minimum spanning tree of the original graph is a subset of the remaining edges.*

Start by finding a rule that, given a connected edge-weighted graph $G = (V, E)$ that is not a tree, identifies one edge $e_1 \in E$ such that some minimum spanning tree of G is a subset of $E \setminus \{e_1\}$.

Choosing the maximum-weight edge doesn't always work! What about this: let C be any cycle in G, and let $e_1 \in C$ be any edge of maximum weight among edges in C. Exercise: prove this rule works, and derive recursive and iterative MST algorithms from it.

External resources

- text: CLRS Chapter 23
- text: Dasgupta et al. Section 5.1
- text: Kleinberg & Tardos Section 4.5
- text: Edmonds Lecture 20

  http://jeffe.cs.illinois.edu/teaching/algorithms/notes/20-mst.pdf
- MIT lecture (video):

  https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/
  6-046j-introduction-to-algorithms-sma-5503-fall-2005/
  video-lectures/lecture-16-greedy-algorithms-minimum-spanning-trees/