23.1 Minimum Weight Spanning Tree

Throughout this section $G = (V, E)$ denote an undirected graph. We say that $G$

is a weighted graph if to every edge $(u, v) \in E$ there is associated a weight $w(u, v)$, i.e., there is a weight function

$$w : E \rightarrow \mathbb{R}.$$ This weight is sometimes interpreted as a cost or distance.

A graph $H$ is called a subgraph of $G$ if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. $H$ is called a spanning subgraph if $V(H) = V(G)$. A spanning tree in $G$ is a spanning subgraph which is also a tree.

The weight of a subgraph $H$ is the total weight of all its edges:

$$w(H) = \sum_{e \in E(H)} w(e)$$

$T$ is called a minimum weight spanning tree if no spanning tree has lower weight than $T$, i.e., for all spanning trees $T'$,

$$w(T) \leq w(T').$$
**Exercise**
Prove that G contains a spanning tree iff G is connected.

**Exercise**
Prove that if a connected graph G has distinct edge weights, then G contains a unique minimum weight spanning tree.

**Problem (MST)**
Given a connected graph $G = (V, E)$ and a weight function $w: E \to \mathbb{R}$, determine a minimum weight spanning tree in $G$.

We will study two famous algorithms (Kruskal and Prima) which solve this problem.
(23.2) Algorithms of Kruskal & Prim

Both algorithms follow a so-called 'greedy' strategy. While building a spanning tree, select new edges according to some local optimum criterion, i.e., amongst all 'suitable' edges, choose one of minimum weight.

We begin with a high level description of both algorithms. There will be most useful for tracing and proving things about the algorithms. Later we'll consider the more detailed versions in this book.

In the following, $A$ denotes a subset of $E(G)$.

**Kruskal**

1.) $A \not= \emptyset$

2.) While $|A| < |V| - 1$

3.) Amongst all edges in $E - A$ whose addition to $A$ would not form a cycle, let $e$ be one of minimum weight

4.) $A \leftarrow A \cup \{e\}$

**Theorem**

When Kruskal is complete, $T = (V, A)$ is a Mst in $G$.

We will not prove this theorem except to note that Kruskal obviously produces a spanning tree. This follows from the theorem.
Theorem. Since no cycles are ever created and we stop when \(|A| = |V| - 1\).

In the following algorithm again, \(A \subseteq \mathcal{E}(G)\)
and \(W\) denotes a subset of \(V(G)\).

1. \(A \leftarrow \emptyset\)
2. Pick any \(v \in V \setminus W\), \(W \leftarrow W \cup \{v\}\)
3. While \(|A| < |V| - 1\)
4. Amongst all edges in \(E - A\) with exactly
   one end in \(W\) (and the other end in \(V - W\)) let \(e\) be one of minimum weight.
5. \(A \leftarrow A \cup \{e\}\)
6. \(W \leftarrow W \cup \{\text{other end of } e\}\)

Theorem.
When \(\text{Prim}\) is complete, \(T = (V, A)\) is a MST in \(G\) (and \(W = V\)).

Again we will not prove this, but it is obvious that \(\text{Prim}\) produces a spanning tree
by the treelessness theorem.

Observe that both algorithms would be difficult
to implement as stated since it is not
clear how to perform Kruskal line 3 or Prim
line 4. Also neither algorithm is deterministic
as stated since it is not specified how to select
edge \(e\) when more than one choice is
available.
Ex: Prim

Observe that Prim grows a single tree \((W, A)\) from the 'seed' \(v\) until \(W\) becomes all of \(V\).

On the other hand, Kruskal simultaneously grows a number of disjoint trees until they all merge into \((V, A)\).

Ex: Kruskal

\[ w(T) = 44 \]
In the following detailed version of Prim, each vertex \( u \) has two attributes: \( \text{pl}[u] \), the parent of \( u \), and \( \text{key}[u] \) which is used to sort vertices. \( Q \) denotes a min-priority queue which stores vertices according to their key values.

\[
\text{Prim}(G, w, r) \\
1.) \text{ for each } u \in V \\
2.) \text{ key}[u] \leq \infty \\
3.) \text{ pl}[u] \leq \text{ nil} \\
4.) \text{ key}[r] \leq 0 \\
5.) Q \leftarrow V \quad (\text{i.e. build a min-priority heap \( V \)}) \\
6.) \text{ while } Q \neq \emptyset \\
7.) u \leftarrow \text{ ExtractMin}(Q) \\
8.) \text{ for each } v \in \text{Adj}[u] \\
9.) \quad \text{ if } v \in Q \text{ and } w(u, v) < \text{key}[v] \\
10.) \quad \text{ pl}[v] \leftarrow u \\
11.) \quad \text{ key}[v] \leftarrow w(u, v) \quad (\text{i.e. DecreaseKey})
\]

During execution, the set \( V - Q \) contains the set of vertices of the tree \( T \) being built \( (V - Q = W \text{ in earlier version}) \). Specifically:

\[
V(T) = V - Q \\
E(T) = \left\{ (\text{pl}[x], x) \mid x \in V(T) \land \text{pl}[x] \neq \text{ nil} \right\}
\]
Thus when Prim(G,w,r) is complete
the must T in the rooted tree
(with root r) given implicitly by the
parent fields.

Loop 6-11 maintains the following invariant:

For each v ∈ Q, key[v] is the minimum
weight of any edge which joins v to
T. By convention key[v] = ∞ if no
such edge is known to exist.

Thus p[v] names the (future) parent
of v in T, i.e. p[v] is the end of
that minimum weight edge which lies in
T, and v is the so-called 'other end'
of that edge.

Line 7 picks the closest vertex to T
then adds it to the tree by deleting
it from Q. The edge (p[u], u) is being
added to the tree implicitly. Since
T was changed, the keys of some vertices
outside T may be incorrect. Obviously
the vertices affected by the change are
those adjacent to the newly added vertex
u. Loop 8-11 makes the necessary changes
to p[v] and key[v] for those vertices
v which are both outside the tree
(i.e. v ∈ Q) and adjacent to u.
Run Time

We assume CQ is implemented as a min heap.

Use Build-Heap to perform initialization.
Steps 1-5 in time \( O(N^2) \), loop 6-11 executes \( N \) times, and each call to ExtractMin costs \( O(\log N) \) time. Thus the total cost of all calls to ExtractMin is \( O(N \log N) \).

Loop 6-11 executes a total of \( O(1+E) \) times. Within this loop, the test VECQ can be performed in constant time. (Just keep a flag for each vertex recording whether or not it is in \( Q \).) Line 11 involves a call to Decrease Key which costs \( O(\log N) \) time. Thus the total cost of all such operations is \( O(1+E \log N) \).

Hence the total run time of Prim is
\[
O(N \log N + 1+E \log N). 
\]

However, since \( G \) is assumed to be connected, we have \( |E| = N-1 \) (Exercise), whence \( N \leq |E|+1 = O(E) \). Thus the run time of Prim is in the class:
\[
O(E \log N). 
\]