Chapter 23: Minimum Spanning Tree

Let $G = (V, E)$ be a connected (undirected) graph. A spanning tree of $G$ is a tree $T$ that consists of edges of $G$ and connects every pair of nodes.

Let $w$ be an integer edge-weight function. A minimum-weight spanning-tree is a tree whose weight respect to $w$ is the smallest of all spanning trees of $G$. 
Safe edges and cuts

$A$ : expandable to an MST

$e \in E - A$ is safe for $A$ if $A \cup \{e\}$ : expandable to an MST or an MST already

A cut of $G$ : a partition $(S, V - S)$ of $V$

An edge $e$ crosses $(S, V - S)$ if $e$ connects a node in $S$ and one in $V - S$

$(S, V - S)$ respects $A \subseteq E$ if no edges in $A$ cross the cut

For any edge property $Q$, a light edge w.r.t. $Q$ is one with the smallest weight among those with the property $Q$
**Theorem A**  Let $G = (V, E)$ be a connected (undirected) graph with edge-weight function $w$. Let $A$ be a set expandable to an MST, let $(S, V - S)$ be a cut respecting $A$, and let $e = (u, v)$ be a light edge crossing the cut. Then $e$ is safe for $A$.

**Proof** Let $T$ be an MST containing $A$ and not containing $e$. There is a unique path $\rho$ in $T$ from $u$ to $v$. $\rho$ has an edge crossing $(S, V - S)$. Pick one such edge $d$. Then $T' = T \cup \{e\} - \{d\}$ is a spanning tree such that $w(T') = w(T)$ so $T'$ is an MST and $e$ is safe.

**Corollary B**  Every light edge connecting two distinct components in $G_A = (V, A)$ is safe for $A$. 


Kruskal’s Algorithm

Maintain a collection of connected components and construct an MST $A$.

Initially, each node is a connected component and $A = \emptyset$.

Examine all the edges in the nondecreasing order of weights.

- If the current edge connects two different components, add $e$ to $A$ to unite the two components.

The added edge is a light edge; otherwise, an edge with smaller weight should have already united the two components.
Implementation with “disjoint-sets”

1. $A \leftarrow \emptyset$
2. for each vertex $v \in V$ do
   3. Make-Set($v$)
4. reorder the edges so their weights are in nondecreasing order
5. for each edge $(u, v) \in E$ in the order do
   6. if Find-Set($u$) $\neq$ Find-Set($v$) then
      7. $A \leftarrow A \cup \{(u, v)\}$
      8. Union($u, v$)
9. return $A$
The number of disjoint-set operations that are executed is \(2E + 2V - 1 = O(E)\), out of which \(V\) are \texttt{Make-Set} operations.

\[\text{So, what is the total running time?}\]
Well, the total cost of the disjoint-set operation is $O(E \lg^* V)$ if the union-by-rank and the path-compression heuristics are used.

Sorting the edges requires $O(E \log E)$ steps.

We can assume $E \geq V - 1$ and $E \leq V^2$.

So, it's $O(E \log V)$ steps.
Prim’s algorithm

Maintain a set of edges $A$ and a set of nodes $B$. Pick any node $r$ as the root and set $B$ to \{r\}. Set $A$ to $\emptyset$. Then repeat the following $V - 1$ times:

- Find a light edge $e = (u, v)$ connecting $u \in B$ and $v \in V - B$.
- Put $e$ in $A$ and $v$ in $B$. 
Implementation Using a Priority Queue

For each node in $Q$, let $key[v]$ be the minimum edge weight connecting $v$ to a node in $B$. By convention, $key[v] = \infty$ if there is no such edge.

For each node $v$ record the parent in the field $\pi[v]$. This is the node $u$ such that $(u, v)$ is the light edge when $v$ is added to $B$.

An implicit definition of $A$ is

$$\{(v, \pi[v]) \mid v \in V - \{r\} - Q\}.$$
for each $u \in Q$ do $key[u] \leftarrow \infty$

Line 3 forces to select $r$ first. Lines 7-10 are for updating the keys.

Implement $Q$ using a heap. The running time is

\[
V \cdot (\text{the cost of Build-Heap}) \\
+ (V - 1) \cdot (\text{the cost of Extract-Min}) \\
+ E \cdot (\text{the cost of Decrease-Key}).
\]
If either a binary heap or a binomial heap is used, the running time is:

\[
V \cdot O(1) \\
+ (V - 1) \cdot O(\lg V) \\
+ E \cdot O(\lg V) \\
= O((E + V) \lg V) = O(E \lg E),
\]

which is the same as the running time of Kruskal’s algorithm.

If a Fibonacci heap is used, the running time is:

\[
V \cdot O(1) \\
+ (V - 1) \cdot O(\lg V) \\
+ E \cdot O(1) \\
= O(V \lg V + E),
\]

which is better than the running time of Kruskal’s algorithm.