Chapter 23
Minimum Spanning Trees

Let $G(V, E, \omega)$ be a weighted connected graph. Find out another weighted connected graph $T(V, E', \omega)$, $E' \subset E$, such that $T$ has the minimum weight among all such $T$’s.

An important application is how to connect all the sites into a network with minimum cost.

Obviously, the solution $T$ must be a tree. If it’s not, there must be a cycle inside.

We can then remove at least one edge from this spanning graph, and end up with another one with less weight, assuming all the weights are non-negative.

If the weight of all the edges along that cycle is 0, we would have another graph with less edges.
An idea

Assume that we have a weighted connected graph $G(V, E, \omega)$, where $\omega$ is a weight function $\omega : E \rightarrow R$, and we wish to find a minimum spanning tree, $T(V, E', \omega), E' \subset E$, for $G$.

A greedy strategy to grow such a tree adds one edge at a time,

Generic-MST($G, w$)
1. A <- NIL
2. while A does not form a spanning tree do
3. find an edge (u, v) that is safe for A
4. A<-A Union {(u, v)}
5. return A

At each stage, if we start with $A$, a subset of a MST, the algorithm selects another safe edge $e$ such that $A \cup \{e\}$ is also a subset of a MST.
Correctness

We have the following loop invariant for the while look at Lines 2-4, where $A$ is the edges that the algorithm has collected so far:

Prior to each iteration, $A$ is a subset of some minimum spanning tree.

Initially, $A = \emptyset$, clearly a subset of any MST.

If this invariant holds, at the end of the algorithm, $A$ is still a subset of a minimum spanning tree, i.e., for all the vertices it includes, it connects them with edges with minimum weights. Now that it includes all the vertices, it is a minimum spanning tree of the original graph.

*The critical step is certainly Step 3, i.e., how to find a “safe edge” to keep the invariant going.*
A few notions

A cut \((S, V - S)\) of a graph \(G = (V, E)\) is a partition of \(V\).

An edge \((u, v)\) crosses a cut \((S, V - S)\) if one of its end points is in \(S\), while the other is in \(V - S\).

We also say that a cut respects a set \(A\) of edges if no edge in \(A\) crosses the cut. In other words, there does not exist an edge \((u, v) \in A\), such that \(u \in S\), and \(v \in V - S\).

Finally, an edge is a light edge crossing a cut if its weight is the minimum among all the edges that cross that cut.
A critical property

**Theorem 23.1** Let $G = (V, E, \omega)$ be a connected graph with a real-valued weight function $\omega$ defined on $E$.

Let $A(\subset E)$ be included in some MST for $G$, $(S, V - S)$ be any cut of $G$ that respects $A$, and let $(u, v)$ be a light edge crossing $(S, V - S)$, then, $(u, v)$ is safe for $A$.

This means that, if $A$ is part of a MST, then $A \cup (u, v)$ is still part of a MST.

This property establishes the maintenance step of the loop invariant. Then, the Generic-MST algorithm, as shown in Page 3 will work.
A proof

Given $A$, part of a MST, we want to show that $A \cup \{(u, v)\}$ is also part of a minimum spanning tree. Let $T$ be a minimum spanning tree that includes $A$. If the edge $\{(u, v)\}$ also belongs to $T$, we are done. We thus assume $\{(u, v)\} \not\in T$.

We now construct another minimum tree, $T'$, that does include $A \cup \{(u, v)\}$. Since $T$ is a spanning tree, and the edge $\{(u, v)\}$ does not belong to $T$, there must be a path, $p$, that connects $u$ and $v$. Thus, this edge $(u, v)$ forms a cycle with $p$. 
Since \((u, v)\) is a crossing edge, \(u\) and \(v\) are on different sides of the cut \((S, V - S)\). Then, there is at least one edge of \(p\) that crosses the cut.

Let \((x, y)\) be such an edge. Since the cut respects \(A\), \((x, y) \notin A\). We now remove \((x, y)\) from \(T\), and add in \((u, v)\) to construct another tree, \(T'\).

Since \((u, v)\) is a light edge crossing the cut, \(w(u, v) \leq w(x, y)\). Thus,

\[
w(T') = w(T) - w(x, y) + w(u, v) \leq w(T).
\]

Thus, \(T'\) is also a minimum spanning tree.

We can now conclude that, if \(A\) is part of a MST, so is \(A \cup \{(u, v)\}\).
An application

Prim’s algorithm is based on the above result. When applied to a graph, the edges in the set \( A \) always form a single tree.

The tree starts from an arbitrary root vertex \( r \), and grows until the tree includes all the vertices in \( G \). At each step, a light edge, one with minimum weight, is added to the tree that connects \( A \) to another isolated vertex of \( G - A \). Thus, by Theorem 23.1, this light edge is safe for \( A \).

The key to an efficient implementation of the algorithm is to quickly find out the light edge.

One way to do it is to collect all the vertices that are not in the tree yet in a minHeap on a key field. For each \( v \), \( key[v] \) is the minimum weight of any edge connecting \( v \) to a vertex in the tree.
**Homework(**): In the above figure, change $w(a, b)$ to 2, and $w(c, f)$ to 1, then run Prim’s.
MST-Prim(G, w)
1. for each u in V
2. do key[u]<-MaxInt //no idea yet
3. p(u)<-NIL //not connected yet
4. key[r]<-0 //Takes nothing to connect to r
5. Q<-V //Everything thrown into Q, a PQ
6. while !isEmpty(Q) //not done yet
7. do u<-Extract-Min(Q)
8. //take the "cheapest" node that can
9. //be connected to the partial tree
10. for each v in Adj[u]
11. do if v in Q and w(u,v)<key[v]
12. //a better deal found?
13. then p[v]<-u
14. key[v]<-w(u,v)

Thus, we have that

\[ A = \{(v, p[v]) | v \in V - Q - \{r\}\}. \]
Algorithm analysis

If we implement the priority queue, $Q$, with minHeap, then line 5 takes $O(|V|)$ to construct the heap.

Line 6 runs $|V|$ times, for each running, line 7 takes $O(\log |V|)$ with its total being $O(V \log |V|)$.

Lines 10-14 take $O(|E|)$ in total, since each adjacency list is checked out only once, and the sum of the length of all the adjacency lists is $2|E|$.

Line 14 is involved with an implicit Decrease-Key operation, which takes $O(\log |V|)$.

Hence, the total time is $O(|V| \log |V| + |E| \log |V|) = O(|E| \log |V|)$. 

11
Another result

**Corollary 23.2.** Let $G(V, E, \omega)$ be a connected graph with a real-valued weight function $\omega$ defined on $E$. Let $A(\subseteq E)$ be included in some MST for $G$, and let $C(V_C, E_C)$ be a connected component in the forest $G_A(V, A)$. If $(u, v)$ is a light edge connecting $C$ to some other component in $G_A$, then $(u, v)$ is safe for $A$.

**Proof:** Considering the cut $(V_C, V - V_C)$. Since every edge of $A$ either belongs to $C$ or other components of $A$, this cut respects $A$. Moreover, $(u, v)$ is also a light edge for this cut. Hence, by Theorem 23.1, $(u, v)$ is safe for $A$. 


Kruskal’s algorithm

Kruskal’s algorithm is an application of the above corollary, which finds a safe edge to add to the growing forest, containing \( A \), by finding, of all the edges that connect any two trees in the forest, an edge \( e \) with the least weight.

More specifically, let \( C_1 \) and \( C_2 \) be two trees that are connected by \( e \), since \( e \) must be a light edge connecting \( C_1 \) to some other tree, the previous corollary guarantees that \( e \) is safe for \( A \).

This algorithm is greedy since it adds to the forest an edge of the least possible weight.
An example

Homework(*): In Figure 23.4, if we change $w(a, b)$ to 2, and $w(c, f)$ to 1, run Kruskal’s algorithm on this modified graph.
The code

MST-Kruskal(G, w)
1. A<-NULL
2. for each v in V
3. do Make-Set(v)
4. sort the edges E by weight
5. for each edge (u, v) in E
6. do if (u, v) connects two components
7. then add (u, v) into A
8. return A

With proper data structure, this algorithm takes $O(|E| \log |V|)$. 