1 Overview

This lecture introduces two algorithms for minimum spanning tree (MST): Prim’s algorithm and Kruskal’s algorithm. Throughout the notes, we use MST for minimal spanning tree, \( w(e) \) for weight of the edge \( e \), and \( w(T) \) for weight of the tree \( T \). In addition, we assume all the underlying graphs are undirected, connected and weighted.

1.1 Prim’s Algorithm

The pseudocode is:

Algorithm 1 Prim’s Algorithm

```plaintext
1: function PR(G = (V, E))
2:   c[s] = 0
3:   ∀v \neq s \in V, c[v] = +\infty, prev[v] = NIL
4:   E' = \emptyset
5:   H = V
6:   while H \neq \emptyset do
7:     u = deletemin(H)
8:     E' = E' \cup (prev[u], u), if u \neq s
9:     for all (u, v) \in E, where v \in H do
10:        if c[v] > l(u, v) then
11:           c[v] = l(u, v)
12:           prev[v] = u
```

1.1.1 Running Time

Prim’s algorithm has the same running time as Dijkstra’s algorithm, \( O(|E| \log |V|) \), by binary heap. It can be improved to \( O(|E| + |V|\log |V|) \) by Fibonacci heap.

1.1.2 Correctness Proof

We prove that, after each selection of an edge by Prim’s algorithm, there exists a minimum spanning tree \( T = (V, E_t) \) such that \( E' \subseteq E_t \). We prove it by induction. For the base case when \( E' = \emptyset \), it is true. Assume that there exists a minimum spanning tree \( T_n = (V, E_n) \) such that \( E' \subseteq E_n \) when \( E' \) has \( n \) edges. For the \((n+1)th\) selection \( e_{n+1} \), we add \( e_{n+1} \) to \( T_n \). If there is no cycle, \( T_n \) is the tree that we want. If there is a cycle, there exists an edge \( e \neq e_{n+1} \) in the cycle such that \( e \) only has one endpoint in \( V \setminus H_n \), where \( H_n \) denotes \( H \)

\footnote{Some of the material is from a previous note by Yilun Zhou for this course in Fall 2014.}
after $n$ selections, because $e_{n+1}$ only has one endpoint in $V \setminus H_n$. The weight of $e$ must be not smaller than the weight of $e_{n+1}$, otherwise, $e$ should be selected by the algorithm instead of $e_{n+1}$. Therefore, the tree constructed by adding $e_{n+1}$ to $T_n$ and deleting $e$ from $T_n$ is also a minimum spanning tree.

1.2 Kruskal’s Algorithm

The pseudocode is:

Algorithm 2 Kruskal’s Algorithm

1: function KR($G = (V, E)$)
2: $E' = \emptyset$
3: $\forall v \in V$, make $V_i = \{v\}$
4: Sort edges in nondecreasing order
5: for each edge $(u, v) \in E$, taken in nondecreasing order do
6: if adding $(u, v)$ does not form a cycle then
7: $E' = E' \cup (u, v)$
8: Return $E'$

Lemma 1. $G' = (V, E')$ is a spanning tree.

Proof. If $G'$ is not connected, there exist edges that should be selected by the algorithm but not in $E'$, contradiction. Line 6 guarantees that $G'$ is acyclic.

Theorem 2. $G' = (V, E')$ is a minimum spanning tree.

Proof. Let $T$ be the spanning tree generated by Kruskal’s algorithm, and let $T^*$ be a minimal spanning tree. We will prove that $w(T) = w(T^*)$.

If $T = T^*$, then $w(T) = w(T^*)$. If $T \neq T^*$, let $e$ be the edge with minimal weight that is in $T^*$ but not in $T$. Then $T \cup \{e\}$ contains a cycle $C$ such that, by the greedy nature of the construction process of $T$, $w(e) \leq w(f)$ for all $f$ in the cycle $C$. In addition, there exists $f^*$ in $C$ such that $f^*$ is in not $T^*$. Otherwise, $T^*$ will contain a cycle. Let $T_2 = T \setminus \{f^*\} \cup \{e\}$. Then $w(T_2) \geq w(T)$ and $T_2$ has more common edges with $T^*$ than $T$. We can repeat this process and until $T_k = T^*$ for some $k$. Now we have

$$w(T) \leq w(T_2) \leq \cdots \leq w(T_k) = w(T^*).$$

Since $T^*$ is an MST, we must have

$$w(T) = w(T_2) = \cdots = w(T_k) = w(T^*).$$

Therefore $T$ is also an MST.

1.2.1 Running Time

The running time of Kruskal’s algorithm depends on the implementation of “Union” and “Find”. We will discuss it in the next set of notes.