COMP 251: Data Structures and Algorithms Fall 2006

Minimum Spanning Trees

Lecturer/TA: Ethan Kim *November 7th, 2006*

In this lecture, we study two algorithms for computing minimum spanning trees of a graph.

1 Motivation

There are n houses in town. You would like to connect all the houses by power line connections so that each house gets electricity. Some pair of houses costs more to connect, either because they are too far apart, or because they are blocked by water pipelines running in between. Given the cost of building a power line connection between every pair of houses, can you connect them all with minimum overall cost?

2 Minimum Spanning Tree(MST)

Let $G = (V, E)$ be an undirected weighted graph with a weight function $w : E \to \Re$. If $T \subseteq E$ is a tree and every vertex in V is touched by edges in T, we call T a spanning tree. A minimum spanning tree of G is a spanning tree where the sum of edge weights on T is minimized. More formally, we would like to compute $T \subseteq E$, such that

- Every vertex in G is connected to every other vertices.
- T is a tree. (ie. no cycles)
- $\sum_{e \in T} w(e)$ is minimum.

3 Greedy Strategy

Some definitions are necessary.

Definition. *A subset of edges $A \subseteq E$ is* promising *if A is a subset of some minimum spanning tree. We call an edge (u, v)* safe *for A if $A \cup \{(u, v)\}$ is also promising.*

We maintain a loop invariant to show correctness of our algorithm. The strategy is to add a new edge (u, v) to A such that $A \cup \{(u, v)\}$ is also a subset of a MST.

GENERIC-MST(G, w)
1 $A \leftarrow \emptyset$
2 **while** A is not a spanning tree
3 **do** find a safe edge (u, v) for A
4 $A \leftarrow A \cup \{(u, v)\}$
5 **return** A

1. Loop invariant: A is promising prior to each iteration.
2. Initialization: $\emptyset$ is always promising since MST exists for all graphs.
3. Maintenance: Adding a safe edge to promising set creates another promising set.
4. Termination: A promising spanning tree is an MST.

Lemma. *Let $G = (V, E)$ be a connected, undirected graph with weight function w. Let $B \subset V$ be a strict subset of vertices. Let $A \subseteq E$ be a promising set of edges such that no edge in A leaves B. Let e be a lightest edge leaving B. Then $A \cup \{e\}$ is promising.*

Proof. Take an arbitrary MST T such that $A \subseteq T$. If $e \in T$, we're done. So, assume $e \notin T$. Now, we add e to T. Since T is a spanning tree, adding e creates a cycle. Furthermore, since e leaves B, there must exist another edge f in T also leaving B. We remove f from T. Now we obtained a new spanning tree T' by adding e and removing f.

To show T' is also minimum, consider the edges e and f. We have that $w(e) \leq w(f)$, and so $w(T') \leq w(T)$. Thus, T' must be an MST. Furthermore, $A \subseteq T'$ and $e \in T'$. This completes the proof. □

Corollary. *Let $G = (V, E)$ be a connected undirected graph with weight function w. Let $A \subseteq E$ be a promising set, and $C = (V_c, E_c)$ be a connected component(tree) in (V, A). If (u, v) is a shortest edge connecting C to some other component of (V, A), then (u, v) is promising.*

Proof. Apply Lemma with $B = C$. No edge of A leaves B. □

Note that an empty set $\emptyset$ is always a promising set, since there exists an MST for every graphs.

4 Kruskal's Algorithm

Kruskal's algorithm works by "growing the forest".

```
MST-KRUSKAL(G, w)
1  A ← ∅
2  for each v ∈ V
3        do MAKE-SET(v)
4  sort the edges into nondecreasing order by w
5  for each (u, v) ∈ E
6        do if FIND-SET(u) ≠ FIND-SET(v)
7              then A ← A ∪ {(u, v)}
8                   UNION(u, v)
9  return A
```

See Figure 23.4 on page 568 for an example. To see the running time, we can analyze as follows:

- $|V|$ MAKE-SET operations: $O(|V|)$
- sorting edges: $O(|E| \lg |E|)$
- $O(|E|)$ FIND and UNION: $O(|E| \lg^* |V|)$

But $\lg^* |V| = O(\lg |V|) = O(\lg |E|)$, so we have that $O(|V|) + O(|E| \lg |E|) + O(|E| \lg^* |V|) = O(|E| \lg |E|) = O(|E| \lg |V|)$.

5 Prim's Algorithm

Contrary to Kruskal's algorithm (where we grow forests), in Prim's algorithm, we grow a single tree. We first pick some vertex r to be the root of the tree. At every step, we grow the tree by adding a new edge that connects the current tree A to some isolated vertex in (V, A). The algorithm takes three things as input: the graph G, the weight function w, and the root r. A simple algorithm can be devised as follows:

```
MST-SimplePrim(G, w, r)
1  A ← ∅
2  B ← {r}
3  while B ≠ V
4        do find e = (u, v) of min weight such that u ∈ B and v ∈ V − B.
5           A ← A ∪ e
6           B ← B ∪ v
7  return A
```

However, we can implement the **find** operation efficiently using priority queue.

- $key[v]$ defines the minimum weight of an edge from v to any vertex of A.
- $\pi[v]$ defines parent of v in the tree.

Throughout the algorithm, all the vertices that are *not* in the tree reside in the min-priority queue Q based on a *key* field. Hence, we implicitly keep the invariant $A = \{(v, \pi[v]) | v \in V - \{r\} - Q\}$.

```
MST-Prim(G, w, r)
 1  for each u ∈ V
 2        do key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V
 6  while Q ≠ ∅
 7        do u ← Extract-Min(Q)
 8           for each v ∈ Adj[u]
 9                 do if v ∈ Q and w(u, v) < key[v]
10                       then π[v] ← u
11                            key[v] ← w(u, v)
```

See Figure 23.5 on page 571 for an example. To analyze the running time:

- First loop: $O(|V|)$
- EXTRACT-MIN: $O(\lg |V|)$, iterating $O(|V|)$ times
- DECREASE-KEY: $O(\lg |V|)$ for $O(|E|)$ times in total

Overall, we have $O(|V|) + O(|V| \lg |V|) + O(|E| \lg |V|) = O(|E| \lg |V|)$

Note that we can improve this running time even more by using Fibonacci Heaps.

- EXTRACT-MIN: $O(\lg |V|)$ amortized
- DECREASE-KEY: $O(1)$ amortized

Overall, we have $O(|V|) + O(|V| \lg |V|) + O(|E|) = O(|E| + |V| \lg |V|)$.

6 Exercise Problem

Recall the first problem discussed at the beginning of the lecture. In this problem, assume some of the houses are already connected. That is, you are given a set of points in 2D, which represent the position of the houses. You are also given the list of pairs of points, which represent the pairs of houses that are already connected. Define the weight function by the euclidean distances of pairs of points. Given this information, compute the new connections so that all the houses are interconnected using the minimum cost.

Note: The existing connections may contain cycles. Hence, the final output of the problem may not be an MST. However, the new connections must not introduce new cycles.