Minimum Spanning Trees

October 1, 2014

1 Introduction

As we have realized in the class of Graph Algorithms, many problems can be represented as a graph. For example, we could be a electrical machine company with different possible sources of parts to the production of a particular engine. The process and production departments gives a total cost of each of the possible production sites around the world. Then, some clever guy decides to do the following

- Each sites will be represented as a node v_i and belong to a production system T.
- The cost = manufacture + moving $w(v_i, v_j)$ cost is given to move a particular product from v_i to v_j, he points out that we could consider this cost as a bidirectional cost. After all this is an attempt to have a provisional answer.
- We add a node s as the final destination linked to the final assembly nodes with edges that are equal to the moving cost.

Then, Voilà!!! The clever guy runs an algorithm that produces a minimum size tree based in the formulation:

$$min_T \sum_{v_i, v_j \in T} w(v_i, v_j). \quad (1)$$

Then, the guy produces an initial analysis of the cost of producing the electrical engine. OK, How he did? Ah! The Minimum Spanning Tree (MST) problems and solutions.

2 The Setup

Given any graph with undirected edges $G = (V, E)$ such a there is a function $w : P(E) \to \mathbb{R}$, we want to be able to solve the following minimization problem:

$$min_T \sum_{(u,v)\in T} w(u,v)$$
$$s.t.\ T \subseteq E$$

with the particularity that the set of edges T connects all of the vertices's on G. We will examine two algorithms to solve this problem:

- Kruskal's algorithm.
- Prim's algorithm.

And remember the Fibonacci Heap? This will allow to minimize the complexity of one of them.

3 Growing the MST

For the two algorithms considered here, a greedy strategy is used that can be exemplified by the following phrase:

- Prior to each iteration of the algorithm, A is a subset of some minimum spanning tree.

Algorithm 1 Generic-MST

```
Generic-MST(G, w)
        A = ∅
        while A does not form a spanning tree
                find an edge (u, v) that is safe for A
                A = A ∪ (u, v)
        return A
```

Note

The edge that can be added to A is called a safe edge and makes $A \cup \{(u,v)\}$ a subset of a minimum spanning tree.

4 Basic Definitions

We have the following definitions.

Definition 1. A **cut** $(S, V - S)$ is a partition of V.

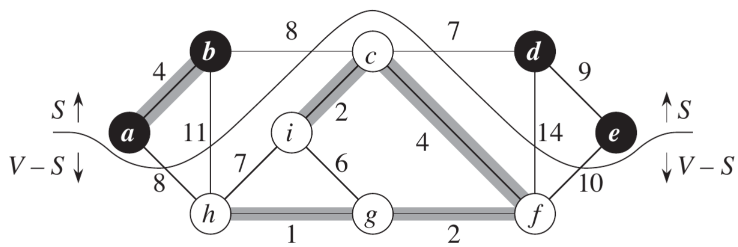

Figure 1: Cut in a Graph

From this, we have the following definitions.

Definition 2. We have the following

1. (u, v) in E **crosses** the cut $(S, V - S)$ if one end point is in S and the other in $V - S$.

2. The cut **respects** A if no edge in A crosses the cut.

3. A **light edge** is a edge crossing the cut with minimum weight with respect the other edges crossing the cut.

5 Recognizing Safe Edges

For this, we will prove the following algorithm.

Theorem 1. *Let G=(V,E) be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let (S,V-S) be any cut of G that respects A, and let (u,v) be a light edge crossing (S,V-S). Then, edge (u,v) is safe for A.*

Proof. Let T be a MST that includes A, we have two cases

Case 1

The light edge is in T, we are done.

Case 2

If this is not the case, we build a spanning tree T' that includes $A \cup \{(u, v)\}$.

1. Build a new spanning tree.

   Simply, we realized that for the cut $(S, V - S)$ exist a edge $(x, y) \in T$ different from (u, v) such that together with (u, v) forms a cycle between u and v (Fig. 2). The edge (x, y) is not in A because the cut respects A, and in addition removing (x, y) breaks set T in two parts (After all (x, y) belongs to a simple path between u and v). Thus adding (u,v) into a new set $T' = T - \{(x, y)\} \cup \{(u, v)\}$ that is a spanning tree.

2. Prove is a MST.

   Since (u, v) is a light edge of $(S, V - S)$ and (x, y) crosses also the cut $(S, V - S)$, we have that

$$w(T') = w(T) - w(x, y) + w(u, v) \leq w(T). \tag{2}$$

   In addition T is a MST or $w(T) \leq w(T')$, thus $w(T) = w(T')$ i.e. T' is a minimum spanning tree.

3. See that (u,v) is a safe edge.

   We have that $A \subseteq T'$, since $A \subseteq T$ and $(x, y) \notin A$. Then, $A \cup \{(u, v)\} \subseteq T'$. Thus, T' is a MST, then by definition (u, v) is safe for A.

$\square$

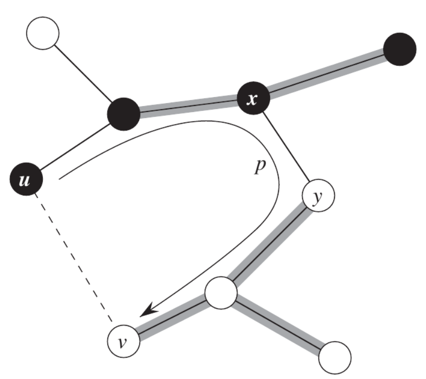

Figure 2: A cycle

Using this theorem, we realized the following about the Generic-MST:

1. The set A at each iteration is always acyclic.
2. The Graph $G_A(V, A)$ is a forest.
3. Each connected component is a tree, if not a contradiction arises.
4. Each iteration connects distinct component of G_A.
5. The while loop only repeats itself $|V| - 1$ times more will produce a contradiction.

Understanding this will allows to understand that corollary 23.2 is why Prim and Kruskal can work.

Corollary. *23.3 Let G=(V,E) be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, and let* $C = (V_c, E_c)$ *be a connected component (tree) 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.*

6 Code and Complexity of Prim's and Kruskal's Algorithms

6.1 Kruskal's Complexity

Based in the code for Kruskal (Algo. 2), it is possible to calculate the following complexity.

Algorithm 2 Kruskal's Algorithm

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

Using Disjoint-set Implementation

Using the Disjoint-Set we have the following complexities

- **Lines 1**. Initialization of A takes $O(1)$.
- **Lines 2-3**. The complexity of these lines is $O(|V|)$ for making all the necessary sets.
- **Lines 4**. Time of sorting the edges is $O(E \lg E)$.
- **Lines 5-8**. They perform $O(E)$ Find-Set and Union operations on the disjoint-set forest. Because these operations can be taken along with the ones in Lines 2-3, we can use the following trick: A constant time $O(1)$ is bounded by the inverse Ackermann function over the set of vertices $O(\alpha(V))$ or $O(V) = O(V\alpha(V))$. Thus

$$O((V + E)\,\alpha(V)) \tag{3}$$

Now assuming the following:

- G is connected therefore

$$|E| \geq |V| - 1 \Longrightarrow O\left(\left(V+E\right)\alpha\left(V\right)\right) = O\left(E\alpha\left(V\right)\right) \tag{4}$$

- In addition, the Ackermann function is bounded by

$$\alpha\left(|V|\right) = O(\lg V) = O\left(\lg E\right) \tag{5}$$

Finally, we have that Kruskal's Algorithm has the following complexity: $O\left(E \lg E\right)$. However, by making the following observation, we have that $|E| < |V|^2$, thus

$$\lg |E| = O\left(\lg V\right) \tag{6}$$

Then, we can restate the running time of Kruskal's algorithm as $O(E \lg V)$**.**

6.2 Prim's Algorithm

We have the following code for Prim's (Algo.).

Algorithm 3 Prim's Algorithm

```
MST-PRIM(G, w, r)
1  for each u ∈ G.V
2      u.key = ∞
3      u.π = NIL
4  r.key = 0
5  Q = G.V
6  while Q ≠ ∅
7      u = EXTRACT-MIN(Q)
8      for each v ∈ G.Adj[u]
9          if v ∈ Q and w(u, v) < v.key
10             v.π = u
11             v.key = w(u, v)
```

Using Min-Heap

Using this data structure, we have that:

- **Lines 1-5**. The priority queue Q can be implemented using Build-Min-Heap in $O\left(V\right)$ time.
- **Line 6**. The while loops is executed $|V|$ times.

- **Line 7**. The Extract-Min is then bounded by $O(V \lg V)$ complexity.
- **Lines 8-11**. They execute $O(E)$ because the total number of elements in the adjacency representation is $2|E|$.
- **Line 9**. The "belonging" operation $\in$ can be implemented using a dirty bit that is turned to zero once the element is remvoed from the priority queue Q.
- **Line 11**. It implies a decreasing of the key $v.key$ using the heapify operation. This can be implemented in $O(\lg V)$ time.

Finally, we have the following complexity for Prim's:

$$O(V \lg V + E \lg V) = O(E \lg V). \tag{7}$$

Using Fibonacci Heap

Here the extract operation can be implemented in $O(\lg V)$ amortized time and the decrease key operation can be implemented in $O(1)$ amortized time. Using these two implementations for the priority queue Q, we have that

$$O(E + V \lg V). \tag{8}$$

Thus, in amortized time Fibonacci heap is better.