COMP. 215 Minimum Spanning Tree

Winter 2016

I. Minimum Spanning Tree in a Weighted Graph

The problem of finding a Minimum Spanning Tree (MST) for a given graph comes up frequently in networking, distributed systems, and other types of problems. both of the algorithms given here, Kruskal's and Prim's, will provide an MST, but keep in mind there may be many spanning trees with the same minimal weight or MST. Key to understanding how MSTs are created is the idea of a "cut" of a graph. All of these algorithms re-establish a cut of the total graph with the MST on one side of the cut and all the vertices and edges *not* in the MST on the other. Edges are added to the tree if and only if the edge crosses this cut.

II. Generic MST

Algorithm 1 Generic-MST(G, w)

```
1: procedure GENERIC-MST((G,w))
2:     A = ∅
3:     while A does not form a spanning tree do
4:         find an edge(u, v) that is safe for A
5:         A = A ∪ {(u, v)}
6:     end while
7:     return A
8: end procedure
```

III. Kruskal's Algorithm

This algorithm is presented on page 631 and forms an MST by creating a forest of trees an safely merging them into a single tree if possible. If this is not possible, the graph is not connected and one spanning tree per disconnected subgraph is formed. The Union(u, v) merges two trees into one tree, A. Kruskal's algorithm forms an MST at an arbitrary root.

Algorithm 2 MST-Kruskal(G, w)

```
1: procedure MST-Kruskal(G[],w[])
2:     A = ∅
3:     for each vertex v ∈ G.V do
4:         Make-Set(v)
5:     end for
6:     sort the edges of G.E in nondecreasing order by weight w
7:     for each edge(u, v) ∈ G.E, taken in nondecreasing order by weight do
8:         if Find-Set(u) ≠ Find-Set(v) then
9:             A = A ∪ (u, v)
10:            Union(u, v)
11:        end if
12:    end for
13:    return A
14: end procedure
```

Time Complexity: Kruskal's Algorithm runs in $O(|E|lg|V|)$

IV. Prim's Algorithm

This algorithm is presented on page 634 and forms an MST by growing an MST from the root (chosen before hand and passed to the algorithm) following much the same form as Dijkstra's algorithm. The adjacency list for the most recently added node is searched to see which path to the adjacent nodes is shortest. All nodes not in the tree are kept in a priority queue (A) as the MST is grown.

Prim's Algorithm grows an MST from a predetermined root. This makes it more useful than Kruskal's when the root of the MST is known.

Algorithm 3 MST-Prim(G, w, r)

```
1:  procedure MST-PRIM(G[],w[],r)
2:      A = ∅
3:      for each vertex u ∈ G.V do
4:          u.d = ∞
5:          u.π = null
6:      end for
7:      r.key = 0
8:      Q = G.V
9:      while Q ≠ ∅ do
10:         u = EXTRACT-MIN(Q)
11:         A = A ∪ {u}
12:         for each v ∈ G.Adj[u] do
13:             if v ∈ Q and w(u, v) < v.key then
14:                 v.π = u
15:                 v.key = w(u, v)
16:             end if
17:         end for
18:     end while
19:     return A
20: end procedure
```

Time Complexity: Prim's Algorithm runs in $O(|E| + Vlg|V|)$