CSc 3102: Spanning Trees

Prim's & Kruskal's Algorithms

- Introduction
- Kruskal's Algorithm
- Prim's Algorithm

1 Introduction

Definition 1. A **spanning tree** is an acyclic subgraph T of a graph G on all the vertices of G. A **minimum spanning tree** is a minimum-weight acyclic subgraph of a weighted graph on all its vertices.

The problem of finding a minimum spanning tree in a weighted graph has many applications. The design of a physical network with feasibility constraint between nodes. Such networks may include telecommunications networks, transportation networks, energy pipelines, VLSI chips, etc. The minimum spanning tree gives a lower-bound on the cost of building the network.

2 Kruskal's Algorithm

```
ALGORITHM: Kruskal(G,w,T)
   {Input: G − a weighted undirected graph with n vertices.
           W − a weight function on E(G), |E| = M
   Output: T − a minimum spanning tree with n vertices.}
   Let T ← {}
   for i ← 1:n−1
      e ← minimum edge in G not in T, T union e is acyclic.
      T ← T ∪ e
   endfor
```

The algorithm has a worst-case performance of $O(|E| \lg |E|)$. Note that the Kruskal's algorithm does not require that the edge selected is incident to some vertex already in the tree. So we may have a forest in an intermediate stage while the algorithm is executing. This algorithm can be implemented by storing the edges in a priority queue using the weights on the edges as key. If the graph is not connected, the algorithm above has to be run in each component of the graph. If the graph is not connected and the algorithm is applied to each component of the graph, we obtain a minimum spanning forest.

3 Prim's Algorithm

```
ALGORITHM: Prim(G,w,T, root)
   Input: G - a weighted undirected graph with n vertices.
          W - a weight function on E the edge set with m edges
          root - some vertex v in V(G)
   Output: T - parent implementation of a minimum spanning tree
           or forest with n vertices.

   Let T[root] ← -1
   for v ← 1:|V| do
      Let v.key ← ∞
   endfor
   Let root.key ← 0
   Let Q ← V(G)
   while Q ≠ {}
      Let u ← Extract-Min(Q)
      for v : Adj[u]
         if v in Q and weight(u,v) < v.key
            Let T[v] ← u
            Decrease-Key(Q,v,w(u,v))
         endif
      endfor
   endWhile
```

Note that when the vertex u is extracted from the priority queue, if its key is still ∞, that means the graph is not connected and a new tree rooted at that vertex becomes the root of a minimum spanning tree for another component. The minimum spanning trees for all components of the graph constitute a minimum spanning forest that contains all the vertices of the G.

The choice of priority queue determines the performance of the algorithm:

1. Indexed Array: $\Theta\left(|V|^2\right)$
2. Binary Heap: $\mathrm{O}\left((|V| + |E|) \lg |V|\right)$
3. Fibonacci Heap: $\mathrm{O}(|E| + |V| \lg |V|)$

Let's look at how these two algorithms compare. Which is more efficient? Well, this depends on the configuration of the graph. When $\frac{m}{n}$ is small, Kruskal is better. In this case, the algorithm has $\mathrm{O}(|E| \lg |V|)$ worst-case complexity. This is significantly better than $\mathrm{O}(|V|^2)$. When the number of edges is quadratic in the number of vertices, Kruskal has worst-case complexity $\mathrm{O}(|E| \lg |V|)$, as compared to the $\mathrm{O}(|V|^2)$ worst-case complexity of Prim.

In practice, the performance of Prim's algorithm also depends on whether the graph is sparse of dense. The array-based implementation is optimal on dense graphs. Binary heap implementation gives a better performance on sparse graphs. In theory, Fibonacci heap should give the best performance but due to additional computation overhead this does not occur in practice.

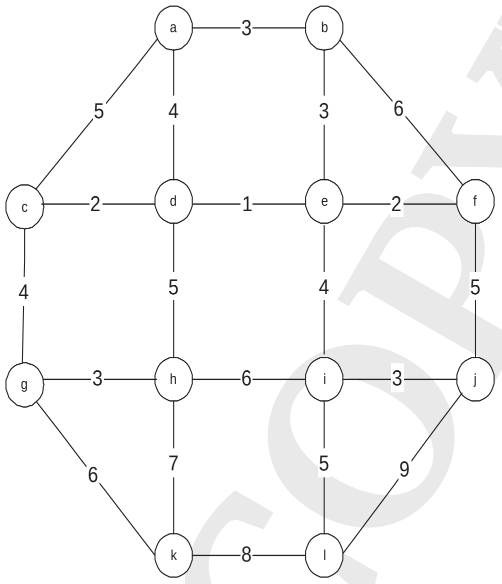

Figure 1: A Connected Graph

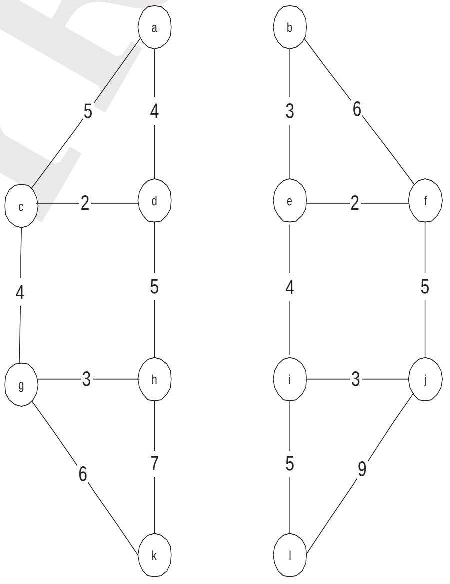

Figure 2: A Disonnected Graph

Problem 1.

1. Draw the minimum spanning tree generated by Kruskal's algorithm on the graph in Figure 1. Assume that during the execution of the algorithm when there are tying edges, the ties are resolved using the lexicographical ordering of the endpoints of the tying edges. Also, assume that when two trees rooted at r_1 and r_2 are merged during the execution of the algorithm and r_1 comes before r_2 in lexicographical order, the root of the merged trees will be r_1. Also, when adding an edge (e_1, e_2) and neither endpoint is already in the tree and e_1 comes before e_2 lexicographically, e_1 becomes the ancestor of e_2 in the tree.

2. What is the weight of the minimum spanning tree? List the edges in the order in which they are added to the minimum spanning tree.

3. Using the same constraints from Problem 1, draw the minimum spanning forest generated by Kruskal's algorithm on the graph in Figure 2. What is the weight of the minimum spanning forest?

4. Draw the minimum spanning tree rooted at $\boldsymbol{a}$ generated by Prim's algorithm on the graph in Figure 1. Assume that when making a choice between adding vertex v_i or v_j with tying keys and vertex v_i comes before v_j in lexicographical order, the tie is resolved by selecting vertex v_i.

5. What is the weight of the minimum spanning tree generated in Problem 4? List the vertices in the order that they were added to the minimum spanning tree in Problem 4.

6. using the constraints given in Problem 4, draw the minimum spanning forest generated by Prim's algorithm on the graph in Figure 2. Assume that when generating the tree in the next component of the graph, the root vertex of the tree comes lexicographically before all other vertices in that component.

7. What is the weight of the minimum spanning forest drawn in Problem 6?

8. Repeat Problems 4 and 5 but using $\boldsymbol{d}$ as the root vertex of the minimum spanning tree generated by Prim's algorithm.