Cosc-320
Minimum Spanning Trees in Connected Weighted Graphs
December 7, 2003

Introduction

These notes on a minimum spanning trees expound on Chapter 16 of "Data Structures with C++ using STL 2nd Edition," William Ford and William Topp, Prentice-Hall, 2002, Section 16.6.)

◇ **Definition**: A *spanning tree* of a connected graph $G = (V, E)$ is a minimal (lowest number of edges) subgraph, G', of G such that G' is connected, $V(G') = V(G)$, and $E(G') \subset E(G)$.

• Less formally, a *spanning tree* of a connected graph G is a tree consisting solely of edges from G and including all the vertices of G.

• Note: The above definition is of a *spanning tree*, not a *minimum spanning tree*.

• If the graph G is not connected, then we get a set of spanning trees, known as a *spanning forest*. We will only consider connected graphs.

• A spanning tree of a graph can be found by doing a breadth-first or depth-first traversal of the graph. The edges traversed form a spanning tree.

• For a graph with $|V|$ vertices, there are exactly $|V| - 1$ edges in any spanning tree.

◇ **Definition**: A *minimum spanning tree* is a spanning tree of a weighted connected graph such that the sum of the weights of the edges in the tree is minimal.

The Algorithms

There are two major algorithms for finding minimum spanning trees in connected weighted graph. These are Prim's algorithm and Kruskal's algorithm. Prim's algorithm is the algorithm in the text. Kruskal's is not in the text.

Both algorithms are greedy and have the following structure:

```
Let T be an empty set that will contain a MST
while (T is not a complete MST)
{
  (u,w) is an edge in E that can be added to T,
     keeping T as a subset of a MST
  Add (u,w) to T.
}
```

Prim and Kruskal differ in the initial definition of `T` and in how they handle selection of the next edge to be added to the MST. In Prim, `T` always contains a single tree that is extended during the algorithm. In Kruskal, `T` is really a forest of trees, each initially with one vertex in it. The forest is combined into a single tree during the algorithm.

Prim's Algorithm

Define a set U that contains any one vertex from V.
Define an empty set T that will contain all the edges of the MST.

```
while ( (V-U) is not empty )
   {
    Choose a lowest cost edge (u,w), u in U and w in V-U
    Add (u,w) to T.
    Add w to U (thus removing it from (V-U)).
   }
```

Kruskal's Algorithm

Store the edges in E in a minimum priority queue Q (lowest weight is chosen first).
Define an empty set T that will contain all the edges of the MST.
Put each vertex in its own set. There will be $|V|$ such sets, initially with one member each.

```
while (the number of edges chosen from Q is less than |V|-1)
    {
     (u,w) = Q.top();  Q.pop();
     if (u's set and w's set are not the same)
       {
         Add (u,w) to T.
         Union the vertex sets of u and w into one set.
       }
    }
```

Performance

Prim's Algorithm

Prim's algorithm works like Dijkstra's minimum path algorithm, greedily extending the tree by selecting the next edge from a priority queue of edges. The performance of Prim's algorithm is therefore $O(|V|+|E|\lg|E|)$

Kruskal's Algorithm

Kruskal's performance depends on how well we can find the set to which a given vertex belongs and how well we can union two such sets. We do not cover this topic in this course (it's known as the "Union-Find" algorithm). Suffice it to say that the operations can be done in $O(\lg|E|)$ time.

We take $O(|V|)$ time to set up the algorithm and $O(|E|\lg|E|)$ time to sort the edges. The `while` loop is executed $O(|E|)$ times and each iteration takes $O(\lg|E|)$ time. Thus the run time of Kruskal is $O(|V|+|E|\lg|E|)$.