Graph Traversals and Connectivity

Background

For non-weighted graphs, we are primarily concerned with can we get from a given node to some other node. This is called the connectivity of that node or of the graph in whole.

The way that we determine connectivity is to start at a given node and walk or traverse the graph edges to see where we can get. This is similar to taking a road trip where you start out on the highway and see what you can find. The algorithms are identical for directed and non-directed graphs. The only difference is the number of edges between them. In non-directed graphs, we essentially have two edges between each connected node – one starting at each node and ending at the other node. For non-directed graphs, we might have one or two edges, depending on how the graph was defined.

The two algorithms we use for traversing a graph are ***breadth first*** or ***depth first***. If we keep track of the edges that we traverse, we can create a minimum spanning tree for the graph. The tree will be different depending on which algorithm we used, but it will have the same connectivity – that is, if we can get from the starting node to some other node using breadth first, we can also get there using depth first – it just might be by a different path.

Breadth First

In a breadth first traversal of a graph, you start at a given node, mark it as visited, and visit all of its neighbors. Any neighbors who have not been visited before (check the visited flag) are added to a FIFO. The next node is removed from the FIFO and all of its unvisited neighbors are added. This continues until the FIFO is empty.

Displaying the Nodes as they are processed generates a breadth first listing.

Displaying the edges as they are generated generates a breadth first minimum spanning tree.

The algorithm can be stated in pseudo code as:

```
 mark starting node as visited and push on FIFO

 while FIFO not empty
     current = FIFO head
     remove FIFO head

     foreach edge starting at current
         if edge.otherEnd is non-visited
             add otherEnd to FIFO
             process otherEnd (depending on algorithm)
             mark otherEnd as visited

reset all nodes to not visited
```

Depth First

Similarly to breadth first, you can traverse a graph depth first. Here you start at a given node, mark it as visited, and save it on a LIFO. Consider the LIFO top, if it has a non-visited neighbor, mark that neighbor as visited and push it on the LIFO. If it does not have a non-visited neighbor, then pop it.

Just as with breadth first, you can generate a list of nodes as visited or a minimum spanning tree by listing edges as traversed.

The algorithm can be stated in pseudo code as:

```
 mark starting node as visited and push on LIFO

 while LIFO not empty
     current = LIFO top
     remove current

     foreach edge starting at current
         if edge.otherEnd is non-visited
             push current
             push otherEnd
             process otherEnd (depending on algorithm)
             mark otherEnd as visited
             break

reset all nodes to not visited
```

Connectivity Table

Since all connections are both ways in non-directed graphs, it does not matter which node we start at to see what nodes can be reached. For a directed graph, this is not true. The starting node determines the possible nodes that can be reached.

A connectivity table is a way to show what nodes can be reached from each starting node in the graph. It can be generated by either a depth first or a breadth first traversal starting at each node in the graph in turn and then displaying a list of those nodes that can be reached from that node.