Weighted Graphs

Overview

In the previous section we considered both directed and non-directed graphs with no weights on the edges. That is, all we were concerned about was whether an edge existed from some node A to some node B.

In weighted graphs, there is a value assigned to the edges. This might be miles between cities on a highway or cost to fly to some destination on an airline.

Nomenclature

The same nomenclature is used as in non-weighted graphs, except that we add the concept of a min-weight spanning tree.

Weighted graphs may be either directed or non-directed.

Representation

Nodes are represented in the same way is in non-weighted graphs.

There are two ways that edges are represented in a program.

The first way to represent the edges in a graph is by using an ***adjacency list***. In an adjacency list, the edges are structures containing a weight, the identifier for the end of the edge, and a pointer to the next edge in the list. For a non-directed graph, there will be an edge in the list for the start and an edge in the list for the end. For a directed graph, each edge is present only once.

The second way to represent the edges is by using an ***NxN adjacency matrix*** where a connection between node A and node B is represented by a non-zero weight at the location array[A][B]. For a non-directed graph, there is symmetry around the diagonal where the value at array[A][B] == array[B][A].

Example Code

The only change from a non-weighted graph is that when you are creating an edge, you need to input a weight. This code sets up bi-directional edges, as in a non-directed graph.

```
        // when using an edgelist, you have a set of links
    class Edge
        endIndex // the index in the Node array of the other end of the edge
        weight   // for a weighted graph, the weight of this edge
        next     // a reference or pointer to the next edge in the list


        // add a new edge to the graph
        // return false and do nothing if either end is invalid
        // otherwise add to both nodes edge lists and to the matrix
    addEdge(startNode name, endNode name, weight)
           // ignore edges from a node to itself
        if starts == ends
            return false

        startIndex = findNode(starts)
        endIndex = findNode(ends)

           // if either name is not in the node array
        if startIndex == -1 or endIndex == -1
            return false

            // set both links in edgeMatrix
        edgeMatrix[startIndex][endIndex] = weight
        edgeMatrix[endIndex][startIndex] = weight

            // create two new edges (one for each direction)
            // and add one to each nodes list of edges
            // Note that C++ needs to use pointers for this
        Edge startEnd = new Edge
        startEnd.endIndex = endIndex
        startEnd.weight = weight
        startEnd.next = nullptr
        startEnd.next = nodeList[startIndex].connects
        nodeList[startIndex]->connects = startEnd

        Edge endStart = new Edge
        endStart.endIndex = startIndex
        endStart.weight = weight
        endStart.next = nullptr
        endStart.next = nodeList[endIndex].connects
        nodeList[endIndex]->connects = endStart

        return true
```