# Weighted Graph Algorithms

## Background

We have considered simple graph traversals (breadth first and depth first). These are the same for both directed and non-directed graphs. We can also use traversals on weighted graphs to determine what the connections are – for example you could say that we can get to both Seattle and Portland on I5 from Eugene, without considering how long it might take. This is the connectivity approach to the graph.

For weighted graphs, the edges provide additional information. For example, we know that the mileage from Eugene to Portland on I5 is about 110 miles, while Seattle is about 280 miles.

One of the things we want to know about weighted graphs is what is the shortest path from node A to node B. For example, we need to fly from Eugene to Atlanta and we want to spend the least amount of time in the air. For this we use a shortest path algorithm as described below. If you want to know the shortest path from every node in a graph to every other node, one solution is Dijkstra's Algorthm. Here is a Wikipedia link https://en.wikipedia.org/wiki/Shortest_path_problem

Another thing we might want to know is what is the minimum cost spanning tree for the graph. This would be useful if we needed to connect a set of cities with fiber optic cable, were the cost is determined by the number of total miles of cable that must be laid. For this we use a minimum spanning tree algorithm as described below. Again, here is a Wikipedia link https://en.wikipedia.org/wiki/Minimum_spanning_tree

## Shortest Path Algorithm

To determine the shortest path between two nodes, we follow all paths from the beginning to the end and keep track of the cost along each path. (This algorithm description is from geeksforgeeks.org).

1) **Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.**

2) **Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.**

3) **While sptSet doesn't include all vertices**

    a. **Pick a vertex u which is not yet in sptSet and has the minimum distance.**
    b. **Add u to sptSet.**
    c. **Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u–v, is less than the distance value of v, then update the distance value of v.**

## Minimum Spanning Tree

While we can create a spanning tree using either depth first or breadth first (LIFO or FIFO), for a minimum spanning tree we use a priority queue to keep track of the edges.

We start at a given node, add all edges beginning at that node to the priority queue, then while the queue is not empty we take the shortest path from the queue. Process the node at its end and continue as shown below.

```
mark starting node as visited and add all its edges to PQueue

while PQueue not empty
    get and remove shortest edge
    set current to destination of that path

    foreach edge starting at current
        if edge->otherEnd is non-visited
            if any edges in PQueue terminate at otherEnd
               if the edge already in PQueue is longer
                   delete the old edge
                   add the new edge
            process otherEnd (depending on algorithm)
            mark otherEnd as visited

reset all nodes to not visited
```