Shaun Carland
COS 485
Program 3: Steiner Tree

The first step of the algorithm is to construct an adjacency matrix (adj[][]) of the Graph g. In building the adjacency matrix, the algorithm iterates over every vertex in the graph, checking to see what other vertices it is connected to. For example, if vertex A connected to vertex B, adj[A][B] would equal the value of the edge that connects A to B. If vertex A is equal to vertex B, adj[A][B] would equal 0. If vertex A does not connect with vertex B, then adj[a][b] would be infinity. The time to build the adjacency matrix would be O(V^2), since V vertices are compared to V other vertices.

The second step of the algorithm is to build the initialized path matrix (path[][]). The path matrix will be used later on in the algorithm, but in the mean time, it must be initialized. To initialize the path matrix, V vertices are compared to V vertices. If two vertices A and B connect, path[A][B] would equal the edge that connects the two vertices. Otherwise, if vertices A and B are equal or A and B do not connect, path[A][B] is assigned a value of -1 (as a flag). The time to initialize the path matrix would be O(V^2), since V vertices are compared to V other vertices.

The third step of the algorithm is to use Floyd's all pairs shortest path algorithm to build a matrix SP[i][j] that contains the distance of the shortest path between vertex i and vertex j. This is accomplished by searching for all non-direct paths between two vertices that have a smaller weight than the best path found so far between the vertices. If such path is found, the optimized weight is put into the matrix. At the end of the algorithm, SP[A][B] corresponds to the optimized path between vertex A and vertex B.

While the Floyd's algorithm is being run, the program simultaneously updates the path matrix so that we can reconstruct the path between two vertices. Since we want to be able to reconstruct the shortest path between any nodes i, j, we want to look at the corresponding value in path[i][j]. If the value is zero, then there is no path between i and j. Else, the value denotes the predecessor of j on the path from i to j. This procedure can be repeated while the preceding node is not equal to i in order to reconstruct the shortest path between i and j.

The runtime of building SP[][] and path[][] is O(V^3), since the runtime of Floyd's algorithm is O(V^3).

At this point, the algorithm has the ability to find the shortest path between any two vertices in constant time due to the construction of the SP and path data structures. Using these data structures, a modified version of Prim's algorithm will be used to find the minimum spanning tree that connects all of the target vertices. One of the target vertices is picked (in the code, the first target is hardwired to be picked. But this choice is arbitrary and doesn't affect the performance/result of the algorithm).

A data structure (implemented as an Array List in the code) called “visited” contains the targets that have been picked. While there are still targets to connect, our goal is to find the optimal path that connects a visited target (the source) with an unvisited target (the sink). The set of visited targets are selected as a candidate source one by one. Then, the selected visited target iterates over every other target as a candidate sink. If the current candidate sink has already been visited or is equal to the current candidate source, it is ignored. Otherwise, the weight of the shortest path between the candidate source and the candidate sink is calculated. If the weight of the path between the candidate source and this specific candidate sink is smaller than the smallest path that has been previously found between the candidate source and the previously checked candidate sinks, the candidate sink and the weight is stored.

Once this process is complete, for each visited target, we know the optimized path that connects it to an unvisited target. We then choose the smallest path that connects a visited vertex with an unvisited vertex. Once the path has been determined, it is drawn on the graph data structure by coloring the edges traversed and setting their mark as 1. While the algorithm is drawing on the graph, it it encounters an edge that has a mark of zero (e.g. has not been traveled on through any previous iteration of prims()), the weight of that edge is added to the total distance traveled. Once all of the target vertices are connected, the total distance traveled is returned.

The runtime of Prim’s algorithm will equal $O(T^2 + P)=O(T^2)$, where T is the number of targets and P is the number of shortest paths between all of the targets.

Overall Run Time:
The overall run time will be $O(V^2) + O(V^2) + O(V^3) + O(T^2) = O(V^3)$.