CS577
Homework #6, Problem #4
Sreeram Danda

#4) Min Edges, Min Weight Path Algorithm

```
path (graph G, vertex s, vertex v)
      Q = set of all vertices in G;

      for(vertex x: G)
            if(x != s)
                  x.distance = INFINITY;
                  x.numEdges = INFINITY;
                  x.previous = null;
            Q.add(x)

      while(Q != empty)
            u = vertex with min distance in Q
            Q.remove(u);

            for(each neighbor n of u)
                  tempDistance = u.distance + distance_between(n, u)
                  tempNumEdges = u.numEdges + 1;
                  if(tempDistance < n.distance)
                        n.distance = tempDistance
                        n.numEdges = tempNumEdges
                        n.previous = u;
                  else if(tempDistance == n.distance)
                        if(tempNumEdges < n.numEdges)
                              n.numEdges = tempNumEdges
                              n.previous = u;

      currentVertex = v;
      while(currentVertex.previous != s)
            shortestPathMinEdges += currentVertex;
            currentVertex = currentVertex.previous;
      shortestPathMinEdges += source;

      return shortestPathMinEdges;
```

#4)Correctness

This algorithm follows dijkstra's shortest path algorithm very closely. The major difference being that at each vertex we also compute the number of edges that it takes to get to that vertex. Consider a vertex v1, the number of edges it takes to get to v1 is equal to the number of edges it takes to get to its parent

vertex plus the additional edge to get to v1. Thus each vertex is correctly assigned the right number of edges. Furthermore, the algorithm updates the number of edges required when it gets to a vertex that has already been seen and has a distance that is equal to a new path with the same distance. At this moment both paths have the same distance to the vertex but we need to ensure to pick the path with the least number of edges. To do this the algorithm compares the two paths and updates the number of edges to the smaller one. The algorithm does not update the number of edges if the new path is longer because it then is no longer the shortest distance regardless of the number of edges. At the end of the first while loop, each vertex in the graph has the shortest distance and number of edges required assigned to it. Additionally, each vertex is assigned a previous vertex to which the shortest path is obtained from. This is assigned and updated exactly like the number of edges. Thus to acquire the shortest path, one simply needs to start at the target vertex and continually log the current vertex and keep track of all previous vertices until the source vertex is reached. At this point we can return the shortest path of minimum edges.

#4)Time Complexity

This algorithm follows closely to dijkstra's shortest path algorithm. The major difference being that at each vertex we also keep track of the number of edges that have been used to get to that vertex. Since this step is all done in constant time the running time of the algorithm is the same as dijkstra's shortest path algorithm of O(|E|+|V| log(|V|)). Additionally, we perform a reverse search to obtain the path itself which takes O(|V|) time. Therefore, the runtime of the algorithm is O(|E|+|V|+|V|log(|V|)). Simplifying we get O(|E|+|V|log(|V|)).