Experiment No: 8

Title: Write program to find Single Source Shortest Path

Theory/Description:

Dijkstra’s algorithm is a graph algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, outputting a shortest path tree. This algorithm is often used in routing. For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path first is widely used in network routing protocols, most notably IS-IS and OSPF (Open Shortest Path First). Algorithm: Let's call the node we are starting with an initial node. Let a distance of a node X be the distance from the initial node to it.

Algorithm

```
ShortestPaths(V,cost,dist, n)
// dist[j] 1,<j <n, is set to the length of the shortest path from vertex v to vertex j in a digraph
//G with n vertices.
// dist[v] is set to zero. G is represented by its cost adjacency matrix cost[l :n, 1:n].
{
     for i :=1 to n do
     {// Initialize S.
     S[i]:=false;
     dist[i] :=cost[v,i];
     }
     S[u]:=true;
     dist[v] :=0.0;// Put v in S.
     for num :=2 to n - 1do
       { // Determine n – 1 paths from v.
          Choose u from among those vertices not in S such that dist[u] is minimum;
          S[u]:=true;// Put u in S.
              for (each w adjacent to u with S[w]= false) do
              // Update distances.
                   if (dist[w] >dist[u]+cost[u,w]) then
                       dist[w] :=dist[u]+cost[u,w];
     }
}
```

Complexity

The time taken by the algorithm on a graph with n vertices is O(n^2). To see this, note that first for loop takes 0(n) time. The second for loop is executed n - 2 times. Each execution of this loop requires O(n) times and again at the for loop to update distances So the total time for this loop is O(n^2). In case a list t of vertices currently not in s is maintained, then the number of nodes on this list would at any time be n - num. Hence asymptotic time would remain O(n^2).