**Aim :** Write a program to find single source shortest path.

**Theory :**

Dijkstra's shortest path algorithm, a greedy algorithm that efficiently finds shortest paths in a

graph. Dijkstra's algorithm solves the single-source shortest-path problem when all edges have non-negative weights. It is a greedy algorithm and similar to Prim's algorithm. algorithm starts at the source vertex, s, it grows a tree, T, that ultimately spans all vertices reachable from S. Vertices are added to T in order of distance i.e., first S, then the vertex closest to S, then the next closest, and so on. Following implementation assumes that graph G is represented by adjacency lists.

**Algorithm :**

1. Let „adj‟ be the adjacency matrix of graph „G‟ having „v‟ vertices numbered from 1 to v

and having „e‟ edges.

2. Let distance, path and visited be arrays of „v‟ elements each.

3. Array „distance‟ is initialized to ∞, while „path‟ array and „visited‟ array is initialized to

0.

4. Let current = source vertex.

5. visited[current]=1

6. T = 0

7. Let number of vertices already added to the trace be given as nv and let nv=1.

8. Repeat steps 9 to 13 while nv ≠ v.

9. for i= 1 to v

if ( adj[current][i] ≠ 0)

if ( visited[i] ≠ 1)

if ( distance[i] > adj[current][i])

{

distance[i] = adj[current][i] + T

path[i]=current

}

10. min = ∞ (in program min=32767)

for i= 1 to v

if ( visited[i] ≠ 1)

if ( distance[i] < min)

{

min = distance[i]

current = i

}

11. visited[current]=1

12. nv = nv+1

13. T = distance[current]

14. Let dest be the destination vertex

Shortest distance from the sorce vertex to the destination vertex = distance[dest]

y = dest

do

{

x = path[y]

Display vertex i is connected to vertex x.

}

15. while y ≠ sorce vertex repeat steps 2 to 15 for each vertex as the sorce vertex.

16. Stop.

**Conclusion:** An upper bound of the running time of Dijkstra's algorithm on a graph with edges *E* and vertices *V* can be expressed as a function of |*E*| and |*V*| using the Big-O notation.

For sparse graphs, that is, graphs with fewer than |*V*|2 edges, Dijkstra's algorithm can be

implemented more efficiently by storing the graph in the form of adjacency lists and using a

binary heap, pairing heap, or Fibonacci heap as a priority queue to implement the Extract-Min function efficiently. With a binary heap, the algorithm requires *O*((|*E*|+|*V*|) log |*V*|) time (which is dominated by *O*(|*E*| log |*V*|) assuming every vertex is connected, that is, |*E*| ≥ |*V*| - 1), and the Fibonacci heap improves this to *O*( | *E* | + | *V* | log | *V* | ).