**G. H. RAISONI COLLEGE OF ENGG., NAGPUR**

**(An Autonomous Institute)**

**Department of Computer Science & Engg.**



**Date: 26 -08-2021**

**Practical Subject: DCN Pr A2**

**Session: 2021-22**

**Student Details:**

| **Roll Number** | 44 |
| --- | --- |
| **Name** | Anand Suralkar |
| **Semester** | 9th |
| **Section** | A |
| **Batch** | CSE |

**Practical Details: Practical Number-4;**

| Practical Aim | **Implement Dijkstra’s Routing algorithm to calculate shortest path.** |
| --- | --- |
| Theory & Syntax | One algorithm for finding the shortest path from a starting node to a target node in a weighted graph is Dijkstra’s algorithm. The algorithm creates a tree of shortest paths from the starting vertex, the source, to all other points in the graph.  Dijkstra’s algorithm, published in 1959 and named after its creator Dutch computer scientist EdsgerDijkstra, can be applied on a weighted graph. The graph can either be directed or undirected. One stipulation to using the algorithm is that the graph needs to have a nonnegative weight on every edge.  Suppose a student wants to go from home to school in the shortest possible way. She knows some roads are heavily congested and difficult to use. In Dijkstra's algorithm, this means the edge has a large weight--the shortest path tree found by the algorithm will try to avoid edges with larger weights. If the student looks up directions using a map service, it is likely they may use Dijkstra's algorithm, as well as others.  For example, find the shortest path from home to school in the following graph:  The shortest path, which could be found using Dijkstra's algorithm, is  Dijkstra's Algorithm:  Dijkstra’s algorithm finds a shortest path tree from a single source node, by building a set of nodes that have minimum distance from the source.  The graph has the following:  vertices, or nodes, denoted in the algorithm by v or u;  weighted edges that connect two nodes: (u,v) denotes an edge, and w(u,v) denotes its weight. In the diagram on the right, the weight for each edge is written in gray.    This is done by initializing three values:  dist, an array of distances from the source node s to each node in the graph, initialized the following way: dist(s) = 0; and for all other nodes v, dist(v) = ∞. This is done at the beginning because as the algorithm proceeds, the dist from the source to each node v in the graph will be recalculated and finalized when the shortest distance to v is found  Q, a queue of all nodes in the graph. At the end of the algorithm's progress, Q will be empty.  S, an empty set, to indicate which nodes the algorithm has visited. At the end of the algorithm's run, S will contain all the nodes of the graph.    The algorithm proceeds as follows:  While Q is not empty, pop the node v, that is not already in S, from Q with the smallest dist (v). In the first run, source node s will be chosen because dist(s) was initialized to 0. In the next run, the next node with the smallest dist value is chosen.  Add node v to S, to indicate that v has been visited  Update dist values of adjacent nodes of the current node v as follows: for each new adjacent node u,  if dist (v) + weight(u,v) < dist (u), there is a new minimal distance found for u, so update dist (u) to the new minimal distance value;  otherwise, no updates are made to dist (u).  The algorithm has visited all nodes in the graph and found the smallest distance to each node. dist now contains the shortest path tree from source s.  Note: The weight of an edge (u,v) is taken from the value associated with (u,v) on the graph. |
| Program | #include<limits.h>  #include<stdio.h>  #define V 9  intminDistance(intdist[],boolsptSet[]){  int min = INT\_MAX,min\_index;  for(int v =0; v < V; v++)  if(sptSet[v]==false&&dist[v]<= min)  min=dist[v],min\_index= v;  returnmin\_index;  }  intprintSolution(intdist[],int n){  printf("Vertex Distance from Source\n");  for(int i =0; i < V; i++)  printf("%d \t %d\n", i,dist[i]);  }  voiddijkstra(int graph[V][V],intsrc){  intdist[V];  boolsptSet[V];  for(int i =0; i < V; i++)  dist[i]= INT\_MAX,sptSet[i]=false;  dist[src]=0;  for(int count =0; count < V -1; count++){  int u =minDistance(dist,sptSet);  sptSet[u]=true;  for(int v =0; v < V; v++)  if(!sptSet[v]&& graph[u][v]&&dist[u]!= INT\_MAX &&dist[u]+ graph[u][v]<dist[v])dist[v]=dist[u]+ graph[u][v];  }  printSolution(dist, V);  }  int main(){  int graph[V][V]={{0,6,0,0,0,0,0,8,0},  {6,0,8,0,0,0,0,13,0},  {0,8,0,7,0,6,0,0,2},  {0,0,7,0,9,14,0,0,0},  {0,0,0,9,0,10,0,0,0},  {0,0,6,14,10,0,2,0,0},  {0,0,0,0,0,2,0,1,6},  {8,13,0,0,0,0,1,0,7},  {0,0,2,0,0,0,6,7,0}  };  dijkstra(graph,0);  return0;  } |
| Output |  |
| Conclusion | Thus, we have understood and implemented the Dijkstra’s Routing algorithm to calculate shortest path |