Data Structures Odyssey: Exploring the Foundations of Computing

Ex. No.:14	Graph Traversal	Date: 16/05/2024

Write a C program to create a graph and find the shortest path using Dijkstra's Algorithm.

Algorithm:

1) Start
2) Initialize the distance from the start node to all other nodes as infinity, except for the start node itself which is 0.
3) Create a priority queue to store nodes and their distances from the start node.
4) Add the start node to the priority queue with a distance of 0.
5) While the priority queue is not empty:
a. Extract the node with the smallest distance from the priority queue.
b. For each neighbor of the extracted node:
i. Calculate the distance from the start node to the neighbor through the extracted node.
ii. If this distance is smaller than the current distance stored for the neighbor, update the distance. iii. Add the neighbor to the priority queue with the updated distance.
6) Repeat step 4 until all nodes have been processed.
7) The distances stored for each node after the algorithm completes represent the shortest path from the start node to that node. 8) Stop

PROGRAM;

```
#include <stdio.h>

#include <limits.h>
```

Data Structures Odyssey: Exploring the Foundations of Computing

```
#define MAX_VERTICES 100
int minDistance(int dist[], int sptSet[], int vertices) {
int min = INT_MAX, minIndex; for (int v = 0; v <
vertices; v++) { if (!sptSet[v] && dist[v] < min) { min
= dist[v]; minIndex = v;
}
}
return minIndex;
}
void printSolution(int dist[], int vertices) {
printf("Vertex \tDistance from Source\n");
for (int i = 0; i < vertices; i++) { printf("%d
\t%d\n", i, dist[i]);
}
}
void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int src, int
vertices) { int dist[MAX_VERTICES]; int sptSet[MAX_VERTICES];
for (int i = 0; i < vertices; i++) {
dist[i] = INT_MAX; sptSet[i] =
0;
} dist[src] =
0; for (int
count = 0;
count <
vertices - 1;
count++) {
int u =
minDistance
(dist, sptSet,
vertices);
sptSet[u] =
1;
```

```
for (int v = 0; v < vertices; 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, vertices);
}
int main() { int vertices; printf("Input the
number of vertices: "); scanf("%d", &vertices); if
(vertices <= 0 || vertices > MAX_VERTICES) {
printf("Invalid number of vertices. Exiting...\n");
return 1;
}
int graph[MAX_VERTICES][MAX_VERTICES]; printf("Input the adjacency matrix
for the graph (use INT_MAX forinfinity):\n"); for (int i = 0; i < vertices; i++) {

for (int j = 0; j < vertices; j++) { scanf("%d",
&graph[i][j]);
}
}
int source;
printf("Input the source vertex: ");
scanf("%d", &source); if (source < 0 ||
source >= vertices) { printf("Invalid source
vertex. Exiting...\n"); return 1;
}
dijkstra(graph, source, vertices); return
0;
}
```

OUTPUT:

Data Structures Odyssey: Exploring the Foundations of Computing

```
Input the number of vertices: 3
Input the adjacency matrix for the graph (use INT_MAX forinfinity):
1
2
3
4
5
6
7
8
9
Input the source vertex: 1
Vertex  Distance from Source
0       4
1       0
2       6
```

RESULT: Thus, the program was successfully executed.