



**S. B. JAIN INSTITUTE OF TECHNOLOGY,
MANAGEMENT & RESEARCH, NAGPUR.**

Practical No. 4

Aim: Create a program that finds the shortest paths from a given source vertex in a weighted connected graph to all other vertices using Dijkstra's algorithm.

Name of Student: Shrutika Pradeep Bagdi

Roll No: CS22130

Semester/Year: V/III

Academic Session:2024-2025

Date of Performance:

Date of Submission:

AIM: Create a program that finds the shortest paths from a given source vertex in a weighted connected graph to all other vertices using Dijkstra's algorithm.

OBJECTIVE/EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- To understand the concepts of single source shortest path.
- To implement the Dijkstra algorithm to find single source shortest path.

THEORY:

Dijkstra's algorithm is a popular algorithm for solving many single-source shortest path problems having non-negative edge weight in the graphs i.e., it is to find the shortest distance between two vertices on a graph. It was conceived by Dutch computer scientist Edsger W. Dijkstra in 1956.

The algorithm maintains a set of visited vertices and a set of unvisited vertices. It starts at the source vertex and iteratively selects the unvisited vertex with the smallest tentative distance from the source. It then visits the neighbors of this vertex and updates their tentative distances if a shorter path is found. This process continues until the destination vertex is reached, or all reachable vertices have been visited.

The need for Dijkstra's algorithm arises in many applications where finding the shortest path between two points is crucial.

For example, It can be used in the routing protocols for computer networks and also used by map systems to find the shortest path between starting point and the destination.

ALGORITHM:

Algorithm 1: Dijkstra's algorithm

Input: Graph $G = (V, E)$

- 1 $(\forall x \neq s) dist[x] = +\infty$ //Initialize dist[]
- 2 $dist[s] = 0$
- 3 $S = \emptyset$
- 4 $Q = V$ // Keyed by $dist[]$.
- 5 **while** $Q \neq \emptyset$ **do**
- 6 $u = extract_min(Q)$
- 7 $S = S \cup \{u\}$
- 8 **foreach** vertex $v \in Adj(u)$ **do**
- 9 $dist[v] = \min(dist[v], dist[u] + w(u, v))$
- 10 // "Relax" operation.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define MAX_VERTICES 100
#define INF INT_MAX

void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int numVertices, int start, int dist[]) {
    int visited[MAX_VERTICES] = {0};

    for (int i = 0; i < numVertices; i++) {
        dist[i] = INF;
    }
    dist[start] = 0;

    for (int count = 0; count < numVertices - 1; count++) {
        int min = INF;
        int u = -1;
        for (int v = 0; v < numVertices; v++) {
            if (!visited[v] && dist[v] <= min) {
                min = dist[v];
                u = v;
            }
        }
    }
}
```

```
        if (u == -1) {
            break;
        }
        visited[u] = 1;
        for (int v = 0; v < numVertices; v++) {
            if (!visited[v] && graph[u][v] && dist[u] != INF && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }
}
```

```
int main() {
    int numVertices;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the adjacency matrix (%d x %d):\n", numVertices, numVertices);
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

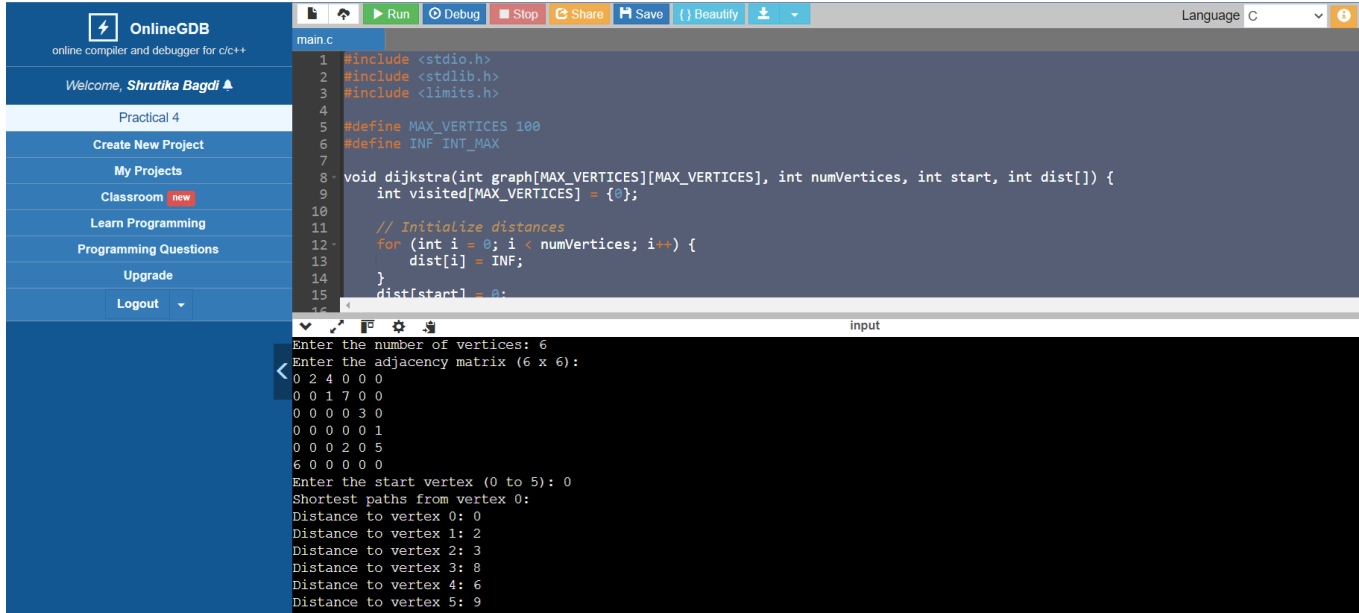
    int startVertex;
    printf("Enter the start vertex (0 to %d): ", numVertices - 1);
    scanf("%d", &startVertex);

    if (startVertex < 0 || startVertex >= numVertices) {
        printf("Invalid start vertex.\n");
        return 1;
    }

    int dist[MAX_VERTICES];
    dijkstra(graph, numVertices, startVertex, dist);
    printf("Shortest paths from vertex %d:\n", startVertex);
    for (int i = 0; i < numVertices; i++) {
        if (dist[i] == INF) {
            printf("Distance to vertex %d: INF\n", i);
        } else {
            printf("Distance to vertex %d: %d\n", i, dist[i]);
        }
    }

    return 0;
}
```

INPUT & OUTPUT WITH DIFFERENT TEST CASES:



The screenshot displays the OnlineGDB interface. On the left is a sidebar with navigation links: 'Welcome, Shrutika Bagdi', 'Practical 4', 'Create New Project', 'My Projects', 'Classroom' (marked as new), 'Learn Programming', 'Programming Questions', 'Upgrade', and 'Logout'. The main area shows a C program implementing Dijkstra's algorithm. The code includes headers for stdio, stdlib, and limits, defines MAX_VERTICES as 100 and INF as INT_MAX, and contains a void function 'dijkstra' that takes a graph, number of vertices, start vertex, and a distance array. It initializes distances to INF and sets the start distance to 0. The output window at the bottom shows the program's execution: it prompts for the number of vertices (6) and the adjacency matrix (6x6), which is entered as a 6x6 grid of integers. It then prompts for the start vertex (0) and outputs the shortest paths from vertex 0, listing distances to vertices 0 through 5.

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <limits.h>
4
5 #define MAX_VERTICES 100
6 #define INF INT_MAX
7
8 void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int numVertices, int start, int dist[]) {
9     int visited[MAX_VERTICES] = {0};
10
11     // Initialize distances
12     for (int i = 0; i < numVertices; i++) {
13         dist[i] = INF;
14     }
15     dist[start] = 0;
16
17     // Dijkstra's algorithm implementation
18 }
19
20 int main() {
21     int numVertices;
22     int graph[MAX_VERTICES][MAX_VERTICES];
23     int start;
24     int dist[MAX_VERTICES];
25
26     printf("Enter the number of vertices: ");
27     scanf("%d", &numVertices);
28     printf("Enter the adjacency matrix (6 x 6): ");
29     for (int i = 0; i < numVertices; i++) {
30         for (int j = 0; j < numVertices; j++) {
31             scanf("%d", &graph[i][j]);
32         }
33     }
34     printf("Enter the start vertex (0 to 5): ");
35     scanf("%d", &start);
36     dijkstra(graph, numVertices, start, dist);
37     printf("Shortest paths from vertex 0:\n");
38     for (int i = 0; i < numVertices; i++) {
39         printf("Distance to vertex %d: %d\n", i, dist[i]);
40     }
41     return 0;
42 }
```

input

```
Enter the number of vertices: 6
Enter the adjacency matrix (6 x 6):
0 2 4 0 0 0
0 0 1 7 0 0
0 0 0 0 3 0
0 0 0 0 0 1
0 0 0 2 0 5
6 0 0 0 0 0
Enter the start vertex (0 to 5): 0
Shortest paths from vertex 0:
Distance to vertex 0: 0
Distance to vertex 1: 2
Distance to vertex 2: 3
Distance to vertex 3: 8
Distance to vertex 4: 6
Distance to vertex 5: 9
```

CONCLUSION:

DISCUSSION AND VIVA VOCE:

- Explain single source shortest path problem.
- Discuss the greedy method.
- Explain Dijkstra algorithm and its advantages and disadvantage.
- Discuss the complexity of Dijkstra algorithm.

REFERENCES:

- <https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/>
- <https://www.javatpoint.com/dijkstras-algorithm>
- <https://www.programiz.com/dsa/dijkstra-algorithm>
- https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php

