Bharatiya Vidya Bhavan's

Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

SE – COMP (SE-A/08) **Sub- DAA Lab**

Name	**Naman Badlani**
UID No.	**2021300008**
Subject	**Design And Analysis Of Algorithm**
Class	**Comps A**
Experiment No.	**8**
AIM	To implement Dijkstra's Algorithm in C.

Theory –

Dijkstra's algorithm is a greedy algorithm used to find the shortest path between a source vertex and all other vertices in a weighted graph. It works by maintaining a set of visited vertices and a set of unvisited vertices. Initially, the set of visited vertices is empty, and the set of unvisited vertices includes all vertices in the graph.

Algorithm –

1. Create a set sptSet (shortest path tree set) that keeps track of vertices included in the shortest path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.
2. Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign the distance value as 0 for the source vertex so that it is picked first.
3. While sptSet doesn't include all vertices
   - Pick a vertex u that is not there in sptSet and has a minimum distance value.
   - Include u to sptSet.
   - Then update the distance value of all adjacent vertices of u.
     - To update the distance values, iterate through all adjacent vertices.
     - For every adjacent vertex v, if the sum of the distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

Bharatiya Vidya Bhavan's

Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India

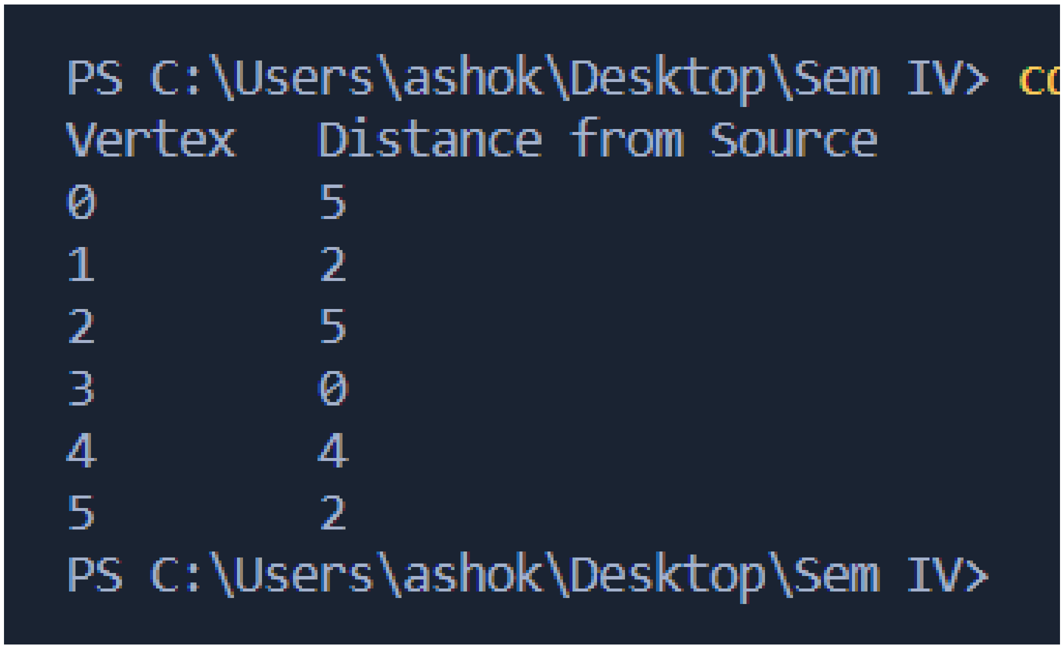
(Autonomous College Affiliated to University of Mumbai)

SE – COMP (SE-A/08) **Sub- DAA Lab**

Program –

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

#define V 6 // Number of vertices in the graph

int minDistance(int dist[], bool sptSet[]) {
    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;
    return min_index;
}

void printSolution(int dist[]) {
    printf("Vertex \t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src) {
    int dist[V]; // The output array. dist[i] will hold the shortest distance
from src to i
    bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
path tree or shortest distance from src to i is finalized
    // Initialize all distances as INFINITE and sptSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;
    // Distance of source vertex from itself is always 0
    dist[src] = 0;
    // Find shortest path for all vertices
    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);
}

int main() {
    int graph[V][V] = { {0, 3, 0, 7, 0, 0},
                        {3, 0, 4, 2, 0, 0},
                        {0, 4, 0, 5, 6, 0},
                        {7, 2, 5, 0, 4, 2},
                        {0, 0, 6, 4, 0, 5},
                        {0, 0, 0, 2, 5, 0} };
    dijkstra(graph, 3); // Source vertex is 3
    return 0;
}
```

Result Analysis –



```
PS C:\Users\ashok\Desktop\Sem IV> co
Vertex   Distance from Source
0        5
1        2
2        5
3        0
4        4
5        2
PS C:\Users\ashok\Desktop\Sem IV>
```

- The time complexity of Dijkstra's Algorithm is O(V^2) for the standard implementation using an adjacency matrix, where V is the number of nodes in the graph. However, with the use of a priority queue data structure, the time complexity can be reduced to O (E log V), where E is the number of edges in the graph.

- The space complexity of the algorithm is O(V) since it needs to maintain the visited set and distances for each node.