Government College of Engineering, Jalgaon
(An Autonomous Institute of Government of Maharashtra)

Name : Rajnandini Patil
PRN : 2241047
Class : T. Y. B.Tech Computer
Academic Year : 2024-25
Subject : DAA Lab
Course Teacher : Mr. Vinit Kakde
Date of Performance :
Date of Completion:

Practical No:7

Aim: To implement **Dijkstra's Algorithm** in C to find the shortest paths from a given source vertex to all other vertices in a **weighted connected graph**.

Theory: Dijkstras shortest path algorithm is similar to that of Prims algorithm as they both rely on finding the shortest path locally to achieve the global solution. However, unlike prims algorithm, the dijkstras algorithm does not find the minimum spanning tree; it is designed to find the shortest path in the graph from one vertex to other remaining vertices in the graph. Dijkstras algorithm can be performed on both directed and undirected graphs.

Dijkstra's Algorithm:

1.**Initialization:**

- Set the distance of every vertex to **infinity (∞)**.
- Mark all vertices as **unvisited**.
- Set the **distance of the source vertex to 0**.

2 . **For all vertices in the graph:**

a. Select the **unvisited vertex** with the **smallest known distance** from the source .

b. For each **unvisited neighbor** of the selected vertex:

- Calculate the tentative distance through the selected vertex.
- If this new distance is **smaller**, update the shortest distanc

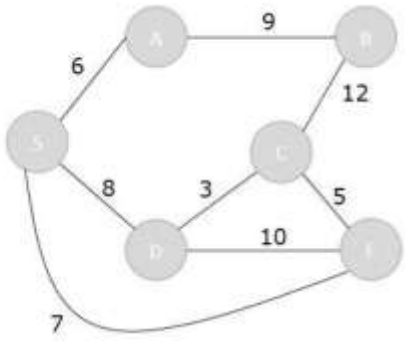
c. After checking all neighbors, **mark the current vertex as visited**.

3 **Repeat Step 2** until all vertices are **visited** or the smallest tentative distance among the unvisited vertices is infinity (i.e., unreachable).

4 The algorithm ends when **all shortest distances** from the source to other vertices are known.

Example:

To understand the dijkstras concept better, let us analyze the algorithm with the help of an example graph −



Step 1

Initialize the distances of all the vertices as ∞, except the source node S.

Vertex	S	A	B	C	D	E
Distance	0	∞	∞	∞	∞	∞

Now that the source vertex S is visited, add it into the visited array.

visited = {S}

Step 2

The vertex S has three adjacent vertices with various distances and the vertex with minimum distance among them all is A. Hence, A is visited and the dist[A] is changed from ∞ to 6.

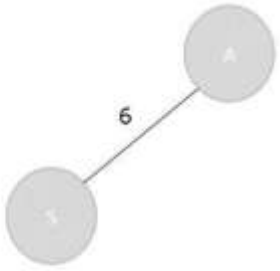
S → A = 6

S → D = 8

S → E = 7

Vertex	S	A	B	C	D	E
Distance	0	6	∞	∞	8	7

Visited = {S,A}



Step 3

There are two vertices visited in the visited array, therefore, the adjacent vertices must be checked for both the visited vertices.

Vertex S has two more adjacent vertices to be visited yet: D and E. Vertex A has one adjacent vertex B.

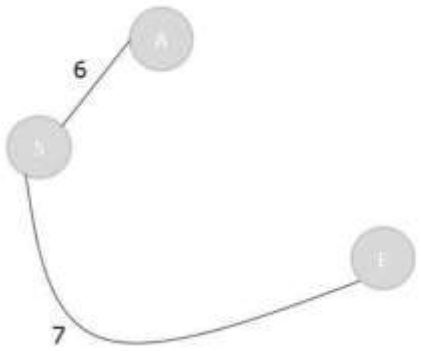
Calculate the distances from S to D, E, B and select the minimum distance –

S → D = 8 and S → E = 7.

$S \rightarrow B = S \rightarrow A + A \rightarrow B = 6 + 9 = 15$

Vertex	S	A	B	C	D	E
Distance	0	6	15	∞	8	7

Visited = {S, A, E}



Step 4

Calculate the distances of the adjacent vertices S, A, E of all the visited arrays and select the vertex with minimum distance.

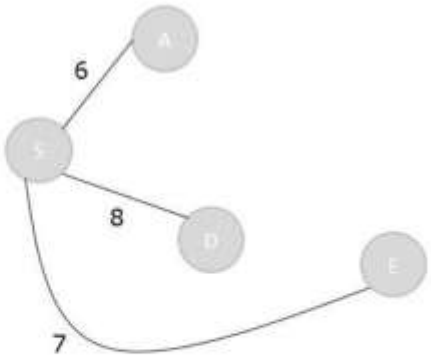
$S \rightarrow D = 8$

$S \rightarrow B = 15$

$S \rightarrow C = S \rightarrow E + E \rightarrow C = 7 + 5 = 12$

Vertex	S	A	B	C	D	E
Distance	0	6	15	12	8	7

Visited = {S, A, E, D}



Step 5

Recalculate the distances of unvisited vertices and if the distances minimum than existing distance is found, replace the value in the distance array.

S → C = S → E + E → C = 7 + 5 = 12

S → C = S → D + D → C = 8 + 3 = 11

dist[C] = minimum (12, 11) = 11

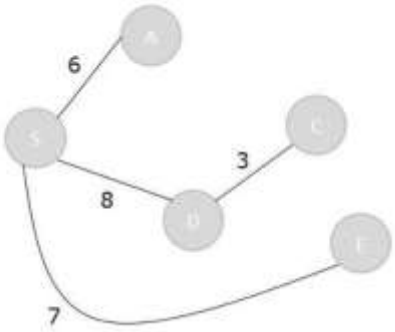
S → B = S → A + A → B = 6 + 9 = 15

S → B = S → D + D → C + C → B = 8 + 3 + 12 = 23

dist[B] = minimum (15,23) = 15

Vertex	S	A	B	C	D	E
Distance	0	6	15	11	8	7

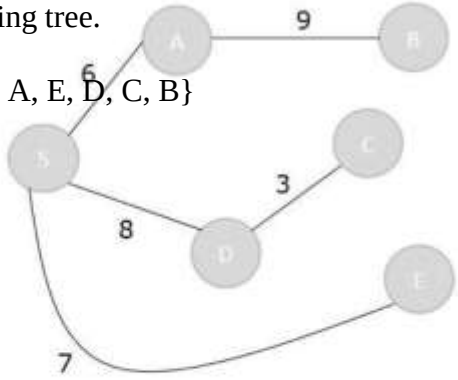
Visited = { S, A, E, D, C}



Step 6

The remaining unvisited vertex in the graph is B with the minimum distance 15, is added to the output spanning tree.

Visited = {S, A, E, D, C, B}



Time Complexity:

Implementation Method		Time Complexity
Adjacency Matrix + Array	-	$O(V^2)$
Adjacency List + Min Heap	-	$O((V + E) \log V)$

- **V**: Number of vertices

- **E**: Number of edges
- For dense graphs, $O(V^2)$ is acceptable. For sparse graphs, the heap-based approach is preferred.

Applications of Dijkstra's Algorithm:

1. **GPS and Navigation Systems** – Finding the shortest route between locations.
2. **Network Routing Protocols** – Like OSPF (Open Shortest Path First).
3. **Flight and Transport Systems** – Calculating shortest travel paths.
4. **Robotics & AI Pathfinding** – Navigation for autonomous systems.
5. **Telecommunication** – Optimizing signal paths.

Advantages:

- Guarantees **shortest path** for graphs with **non-negative weights**.
- Efficient with **priority queues** on sparse graphs.
- Works well in **real-world applications** like maps and networks.
- Can be extended for **shortest path tree** creation.

Disadvantages:

- **Does not work with negative weights** (unlike Bellman-Ford).
- Requires additional structures (like **heaps**) for optimal efficiency.
- **Relatively slower** for very large graphs compared to other specialized algorithms.
- Cannot detect **negative weight cycles**.

Program:

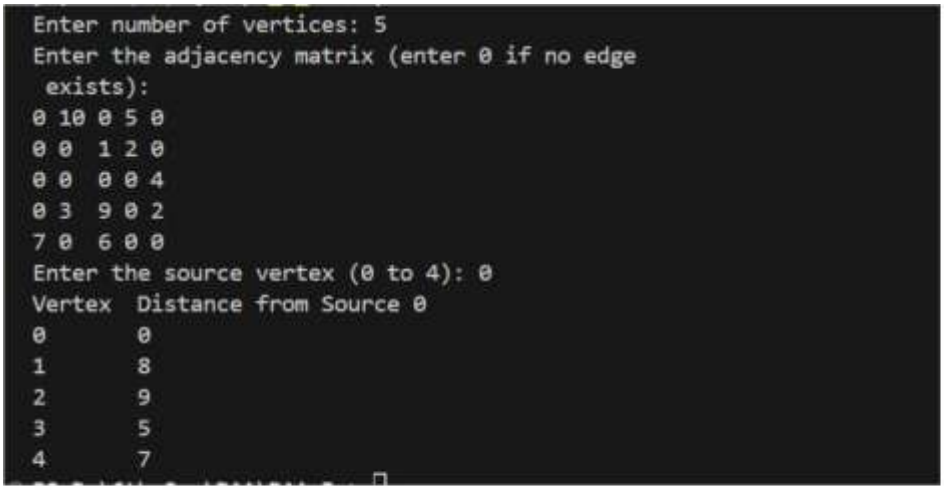
```
#include <stdio.h>
#include <limits.h>
#define MAX 100
int minDistance(int dist[], int visited[], int V) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (!visited[v] && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}
void dijkstra(int graph[MAX][MAX], int V, int src) {
    int dist[MAX];
    int visited[MAX];
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, visited[i] = 0;
    dist[src] = 0;
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, visited, V);
        visited[u] = 1;
        for (int v = 0; v < V; v++)
            if (!visited[v] && graph[u][v] && dist[u] != INT_MAX &&
                dist[u] + graph[u][v] < dist[v])
```

```
                dist[v] = dist[u] + graph[u][v];
    }
    printf("Vertex\tDistance from Source %d\n", src);
    for (int i = 0; i < V; i++)
        printf("%d\t%d\n", i, dist[i]);
}
int main() {
    int V;
    int graph[MAX][MAX];
    printf("Enter number of vertices: ");
    scanf("%d", &V);
    printf("Enter the adjacency matrix (enter 0 if no edge exists):\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    int src;
    printf("Enter the source vertex (0 to %d): ", V - 1);
    scanf("%d", &src);
    dijkstra(graph, V, src);
    return 0;
```

}

Output:



Questions & Answer:

Q1: What is the main purpose of Dijkstra's Algorithm?

Answer:
Dijkstra's Algorithm is used to find the **shortest paths from a single source vertex** to all other vertices in a **weighted graph** with **non-negative edge weights**.

Q2: Why do we use INFINITY (9999) in the C program?

Answer:
We use INFINITY (represented by 9999 in the program) to indicate that **no direct edge** exists between two vertices, i.e., the vertices are not directly connected.

Q3: Can Dijkstra’s algorithm work with negative weights? Why or why not?

Answer:
No, Dijkstra's Algorithm **does not work with negative edge weights** because it assumes that once a node’s shortest distance is found, it will not change. Negative weights can violate this assumption and lead to incorrect results.

Q4: What is the time complexity of Dijkstra's Algorithm using a simple array (as in the C code)?

Answer:
The time complexity is $\mathbf{O(n^2)}$, where n is the number of vertices. This is because we check all vertices in each iteration to find the minimum distance node.

Q5: What data structures could be used to improve Dijkstra’s performance for large graphs?

Answer:
For large graphs, using a **priority queue** (min-heap) along with an **adjacency list** representation reduces the time complexity to **O((V + E) log V)**, where V is the number of vertices and E is the number of edges.

Conclusion:

Dijkstra's Algorithm is successfully implemented to find the shortest path from a single source to all other nodes in a weighted connected graph. The result shows the minimum distances correctly as expected.

Sign of Course teacher

Mr. Vinit Kakde