Autonomous Institute of Government of Maharashtra)	
Name :	**PRN :** 2241004
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.

AIM:

Implement Dijkstra's Algorithm for finding the shortest path from a single source to all other vertices in a weighted graph with non-negative weights.

THEORY:

Dijkstra's Algorithm is a **Greedy Algorithm** used for finding the shortest paths from a single source vertex to all other vertices in a **weighted graph** with **non-negative edge weights**. It maintains a set of vertices whose minimum distance from the source is known and repeatedly selects the vertex with the minimum known distance, updating the distances to its adjacent vertices accordingly.

ALGORITHM:

1. Set the distance of the source vertex to 0 and all other vertices to infinity.
2. Mark all vertices as unvisited.
3. Select the unvisited vertex with the smallest distance.
4. For the selected vertex, update the distances of its neighboring vertices.
5. Mark the selected vertex as visited.
6. Repeat steps 3–5 until all vertices are visited or the smallest distance among unvisited vertices is infinity.

TIME COMPLEXITY:

- Using array: $O(V^2)$
- Using min-priority queue (heap): $O((V + E) \log V)$

ADVANTAGES:

1. Efficient for graphs with non-negative weights.
2. Finds the shortest path to all vertices from the source.
3. Can be optimized using priority queues.
4. Guarantees the globally shortest path.

DISADVANTAGES:

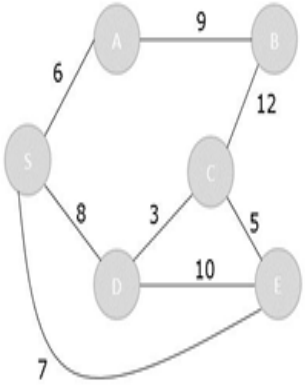
1. Cannot handle negative edge weights.
2. Slower for dense graphs if implemented without a heap.
3. Requires additional space for distance and visited arrays.

APPLICATIONS:

1. GPS Navigation Systems
2. Network Routing Protocols
3. Robot Path Planning
4. Map Services (Google Maps, etc.)
5. Urban Traffic Navigation

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.

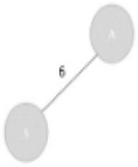
$S \rightarrow A = 6$

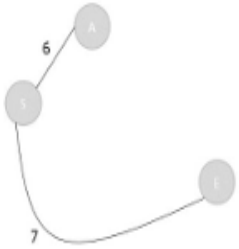
$S \rightarrow D = 8$

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 → B = S → A + A → 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.

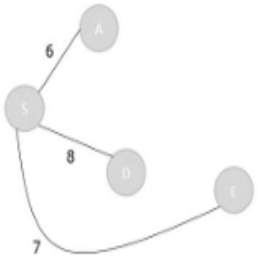
S → D = 8

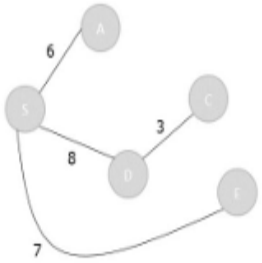
S → B = 15

S → C = S → E + E → 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.

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

$S \rightarrow C = S \rightarrow D + D \rightarrow C = 8 + 3 = 11$

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

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

$S \rightarrow B = S \rightarrow D + D \rightarrow C + C \rightarrow B = 8 + 3 + 12 = 23$

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

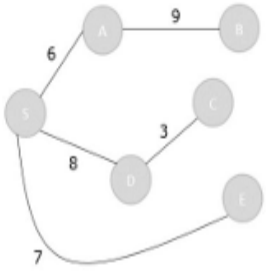
Vertex	S	A	B	C	D	E
Distance	0	6	15	11	8	7

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}



The shortest path spanning tree is obtained as an output using the dijkstras algorithm.

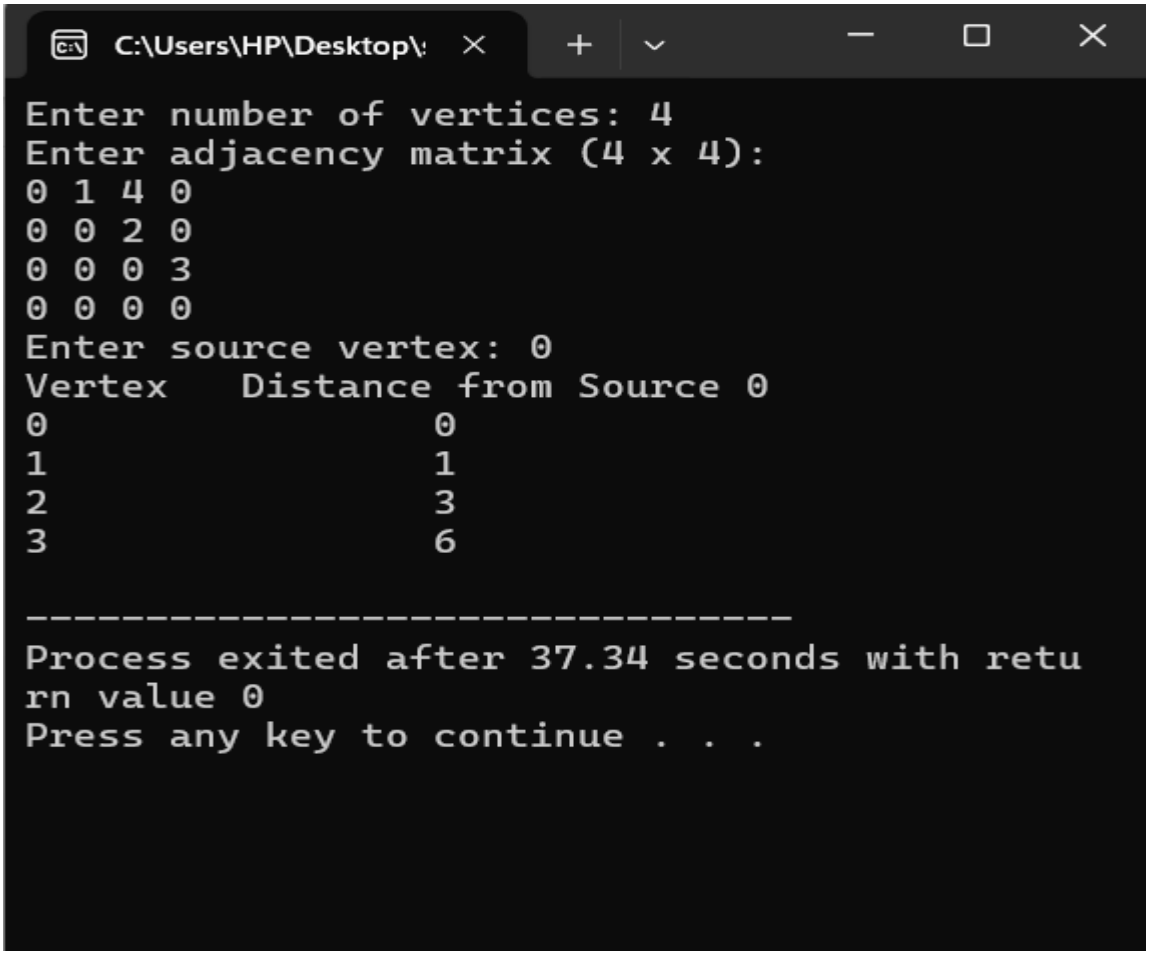
PROGRAM:

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

#define V 10

int minDistance(int dist[], int visited[], int n) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < n; v++)
        if (!visited[v] && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}

void dijkstra(int graph[V][V], int src, int n) {
    int dist[V], visited[V];

    for (int i = 0; i < n; i++)
        dist[i] = INT_MAX, visited[i] = 0;

    dist[src] = 0;

    for (int count = 0; count < n - 1; count++) {
        int u = minDistance(dist, visited, n);
        visited[u] = 1;

        for (int v = 0; v < n; 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 \t Distance from Source %d\n", src);
    for (int i = 0; i < n; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

int main() {
    int n;
    int graph[V][V];

    printf("Enter number of vertices: ");
    scanf("%d", &n);

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

    int source;
    printf("Enter source vertex: ");
    scanf("%d", &source);
```

```
 dijkstra(graph, source, n);

    return 0;

}
```

#OUTPUT:



QUESTIONS:

1. **What type of graphs is Dijkstra's Algorithm used for?**
   → Weighted, connected graphs with non-negative weights.
2. **What is the key limitation of Dijkstra's Algorithm?**
   ➔ It does not work with negative edge weights.
3. **How does Dijkstra's Algorithm differ from Prim's Algorithm?**
   ➔ Dijkstra's finds the shortest path, while Prim's finds the Minimum Spanning Tree.
4. **What data structure improves the efficiency of Dijkstra's Algorithm?**
   → A Min Heap (Priority Queue) helps improve performance.
5. **Name one real-world application of Dijkstra's Algorithm.**
   → GPS Navigation Systems use Dijkstra's to find shortest driving routes.

☑ **Conclusion:**

Dijkstra's Algorithm is an efficient and widely-used method for finding the shortest path from a single source in a weighted graph with non-negative weights. It ensures optimal path discovery and is highly applicable in areas such as routing, navigation, and AI-based pathfinding.

.

Sign of course Teacher

Mr. Vinit Kakde