**Exp 5**

**Name : Rais Saaim Ravish**

**Roll No : 24DDS03**

**Batch : 2**

**Aim:**Implement Single Source Shortest Part Algorithm(Dijkstra)

**Theory:**

Dijkstra's algorithm is a famous algorithm used for finding the shortest path between a source node and all other nodes in a weighted graph. It works efficiently on graphs with non-negative edge weights. The primary use of this algorithm is in routing and as a subroutine in other graph algorithms.

Dijkstra's algorithm is based on greedy strategy, which means it always chooses the next "best" option available at each step. The idea is to keep track of the shortest known distance from the source node to each node, and progressively improve this estimate as the algorithm explores the graph.

Here’s how the algorithm works step-by-step:

1. Initialization:  
   * Create a set of distances to all nodes, initially setting the distance of the source node to 0 and all other nodes to infinity (representing unknown distances).
   * Maintain a set of unvisited nodes.
   * Keep track of the shortest known distance to each node.
2. Iteration:  
   * Start from the source node.
   * For each unvisited node, calculate the distance from the current node to its neighbors through the edge, and if a shorter path is found, update the shortest path estimate.
   * Once all neighbors of the current node are processed, mark the node as "visited."
   * Among all unvisited nodes, choose the one with the smallest known distance (the "greedy" step).
3. Termination:  
   * The algorithm terminates once all nodes are visited, or if there are no reachable nodes left.
4. Output:  
   * The algorithm outputs the shortest path from the source node to each node in the graph.

**Program:**

#include <stdio.h>

#include <stdbool.h>

#define MAX 30

#define INF 9999

int operationsCount = 0;

int selectNext(int d[], bool v[], int n)

{

int idx = -1, min = INF;

for (int i = 0; i < n; i++)

{

operationsCount++;

if (v[i] == false && d[i] < min)

{

min = d[i];

idx = i;

}

}

return idx;

}

void dijkstras(int graph[MAX][MAX], int src, int n)

{

int dist[MAX];

bool visited[MAX];

for (int i = 0; i < n; i++)

{

dist[i] = INF;

visited[i] = false;

}

int newPath, curr = src;

dist[src] = 0;

while (curr != -1)

{

for (int i = 0; i < n; i++)

{

operationsCount++;

if (!visited[i] && graph[curr][i] != 0)

{

newPath = dist[curr] + graph[curr][i];

if (newPath < dist[i])

dist[i] = newPath;

}

}

visited[curr] = true;

curr = selectNext(dist, visited, n);

}

printf("\nNode \t\t Distance From Source\n");

for (int i = 0; i < n; i++)

{

printf("%d \t\t\t %d\n", i + 1, dist[i]);

}

}

int main()

{

int n, source;

int graph[MAX][MAX];

printf("Dijktra's Algorithm for Single Source Shortest Path(Greedy Approach):\n");

printf("Enter the number of vertices: ");

scanf("%d", &n);

printf("Enter weighted adjacency matrix for Graph:\n");

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

scanf("%d", &graph[i][j]);

}

}

printf("Enter Source Node between 1 - %d: ", n);

scanf("%d", &source);

source--;

operationsCount = 0;

dijkstras(graph, source, n);

printf("\n Total Number of Operations Performed (Time Complexity Estimate) is: %d\n", operationsCount);

}

**Output:**

Dijktra's Algorithm for Single Source Shortest Path(Greedy Approach):

Enter the number of vertices: 5

Enter weighted adjacency matrix for Graph:

0 3 1 0 0

3 0 7 5 1

1 7 0 2 0

0 5 2 0 7

0 1 0 7 0

Enter Source Node between 1 - 5: 3

Node Distance From Source

1 1

2 4

3 0

4 2

5 5

Total Number of Operations Performed (Time Complexity Estimate) is: 50

**Conclusion:**

In this practical, we successfully implemented Dijkstra’s Algorithm for finding the shortest path from a single source node to all other nodes in a weighted graph using the greedy approach. The algorithm correctly computes the minimum distance to each node starting from the source node by progressively selecting the unvisited node with the smallest tentative distance.

From the program's output, we observe that the algorithm effectively calculates the shortest paths and provides the number of operations performed, giving us an insight into the algorithm's computational complexity.

The algorithm works as expected and outputs the shortest distance from the source node to every other node in the graph.

We also tracked the number of operations performed during the algorithm’s execution, which serves as an estimate for its time complexity. This helps us understand the efficiency of the algorithm in terms of the number of operations and the number of nodes and edges in the graph.