

Aim:

Given a graph G and source vertex S , Dijkstra's shortest path algorithm is used to find the shortest paths from source S to all vertices in the given graph.

The Dijkstra algorithm is also known as the single-source shortest path algorithm. It is based on the greedy technique. A little variation in the algorithm can find the shortest path from the source nodes to all the other nodes in the graph.

The function `void dijkstra(int G[MAX][MAX], int n, int startnode)` computes and prints the shortest path distances and corresponding paths from the given source node to all other nodes in a weighted directed graph using Dijkstra's algorithm. It outputs the distance or "INF" if unreachable, along with the path or "NO PATH" for each node.

Note:

- Vertices are numbered from 1 through V .
- All input values are separated by spaces and/or newlines.

Sample Input and Output:

```
Enter the number of vertices : 4
Enter the number of edges : 5
Enter source : 1
Enter destination : 2
Enter weight : 4
Enter source : 1
Enter destination : 4
Enter weight : 10
Enter source : 1
Enter destination : 3
Enter weight : 6
Enter source : 2
Enter destination : 4
Enter weight : 5
Enter source : 3
Enter destination : 4
Enter weight : 2
Enter the source :1
```

Node	Distance	Path
2	4	2<-1
3	6	3<-1
4	8	4<-3<-1

Source Code:Dijkstras.c

```
#include <limits.h>
#include <stdio.h>
#define MAX 20
int V, E;
int graph[MAX][MAX];
```

```
#define INFINITY 99999

void dijkstra(int G[MAX][MAX], int n, int startnode) {
    int cost[MAX][MAX], distance[MAX], pred[MAX];
    int visited[MAX], count, mindistance, nextnode, i, j;

    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            if(graph[i][j] == 0 && i!=j)
                cost[i][j] = INFINITY;
            else
                cost[i][j] = graph[i][j];

    for(i = 1; i <= n; i++) {
        distance[i] = cost[startnode][i];
        pred[i] = startnode;
        visited[i] = 0;
    }

    distance[startnode] = 0;
    visited[startnode] = 1;
    count = 1;

    while(count < n) {
        mindistance = INFINITY;
        for(i = 1; i <= n; i++)
            if(distance[i] < mindistance && !visited[i]) {
                mindistance = distance[i];
                nextnode = i;
            }
        visited[nextnode] = 1;
        for(i = 1; i <= n; i++)
            if(!visited[i])
                if(mindistance + cost[nextnode][i] < distance[i]) {
                    distance[i] = mindistance + cost[nextnode][i];
                    pred[i] = nextnode;
                }
        count++;
    }
    printf("Node\tDistance\tPath\n");
    for(i = 1; i <= n; i++) {
        if(i != startnode) {
            if(distance[i] == INFINITY) {
                printf("    %d\t    INF\tNO PATH\n", i);
            } else {
                printf("    %d\t    %d\t", i, distance[i]);
                j = i;
                while(j != startnode) {
                    printf("%d<- ", j);
                    j = pred[j];
                }
                printf("%d\n", startnode);
            }
        }
    }
}
```

```

int main() {
    int s, d, w, i, j;
    printf("Enter the number of vertices : ");
    scanf("%d", &V);
    printf("Enter the number of edges : ");
    scanf("%d", &E);
    for(i = 1 ; i <= V; i++) {
        for(j = 1; j <= V; j++) {
            graph[i][j] = 0;
        }
    }
    for(i = 1; i <= E; i++) {
        printf("Enter source : ");
        scanf("%d", &s);
        printf("Enter destination : ");
        scanf("%d", &d);
        printf("Enter weight : ");
        scanf("%d", &w);
        if(s > V || d > V || s <= 0 || d <= 0) {
            printf("Invalid index. Try again.\n");
            i--;
            continue;
        } else {
            graph[s][d] = w;
        }
    }
    printf("Enter the source :");
    scanf("%d", &s);
    dijkstra(graph, V, s);
    return 0;
}

```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices : 4
Enter the number of edges : 5
Enter source : 1
Enter destination : 2
Enter weight : 4
Enter source : 1
Enter destination : 4
Enter weight : 10
Enter source : 1
Enter destination : 3
Enter weight : 6
Enter source : 2
Enter destination : 4
Enter weight : 5
Enter source : 3
Enter destination : 4
Enter weight : 2

Enter the source : 1		
Node	Distance	Path
2	4	2<-1
3	6	3<-1
4	8	4<-3<-1

Test Case - 2		
User Output		
Enter the number of vertices : 5		
Enter the number of edges : 6		
Enter source : 1		
Enter destination : 2		
Enter weight : 2		
Enter source : 1		
Enter destination : 5		
Enter weight : 3		
Enter source : 2		
Enter destination : 4		
Enter weight : 4		
Enter source : 2		
Enter destination : 3		
Enter weight : 7		
Enter source : 4		
Enter destination : 3		
Enter weight : 2		
Enter source : 5		
Enter destination : 4		
Enter weight : 1		
Enter the source : 2		
Node	Distance	Path
1	INF	NO PATH
3	6	3<-4<-2
4	4	4<-2
5	INF	NO PATH

Test Case - 3		
User Output		
Enter the number of vertices : 4		
Enter the number of edges : 5		
Enter source : 1		
Enter destination : 2		
Enter weight : 4		
Enter source : 3		
Enter destination : 2		
Enter weight : 5		
Enter source : 4		
Enter destination : 1		
Enter weight : 1		
Enter source : 4		
Enter destination : 2		

Enter weight : 3		
Enter source : 4		
Enter destination : 3		
Enter weight : 8		
Enter the source : 1		
Node	Distance	Path
2	4	2<-1
3	INF	NO PATH
4	INF	NO PATH