

Date: 10/03/2021

# Practical 5:

2CSDE56 – Graph Theory

Name: Shrey Viradiya

Roll No: 18BCE259

Aim:

Write a program to implement single source shortest path algorithm.

## Code:

### Prac5\_Djikstra.cpp

---

```
// Write a program to implement single source shortest path algorithm.

#include <iostream>
#include "Djikstra.h"

int main(){
    using namespace std;

    int V, src;

    cout << "Enter the number of vertices: " << endl;
    cin >> V;

    cout << "Enter the adjacency matrix of the graph: " << endl;
    int **graphA = new int *[V];

    int error = 0;
    for (int i = 0; i < V; i++)
    {
        graphA[i] = new int[V];
        for (int j = 0; j < V; j++)
        {
            cin >> graphA[i][j];
            if (graphA[i][j] < 0) error = 1;
        }
    }

    if(error){
        cout << "Djikstra can not perform with negative weights! " << endl;
        // Releasing Memory
        for (int i = 0; i < V; i++)
        {
            delete [] graphA[i];
        }
        delete [] graphA;
        return 0;
    }

    cout << "Enter the Source Vertice: " << endl;
    do
    {
        cin >> src;
        if (src >= V){
            cout << "Invalid Source.. must be between 0 to" << (V-1) << endl;
        }
    } while (src >= V);

    // Implementing Dijkstra's Algorithm
    // 1. Create a source vertex and a visited array
    // 2. Create a priority queue to store the vertices
    // 3. Push the source vertex into the priority queue
    // 4. While the priority queue is not empty
    // 5. Pop the vertex with the minimum distance
    // 6. If the vertex is visited, skip it
    // 7. For each neighbor of the vertex
    // 8. Calculate the distance to the neighbor
    // 9. If the distance is less than the current distance, update it
    // 10. Push the neighbor into the priority queue
    // 11. Mark the vertex as visited
    // 12. Return the distance to the destination vertex
```

```

    }
} while (src >= V);

Djikstra(graphA, V, src);

// Releasing Memory
for (int i = 0; i < V; i++)
{
    delete [] graphA[i];
}
delete [] graphA;
return 0;
}

```

## Djikstra.h

---

```

#pragma once

#include <limits.h>
#include <iostream>

void Djikstra(int **graph, int V, int src){
    int *dist = new int[V];
    bool *visited = new bool[V];

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

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++)
    {
        int min = INT_MAX, min_index;

        // Finding Minimum Distance
        for (int v = 0; v < V; v++){
            if (visited[v] == false && dist[v] <= min)
                min = dist[v], min_index = v;
        }

        visited[min_index] = true;
        for (int v = 0; v < V; v++){
            if (!visited[v] && graph[min_index][v] && dist[min_index] != INT_MAX && dist[min_index] + graph[min_index][v] < dist[v]){
                dist[v] = dist[min_index] + graph[min_index][v];
            }
        }
    }
}

```

```

    }
}

using namespace std;
cout << "Vertices \t Distance from Source\n" << endl;
for (int i = 0; i < V; i++)
    cout << src << "-->" << i << " : " << dist[i] << endl;
}

```

Snapshot of the output:

```

Prac5_Dijkstra.cpp
1 // Write a program to implement single source shortest path al
2
3 #include <iostream>
4 #include "Dijkstra.h"
5
6 int main(){
7     using namespace std;
8
9     int V, src;
10
11     cout << "Enter the number of vertices: " << endl;
12     cin >> V;
13
14     cout << "Enter the adjacency matrix of the graph: " << endl;
15     int **graphA = new int *[V];
16
17     int error = 0;
18     for (int i = 0; i < V; i++)
19     {
20         graphA[i] = new int[V];
21         for (int j = 0; j < V; j++)
22         {
23             cin >> graphA[i][j];
24             if (graphA[i][j] < 0) error = 1;
25         }
26     }
27
28     if(error){
29         cout << "Dijkstra can not perform with negative weight
30         // Releasing Memory
31         for (int i = 0; i < V; i++)
32         {
33             delete [] graphA[i];
34         }
35         delete [] graphA;
36         return 0;
37     }
38
39     cout << "Enter the Source Vertex: " << endl;
40     do

```

```

Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

S:\SEM 6\GraphTheory\GraphTheory>cd "S:\SEM 6\GraphTheory\GraphTheory\" && g++ Prac5_Dijkstra
.cpp -o Prac5_Dijkstra && "S:\SEM 6\GraphTheory\GraphTheory\Prac5_Dijkstra
Enter the number of vertices:
5
Enter the adjacency matrix of the graph:
0 2 4 1 0
2 0 4 7 8
1 1 0 0 0
0 5 6 0 7
7 8 8 0 0
Enter the Source Vertex:
0
Vertices      Distance from Source
0-->0 : 0
0-->1 : 2
0-->2 : 4
0-->3 : 1
0-->4 : 8
S:\SEM 6\GraphTheory\GraphTheory>

```

Conclusion:

Using Dijkstra's Algorithm, we can find single source shortest path. Negative edges are not allowed.