	Bansilal Ramnath Agarwal Charitable Trust's Vishwakarma Institute of Information Technology **Department of Artificial Intelligence and Data Science**

Name: Siddhesh Dilip Khairnar		
Class: SY	Division: B	Roll No: 272028
Semester: IV	Academic Year: 2022-2023	
Subject Name & Code: Advance Data Structure: ADUA22202		
Title of Assignment: For any application find Single source shortest path using Dijkstra's algorithm		
Date of Performance: 01-03-2023	Date of Submission: 08-03-2023	

ASSIGNMENT NO. 6

Name: Siddhesh Khairnar
ROLL NO: 272028

ADS Assignment 06

Aim: Implementation of Dijkstra's algorithm to find shortest Path.

Theory: Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. like prim's MST, we generate a SPT with a given source as root. Maintain two set, one set contain vertices included in the SPT, other set contain vertices not yet included. At every step of the algorithm find a vertex that is other set and has minimum distance from source.

Follow the step to solve the problem: →

* create a spt set that keep track of vertices included in the spt that whose minimum distance from the source is ~~calted~~ calculated and finalized.

* Assign a distance value to all vertices in the input graph. Initialize all distance value as 'Infinite'. Assign the distance value as 0 for source vertex so that it is picked first.

* while spt set doesn't include all vertices: —

a. Pick a vertex u which is not there in spt set and has a minimum distance value.

b. Include u to spt set.

c. Then update distance value of all adjacent vertices of u.

Conclusion: → we have successfully learn't the implementation of Dijekstra's algorithm to find the shortest path tree.

10/4/23

Program and Output:

```
#include<iostream>

using namespace std;

// Number of vertices in the graph
const int V=7;

// Function to find the vertex with minimum key value
int min_distance(int key[], bool visited[])
{
    int min = 999, min_index;  // 999 represents an Infinite value

    for (int v = 0; v < V; v++) {
        if (visited[v] == false && key[v] < min) {
            // vertex should not be visited
            min = key[v];
            min_index = v;
        }
    }
    return min_index;
}

// Function to print the final MST stored in parent[]
void print_MST(int parent[], int distance[V])
{
    int minCost=0;
    cout<<"Edge \tWeight\n";
    for (int i = 1; i< V; i++) {
        cout<<"0"<<" - "<<i<<" \t"<<distance[i]<<" \n";
        minCost+=distance[i];
    }
    cout<<"Total cost is"<<minCost;
}

// Function to find the MST using adjacency cost matrix representation
void find_MST(int cost[V][V])
{
    int parent[V], distance[V];
    bool visited[V];

    // Initialize all the arrays
    for (int i = 0; i< V; i++) {
        distance[i] = 999;    // 999 represents an Infinite value
        visited[i] = false;
        parent[i]=-1;
    }
```

```
    distance[0] = 0;
    parent[0] = -1;


    for (int x = 0; x < V - 1; x++)
    {

        int u = min_distance(distance, visited);

        visited[u] = true;
        for (int v = 0; v < V; v++)
        {
            int total_distance=distance[u]+cost[u][v];
            if (cost[u][v]!=0 && visited[v] == false && total_distance<
distance[v])
            {
                parent[v] = u;
                distance[v] = total_distance;
            }
        }
    }

    // print the final MST
    print_MST(parent, distance);
}

// main function
int main()
{
    int cost[V][V];
    cout<<"Enter the weigth matrix for a graph with 6 vetices";
    for (int i=0;i<V;i++)
    {
        for(int j=0;j<V;j++)
        {
            cout<<"\n"<<"enter cost from vertex["<<i<<"]"<<"["<<j<<"] :";
            cin>>cost[i][j];
        }
    }
    find_MST(cost);

    return 0;
}
```

```
PS D:\MY FILES\PROGRAM> cd "d:\MY FILES\PROGRAM\" ; if ($?)
Enter the weigth matrix for a graph with 6 vetices
enter cost from vertex[0][0] :0

enter cost from vertex[0][1] :2

enter cost from vertex[0][2] :1

enter cost from vertex[0][3] :0

enter cost from vertex[0][4] :0

enter cost from vertex[0][5] :0

enter cost from vertex[0][6] :0
```

```
 enter cost from vertex[1][0] :2

 enter cost from vertex[1][1] :0

 enter cost from vertex[1][2] :0

 enter cost from vertex[1][3] :3

 enter cost from vertex[1][4] :0

 enter cost from vertex[1][5] :0

 enter cost from vertex[1][6] :0
```

```
enter cost from vertex[2][0] :1

enter cost from vertex[2][1] :0

enter cost from vertex[2][2] :0

enter cost from vertex[2][3] :2

enter cost from vertex[2][4] :0

enter cost from vertex[2][5] :0

enter cost from vertex[2][6] :0
```

```
enter cost from vertex[3][0] :0
enter cost from vertex[3][1] :3
enter cost from vertex[3][2] :2
enter cost from vertex[3][3] :0
enter cost from vertex[3][4] :3
enter cost from vertex[3][5] :0
enter cost from vertex[3][6] :4
enter cost from vertex[4][0] :0
enter cost from vertex[4][1] :0
enter cost from vertex[4][2] :0
enter cost from vertex[4][3] :3
enter cost from vertex[4][4] :0
enter cost from vertex[4][5] :2
enter cost from vertex[4][6] :3
```

```
enter cost from vertex[5][1] :0
enter cost from vertex[5][2] :0
enter cost from vertex[5][3] :0
enter cost from vertex[5][4] :2
enter cost from vertex[5][5] :0
enter cost from vertex[5][6] :1
enter cost from vertex[6][0] :0
enter cost from vertex[6][1] :0
enter cost from vertex[6][2] :0
enter cost from vertex[6][3] :4
enter cost from vertex[6][4] :3
enter cost from vertex[6][5] :1
enter cost from vertex[6][6] :0
```

```
enter cost from vertex[6][6] :0
Edge    Weight
0 - 1   2
0 - 2   1
0 - 3   3
0 - 4   6
0 - 5   8
0 - 6   7
Total cost is27
PS D:\MY FILES\PROGRAM>
```