	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: Implement Prim's/Kruskal algorithm for any application.		
Date of Performance: 01-03-2023	Date of Submission: 08-03-2023	

ASSIGNMENT NO. 5

ADS Assignment no 5

* **Aim:** Implement Prim / Kruskal's algorithm:

* **Theory:**

- Prim's algorithm alway start with a single node and it move through several adjacent nodes, in order to explore all of the connected edges along the ways.
- It start with an empty spanning trees. The ideas is to maintain the set of vertices. The first set contain the vertices already included in the MST, & other set contain the vertices not yet included.
- At every step, it consider all the edges After picking the edges it moves the other endpoint of the edges to the set containing MST.
- A group of edges that connect two set of vertices in a graph is called cut in graph theory. So, at every step of prim's algorithm find a cut, pick The minimum weight edge from the cut and include this vertex to the MST set.
- The idea behind prim's algorithm is simple, a spanning tree means all vertices must be connected so the two disjoint subset of vertices ~~must be~~ must be connected to make a spanning tree. And they must be connected with the minimum weight edge to move it a minimum spanning trees.

Follow given step to find MST using prim's algorithm:
- create a set 'MSTset' that keep track of vertices already included in MST.
- Assign a key value to all vertices in the input graph. Initialize all ~~they~~ Key values to infinite. Assign the key value as 0 for the first vertex so that it is picked first.

- While MST set doesn't include all vertices
  - Pick a vertex u which is not there in MST set and has a minimum key value
  - Include u in the MST set.

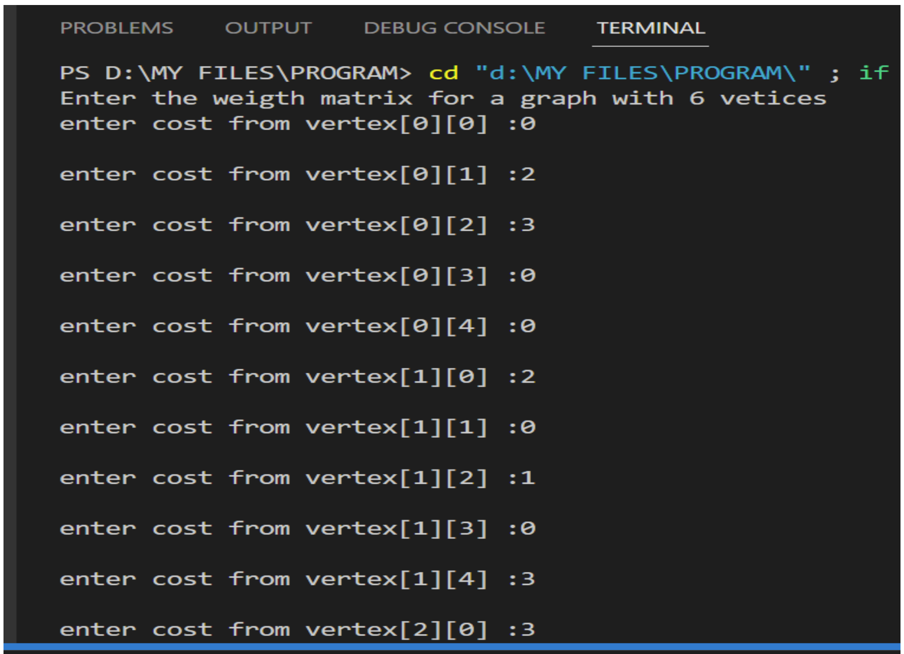
- update key value of all adjacent vertices of u. To update the
iterate through all adjacent vertices. for every adjacent vertex v,
weight of edges u-v is less than previous key value of v, update
value as weight of u-v

Conclusion: we have learnt the implementation of Prim's algo
along with it application in real life.

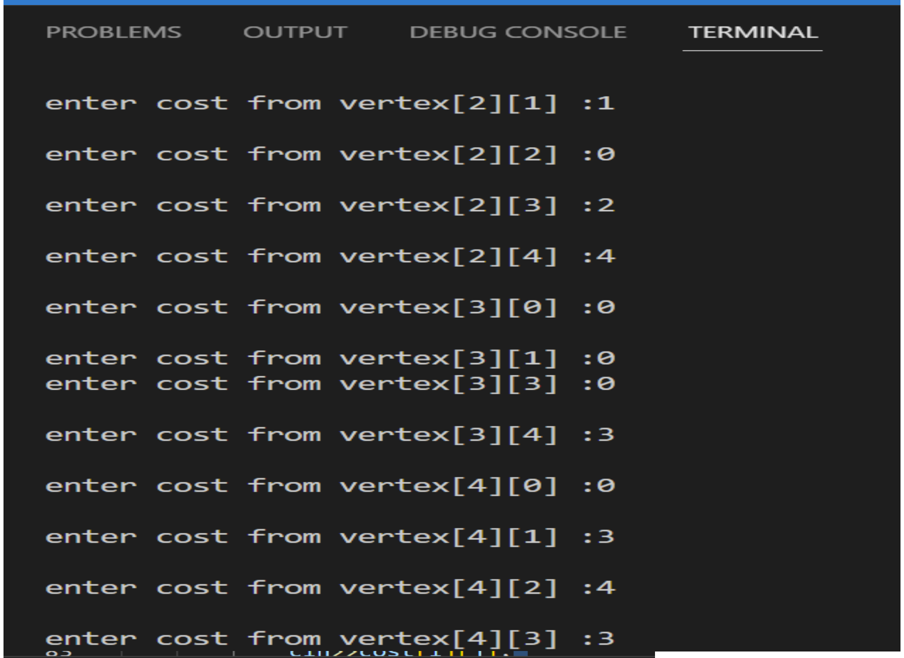
10/04/23

Program and Output:

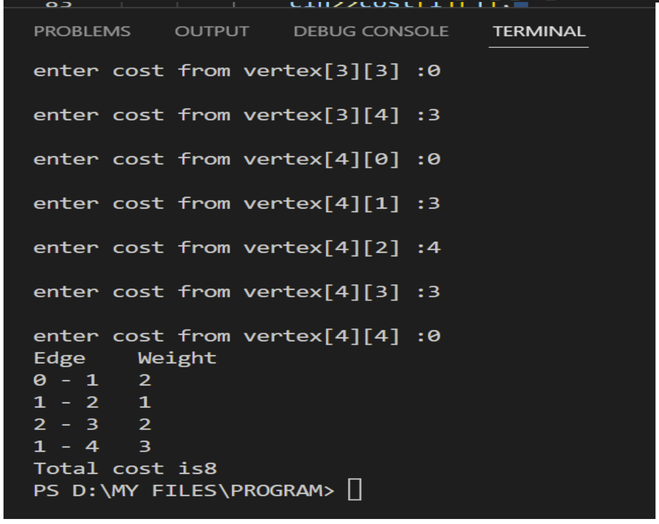
```
#include<iostream>

using namespace std;

// Number of vertices in the graph
const int V=5;

// 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 cost[V][V])
{
    int minCost=0;
    cout<<"Edge \tWeight\n";
    for (int i = 1; i< V; i++) {
        cout<<parent[i]<<" - "<<i<<" \t"<<cost[i][parent[i]]<<" \n";
        minCost+=cost[i][parent[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++)
        {

            if (cost[u][v]!=0 && visited[v] == false && cost[u][v] <
distance[v])
            {
                parent[v] = u;
                distance[v] = cost[u][v];
            }
        }
    }

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

// 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] :3

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

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

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

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

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

enter cost from vertex[1][3] :0

enter cost from vertex[1][4] :3

enter cost from vertex[2][0] :3
```



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

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

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

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

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

enter cost from vertex[3][1] :0
enter cost from vertex[3][3] :0

enter cost from vertex[3][4] :3

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

enter cost from vertex[4][1] :3

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

enter cost from vertex[4][3] :3
```



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

enter cost from vertex[3][4] :3

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

enter cost from vertex[4][1] :3

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

enter cost from vertex[4][3] :3

enter cost from vertex[4][4] :0
Edge     Weight
0 - 1    2
1 - 2    1
2 - 3    2
1 - 4    3
Total cost is8
PS D:\MY FILES\PROGRAM> 
```