NAME:	VEDANT TUSHAR DAPOLIKAR
UID:	2021700016
BRANCH:	CS-DS

EXPERIMENT- 6

- **AIM:** IMPLEMENTING DJISKTRA'S AND PRIM'S ALGORITHM FOR MINIMUM SPANNING TREE USING DYNAMIC PROGRAMMING.

- **ALGORITHM:-**

Step1: All nodes should be marked as unvisited.

Step2: All the nodes must be initialized with the "infinite" (a big number) distance. The starting node must be initialized with zero.

Step3: Mark the starting node as the current node.

Step4: From the current node, analyze all of its neighbors that are not visited yet, and compute their distances by adding the weight of the edge, which establishes the connection between the current node and neighbor node to the current distance of the current node.

Step5: Now, compare the recently computed distance with the distance allotted to the neighboring node, and treat it as the current distance of the neighboring node,

Step6: After that, the surrounding neighbors of the current node, which has not been visited, are considered, and the current nodes are marked as visited.

Step7: When the ending node is marked as visited, then the algorithm has done its job; otherwise,

Step8: Pick the unvisited node which has been allotted the minimum distance and treat it as the new current node. After that, start again from step4.

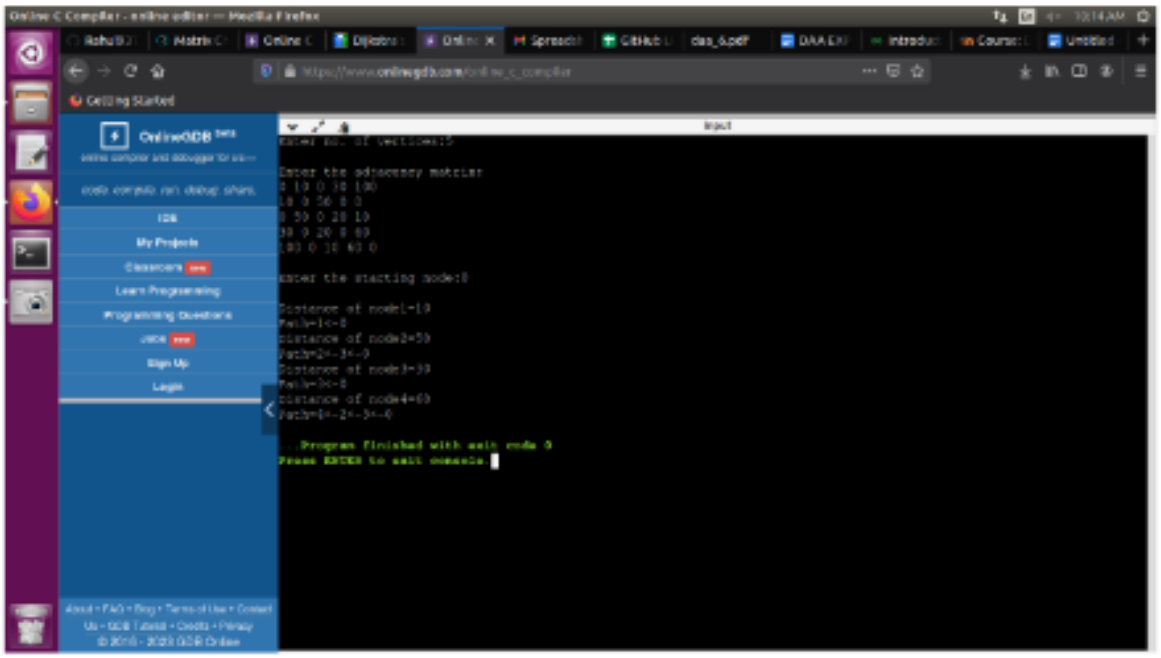
CODE:

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
int G[MAX][MAX],i,j,n,u;
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);
return 0;
}

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;
//pred[] stores the predecessor of each node
//count gives the number of nodes seen so far
//create the cost matrix
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];
//initialize pred[],distance[] and visited[]
for(i=0;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-1)
{
mindistance=INFINITY;
//nextnode gives the node at minimum distance
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
//check if a better path exists through nextnode
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}

//print the path and distance of each node
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
```

```
j=pred[j];
printf("<-%d",j);
}while(j!=startnode);
}
}
```

OUTPUT:



- **PRIM'S ALGORITHM:**

- **ALGORITHM:-**

Step 1: Determine an arbitrary vertex as the starting vertex of the MST.
Step 2: Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
Step 3: Find edges connecting any tree vertex with the fringe vertices.

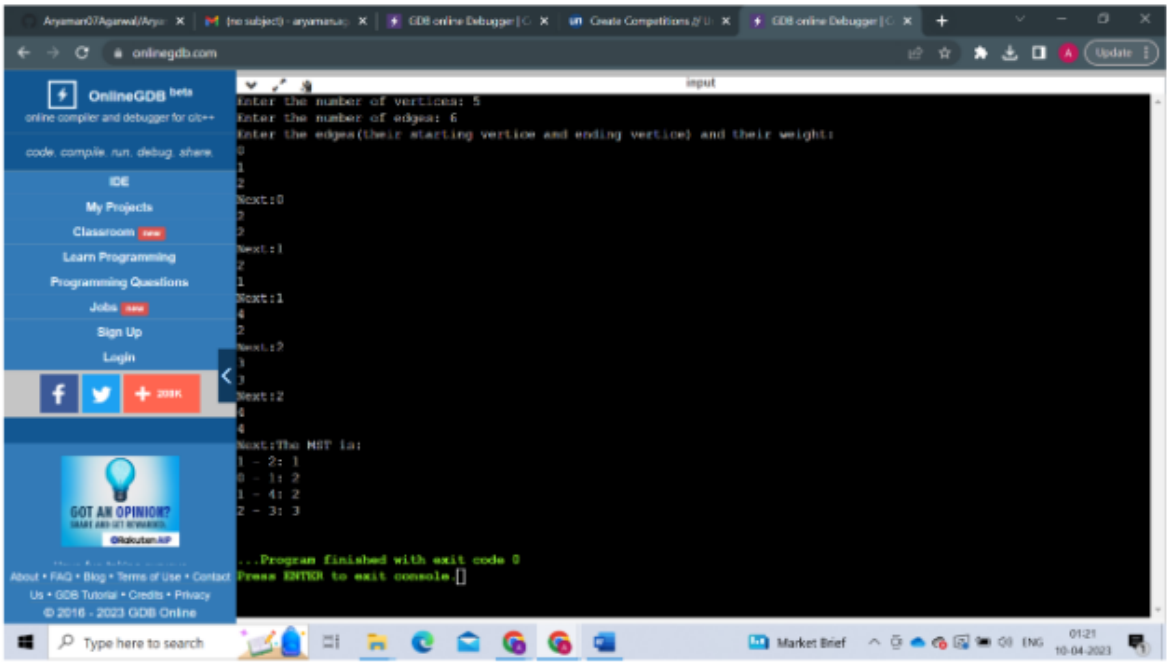
Step 4: Find the minimum among these edges.
Step 5: Add the chosen edge to the MST if it does not form any cycle.
Step 6: Return the MST and exit

PROGRAM:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

#define MAX_VERTICES 100
#define INF INT_MAX

typedef struct {
    int u, v, weight;
} Edge;

int parent[MAX_VERTICES];
Edge edges[MAX_VERTICES];
int num_edges = 0;

int find(int v) {
    if (parent[v] != v) {
        parent[v] = find(parent[v]);
    }
    return parent[v];
}

void union_sets(int u, int v) {
    parent[find(u)] = find(v);
}

// Comparator function for sorting edges by weight

int compare_edges(const void* a, const void* b) {
    Edge* e1 = (Edge*)a;
    Edge* e2 = (Edge*)b;
    return e1->weight - e2->weight;
}

// Find the MST of a graph with n vertices and m edges

void mst(int n, int m, Edge* edges) {
    for (int i = 0; i < n; i++) {
        parent[i] = i;
    }
    // Sort the edges by weight
    qsort(edges, m, sizeof(Edge), compare_edges);
    for (int i = 0; i < m && num_edges < n - 1; i++) {
```

```
        int u = edges[i].u;
        int v = edges[i].v;
        if (find(u) != find(v)) {
            union_sets(u, v);
            edges[num_edges++] = edges[i];
        }
    }
}

int main() {

    int n, m;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    printf("Enter the number of edges: ");
    scanf("%d", &m);

    printf("Enter the edges(their starting vertice and ending
vertice) and their weight:\n");

    for (int i = 0; i < m; i++)
    {
        scanf("%d%d%d", &edges[i].u, &edges[i].v,
&edges[i].weight);
        printf("Next:");
    }
    mst(n, m, edges);
    printf("The MST is:\n");
    for (int i = 0; i < num_edges; i++) {
        printf("%d - %d: %d\n", edges[i].u, edges[i].v,
edges[i].weight);
    }

    return 0;
}
```

RESULT:



CONCLUSION:	I understood how Prims Algorithm works to find out Minimum Spanning Tree.