NAME: Riya Vaid REGISTRATION NUMBER: 21BCI0014

PRIM'S MST:

ALGORITHM:

It falls under a class of algorithms called greedy algorithms that find the local optimum in the hopes of finding a global optimum.

We start from one vertex and keep adding edges with the lowest weight until we reach our goal.

The steps for implementing Prim's algorithm are as follows:

Initialize the minimum spanning tree with a vertex chosen at random.
Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
Keep repeating step 2 until we get a minimum spanning tree.

CODE:

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

#define V 5

int minKey(int key[], bool mstSet[])
{

    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] <
            min)min = key[v], min_index = v;

    return min_index;
}

int printMST(int parent[], int graph[V][V])
{
    printf("Edge
    \tWeight\n");for (int i = 1;
    i < V; i++)
        printf("%d - %d \t%d \n", parent[i],
               i,graph[i][parent[i]]);
}

void primMST(int graph[V][V])
{

    int

    parent[V];int

    key[V];

    bool mstSet[V];

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

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++)
```

```
        {int u = minKey(key, mstSet);

        mstSet[u] = true;

        for (int v = 0; v < V; v++){

            if (graph[u][v] && mstSet[v] ==
                false&& graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
        }

    }

    printMST(parent, graph);
}

int main()
{
    int graph[V][V] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },
                        { 6, 8, 0, 0, 9 },
                        { 0, 5, 7, 9, 0 } };
    primMST(graph)

    ;return 0;

}
```

# OUTPUT:

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 5
int
minKey (int key[], bool mstSet[])
{
  int min = INT_MAX, min_index;
  for (int v = 0; v < V; v++)
    if (mstSet[v] == false && key[v] < min)
      min = key[v], min_index = v;
  return min_index;
}

int
printMST (int parent[], int graph[V][V])
{
  printf ("Edge \tWeight\n");
  for (int i = 1; i < V; i++)
    printf ("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
}

```

input

```
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```

KRUSKAL'S

MSTCODE:

```
#include<stdio.h>

#define MAX 30

typedef struct edge
  {int u, v, w;
} edge;

typedef struct edge_list
  {edge data[MAX];
  int n;
} edge_list;

edge_list elist;

int Graph[MAX][MAX],
n;edge_list spanlist;

void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int
c2);void sort();
void print();

void kruskalAlgo() {
  int belongs[MAX], i, j, cno1,
  cno2;elist.n = 0;

  for (i = 1; i < n; i++)
    for (j = 0; j < i; j++)
    {if (Graph[i][j] != 0) {
        elist.data[elist.n].u = i;
        elist.data[elist.n].v = j;
        elist.data[elist.n].w = Graph[i]
        [j];elist.n++;
      }
    }

  sort();

  for (i = 0; i < n; i+
    +)belongs[i] = i;

  spanlist.n = 0;

  for (i = 0; i < elist.n; i++) {
    cno1 = find(belongs,
    elist.data[i].u);cno2 = find(belongs,
    elist.data[i].v);

    if (cno1 != cno2)
      { spanlist.data[spanlist.n] =
      elist.data[i];

      spanlist.n = spanlist.n + 1;
      applyUnion(belongs, cno1, cno2);
    }
  }
}

int find(int belongs[], int vertexno)
  {return (belongs[vertexno]);
}
```

```
void applyUnion(int belongs[], int c1, int c2)
  {int i;

  for (i = 0; i < n; i++)
    if (belongs[i] ==
    c2)belongs[i] = c1;
}

void sort()
  { int i,  j;
  edge temp;

  for (i = 1; i < elist.n; i++)
    for (j = 0; j < elist.n - 1; j++)
      if (elist.data[j].w > elist.data[j + 1].w)
        {temp = elist.data[j];
        elist.data[j] = elist.data[j +
        1];elist.data[j + 1] = temp;
      }
}

void print()
  { int i, cost =
  0;

  for (i = 0; i < spanlist.n; i++) {
    printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v,
    spanlist.data[i].w);cost = cost + spanlist.data[i].w;
  }

  printf("\nSpanning tree cost: %d", cost);
}

int main() {
  int i, j,

  total_cost;n = 6;

  Graph[0][0] = 0;
  Graph[0][1] = 4;
  Graph[0][2] = 4;
  Graph[0][3] = 0;
  Graph[0][4] = 0;
  Graph[0][5] = 0;
  Graph[0][6] = 0;

  Graph[1][0] = 4;
  Graph[1][1] = 0;
  Graph[1][2] = 2;
  Graph[1][3] = 0;
  Graph[1][4] = 0;
  Graph[1][5] = 0;
  Graph[1][6] = 0;

  Graph[2][0] = 4;
  Graph[2][1] = 2;
  Graph[2][2] = 0;
```

```
Graph[2][3] = 3;
Graph[2][4] = 4;
Graph[2][5] = 0;
Graph[2][6] = 0;

Graph[3][0] = 0;
Graph[3][1] = 0;
Graph[3][2] = 3;
Graph[3][3] = 0;
Graph[3][4] = 3;
Graph[3][5] = 0;
Graph[3][6] = 0;

Graph[4][0] = 0;
Graph[4][1] = 0;
Graph[4][2] = 4;
Graph[4][3] = 3;
Graph[4][4] = 0;
Graph[4][5] = 0;
Graph[4][6] = 0;

Graph[5][0] = 0;
Graph[5][1] = 0;
Graph[5][2] = 2;
Graph[5][3] = 0;
Graph[5][4] = 3;
Graph[5][5] = 0;
Graph[5][6] = 0;

kruskalAlgo()
;print();
}
```

# OUTPUT:

```
    Graph[3][3] = 0;
    Graph[3][4] = 3;
    Graph[3][5] = 0;
    Graph[3][6] = 0;
    Graph[4][0] = 0;
    Graph[4][1] = 0;
    Graph[4][2] = 4;
    Graph[4][3] = 3;
    Graph[4][4] = 0;
    Graph[4][5] = 0;
    Graph[4][6] = 0;
    Graph[5][0] = 0;
    Graph[5][1] = 0;
    Graph[5][2] = 2;
    Graph[5][3] = 0;
    Graph[5][4] = 3;
    Graph[5][5] = 0;
    Graph[5][6] = 0;
    kruskalAlgo ();
    print ();
}
```

input

```
5 - 2 : 2
3 - 2 : 3
4 - 3 : 3
1 - 0 : 4
Spanning tree cost: 14
```

DIJKSTAR'S ALGORITHM:

CODE:

Algorithm:

Dijkstra's Algorithm works on the basis that any subpath B -> D of the shortest path A -> D between vertices A and D is also the shortest path between vertices B and D.
Djikstra used this property in the opposite direction i.e we overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbors to find the shortest subpath to those neighbors.

The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

```
#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;
  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];

  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;

      for (i = 0; i < n; i++)
          if (distance[i] < mindistance && !visited[i])
            {
              mindistance =
              distance[i];nextnode = i;
            }
```

```
    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++;
  }

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:

```
0
1
1
0
1
1
0
0
1
1
1
0
0
1
1
0
1
1
0
1
1
1
1
1
0

Enter the starting node:0

Distance of node1=1
Path=1<-0
Distance of node2=1
Path=2<-0
Distance of node3=2
Path=3<-1<-0
Distance of node4=1
Path=4<-0
```