

PRACTICAL – 3

AIMAN SIDDIQUA

2K18/MC/008

AIM: To write a program to find the minimum spanning tree of a graph using Prim's Algorithm.

CODE:

```
#include <bits/stdc++.h>
using namespace std;

#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;
}

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

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:

 "C:\Users\aiman\Desktop\Semester 7\GT\Practicals\Programs\2K18_MC_008_GT_Practical_3.exe"

```

Edge    Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5

Process returned 0 (0x0)   execution time : 13.949 s
Press any key to continue.

```

PRACTICAL – 4

AIMAN SIDDIQUA

2K18/MC/008

AIM: To write a program to find the minimum spanning tree of a graph using Kruskal's Algorithm.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Edge {
    int src, dest, weight;
};

struct Graph {
    int V, E;

    struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = (struct Graph*) (malloc(sizeof(struct Graph)));
    graph->V = V;
    graph->E = E;

    graph->edge = (struct Edge*) malloc(sizeof( struct Edge)*E);

    return graph;
}

struct subset {
    int parent;
    int rank;
};

int find(struct subset subsets[], int i)
{
    if (subsets[i].parent != i)
        subsets[i].parent
            = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}

void Union(struct subset subsets[], int x, int y)
```

```

{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;

    else
    {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

int myComp(const void* a, const void* b)
{
    struct Edge* a1 = (struct Edge*)a;
    struct Edge* b1 = (struct Edge*)b;
    return a1->weight > b1->weight;
}

void KruskalMST(struct Graph* graph)
{
    int V = graph->V;
    struct Edge
        result[V];
    int e = 0;
    int i = 0;

    qsort(graph->edge, graph->E, sizeof(graph->edge[0]),
        myComp);

    struct subset* subsets
        = (struct subset*)malloc(V * sizeof(struct subset));

    for (int v = 0; v < V; ++v) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    while (e < V - 1 && i < graph->E) {

        struct Edge next_edge = graph->edge[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        if (x != y) {
            result[e++] = next_edge;
            Union(subsets, x, y);
        }
    }
}

```

```

    }

    printf(
        "Following are the edges in the constructed MST\n");
    int minimumCost = 0;
    for (i = 0; i < e; ++i)
    {
        printf("%d -- %d == %d\n", result[i].src,
            result[i].dest, result[i].weight);
        minimumCost += result[i].weight;
    }
    printf("\nMinimum Cost Spanning tree : %d", minimumCost);
    return;
}

int main()
{
    int V = 4;
    int E = 5;
    struct Graph* graph = createGraph(V, E);

    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = 10;

    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 6;

    graph->edge[2].src = 0;
    graph->edge[2].dest = 3;
    graph->edge[2].weight = 5;

    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 15;


    graph->edge[4].src = 2;
    graph->edge[4].dest = 3;
    graph->edge[4].weight = 4;

    KruskalMST(graph);

    return 0;
}

```

OUTPUT:

 "C:\Users\aiman\Desktop\Semester 7\GT\Practicals\Programs\2K18_MC_008_Practical_4.exe"

Following are the edges in the constructed MST

2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10

Minimum Cost Spanning tree : 19

Process returned 0 (0x0) execution time : 7.147 s

Press any key to continue.

■