

## 23CSE –111- DESIGN AND ANALYSIS OF ALGORITHM

### LAB-4

### PRIMS'S AND KRUSKEL'S ALGORITHMS

#### PRIMS'S ALGORITHM:

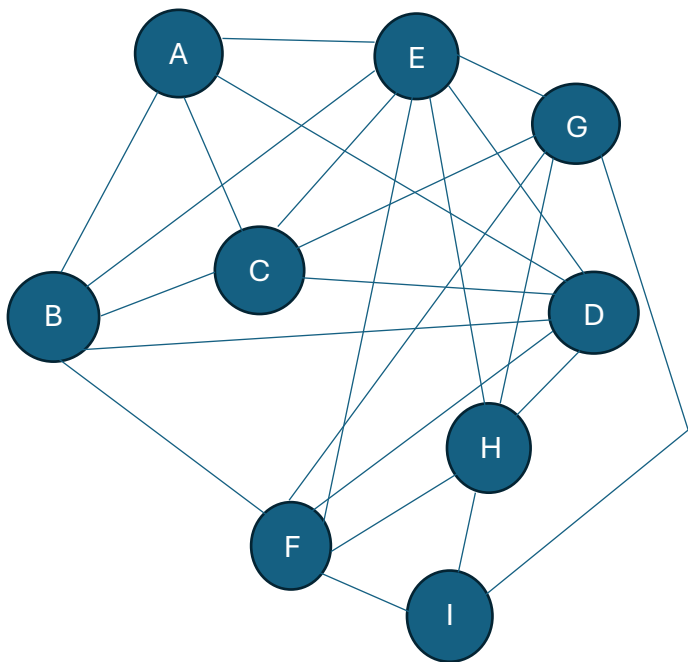
Prim's Algorithm is used to find a Minimum Spanning Tree (MST) of a graph.

- Connect all vertex, no cycles and Total edge cost is minimum

#### ALGORITHM:

1. Start with any vertex and mark it as part of the Minimum Spanning Tree (MST).
2. Initialize the cost of all other vertices as infinity.
3. Find the edge with the minimum weight that connects a vertex in the MST to a vertex outside the MST.
4. Add this edge and the corresponding vertex to the MST.
5. Repeat steps 3 and 4 until all vertices are included in the MST.

#### GRAPH DIAGRAM:



#### Weight of the graph:

- i. A-B = 4   A-C = 8   A-D = 6   A-E = 7

- ii. B-C = 5   B-D = 3   B-F = 9   B-E = 6
- iii. C-D = 4   C-E = 2   C-G = 10
- iv. D-E = 5   D-F = 7   D-H = 6
- v. E-F = 4   E-G = 8   E-H = 9
- vi. F-G = 3   F-H = 5   F-I = 6
- vii. G-H = 4   G-I = 7
- viii. H-I = 2

**CODE:**

```
#include <stdio.h>
#include <limits.h>
#define V 9
int minKey(int key[], int mstSet[]) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++) {
        if (!mstSet[v] && key[v] < min) {
            min = key[v];
            min_index = v;
        }
    }
    return min_index;
}
void printMST(int parent[], int graph[V][V]) {
    int cost = 0;
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++) {
        printf("%c - %c \t%d\n", parent[i] + 'A', i + 'A',
            graph[i][parent[i]]);
        cost += graph[i][parent[i]];
    }
    printf("Total MST Cost = %d\n", cost);
}
void primMST(int graph[V][V]) {
    int parent[V];
    int key[V];
    int mstSet[V];
    for (int i = 0; i < V; i++) {
        key[i] = INT_MAX;
```

```

    mstSet[i] = 0;
}
key[0] = 0;
parent[0] = -1;
for (int count = 0; count < V - 1; count++) {
    int u = minKey(key, mstSet);
    mstSet[u] = 1;
    for (int v = 0; v < V; v++) {
        if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {
            parent[v] = u;
            key[v] = graph[u][v];
        }
    }
}
printMST(parent, graph);
}

int main() {
    int graph[V][V] = {
        {0,4,8,6,7,0,0,0,0},
        {4,0,5,3,6,9,0,0,0},
        {8,5,0,4,2,0,10,0,0},
        {6,3,4,0,5,7,0,6,0},
        {7,6,2,5,0,4,8,9,0},
        {0,9,0,7,4,0,3,5,6},
        {0,0,10,0,8,3,0,4,7},
        {0,0,0,6,9,5,4,0,2},
        {0,0,0,0,0,6,7,2,0}
    };
    primMST(graph);
    return 0;
}

```

**OUTPUT:**

```

vangapandu-kishor@vangapandu-kishor-IdeaPad-Slim-5-14IRH10:~/Downloads$ gcc prims.c -o prims
vangapandu-kishor@vangapandu-kishor-IdeaPad-Slim-5-14IRH10:~/Downloads$ ./prims
Edge    Weight
A - B    4
D - C    4
B - D    3
C - E    2
E - F    4
F - G    3
G - H    4
H - I    2
Total MST Cost = 26
vangapandu-kishor@vangapandu-kishor-IdeaPad-Slim-5-14IRH10:~/Downloads$ S

```

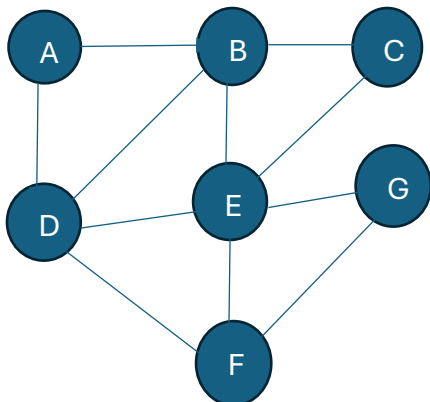
## KRUSKAL'S ALGORITHM:

Kruskal's algorithm constructs a minimum spanning tree by selecting edges in ascending order of weight without forming cycles.

### ALGORITHM:

1. Start with an empty set of edges for the Minimum Spanning Tree (MST).
2. Sort all the edges of the graph in increasing order of their weights.
3. Select the edge with the minimum weight from the sorted list.
4. Add this edge to the MST if it does not form a cycle.
5. Repeat steps 3 and 4 until the MST contains  $V-1$  edges.

### GRAPH DIAGRAM:



### Weight of the graph:

- I.  $A - B = 7$  ,  $A - D = 5$
- II.  $B - C = 8$  ,  $B - D = 9$  ,  $B - E = 7$
- III.  $C - E = 5$
- IV.  $D - E = 15$  ,  $D - F = 6$
- V.  $E - F = 8$  ,  $E - G = 9$
- VI.  $F - G = 11$

**CODE:**

```
#include <stdio.h>
#include <stdlib.h>
#define V 7
#define E 11
struct Edge {
    int src, dest, weight;
};
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 compare(const void *a, const void *b) {
    return ((struct Edge *)a)->weight - ((struct Edge *)b)->weight;
```

```

}
void KruskalMST(struct Edge edges[]) {
    struct Edge result[V];
    int e = 0, i = 0;
    qsort(edges, E, sizeof(edges[0]), compare);
    struct Subset subsets[V];
    for (int v = 0; v < V; v++) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }
    while (e < V - 1 && i < E) {
        struct Edge next = edges[i++];
        int x = find(subsets, next.src);
        int y = find(subsets, next.dest);
        if (x != y) {
            result[e++] = next;
            Union(subsets, x, y);
        }
    }
    int total = 0;
    printf("Edge \tWeight\n");
    for (i = 0; i < e; i++) {
        printf("%c - %c \t%d\n",
            result[i].src + 'A',
            result[i].dest + 'A',
            result[i].weight);
        total += result[i].weight;
    }
    printf("Total MST Cost = %d\n", total);
}

int main() {
    struct Edge edges[E] = {
        {0,1,7},
        {0,3,5},
        {1,2,8},
        {1,3,9},
        {1,4,7},
        {2,4,5},
    }
}

```

```
    {3,4,15},  
    {3,5,6},  
    {4,5,8},  
    {4,6,9},  
    {5,6,11}  
};  
KruskalMST(edges);  
return 0;  
}
```

### OUTPUT:

```
vangapandu-kishor@vangapandu-kishor-IdeaPad-Slim-5-14IRH10:~/Downloads$ gcc krushkal.c -o krushkal  
vangapandu-kishor@vangapandu-kishor-IdeaPad-Slim-5-14IRH10:~/Downloads$ ./krushkal  
Edge    Weight  
A - D    5  
C - E    5  
D - F    6  
A - B    7  
B - E    7  
E - G    9  
Total MST Cost = 39  
vangapandu-kishor@vangapandu-kishor-IdeaPad-Slim-5-14IRH10:~/Downloads$ S
```