

DESIGN AND ANALYSIS OF ALGORITHM

LAB EXPERIMENT -4

KOLATHUR SURYA

[CH.SC.U4CSE24224]

1. PRIM'S ALGORITHM

Code :

```
GNU nano 7.2                                                                    prims.c *
#include <stdio.h>
#include <limits.h>

#define INF 9999

int n;

int minKey(int key[], int mstSet[]) {
    int min = INF, min_index = -1;
    for (int v = 0; v < n; v++)
        if (mstSet[v] == 0 && key[v] < min)
            min = key[v], min_index = v;
    return min_index;
}

void printMST(int parent[], int graph[20][20]) {
    printf("Edge\tWeight\n");
    for (int i = 1; i < n; i++)
        printf("%d-%d\t%d\n", parent[i], i, graph[i][parent[i]]);
}

void primMST(int graph[20][20]) {
    int parent[20];
    int key[20];
    int mstSet[20];

    for (int i = 0; i < n; i++)
        key[i] = INF, mstSet[i] = 0;

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

    for (int count = 0; count < n - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = 1;

        for (int v = 0; v < n; v++)
            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }
    printMST(parent, graph);
}

int main() {
    int graph[20][20];

    printf("Name:KOLATHURSURYA\n");
    printf("RollNo:CH.SC.U4CSE24224\n\n");

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

    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

    primMST(graph);
    return 0;
}
```

Output :

```
bsurya@DESKTOP-FJ2EKA4:~$ nano prims.c
surya@DESKTOP-FJ2EKA4:~$ gcc prims.c -o prims
surya@DESKTOP-FJ2EKA4:~$ ./prims
Name:KOLATHURSURYA
RollNo:CH.SC.U4CSE24224

Enter number of vertices:3
Enter adjacency matrix:
12
32
12
54
32
12
23
43
23
Edge      Weight
0-1       54
0-2       23
surya@DESKTOP-FJ2EKA4:~$
```

Time Complexity: $O(N^2)$

Since the graph is represented using an adjacency matrix, two nested loops are used. The outer loop runs N times to include all vertices in the Minimum Spanning Tree. The inner loop scans all N vertices to find the minimum key value and update the adjacent vertices. Hence, the total number of operations is $N \times N$, resulting in a time complexity of $O(N^2)$.

Space Complexity: $O(N)$

The algorithm uses three auxiliary arrays: key, parent, and mstSet, each of size N .

Therefore, the extra space required is proportional to the number of vertices, giving a space complexity of $O(N)$.

2. KRUSKAL'S ALGORITHM

Code:

```
GNU nano 7.2                                                                    kruskals.c
#include <stdio.h>
#include <stdlib.h>

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

struct Edge edges[100];
int parent[20];
int n, e_cnt = 0;

int find(int i) {
    if (parent[i] == -1)
        return i;
    return find(parent[i]);
}

void Union(int x, int y) {
    int xset = find(x);
    int yset = find(y);
    if (xset != yset)
        parent[xset] = yset;
}

int compare(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() {
    int mst_weight = 0;
    qsort(edges, e_cnt, sizeof(edges[0]), compare);

    for (int i = 0; i < n; i++)
        parent[i] = -1;

    printf("Edge\tWeight\n");

    for (int i = 0; i < e_cnt; i++) {
        int x = find(edges[i].src);
        int y = find(edges[i].dest);

        if (x != y) {
            printf("%d-%d\t%d\n", edges[i].src, edges[i].dest, edges[i].weight);
            mst_weight += edges[i].weight;
            Union(x, y);
        }
    }
    printf("TotalCost:%d\n", mst_weight);
}

int main() {
    printf("Name:KOLATHURSURYA\n");
    printf("RollNo:CH.SC.U4CSE24224\n\n");

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

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

    for (int i = 0; i < e_cnt; i++) {
        printf("Enter src dest weight for edge %d:", i + 1);
        scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);
    }

    kruskalMST();
    return 0;
}
```

Output:

```
surya@DESKTOP-FJ2EKA4:~$ nano kruskals.c
surya@DESKTOP-FJ2EKA4:~$ gcc kruskals.c -o kruskals
surya@DESKTOP-FJ2EKA4:~$ ./kruskals
Name:KOLATHURSURYA
RollNo:CH.SC.U4CSE24224

Enter number of vertices:3
Enter number of edges:3
Enter src dest weight for edge 1:2
3
4
Enter src dest weight for edge 2:1
2
5
Enter src dest weight for edge 3:1
42
43
Edge      Weight
2-3       4
1-2       5
TotalCost:9
surya@DESKTOP-FJ2EKA4:~$ █
```

Time Complexity: $O(E \log E)$ or $O(E \log N)$

The most time-consuming step in Kruskal's algorithm is sorting all edges based on their weights, which takes $O(E \log E)$ time.

The Union-Find operations (find and union) run in almost constant time on average, i.e., $O(\alpha(N))$, where α is the inverse Ackermann function.

Since sorting dominates the overall execution time, the final time complexity is $O(E \log E)$, which is also commonly written as $O(E \log N)$.

Space Complexity: $O(E + N)$

The algorithm uses an array of size E to store all edges and a parent array of size N for the Union-Find data structure.

Hence, the total space required depends on both the number of edges and vertices, resulting in a space complexity of $O(E + N)$.