**Practical-7**

**AIM:** To implement Prim’s and Kruskal’s Algorithm to find the Minimum Spanning Trees using C language.

**SOFTWARE REQUIRED:** Vs Code

**PSEUDO CODE:**

Input: Graph G represented by an adjacency matrix or adjacency list

1. Initialize an empty set MST to store the Minimum Spanning Tree.

2. Select a starting vertex startVertex.

3. Create an empty priority queue pq.

4. Insert (startVertex, 0) into pq, where 0 is the initial key value.

5. While pq is not empty:

a. Extract the vertex u with the minimum key value from pq.

b. Add u to the MST set.

c. For each vertex v adjacent to u:

- If v is not in MST and the edge weight u-v is smaller than the current key value of v:

i. Update the key value of vertex v in pq to the edge weight u-v.

ii. Update the parent of vertex v to be u.

6. Once all vertices are processed, the MST set will contain the Minimum Spanning Tree.

**CODE:**

**a)Prim’s Algorithm:**

#include <stdio.h>

#include <stdbool.h>

#include <limits.h>

#define MAX\_VERTICES 20

int findMinKey(int key[], bool mstSet[], int V) {

    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 primMST(int graph[MAX\_VERTICES][MAX\_VERTICES], int V) {

    int parent[MAX\_VERTICES];

    int key[MAX\_VERTICES];

    bool mstSet[MAX\_VERTICES];

    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 = findMinKey(key, mstSet, V);

        mstSet[u] = true;

        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];

            }

        }

    }

    printf("Edge   Weight\n");

    for (int i = 1; i < V; i++) {

        printf("%d - %d    %d\n", parent[i], i, graph[i][parent[i]]);

    }

}

int main() {

    int V;

    printf("Name: Ananta Walli");

    printf("\nEnorllment Number: A2305221322");

    printf("\nEnter the number of vertices: ");

    scanf("%d", &V);

    int graph[MAX\_VERTICES][MAX\_VERTICES];

    printf("Enter the adjacency matrix:\n");

    for (int i = 0; i < V; i++) {

        for (int j = 0; j < V; j++) {

            scanf("%d", &graph[i][j]);

        }

    }

    primMST(graph, V);

    return 0;

}

**a)Kruskal’s Algorithm:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 20

#define MAX\_EDGES 50

struct Edge {

    int src, dest, weight;

};

struct Graph {

    int V, E;

    struct Edge edges[MAX\_EDGES];

};

int findParent(int parent[], int i) {

    if (parent[i] == -1)

        return i;

    return findParent(parent, parent[i]);

}

void unionSets(int parent[], int x, int y) {

    int xroot = findParent(parent, x);

    int yroot = findParent(parent, y);

    parent[xroot] = yroot;

}

int compareEdges(const void\* a, const void\* b) {

    return ((struct Edge\*)a)->weight - ((struct Edge\*)b)->weight;

}

void kruskalMST(struct Graph\* graph) {

    int V = graph->V;

    struct Edge result[V];

    int e = 0;

    int i = 0;

    qsort(graph->edges, graph->E, sizeof(graph->edges[0]), compareEdges);

    int parent[MAX\_VERTICES];

    for (int v = 0; v < V; v++)

        parent[v] = -1;

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

        struct Edge next\_edge = graph->edges[i++];

        int x = findParent(parent, next\_edge.src);

        int y = findParent(parent, next\_edge.dest);

        if (x != y) {

            result[e++] = next\_edge;

            unionSets(parent, x, y);

        }

    }

    printf("Edge   Weight\n");

    for (i = 0; i < e; i++)

        printf("%d - %d    %d\n", result[i].src, result[i].dest, result[i].weight);

}

int main() {

    struct Graph graph;

    printf("Name: Ananta Walli");

    printf("\nEnrollment Number:A2305221322");

    printf("\nEnter the number of vertices: ");

    scanf("%d", &graph.V);

    printf("Enter the number of edges: ");

    scanf("%d", &graph.E);

    printf("Enter edge details (src, dest, weight):\n");

    for (int i = 0; i < graph.E; i++)

        scanf("%d %d %d", &graph.edges[i].src, &graph.edges[i].dest, &graph.edges[i].weight);

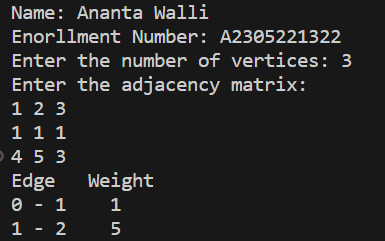
    kruskalMST(&graph);

    return 0;

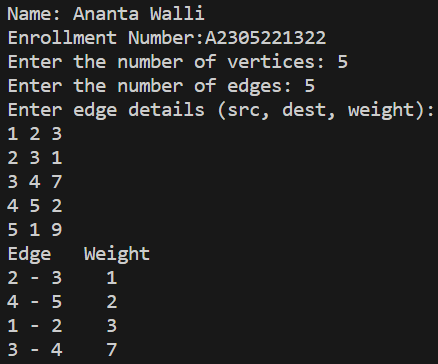
}

**OUTPUT:**

**a)Prim’s Algorithm**



**a)Kruskal’s Algorithm**

****

**RESULT:** The above code implements the Prim’s and Kruskal Algorithm to obtain minimum spanning tress in C programming.