**NAME:** Aditya Jethani

**ROLL NO:** 21BCP166

**DIV-**3 **G-**5

**DAA Assignment 6**

**AIM:**

Implement Greedy Algorithm: Kruskal using union-find data structure.

Kruskal's algorithm is a greedy algorithm used to find the minimum spanning tree (MST) of a connected, undirected graph. The MST of a graph is the subset of edges that connect all the vertices of the graph with the minimum possible total edge weight.

The algorithm starts by sorting all the edges in the graph in increasing order of their weight. It then processes the edges one by one, adding each edge to the MST if it does not create a cycle with the edges already included in the MST. To check for cycles, the algorithm uses the union-find data structure, which keeps track of the subsets of vertices that have been connected by edges.

The algorithm terminates when it has added V-1 edges to the MST, where V is the number of vertices in the graph. This is because a connected graph with V vertices always has exactly V-1 edges in its MST.

**CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// Structure to represent a weighted edge in the graph

struct Edge {

    int src, dest, weight;

};

// Structure to represent a subset for union-find

struct Subset {

    int parent, rank;

};

// Function to compare edges by their weight

bool compareEdges(Edge a, Edge b) {

    return a.weight < b.weight;

}

// Function to find the parent of a subset using path compression technique

int find(Subset subsets[], int i) {

    if (subsets[i].parent != i)

        subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;

}

// Function to do union of two subsets using rank

void unionSubsets(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++;

    }

}

// Function to find the Minimum Spanning Tree of the graph using Kruskal's algorithm

vector<Edge> kruskalMST(int V, vector<Edge> edges) {

    vector<Edge> result;

    Subset subsets[V];

    int e = 0;

    // Sort all the edges in increasing order of their weight

    sort(edges.begin(), edges.end(), compareEdges);

    // Initialize subsets for union-find

    for (int i = 0; i < V; i++) {

        subsets[i].parent = i;

        subsets[i].rank = 0;

    }

    // Process all the edges in sorted order

    for (int i = 0; i < edges.size(); i++) {

        int x = find(subsets, edges[i].src);

        int y = find(subsets, edges[i].dest);

        // If including this edge does not cause a cycle, include it in result and increment the index of result

        if (x != y) {

            result.push\_back(edges[i]);

            unionSubsets(subsets, x, y);

            e++;

        }

        // If we have included V-1 edges, we have found the MST

        if (e == V - 1)

            break;

    }

    return result;

}

// Function to print the edges of the MST

void printMST(vector<Edge> edges) {

    int total\_weight = 0;

    cout << "Edges of the Minimum Spanning Tree:" << endl;

    for (int i = 0; i < edges.size(); i++) {

        cout << edges[i].src << " -- " << edges[i].dest << "  (weight: " << edges[i].weight << ")" << endl;

        total\_weight += edges[i].weight;

    }

    cout << "Total weight of the MST: " << total\_weight << endl;

}

// Driver code

int main() {

    int V, E;

    cout << "Enter the number of vertices and edges in the graph: ";

    cin >> V >> E;

    vector<Edge> edges;

    cout << "Enter the source vertex, destination vertex, and weight of each edge:" << endl;

    for (int i = 0; i < E; i++) {

        Edge e;

        cin >> e.src >> e.dest >> e.weight;

        edges.push\_back(e);

    }

    vector<Edge> mst = kruskalMST(V, edges);

    printMST(mst);

    return 0;

}

**OUTPUT:**

Text

Description automatically generated

**ANALYSIS:**

The algorithm uses the union-find data structure to check for cycles. Kruskal's algorithm has a time complexity of **O(E log E)** and a space complexity of **O(V + E),** where E is the number of edges in the graph and V is the number of vertices. It is efficient for sparse graphs but may not be the best choice for dense graphs or graphs with large edge weights.