kruskal’s algorithm

Name: Raneem Montaser Ibrahim

Teemed with :sara Ashraf sec 3

Sec:3 CS department

Id:1000276904

(a): Required Steps

1. **Input Representation**:
   Represent the graph using an edge list with weights. Each edge is represented as a tuple: (u, v, w), where u and v are vertices and w is the weight.

2. **Sort Edges by Weight**:
   Sort all edges in ascending order of their weights.

3. **Union-Find Data Structure**:
   Use a Union-Find (disjoint set) structure to manage connected components and detect cycles efficiently.

4. **MST Construction**:
   Iterate through the sorted edges and add an edge to the MST if it connects two different components. Stop when the MST contains $n - 1$ edges, where n is the number of vertices.

(b): Analysis

1. Sorting the Edges

- **Purpose**: Sorting ensures that edges are considered in increasing order of weight, as required by Kruskal's algorithm.
- **Time Complexity**: Sorting the E edges takes $O(E \log E)$, where E is the number of edges.

2. Union-Find Operations

- **Purpose**: The Union-Find structure helps efficiently manage connected components and detect cycles. It supports:
  - **Find Operation**: Determines the root (representative) of a component.
  - **Union Operation**: Merges two components into one.
- **Time Complexity**: Using path compression and union by rank, each operation runs in $O\big(\alpha(V)\big)$, where $\alpha(V)$ is the inverse Ackermann function, which grows very slowly.

the total complexity of Union-Find operations is approximately $O\big(E \cdot \alpha(V)\big)$.

3. Constructing the MST

- **Purpose**: Construct the MST by iterating through the sorted edges and checking connectivity using the Union-Find structure.
- **Time Complexity**: Checking and adding E edges involves $O(E)$ operations with the Union-Find structure, contributing to $O\big(E \cdot \alpha(V)\big)$.

Overall Time Complexity

$$O(E\log E) + O\big(E \cdot \alpha(V)\big) \approx O(E\log E),$$

$O(E\log E)$ dominates $O\big(E \cdot \alpha(V)\big)$ for practical graph sizes.

(c): Implementation

C# implementation of Kruskal's algorithm:

```
using System;
using System.Collections.Generic;

class KruskalAlgorithm
{
    static void Main(string[] args)
    {
        Console.Write("Enter the number of vertices: ");
        int vertices = int.Parse(Console.ReadLine());

        Console.Write("Enter the number of edges: ");
        int edgeCount = int.Parse(Console.ReadLine());

        var edges = new List<(int, int, int)>();
        Console.WriteLine("Enter edges in the format: u v weight (0-indexed
vertices)");
        for (int i = 0; i < edgeCount; i++)
        {
            string[] edgeInput = Console.ReadLine().Split();
            int u = int.Parse(edgeInput[0]);
            int v = int.Parse(edgeInput[1]);
            int weight = int.Parse(edgeInput[2]);


            if (u < 0 || v < 0 || u >= vertices || v >= vertices)
            {
```

```
                Console.WriteLine($"Error: Invalid edge ({u}, {v}). Vertex
indices must be between 0 and {vertices - 1}.");
                i--;
                continue;
            }

            edges.Add((u, v, weight));
        }

        Console.WriteLine("Edges in the Minimum Spanning Tree:");
        KruskalMST(edges, vertices);
    }

    static void KruskalMST(List<(int, int, int)> edges, int vertices)
    {

        edges.Sort((a, b) => a.Item3.CompareTo(b.Item3));


        int[] parent = new int[vertices];
        int[] rank = new int[vertices];
        for (int i = 0; i < vertices; i++)
        {
            parent[i] = i;
        }

        var mst = new List<(int, int, int)>();
        int mstWeight = 0;

        foreach (var (u, v, w) in edges)
        {
            if (Find(parent, u) != Find(parent, v))
            {
                mst.Add((u, v, w));
                mstWeight += w;
                Union(parent, rank, u, v);
            }
        }

        foreach (var (u, v, w) in mst)
        {
            Console.WriteLine($"Edge: {u} - {v}, Weight: {w}");
        }
        Console.WriteLine($"Total Weight of MST: {mstWeight}");
    }

    static int Find(int[] parent, int i)
    {
        if (parent[i] != i)
        {
```

```
            parent[i] = Find(parent, parent[i]);
        }
        return parent[i];
    }

    static void Union(int[] parent, int[] rank, int x, int y)
    {
        int rootX = Find(parent, x);
        int rootY = Find(parent, y);

        if (rootX != rootY)
        {
            if (rank[rootX] < rank[rootY])
            {
                parent[rootX] = rootY;
            }
            else if (rank[rootX] > rank[rootY])
            {
                parent[rootY] = rootX;
            }
            else
            {
                parent[rootY] = rootX;
                rank[rootX]++;
            }
        }
    }
}
```

Output Example

Example:

Given the input:

```
Vertices: 4
Edges: 5
Edge List: [(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]
```

Steps:

5. **Sort Edges by Weight:**

$$[(2,3,4),(0,3,5),(0,2,6),(0,1,10),(1,3,15)]$$

6. **Initialize Union-Find:**

- Parent array: [0,1,2,3]
- Rank array: [0,0,0,0]

7. **Construct MST:**
   - Add edges to MST in order, checking connectivity:
     - Edge $(2,3,4)$
     - Edge $(0,3,5)$
     - Edge $(0,1,10)$
8. **Output MST:**
   - MST Edges: $(2,3,4), (0,3,5), (0,1,10)$
   - Total Weight: 19