UNIVERSITY OF TOMORROW

Design & Analysis of Algorithm (Lab)

Name: Ananya

SAPID: 590013832

B-33

Submitted to: Mr.Aryan Gupta

https://github.com/ananya438/DAALAB_ANANYA-590013832

Kruskal's Spanning TREE ALGO

1. Kruskal's Algorithm :

(a) Working Principle :

→ This is a greedy algorithm used to find Minimum Spanning Tree, of a connected weighted graph.

STEP1 : Sort all edges of the graph in increasing order of weights.

STEP2 : Initialize an empty set for the MST.

STEP3 : Pick the smallest edge from sorted list & check if it forms a cycle in MST.

STEP4 : If no cycle is formed, include the edge in MST.

STEP5 : Repeat steps 3 & 4 until the MST contains exactly (V-1) edges, where V = no. of vertices.

The algorithm ensures that the total weight of MST is minimized.

EXAMPLE:

5. Kruskal's Algo (MST) :

Find (MST) using it.

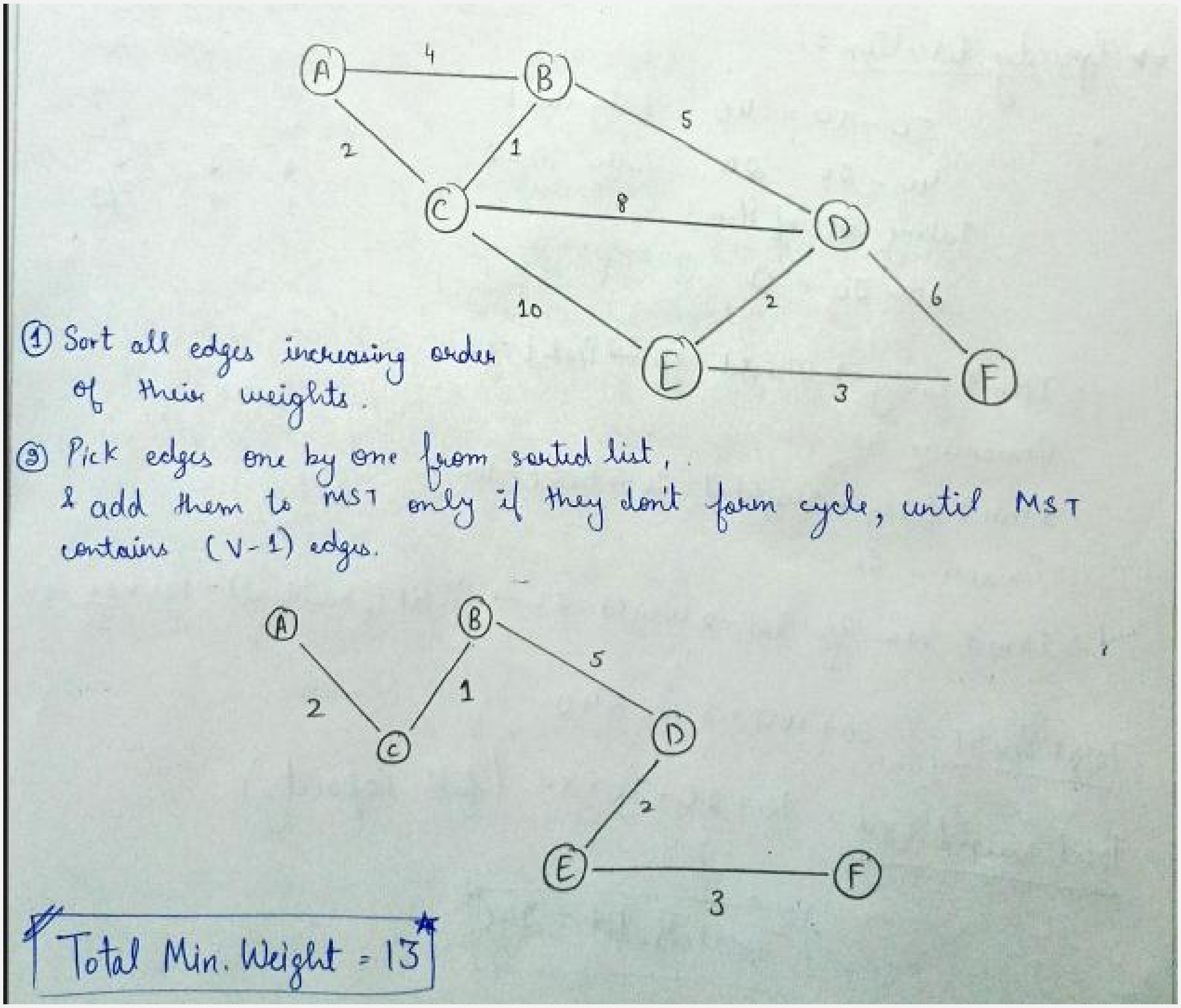
Vertices = {A, B, C, D, E, F}

Edges with weights

(A, B, 4) (A, C, 2), (B, C, 1), (B, D, 5), (C, D, 8), (C, E, 10),

(D, E, 2), (D, F, 6), (E, F, 3).

* Original Weighted Graph :




```java
import java.util.*;

class Edge implements Comparable<Edge> {

   int src, dest, weight;

   Edge(int s, int d, int w) { src = s; dest = d; weight = w; }

   public int compareTo(Edge o) { return this.weight - o.weight; } }

class Subset { int parent, rank; }

public class KruskalMST {

   int V, E; Edge[] edges;

   KruskalMST(int v, int e) { V = v; E = e; edges = new Edge[E]; }

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

       if (subsets[i].parent != i)

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

       return subsets[i].parent; }
```

```
    void union(Subset[] subsets, int x, int y) {
      int xr = find(subsets, x), yr = find(subsets, y);
      if (subsets[xr].rank < subsets[yr].rank) subsets[xr].parent = yr;
      else if (subsets[xr].rank > subsets[yr].rank) subsets[yr].parent = xr;
      else { subsets[yr].parent = xr; subsets[xr].rank++;   }
  void kruskalMST() {
      Arrays.sort(edges);
      Edge[] result = new Edge[V - 1];
      Subset[] subsets = new Subset[V];
      for (int v = 0; v < V; v++) { subsets[v] = new Subset(); subsets[v].parent = v; }
      int e = 0, i = 0, total = 0;
      while (e < V - 1 && i < E) {
          Edge next = edges[i++];
          int x = find(subsets, next.src), y = find(subsets, next.dest);
          if (x != y) { result[e++] = next; union(subsets, x, y); }  }
      for (i = 0; i < e; i++) {
          System.out.println(result[i].src + " - " + result[i].dest + " : " + result[i].weight);
          total += result[i].weight;   }
      System.out.println("Total weight of MST = " + total)   }
  public static void main(String[] args) {
      int V = 6, E = 9;
      KruskalMST g = new KruskalMST(V, E);
      g.edges[0] = new Edge(0, 1, 4);
      g.edges[1] = new Edge(0, 2, 2);
      g.edges[2] = new Edge(1, 2, 1);
      g.edges[3] = new Edge(1, 3, 5);
      g.edges[4] = new Edge(2, 3, 8);
      g.edges[5] = new Edge(2, 4, 10);
      g.edges[6] = new Edge(3, 4, 2);
      g.edges[7] = new Edge(3, 5, 6);
      g.edges[8] = new Edge(4, 5, 3);
      g.kruskalMST();}
}
```

O/P:

```
PS C:\Users\nannu\Desktop\JAVA DSA\JAVA\First lectures>  & 'C:\Pr
nannu\AppData\Roaming\Code\User\workspaceStorage\0af2579802541dcb
1 - 2 : 1
0 - 2 : 2
3 - 4 : 2
4 - 5 : 3
1 - 3 : 5
Total weight of MST = 13
```

Time Complexity

O(E log E), where E is the number of edges. This is because the algorithm is dominated by the time it takes to sort all the edges.

It can also be written as **O(E log V)**, where V is the number of vertices, because in a connected graph, log E is on the same order as log V.

Space Complexity

O(V + E). This is because the algorithm needs to store all the edges and the Disjoint Set Union (Union-Find) data structure to keep track of the vertices.