**Minimum Spanning tree (MST)**

Definition  
Die Teilmenge der Kanten, die alle Scheitelpunkte im Diagramm verbinden und das minimale Gesamtgewicht haben. Wir benutzen Kruskal Algorithmus, um MST zu finden.

1. **Disjoint Sets:**

Definition: eine Datenstruktur, die alle Elemente verfolgt, die durch eine Anzahl disjunkter (nicht verbundener) Teilmengen getrennt sind.

Die Funktionen von DisjointSet():

**add(E)**

Fügt der Struktur eine neue Gruppe hinzu, die nur Kanten enthält. Wenn es bereits zu einigen der disjunkten Mengen gehört, passiert nichts

**inSameSet(e1,e2)**  
Prüft, ob zwei Elemente zur gleichen Menge gehören.

**union(e1,e2)**

Beispiel : union(edge1, edge2) // Verbinde zwei Mengen zu einer einzigen Teilmenge

* Vereinigung der Menge mit edge1 und der Menge mit edge2.
* Nach dieser Operation gibt inSameSet (e1, e2) true zurück.
* Ob  edge1 oder edge2 gehören zu keiner Menge oder sind es bereits gehören zu der gleichen Menge, passiert nichts.

Die Operation union() benutzt die Operation join() und root(), um die Wurzeln der Bäume zu bestimmen, zu denen x und y gehören.

Die Operation join() mit der Rangzuordnung verknüpft den kürzeren Baum immer mit der Wurzel des größeren Baums. Daher ist der resultierende Baum nicht höher als die Originale, es sei denn, sie sind gleich hoch. In diesem Fall ist der resultierende Baum einen Knoten höher.

Die Operation root() folgt der Kette der übergeordneten Zeiger von x bis zu einem Stammelement, dessen übergeordnetes Element es selbst ist. Dieses Wurzelelement ist das repräsentative Mitglied der Menge, zu der x gehört, und kann x selbst sein.   
Durch die Pfadkomprimierung wird die Struktur des Baums geglättet, indem mit root () die einzelnen Knoten auf den Stamm verweisen. Dies ist gültig, da jedes Element, das auf dem Weg zu einer Wurzel besucht wird, Teil derselben Menge ist.

1. **Kruskal Algorithm**

Der Kruskal-Algorithmus zum Auffinden des Spanning Tree mit minimalen Kosten verwendet den Greedy-Ansatz.

KRUSKAL(G):

1 MST = ∅ // List to keep all the edges in MST

2 **foreach** v ∈ G.V:

3 DisjointSet = MAKE-DISJOINT-SET(v)

4 **foreach** (u, v) in G.E ordered by weight(u, v), increasing:

5 **if** UNION(FIND-SET(u), FIND-SET(v)) von DisjointSet :

6 MST = MST ∪ {(u, v)}

7 **if** MST.size == G.getNodeCount -1:

8 break;

9 **return** MST

Wir implementieren wie folgt:

* Sortieren Sie die Diagrammkanten nach ihren Gewichten.
* Fügen Sie dem MST Kanten von der Kante mit dem kleinsten Gewicht bis zur Kante mit dem größten Gewicht hinzu.
* Fügen Sie nur Kanten hinzu, die keinen Zyklus bilden, Kanten, die nur nicht verbundene Komponenten verbinden.
* Wenn die Anzahl der Knoten in MST ist gleich als die Anzahl der Knoten in der Graph -1, dann enden wir den Prozess.
* Wir geben den MST aus.

1. Laufzeit des Algorithmus:

Es kann gezeigt werden, dass der Kruskal-Algorithmus in der Zeit O (E log E) oder äquivalent in der Zeit O (E log V) abläuft, wobei E die Anzahl der Kanten im Diagramm und V die Anzahl der Eckpunkte ist, alle mit einfachen Datenstrukturen

Wir können diese Grenze wie folgt erreichen: Zuerst sortieren wir die Kanten nach Gewicht unter Verwendung einer Vergleichssortierung in O (E log E) -Zeit.

Als Nächstes verwenden wir eine disjunkte Datenstruktur, um zu verfolgen, welche Knoten sich in welchen Set befinden. Wir müssen O (N) Operationen ausführen, da wir in jeder Iteration einen Knoten mit dem Spanning Tree (MST) verbinden, zwei Suchoperationen und möglicherweise eine Vereinigung für jede Kante.