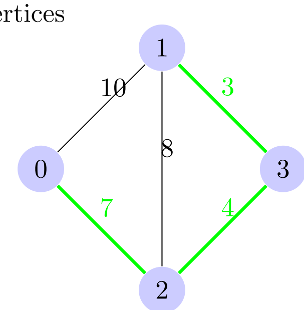
1 Minimum Spanning Tree

Input: Connected, undirected, weighted $G = (V, E)$
Output: A tree on edges that minimizes sum of weights while spanning all vertices



1.1 Cycle Property

Lemma 1.1. *Let C be a cycle in G. Let $e \in C$ be a heaviest edge on C. There, there is always an MST that does not contain e.*

Proof. Let T be an MST where $e \in T$.
Consider $T - \{ e \}$, where $e = (u, v)$
Let $A = \{w \in V | w$ is reachable from $u \in T - \{e\}\}$
There must be an edge e' on C that connected A to $V - A$.
Consider $T' \triangleq T - \{e\} \bigcup \{e'\}$
Claim 1: Weight of T' only smaller or equal to weight of T because e is the heaviest edge.
Claim 2: T' spans all vertices. □

2 Kruskal's Algorithm

Note 2.1. You must pass on connected components, or else you will make a cycle.

2.1 Complexity

$n - 1 \leq m \leq n^2$
$m = |E| \quad n = |V|$
$O(m \log n)$

Note 2.2. The actual complexity is $O(m \log m)$ but it is aways better to write $\log n$ if it is a connected graph.

Algorithm 1 Kruskal's Algorithm

procedure KRUSKAL'S ALGORITHM(G) $\triangleright$ $G(V, E)$ weighted graph w/non-negative weights
    $T \leftarrow \{\}$
    Sort all edges according to weight. $e_1 \le e_2 \le \cdots \le e_m$
    for all edges $e = (u, v)$ in order from to light to heavy **do**
        if u not connected to v **then**
            $T \leftarrow T \bigcup \{e\}$
        end if
    end for
    return T for G
end procedure

3 Union-Find Data Structure

Maintain a collection of disjoint sets. Ex: connected components of a graph. Each disjoint set is represented by one of its members.

- **Make-Set(X):** Creates a new set with element X. $O(1)$
- **Find-Set(X):** Returns the representative of the set X belongs to. $O(\log n)$. depth of X in its tree.
- **Union(X, Y):** Merges the set represented by X and Y. $O(1)$

Represent each set by a tree. Root = representative. Each element points to its parent in the tree.

Union by size: Whenever we union two sets, we make the root of the smaller tree, the child of the root of the larger tree.

4 MST-KRUSKAL

Implementation of Kruskal using Union-Find is described in Algorithm 2.

5 Clustering

Input: Point P_1 - P_n
Distance Function:$d(.\,,.)$
$\forall i\,, j$
$d(P_i, P_j) = 0 \leftrightarrow i = j$
$d(P_i, P_j) = d(P_j, P_i)$
$d(P_i, P_j) \ge 0$
Output: Partition the points to K sets where K is a natural number.

Algorithm 2 Kruskal's Algorithm

```
procedure MST-KRUSKAL(G)                ▷ G(V, E) weighted graph w/non-negative weights
    T ← {}
    for all v ∈ V do
        Make-Set(v)
    end for
    Sort all edges according to weight. e_1 ≤ e_2 ≤ ··· ≤ e_m
    for all edges e = (u, v) ∈ E in order from to light to heavy do
        if Find-Set(u) ≠Find-Set(v) then
            T ← T ⋃{e}
            Union(u,v)
        end if
    end for
    return T for G
end procedure
```

So the spacing between clusters i s max. Spacing = min distance between points in different clusters.

Ex: Netflix ”you may also like”

Algorithm: Run Kruskal on graph $\{p_1, p_2, \ldots, p_n\}$ all edges (p_i, p_j) each weighted with $d(p_i, p_j)$

Stop when you have reached K connected components.

Let D be the spacing Kruskal achieved. D = weight of the $(k-1)^{th}$ heaviest edge in MST.

$\Rightarrow$ edges inside component are of weight $\leq D$