Kruskal's Algorithm

Definition

Kruskal's algorithm is a greedy minimum spanning tree algorithm finds the subset of the edges of the graph which:

- form a tree including every vertex
- has the minimum sum of weights among all the trees taht can be formed from the graph

Kruskal's treats the graph as a ***forest*** (undirected acyclic graph; two vertices connected by at most one path). A tree connects to another *iff* it has the **least cost**.
💡**:** Scans all edges by **increasing weight**; if an edge is safe add it to *F*

Pseudocode

```
function kruskal(V,E):
        sort E by increasing weight
        forest = Ø
        for each vertext v ∈ V do:
                MakeSet(v)
                for each edge (u,v) ∈ E do:
                        if Find(u) != Find(v)
                                add (u,v) to forest
                                Union(u,v)
        return forest
```

Algorithm Description

Union-Find

Kruskal's algorithm maintains a partition of vertices of G into disjoint subsets (the components of our forest), using a data structure that supports the following operations:

- *MakeSet(v)* — Create a set containing only the vertex *v*
- *Find(v)* — Return an identifier unique to the set containing v
- *Union(u, v)* — Replace the sets containing u and v with their union. (This operation decreases the number of sets).

Steps

1. First thing we do in this algorithm is **sort the set of edges *E*** by weight in increasing order. Costs **O(E log V)**
    - *(This will be important in our second for loop)
2. We then **initialize an empty forest** that we'll be filling with our discovered nodes.
3. Iterate through the sorted edge list. Because we sorted our edge list in increasing order, **any edge we examine is safe**.
    - For example, if we found a non-safe edge between two components then there must be a lighter edge with exactly one endpoint in A.
      BUT, this is impossible because inductively every previously examined edge has both endpoints in the same component of the forest.
4. We check if two vertices belong to the same cluster **(this decides whether adding an edge creates a cycle)**.
5. If they don't create a cycle we create a union between them.

Time Complexity

The worst case time complexity of this algorithm is **O(E log V)**.