Kruskal's Algorithm and Union-Find

Throughout, when we say *graph* we mean undirected graph.

1. A *tree* is a graph which is connected and acyclic. Note that a tree which has n vertices must have exactly $n - 1$ edges.
2. A *directed tree* is a directed graph such that
   (a) One vertex, designated to be the *root*, has outdegree 0.
   (b) Each vertex, other than the root, has outdegree 1.
   (c) For each vertex v, there is a directed path from v to the root.
3. A *forest* is a graph which is the disjoint union of trees.
4. A *directed forest* is a directed graph which is the disjoint union of directed trees.

Kruskal's Algorithm

A *component* of a graph G is a maximal connected subgraph of G. A *spanning tree* of a graph G is a subgraph which is a tree and which contains all vertices of G. If G is *weighted*, *i.e.,* every edge of G has a *weight*, a spanning tree is *minimal* if the total weight of its edges is minimal among all spanning trees of G. A *spanning forest* of any graph G consists of a spanning tree of each component of G. If G is weighted, a spanning forest of G is *minimal* if the total weight of its edges is minimal.

Kruskal's algorithm finds a minimal spanning tree of a connected weighted graph, and can be extended to find a minimal spanning forest of any weighted graph G. During the algorithm, edges can be either selected or discarded. At any given step, the *selected subgraph* $\mathcal{S}$ is a forest consisting of all vertices of G together with all edges selected up to that step. Edges are processed in order of weight. Procssing an edge $e = \{x, y\}$ consists of selecting e if x and y belong to different components of $\mathcal{S}$, otherwise discarding e. When an edge is selected, the number of components of $\mathcal{S}$ decreases by 1. Computation ends when each component of $\mathcal{S}$ is a minimal spanning tree of one component of G, Kruskal's algorithm is *greedy* since at each step a minimal edge which does not create a cycle is selected.

In some applications, we only wish to find the set of components of G. meaning that all edges of G can be thought of having equal weight. In this case, the edges can be processed in any order.

Implementation of Kruskal's Algorithm using Union/Find

Given a weighted graph G, the data structure of our implementation consists of a forest $\mathcal{S}$ and a directed forest $\mathcal{D}$. $\mathcal{S}$ is a subgraph of G, a set of trees which include all vertices of G and the edges which have been selected so far. $\mathcal{D}$ is the union of directed trees, each of which corresponds to one of the trees of $\mathcal{S}$, and has the same set of vertices. The number of arcs of one of these directed trees equals the number of edges of the corresponding tree in $\mathcal{S}$, but an arc may not correspond to any edge of $\mathcal{S}$. When an edge is selected and added to $\mathcal{S}$, the number of arcs in $\mathcal{D}$ is increased by 1 After all steps, $\mathcal{S}$ is a minimal spanning forest of G and $\mathcal{D}$ is a spanning directed forest of G.

Parents and Leaders. Each vertex of $\mathcal{D}$, other than the root of one of its components, has a *parent* vertex, and $\mathcal{D}$ contains an arc from that vertex to its parent. From each vertex v, there is a unique directed path in $\mathcal{D}$ from v to a root vertex ℓ, which we call the *leader* of v, and we call v a *follower* of ℓ. The function *Find*(v) recursively computes the leader of a vertex v, while the function *Union*(k,ℓ), where k and ℓ are roots, assigns, say, ℓ to be the parent of k. This has the effect of combining the two sets of followers into one. Initially, $\mathcal{S}$ and $\mathcal{D}$ consist of all vertices of G and no edges or arcs. Edges are processed in order of increasing weight, as shown in the pseudocode below.

```
Process an edge {u, v}:
  k = Find(u);
  ℓ = Find(v);
  If(k = ℓ);
    Discard e;
  Else
    Union(k,ℓ);
    Select e;
```

Time Complexity. The worst case time to process one edge is $O(n^2)$, where n is the number of vertices of G. The worst case time of the algorithm is $O(mn^2)$, where m is the number of edges of G. However, the algorithm can be sped up by using the disjoint-set method introduced by Galler and Fischer in 1964. Tarjan showed that, using this method, Union/Find takes only $O(m\alpha(n))$ time where α is the very slow growing *inverse Ackermann* function.[1] The method improves search time by using *path compression* which resets the parent of a vertex to be its leader whenever *Find* is executed. Time is also improved by making sure that when *Union* is executed, the leader of the larger set becomes the leader of the combined set.

For clarity, we explain these operations using C++ code, after giving a structural type for vertices.

```
struct vertex
 {
  vertex*parent;
  int numfollow;
 };

vertex*find(vertex*v)
 {
  if(v->parent == v) return v;
  else
   {
    vertex*u = v->parent;
    vertex*ell = find(u);
    v->parent = ell;
    return ell;
   }
 }
```

[1] $\alpha(m) \leq 4$ for any number m less than $2^{2^{2^{65536}}} - 2$. Thus, for any application whose input is small enough to fit into the known universe (no kidding) the time is $O(m)$.

```
void union(vertex*u,vertex*v)
 {
  if(u->numfollow <= v->numfollow)
   {
    u->parent = v;
    v->numfollow += u->numfollow;
   }
  else union(v,u);
 }

void initialize(vertex*v)
 {
  v->parent = v;
  v->numfollow = 1;
 }
```