Data Structures in MST Algorithms

Kruskal's Algo.

maintains Components of (V, F_i)

Disjoint set, Union data structure

Makeset (a) // creates set {a}

Find (x) // returns id of the set which contains x

Union(x, y) // merges the sets containing x, y into one

Find(x) = Find(y) if f x and y are in the same set.

```
MST-Kruskal(G)        // G = (V, E, w)
1. for all v ∈ V do
       makeset(v)     // Initialization
   A = ∅
2. Sort elements of E in increasing order
   of their weights
3. for each edge (u,v) in E (picked in the
                             sorted order above)
       if (Find(u) ≠ Find(v))   // if (u,v) connects
                                // distinct components
           Union(u,v)
           A = A ∪ {(u,v)}
return A
```

Complexity Analysis

Step 1: $|V|$

Step 2: $|E| \log |E|$

Step 3: $|E| \times$ (Cost of Find + Union operation)

Any set has at most $|V|$ elements

time of Find (and of union) is $O(\log |V|)$

$O(|E| \times \log |V|)$

Total time: $O(|E| \log |V|)$

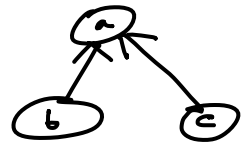
$|E| \leq |V|^2$ $\quad O(\log |E|) = O(\log |V|)$

An Implementation of Disjoint set, Union operations

Sets are represented as trees.

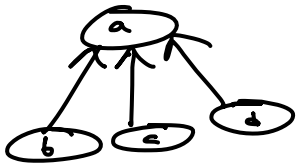
Each node of the tree is contains a member of this set. Each node has a parent pointer and a rank field.

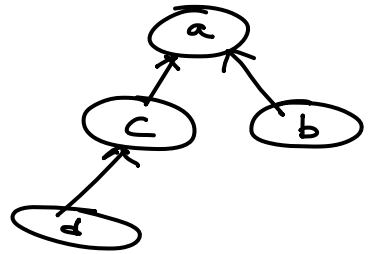
Rank of a node contains height of the tree rooted at this node.



{a, b, c}

{a, b, c, d}



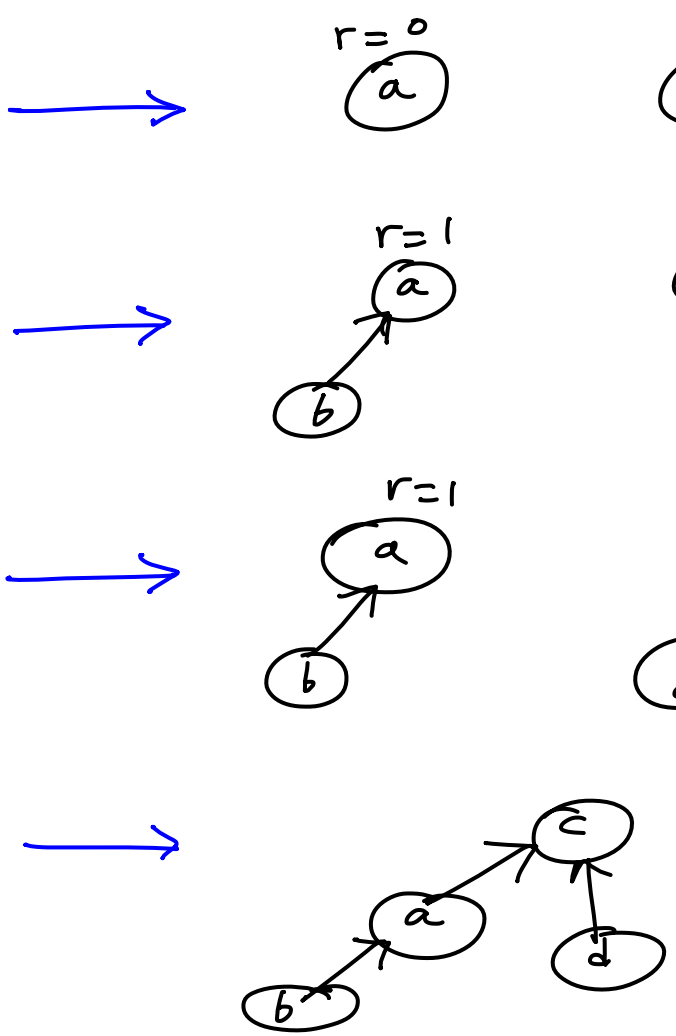



```
Makeset(x)
    n = node(x)
    n.p = nil
    n.rank = 0
```

```
Find(x)
    While x.p ≠ nil do
        x = x.p
    return x
```

```
Union(x, y)
  x1 = Find(x)
  y1 = Find(y)
  if (x1 == y1)
      return x1
  if (x1.rank < y1.rank)
      x1.p = y1
  elsif (y1.rank < x1.rank)
      y1.p = x1
  else: x1.p = y1
      y1.rank = y1.rank + 1
```

Example : makeset (a), makeset (b), Find (a), makeset (c), Union (a, b), makeset (d), Union (c, d), Union (a, c).



Complexity

makeset : $O(1)$

Find (x) : O (height of tree containing x)

Union (x, y) : $O(\max\{ht(x), ht(y)\})$

Lemma: If root node of a set has rank r then the set has at least 2^r elements.

Proof: Exercise (by induction on no. of operations on this data structure)

(Each operation preserves the property) □

Corollary

If the set has n elements, then find operation takes at most $O(\log_2 n)$ time.

This implies Complexity of MST – Kruskal $O(|E| \log |V|)$

Prim's Algorithm

Uses priority queue T_i $V - T_i$

Min-priority queue contains all $v \in V - T_i$
(vertices not in the current tree)

Priority queue is built on 'key'.

$$v.key = \begin{cases} \infty & \text{if there is no edge from } v \text{ to } T_i \\ \min\{w(u,v) \mid u \in T_i, (u,v) \in E\} & \text{otherwise} \end{cases}$$

$v.\pi$ parent of v.

$$v.\pi = \begin{cases} \text{parent of } v & \text{if } v \in T_i \\ u & \text{if } (u,u \text{ is the min weight edge between } T_i \text{ and } v \\ nil & \text{if there is no edge between } T_i \text{ and } v. \end{cases}$$

MST – Prim (G, r) // G = (V, E, w)

|V(

1. for all $v \in V$ do
   $v.\pi = nil$
   $v.key = \infty$

   // Initialization

   $r.key = 0$
   Create priority queue Q (using field key)
   containing all $v \in V$.

2. While $Q \neq$ empty do
   v = delete-min(Q) // Adding edge $(v.\pi, v)$ to our current tree
   for all (v, t) in E do
   if $(t \in Q)$ and $t.key > w(v,t)$
   $t.\pi = v$
   Decrease-key(Q, t, $w(v,t)$) // Decreases key field of node t to $w(v,t)$

output $\{(v.\pi, v) \mid v \in V, v.\pi \neq nil\}$

Initialization takes $O(|V|)$ step

Aggregate Analysis: Each $v \in V$ is deleted from Q exactly once.
(we never insert a vertex into Q after initialization)

Adjacency list of v is examined (for loop) when it is deleted.

Total iterations of for loop (over all while iterations)

$$= \sum_{v \in V} |\{(v,u) \mid (v,u) \in E\}|$$

$$= \sum_{v \in V} \deg(v) = 2|E|$$

In each iteration of the for loop time reqd $O(\log |V|)$ [Q contains at most $|V|$ elements]

overall time $O(|E| \log |V|)$.