A slightly better analysis

$e = (u, v)$ is added s.t

$comp(u) \neq comp(v)$

Additionaly store $size(u)$ & $size(v)$

if $size(u) > size(v)$

$size'(u) - size(u) + size(v) \geq 2\,size(u)$

{ Perform DFS on $cc(v)$ and

$\forall w \in cc(v)$ change $comp(w) = comp(u)$

else

Perform DFS on $cc(u)$ &

$\forall w \in cc(u)$, change $comp(w) = comp(v)$

Amortized analysis: Instead of upper bounding the cost of each operation with the worst-case bound, bound the cost of a sequence of operations

Observation: Each time the label of a vertex changes, it moves to a component with size at least twice the old size.

Running time: $O(\sum_{u \in V} \#\text{ of label changes of } u)$

$= O(V \log V) + O(E)$ → Checking $comp(u) = comp(v)$

Data structures for disjoint sets

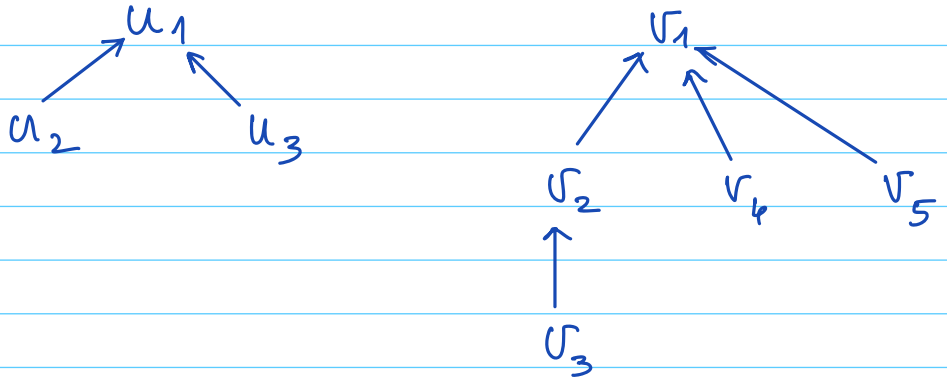
Maintain a collection of sets supporting the following operations

1. Makeset(u) - Create a set $\{u\}$
2. Find(u) - find the id of the set containing u
3. Union(u,v) - Compute the union of the sets containing u & v

Up-trees

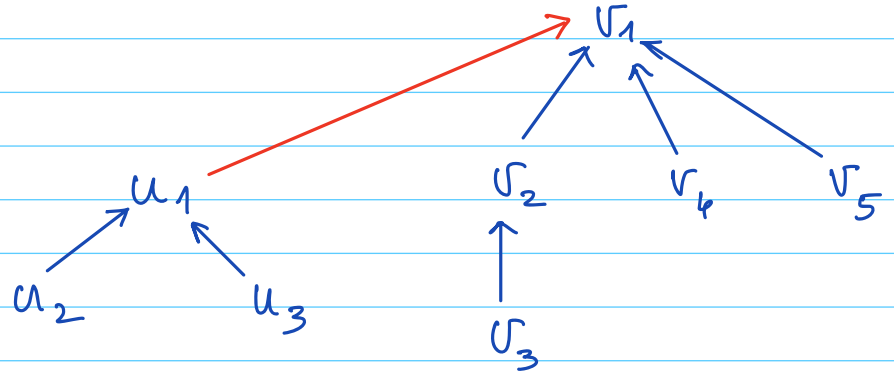
- Sets represented as rooted directed trees
- Node points to its parent
- Root is the id of the set

$F_1 = \{u_1, u_2, u_3\}$ $F_2 = \{v_1, v_2, v_3, v_4, v_5\}$



Find (u): Trace the path to the root

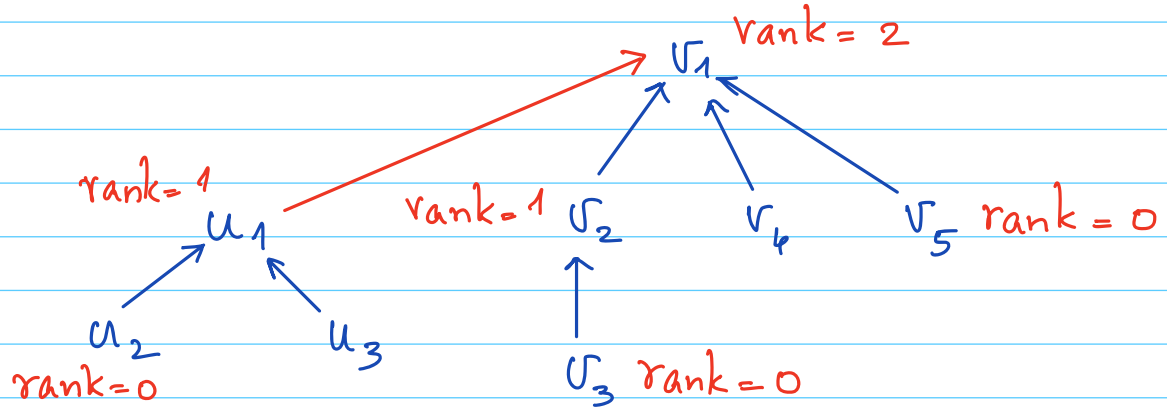
Union (u, v): Make one root the child of the other



* Complexity of Find - depends on the depth of the tree

Union-by-rank

rank (u) - height of the subtree rooted at u

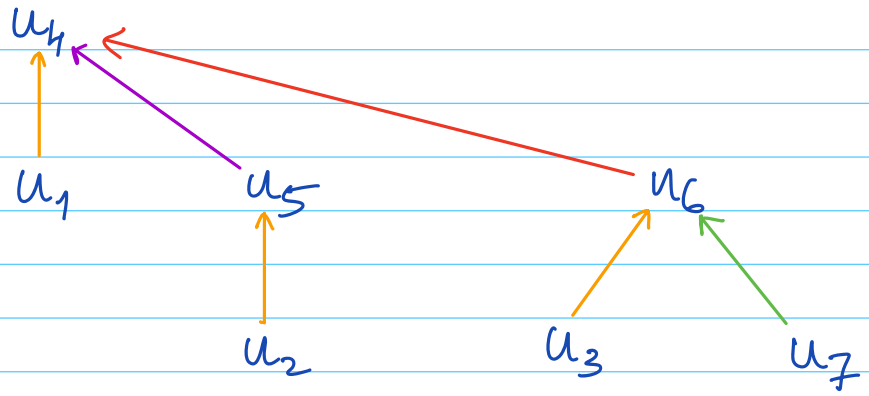


Union (u, v): If rank (Find (u)) > rank (Find (v)) then make u the parent of v

rank of all the nodes remain unchanged

else make v the parent of u.

↳ rank of the root increases by 1

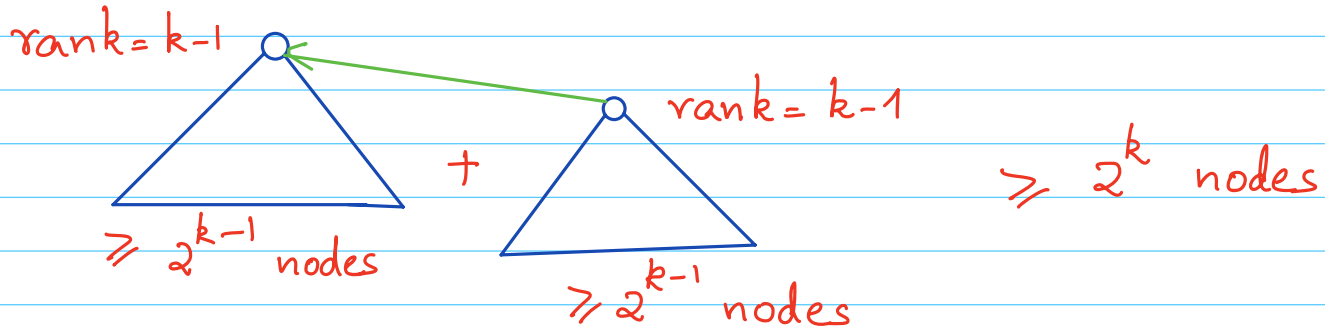


- Union (u_1, u_4), Union (u_2, u_5), Union (u_3, u_6)
- Union (u_3, u_7)
- Union (u_1, u_2)
- Union (u_7, u_5)

① For any node u $\underbrace{\text{rank}(u) < \text{rank}(\text{parent}(u))}_{\text{depth of the subtree}}$

② For any node u of rank k, there are at least 2^k nodes in its subtree

Proof by induction



(3) If there are n elements in total, then there are at most $n/2^k$ elements of rank k

- Subtrees are disjoint
- Each subtree contains $\geq 2^k$ nodes

Corollary: The highest possible rank = $\log n$

Running time: $O(E \log V)$ for Kruskal using up trees