Lecture 7

CS 124

Kruskal's alg. operations

Makeset (x) - creates set $\{x\}$
Find (x) - return "name" of the set containing x
Union (x, y) - union sets containing x and y

Kruskal's alg

Set X = {}
Set E = of edges
Sort E — assume ordered — $O(E \log E)$
for $u \in V$ makeset (u) — $O(V \cdot \text{makeset})$
for $(u,v) \in E$ in increasing order, do
  if find (u) ≠ find (v) — $O(E \cdot \text{find})$
    X = X ∪ {(u, v)}
    union (u, v) — $O(V \cdot \text{union})$

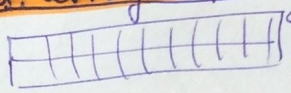
equivalence of [redacted] data structure and no cycle checking

edge connects disconnected components ⟺ no cycle

take cut through disconnected components, take min edge

cut property

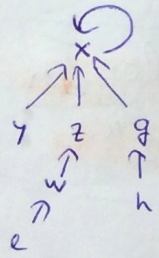
Baseline: array implementation



makeset O(1)
find O(1)
union O(V)

$O(E + V^2)$

Represent set by a tree



name of set

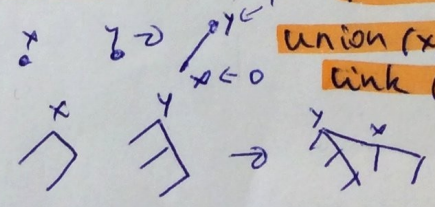
find O(Depth)

Makeset (x)
  p(x) := x
  Rank (x) := 0

Find (x)
  if x ≠ p(x)
    return (find (p(x)) [redacted])
  else return (x)

Link (x, y) ← put y on top of x; shallower to deeper
  if rank (x) > rank (y)
    x ⟺ y
  if rank (x) = rank (y)
    rank (y) := rank (y) + 1
  p(x) := y

union (x, y)
  Link (Find (x), Find (y))



$O([E+V] \text{ find})$
$O(E \log V)$

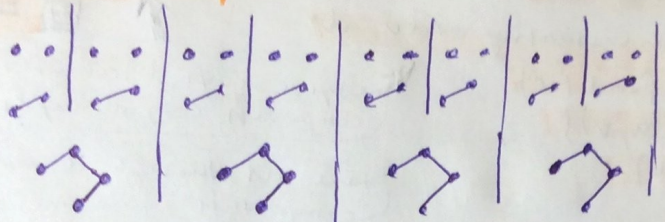
Think of rank as depth of the tree from the element
↳ no compression

not binary trees, but max depth is $\leq \log_2 n$, where n is # of nodes in a tree

rank increases only if two trees have the same rank, otherwise rank stays.

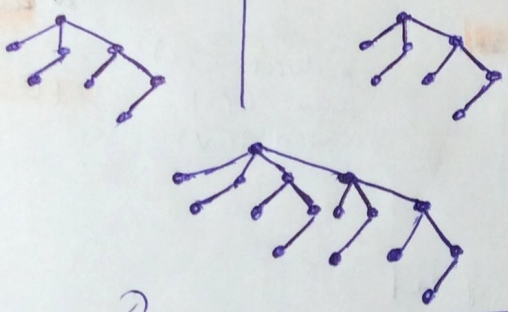
↳ worst case

get highest rank from n nodes → # increase rank each time two trees are joined



n	k ← rank
1 node	0 depth
2 nodes	1 depth
4 nodes	2 depth
8 nods	3 depth
16 nods	4 depth

induction

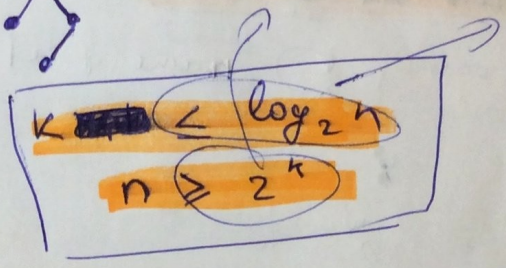


this case $k = \log_2 n$

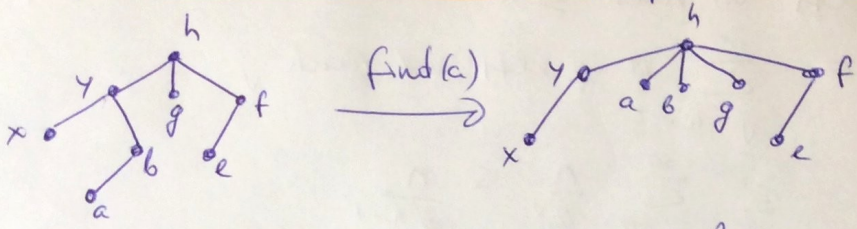
$n = 2^k$

other cases $k < \log_2 n$

$n > 2^k$



improve data structure with path compression (2



$O([E+V] \log^* V)$

path compression: $\log n \to 2 \log n$ constant factor cost

# of times to take $\log_2$ to get ≤ 1

Break even point after repeating find(a)

insurance: _not doing anything that is [redacted] unnecessary

idea [redacted] → add a constant factor to what is necessary anyway

↓

amortized analysis

maybe have to do a $\log n$ operation once but not again

→ amortized not just worst cost but consider cost over a sequence of operations

$\log^*$

$\log^* 2 = 1$

$\log^* 4 = 2 \quad (\log_2(\log_2 4)) = 1$

$\log^* 16 = 3$

$\log^* 65536 = 4$ (2^{16})

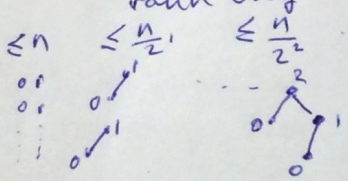
$\log^* 2^{65536} = 5$ ($2^{2^{16}}$)

Prove $O([E+V] \log^* V)$

lemmas: (holds with path compression or not)

1) if $v \neq p(v)$, then $\text{rank}(p(v)) > \text{rank}(v)$ (linked with higher up the tree)
2) when v's parent is updated, $\text{rank}(p(v))$ increases
3) # of elts w/ rank k $\leq n/2^k$ ✓
4) # of elts w/ rank $\geq k \leq n/2^{k-1}$

3) by induction rank only changes, if [redacted] root



4) # of elts w/ rank $\geq k$

$$= \sum_{j=k}^{\infty} \#\text{ of elts w/ rank } j$$

$$\leq \sum_{j=k}^{\infty} \frac{n}{2^j} = \frac{n}{2^{k-1}}$$

$$\sum_{j=0}^{\infty} \frac{n}{2^j} - \sum_{j=0}^{k-1} \frac{n}{2^j} = \frac{n}{1-\frac{1}{2}} - \frac{n\left(1-\frac{1}{2^k}\right)}{\frac{1}{2}}$$

$$= 2n - 2n\left(1-\frac{1}{2^k}\right) =$$

$$= 2n\left(1-1+\frac{1}{2^k}\right) = \frac{n}{2^{k-1}} \quad \checkmark$$

Group i = # of non-root elts with rank r satisfying $\log^* r = i$

(group 3 = ranks $(4, 16]$)

$\leq$ group $\log^* n$

① pointer to root (constant)

Type1: follow a pointer from u to v ← find operation $O(E \log^* n)$
u, v in different groups

Type2: same group ← group pays for this

take u in group $(k, 2^k]$

assign u 2^k tokens to pay for type 2 links

group $(k, 2^k]$, has $\leq n/2^k$ elts. (Lemma 4)

→ group $(k, 2^k] \leq n$ tokens

⇒ $\leq n \log^* n$ tokens needed

total: $\leq O((E+V)\log^* V)$