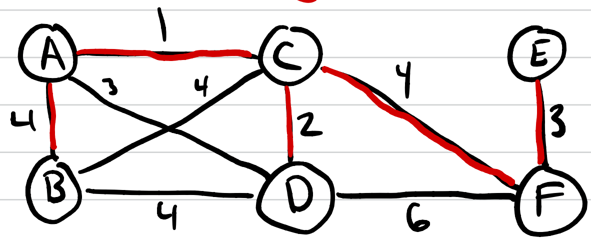
Recap: Minimum Spanning Tree (MST)



Graph $G=(V,E)$
with edge weights w_e

Goal: Find a tree $T \subseteq E$ connecting all the vertices w/ minimum total edge cost

Meta-algorithm

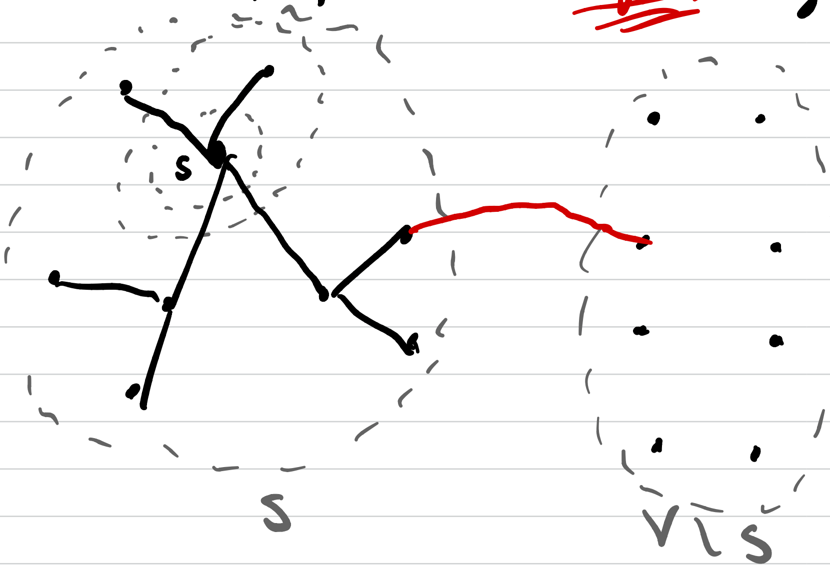
$X = \{\}$
repeat until $|X| = |V| - 1$
  pick a set $S \subset V$ s.t. X has no edge from S to $V \setminus S$
  let $e \in E$ be the minimum-weight edge from S to $V \setminus S$
  $X = X \cup \{e\}$

Example: Kruskal's algorithm, Prim's algorithm

Prim's algorithm

X always forms a (connected) tree on a set of vertices $S \subseteq V$

At each step, pick the **lightest** edge from S to $V \setminus S$



Similar to Dijkstra's. Implement w/ priority queue

prim(G, w)

$X = \{\}$
Q = priorityQueue()
for each $u \in V$, Q.insert(u, ∞)
                        from[u] = null
pick start vertex $s \in V$, Q.decreaseKey(s, 0)

while $|X| \leq |V| - 1$
    u = deleteMin(Q)
    if $u \neq s$, $X = X \cup \{($from[u]$, u)\}$

    for all $v \in V$ s.t. $(u, v) \in E$
        if $v \in Q$ still and $w(u,v) <$ v.key()
            Q.decKey(v, w(u,v))
            from[v] = u

Runtime

$n = |V|$, $m = |E|$

n inserts

n deleteMins

$O(m)$ decreaseKeys

$O((m+n)\log n)$

w/ binary heap

$O(m + n\log n)$

w/ Fib heap

MST runtimes

Kruskal: $O((m+n)\log n)$ time

Prim: $O(m + n\log n)$

Karger, Klein, Tarjan 1995: $O(m+n)$ expected time
(randomized)

Chazelle 2000: Deterministic $O(m \cdot \alpha(m,n))$
inverse Ackermann function
$\alpha(m,n) \leq 5$ for m,n reasonable

Pettie, Ramachandran 2002: Deterministic $O(\text{optimal})$
optimal $= ?$

Rest of lecture: disjoint set/union find data structure

implementation: disjoint-set forest

	runtime
• makeset(x) (creates a singleton set containing x)	$O(1)$ time
• find(x) (which set is x in?)	$O(\log n)$
• union(x,y) (merges the sets containing x,y)	$O(\log n)$

Kruskal's algorithm

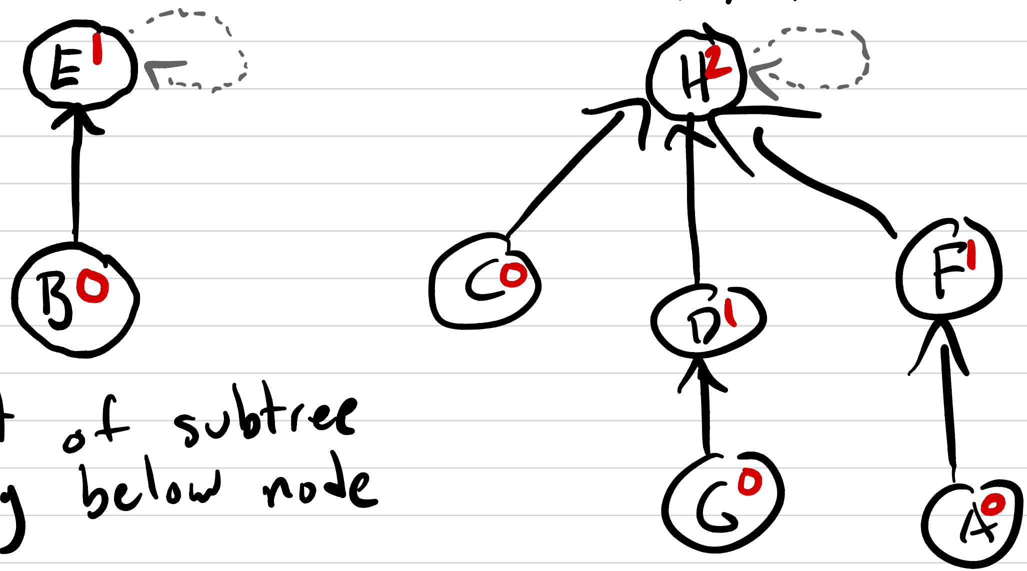
• n makesets	$O(n)$
• $2m$ finds	$O(m \log n)$
• $n-1$ unions	$O(n \log n)$
• sorts edges	$+ \; O(m \log m) = O(m \log n)$
	$O((m+n) \log n)$

(We will also see how to improve union find)

Union find using directed trees

{B, E} {A, C, D, F, G, H}

representative/ name of set



rank: height of subtree hanging below node

---

makeset (x)

$\pi(x) = x$ ($\pi(x)$ = x's parent)

$\text{rank}(x) = 0$

find (x)

while $x \neq \pi(x)$

  $x = \pi(x)$

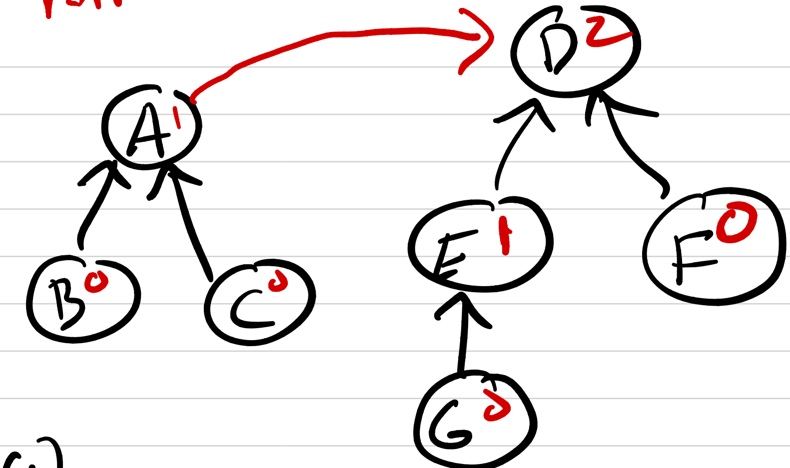
return x

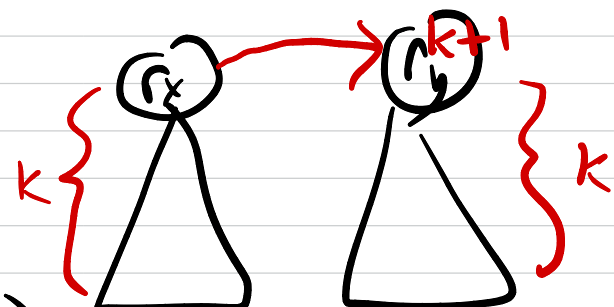
union(x, y) "union by rank"

$r_x = \text{find}(x)$
$r_y = \text{find}(y)$

if $r_x = r_y$, return

if $\text{rank}(r_x) > \text{rank}(r_y)$
  $\pi(r_y) = r_x$
else if $\text{rank}(r_x) < \text{rank}(r_y)$
  $\pi(r_x) = r_y$
else
  $\pi(r_x) = r_y$
  $\text{rank}(r_y) = \text{rank}(r_y) + 1$





runtime = O(runtime of find)

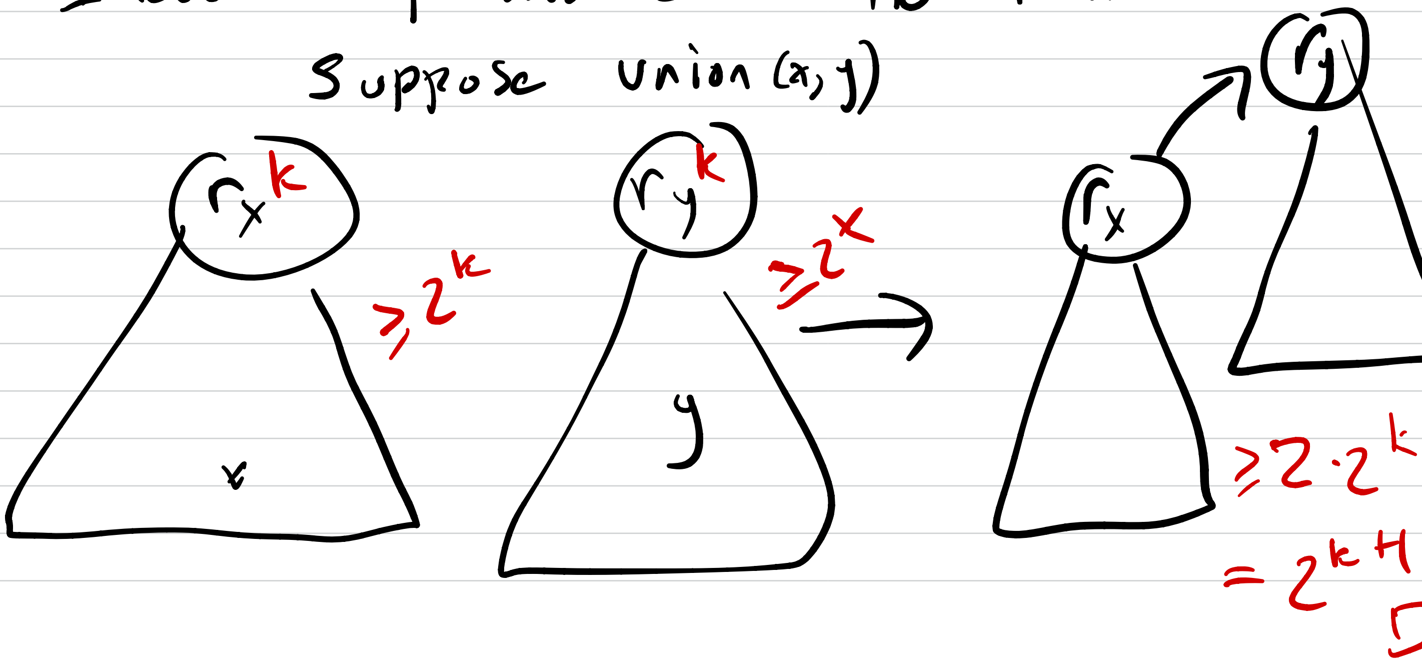
Claim 1: For all x, $\text{rank}(x) < \text{rank}(\pi(x))$ (unless $x = \pi(x)$)

Claim 2: Any root node of rank k has $\geq 2^k$ nodes in its tree

Pf: By induction on k. Base case $k = 0$ ✓

Inductive step. Assume true for rank k.

Suppose union(x, y)

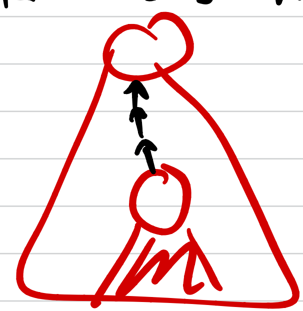


Claim 1: For all x, $\text{rank}(x) < \text{rank}(\pi(x))$ (unless $x = \pi(x)$)

Claim 2: Any root node of rank k has $\geq 2^k$ nodes in its tree
(Also holds for nonroot nodes)

Claim 3: If n elements overall, at most $\frac{n}{2^k}$ elements of rank k.

Pf: Rank k nodes have disjoint subtrees.



$\therefore$ (# of rank k nodes) $\cdot\, 2^k \leq n$

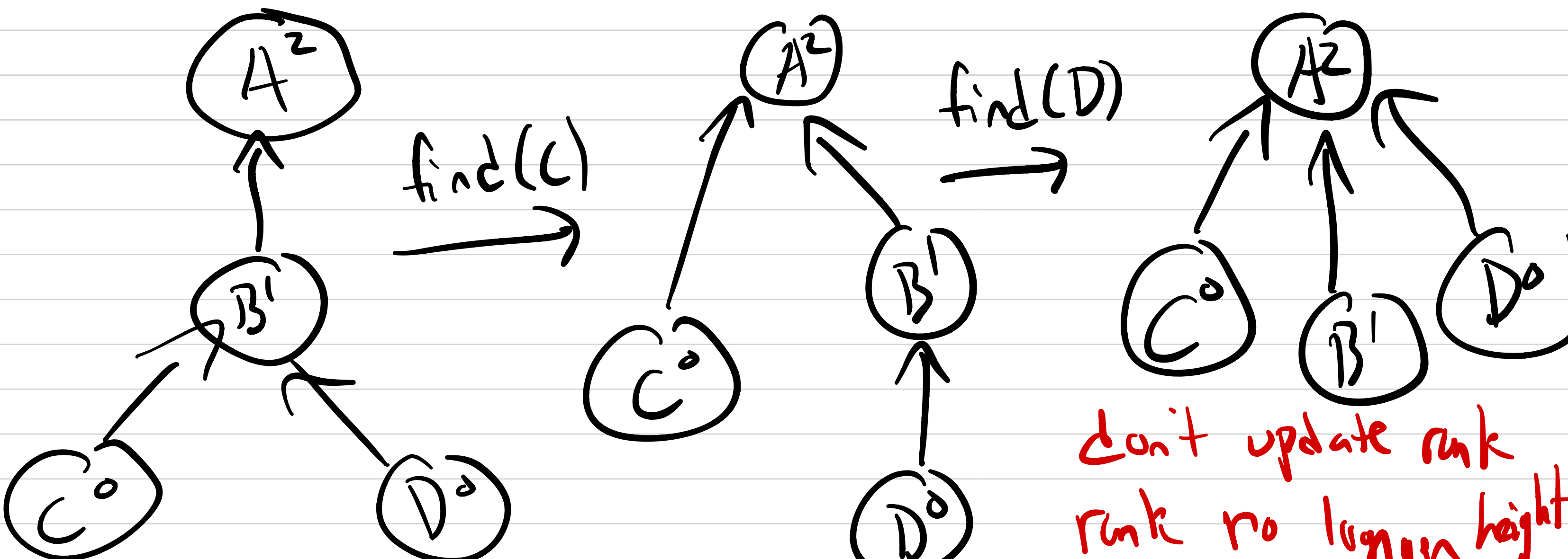
Corollary: All nodes have rank $\leq \log(n)$
Hence, find and union take time $O(\log(n))$

Improving union find via path compression

```
find(x)
    while x ≠ π(x)
        x = π(x)
    return x
```

→

```
find(x)
    if x = π(x), return π(x)
    else, return find(π(x))
        π(x) = find(π(x))
        return π(x)
```



don't update rank

rank no longer height

Runtime of union find with path compression

A sequence of n makesets and m union/finds take

$$O(n + m \cdot \alpha(m,n)) \text{ time}$$

inverse Ackermann

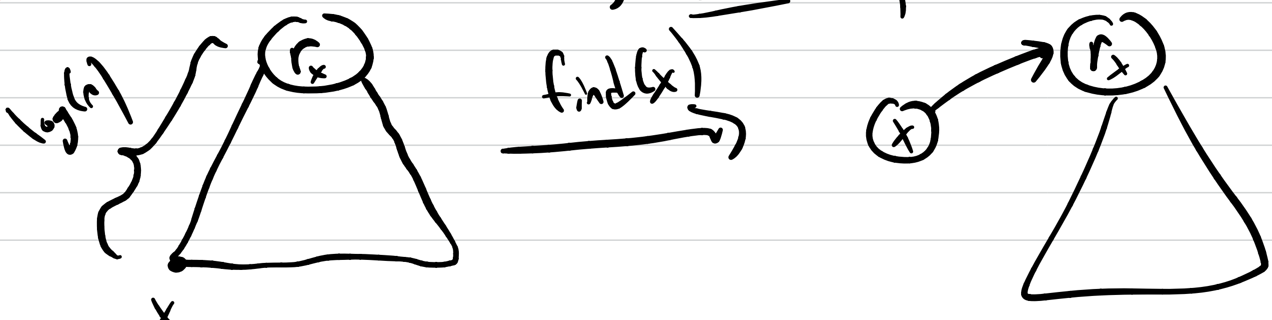
We will show: $O((n+m) \cdot \log^* n)$ time

$\therefore$ average operation takes time $\log^* n$

$\log^*(n) = \#\ \log_2()$'s needed to bring n to ≤ 1

$\log^*(2) = 1 \quad \log^*(2^2) = 2 \quad \log^*(2^{2^{2^{2^2}}}) = \cancel{5}$

Note: does not say each operation takes $\log^*(n)$ time



Consider intervals $[0], [1], (1,2], (2,4], (4,16], (16, 65536], \ldots$

"interval -1" "interval 0" $(2^0, 2], (2, 2^2], (2^2, 2^{2^2}], (2^{2^2}, 2^{2^{2^2}}], \ldots$

interval 1

For $i \geq 1$, interval i is $(2\uparrow(i-1), 2\uparrow i]$, $2\uparrow i = \left.2^{2^{\cdot^{\cdot^{2}}}}\right\}i$

Element x is in interval i at some time if:

- x is not a root
- x's $\text{rank}(x)$ is in interval i

Largest possible rank is $\log(n)$. It is in interval i where

$$\left.2^{2^{\cdot^{\cdot^{2}}}}\right\}i-1 < \log(n) \leq \left.2^{2^{\cdot^{\cdot^{2}}}}\right\}i$$

$$\text{So } i = \log^*(\log(n)) = \log^*(n) - 1.$$

Let $k = 2\uparrow(i-1)$. # of elements in $(k, 2^k]$ (interval i)

$$\leq \frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \cdots \leq \frac{n}{2^k}$$