CS314 - Union-Find

Note Title 10/3/2013

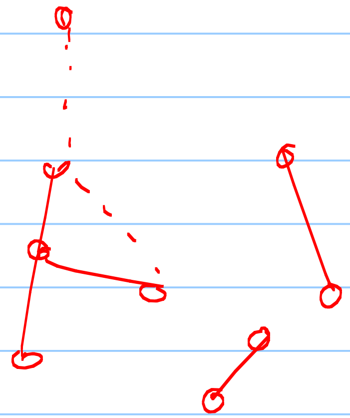
Announcements

- Grading next Tuesday
- Exam in 2 weeks
  (review the Wed. before exam)

Goal: Implement Kruskal's algorithm

Need: A data structure to maintain a collection of disjoint sets

Notation: Each set will have a unique "leader", to identify the set.



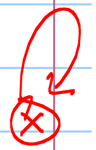
Operations:

- MakeSet(x): create a new set $\{x\}$ (x not in any other set)

- Find(x): Find (leader of) the set containing x

- Union(A, B): Replace 2 sets A, B with the set $A \cup B$.
  - Union(x, y) = Union(Find(x), Find(y))

Simple implementation — trees



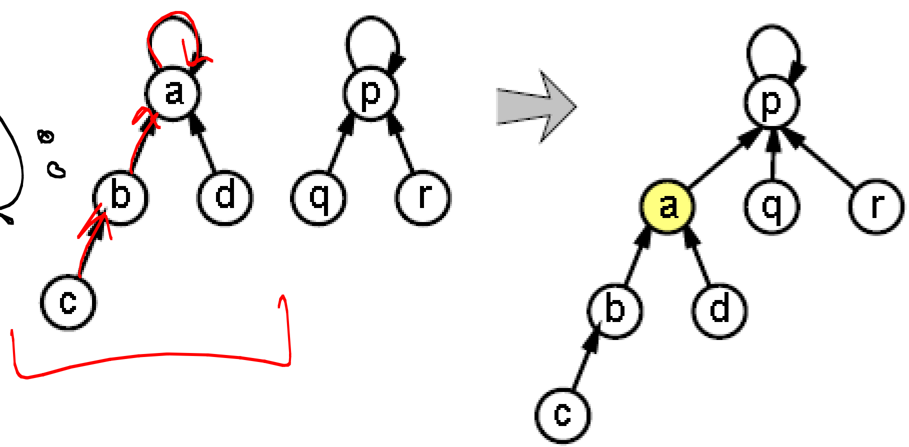
MakeSet(x):
  $parent(x) \leftarrow x$

Find(x):
  while $x \neq parent(x)$
    $x \leftarrow parent(x)$
  return x

Union(x, y):
  $\bar{x} \leftarrow \textsc{Find}(x)$
  $\bar{y} \leftarrow \textsc{Find}(y)$
  $parent(\bar{y}) \leftarrow \bar{x}$

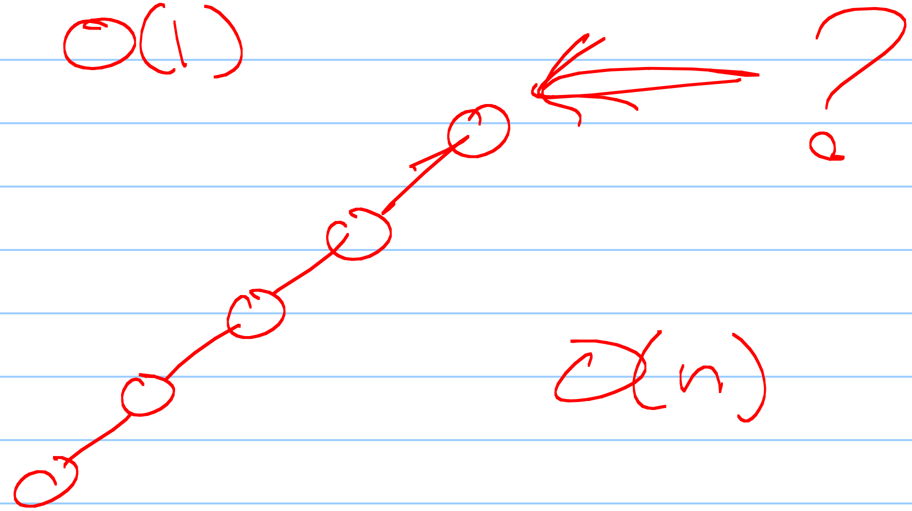
Example:

Union(c, r):



Runtimes:

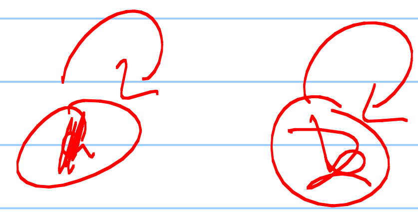
MakeSet: $O(1)$



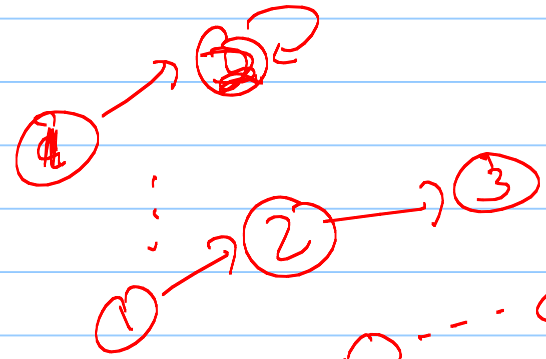
Find: $O(n)$

Union: $2 \cdot \text{Find} + O(1) \Rightarrow O(n)$

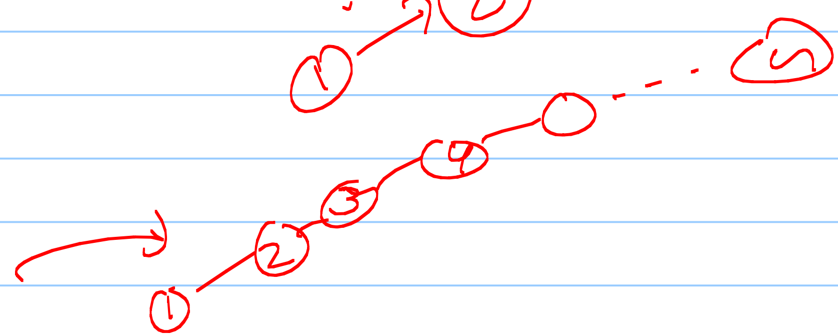
Makeset(1)
Makeset(2)
UNION(1,2)



makeset(3)
union(2,3)



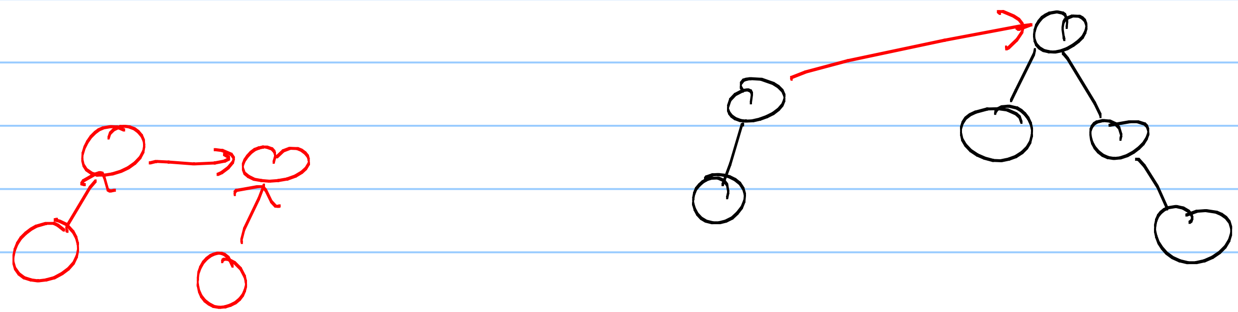
for i ← 1 to n
    Makeset(i)
    union(i-1, i)



Simple improvement:

Store depth of tree & make shallower one the child.

This results in better depth.



Claim: depth (& hence Find's runtime) is $\Theta(\log n)$.

Lemma: For any leader x, the size of x's set is at least $2^{\text{depth}(x)}$.

pf: induction! on d = depth:

Base Case d = 0.

size $1 = 2^0$ ✓

$\geq 2^{\text{depth}}$

cont; Ind Step:

For any $d > 0$, consider the Union which makes x's depth $= d$ for the first time.

Say Union(x, y)

$\Rightarrow$ y's depth was same as x's.

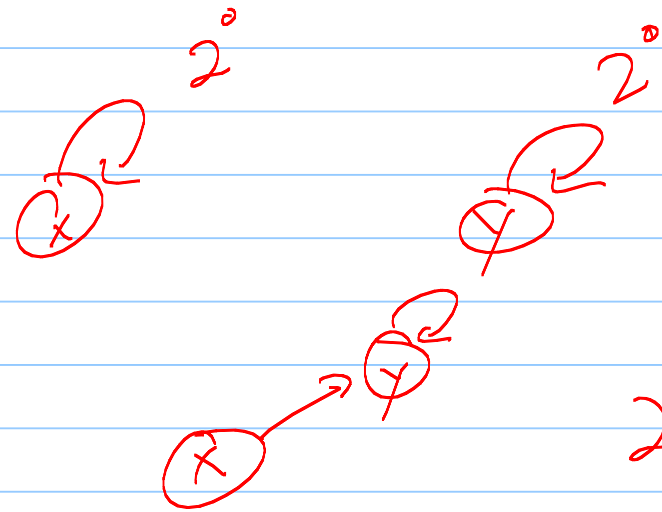
$\Rightarrow$ x & y both had depth $d-1$.

$\Rightarrow$ before union, size x's set $\geq 2^{d-1}$ & size y's set $\geq 2^{d-1}$ by I.H.

Now, x's set $\geq 2^{d-1} + 2^{d-1} = 2^d$ $\blacksquare$

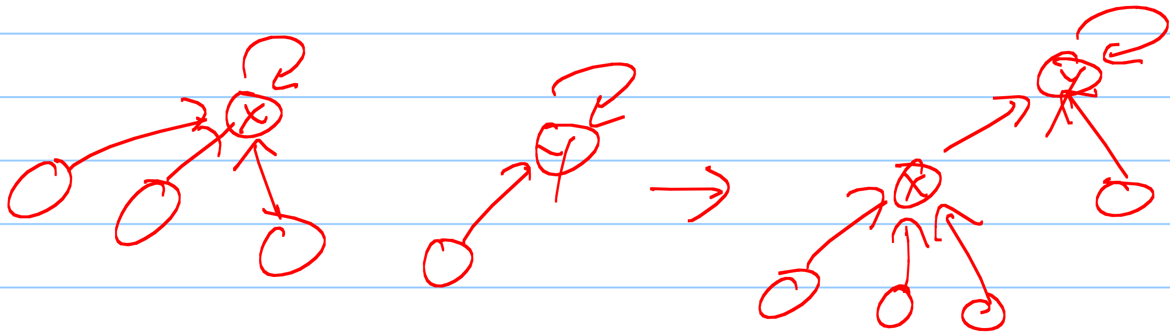
$size \geq 2^{depth}$

$\lg size \geq depth$



$\log_a n = O(\log_b n)$

$2^0 + 2^0 = 2^1$



$\geq 2^1 + 2^1 = 2^2$

Finally, since only n elements total, max depth of any set is $O(\log n)$

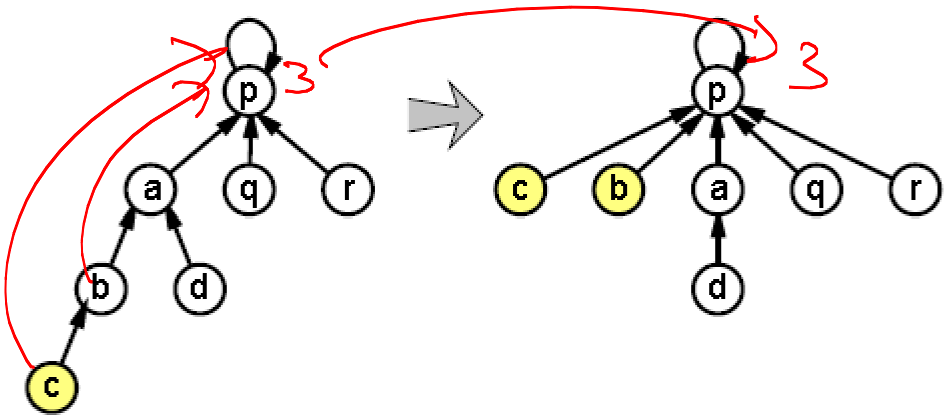
$\Rightarrow$ Find (& Union) are $O(\log n)$.

New Union:

```
UNION(x, y)
    x̄ ← FIND(x)
    ȳ ← FIND(y)
    if depth(x̄) > depth(ȳ)
        parent(ȳ) ← x̄
    else
        parent(x̄) ← ȳ
        if depth(x̄) = depth(ȳ)
            depth(ȳ) ← depth(ȳ) + 1
```

Observation:

In any Find, once we have the leader, we can speed up future operations with one easy modification:



Code: use recursion!

```
FIND(x)
    if x ≠ parent(x)
        parent(x) ← FIND(parent(x))
    return parent(x)
```

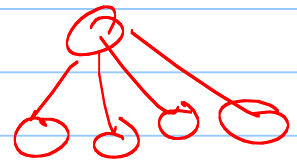
(This is called path compression.)

Problem:

This messes with our previous "depth" field.

How to fix?

How to compute depth?



Continue to maintain depth

However, even if "depth" is wrong, still ensures smaller is child of larger

$\Rightarrow O(\log n)$.

So keep it!

Union by "rank":

```
MAKESET(x):
    parent(x) ← x
    rank(x) ← 0
```

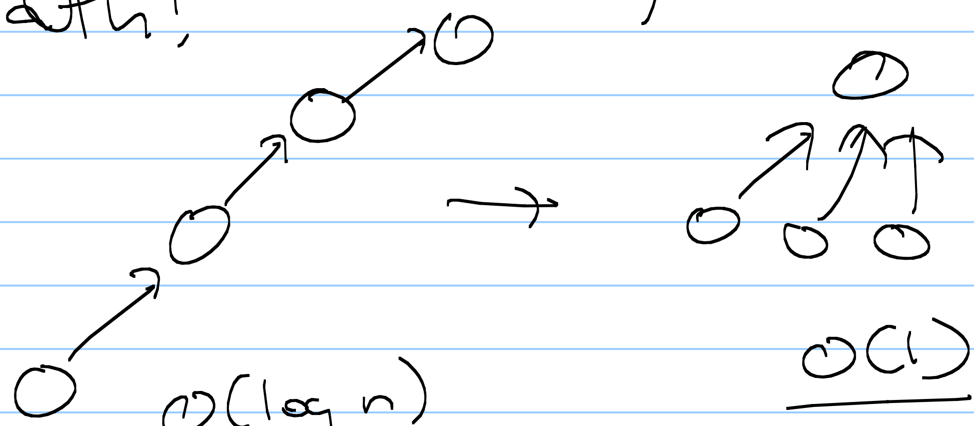
```
UNION(x, y)
    x̄ ← FIND(x)
    ȳ ← FIND(y)
    if rank(x̄) > rank(ȳ)
        parent(ȳ) ← x̄
    else
        parent(x̄) ← ȳ
        if rank(x̄) = rank(ȳ)
            rank(ȳ) ← rank(ȳ) + 1
```

But...

Seems like $O(\log n)$ is not tight.

Path compression should help us:

Any "long" Find destroys a deep path!



$O(\log n)$ $\rightarrow$ $O(1)$

Thm: The amortized cost of a find operation is $O(\log^* n)$.

$\Downarrow$ $O(\alpha(n))$

Iterated log: $\log^* n$

$$\lg^* n = \begin{cases} 1 & \text{if } n \leq 2 \\ 1 + \lg^*(\lg n) & \text{otherwise} \end{cases}$$

= # of times you take a logarithm before reaching ≤ 1

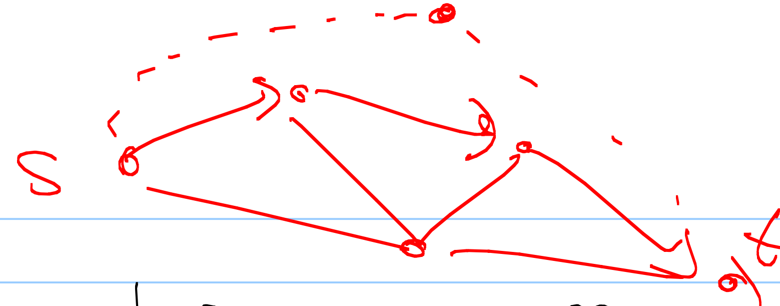
Note: This doesn't actually help us implementing Kruskal's algorithm.

Why?

Sorting edges takes $O(m \log n)$

maintaining the forest is $O(n \log^* n)$

Shortest path~~s~~ Trees

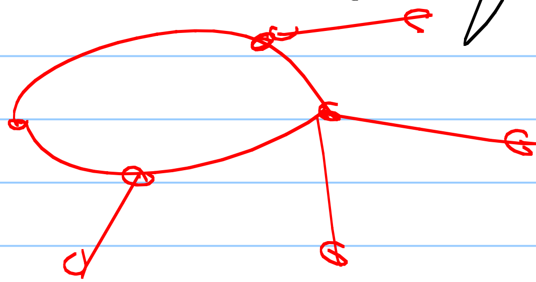


Consider weighted graph, plus a source vertex s.

Goal: Find shortest path from s to t
(or s to every other vertex).

Note: this is a tree if shortest path lengths are unique.
Why?

this would result in 2 shortest paths.



Shortest path trees versus MST.

Different!