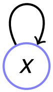
Today

- Union-Find Datastructure to implement Kruskal
- Path Compression

Disjoint Set Data Structure

Maintain pointers: $\pi(x)$ for each x.

makeset(x) $\pi(x) = x$.



find(x)

  if $\pi(x) == x$

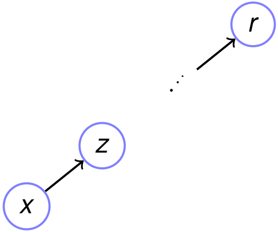
    return x

  else

    find($\pi(x)$)

union(x,y)

  $\pi(\text{find}(x)) = \text{find}(y)$



How long does find take?

(A) $O(n)$

(B) $O(1)$

(C) Depends.

Want depth to be small!

Disjoint Set Data Structure

Maintain pointers: $\pi(x)$ for each x.

makeset(x) $\pi(x) = x$.

find(x)

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

Make a bit less deep: union-by-rank.

union(x,y)

Use roots of x and y.

Which points to which?

“smaller” to “larger” in terms of the height (or what we will call rank)

Union by rank.

makeset(x) $\pi(x) = x$.
rank(x) $= 0$.

union(x,y)

r_x = find(x)
r_y = find(y)
if rank(r_x) $<$ rank(r_y):
  $\pi(r_x) = r_y$
else:
  $\pi(r_y) = r_x$
  if rank(r_x) == rank(r_y):
    rank(r_x) $+= 1$

Property of rank

Lemma: Dad's got a higher rank:

$$\text{rank}(x) < \text{rank}(\pi(x))$$

if $x \neq \pi(x)$.

Code enforces it.

union(x,y):

```
  ⋮
  if rank(r_x) < rank(r_y):
      π(r_x) = r_y
  else:
      π(r_y) = r_x
      if rank(r_x) == rank(r_y):
          rank(r_x) += 1
```

Test your understanding: Can the rank of a node that is not a root change?

Big rank corresponds to the bigger tree!

union(x,y):

```
  ⋮
  if rank(r_x) < rank(r_y):
      π(r_x) = r_y
  else:
      π(r_y) = r_x
      if rank(r_x) == rank(r_y):
          rank(r_x) += 1
```

Lemma: Any rank k root node has $\geq 2^k$ nodes in its tree.
Induction:
Base Case ?

(A) $2^0 \geq 1$

(B) $2^1 \geq 1$

A. Initially $rank(x) = 0$, 1 node in tree.

Induction step:
When rank(x) goes up to k, but it goes up by at most 1.
rank(x) was previously $k-1$ so it already has $\geq 2^{k-1}$ nodes. by ind. hyp.
gains nodes from another rank $k-1$ node y with $\geq 2^{k-1}$ other nodes
$\implies \geq 2^{k-1} + 2^{k-1} = 2^k$ nodes. □

Check your understanding?

Exactly 2^k nodes in tree of rank k? Yes or No?

No.

$\vdots$

if rank(r_x) $<$ rank(r_y):
$\quad\quad \pi(r_x) = r_y$

$\vdots$

Gains nodes without gaining rank!

Back to complexity for Kruskal.

Kruskal: Sort edges, $O(n)$ union, $O(m)$ finds.

Find(x) is

(A) $O(\log n)$ time.

(B) $O(1)$ time

(C) $O(n)$ time.

A. (and (C)).

Rank k node has $\geq 2^k$ nodes.

Only n nodes.

Every rank at most $\log n$, (otherwise, $> 2^{\log n} = n$ nodes.)

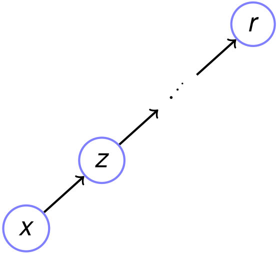
Since parent has higher rank, find time is at most $O(\log n)$.

Total find time is $O(m \log n)$. Yay!

Can we do better?

Path Compression

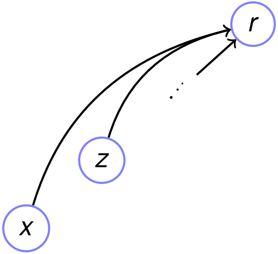
find(x)
  if $\pi(x) == x$
    return x
  else
    find($\pi(x)$)



What happens if we find(x) again? We go up the tree again?

Can we avoid this work the next time?

find(x)
  if $\pi(x) == x$
    return x
  else
    $\pi(x) = \text{find}(\pi(x))$
    return $\pi(x)$



Path Compression Analysis

Union is same. Only affects root nodes.

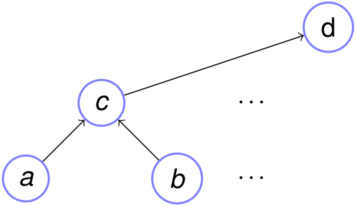
Rank properties still hold:
rank of parent is higher
and $\geq 2^k$ nodes were below a rank k node when it was root

Every find is asymptotically faster?

(A) Yes

(B) No

No. Can make a find take $\Theta(\log n)$ time.



union(a,c) union(b,c)
$\cdots$ union(c,d)
union roots to build complete binary tree
find(a)
$\Theta(\log n)$ time for this find.

Amortized Analysis.

Show that m **finds** take $O(m \log^* n)$ time in total.

$O(\log^* n)$ time on average!

Amortize cost = average over many operations.

What is log star?

$\log^* n$ is number of times one take $\log$ to get to 1.

$\log^*(16)$?

(A) 4

(B) 2

(C) 3

C. $\log 16 = 4$, $\log 4 = 2$, $\log 2 = 1$. 3 times.

Also $2^{2^2} = 16$. height of powers of two!

$\log 1,000,000$ versus $\log^* 1,000,000$?

20 versus 5.

$\log 1,000,000^{1,000,000}$ versus $\log^* 1,000,000^{1,000,000}$?

$20,000,000$ versus 6.

Grows very slowly.

Amortized Analysis.

Show that m **finds** take $O(m \log^* n)$ time in total.

$O(\log^* n)$ time on average!

Amortize cost = average over many operations.

How to do amortized analysis?

Hand out some money
..... use it to pay for each pointer change.

Only hand out $O(m \log^* n)$ dollars.

Handing out dollars.

Will hand out money to internal nodes
.....since they change pointers in find.

Notice: When a node stops being a root
rank will no longer change!

Divide non-zero ranks into levels.
$\{1\},\{2,3,4\},\{5,\ldots,16\}\cdots\{k+1,\ldots 2^k\}\cdots$

How many groups of ranks?

(A) $\Theta(\log n)$

(B) $\Theta(\log^* n)$

B. Each group grows by powering two!

How many internal nodes ever get rank r?

No node contained in more one rank r internal node.
n nodes in total.
Each rank r node contains 2^r nodes.
$\implies < \frac{n}{2^r}$ rank r nodes

Handing out money!

Will hand out money to internal nodes
.....since they change pointers in find.

Notice: When a node stops being a root
rank will no longer change!

If in set of ranks $\{k+1,\ldots,2^k\}$ give node 2^k dollars.

$O(n/2^r)$ internal nodes of rank r.

Total Doled out:
In a group: $2^k(n/2^{k+1} + n/2^{k+2}\cdots) = O(n)$.
$O(\log^* n)$ groups. Total money: $O(n\log^* n)$.

Bounding find cost.

Bound cost of find operation.

$O(1)$ plus
  cost of changing pointers to point to higher ranked nodes

$O(\log^* n)$ pointers that point to a node to a higher group.

  Total cost: $O(m \log^* n)$.

Node pays for changing a pointer within group.
Recall group: $\{k+1, \ldots, 2^k\}$

  Enough money?
    fewer than 2^k ranks in group
    each node in group has 2^k dollars.   Enough money!

Total money: $O(n \log^* n)$.

Total cost of finds: $O((m+n) \log^* n)$!

Instant Replay

Intuition:

Some operations may be expensive.
...but modify data structure so they won't be in future.

Place credits in data structure to pay for some modifications.

Still..

tough business.