Advanced Data Structures

Union-Find
Analysis
(End!)

Notes

- First HW - posted next week
- Sub next Friday 1/31,
  no class Monday 2/3,
  sub Wed 2/5

Formally: 3 operations:

makeSet(x): take an item & create a one element set for it

find(x): return "canonical" element of set containing x

union(x, y): Assuming that $x \neq y$, form a new set that is the union of the 2 sets holding x & y, destroying the 2 old sets. (Also selects & returns a canonical element for new set)

How to implement?

- certainly use existing DS.

Use a table:

For each entry, record its set label!

Runtime?	Why?
makeset : $O(1)$	create 1 new entry
find : $O(1)$	lookup one entry
union : $O(n)$	linear loop

So tradeoff w/ this approach:

Bad if many unions.

Better: Use trees!

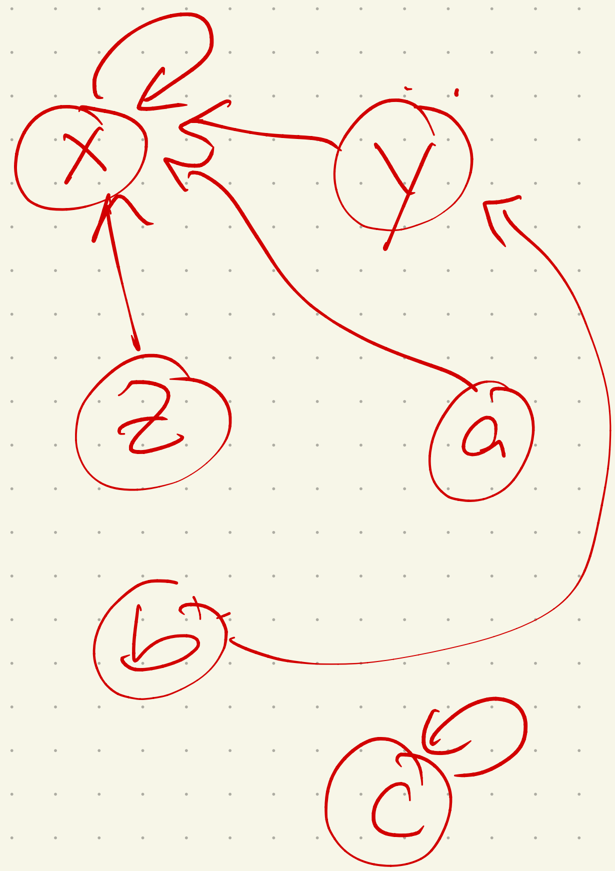
(Galler + Fisher, 1964)

Each set will be a rooted tree, where elements are in the tree & the root is the canonical element.

So each element has a pointer to its parent (& root points to itself)

Ex:

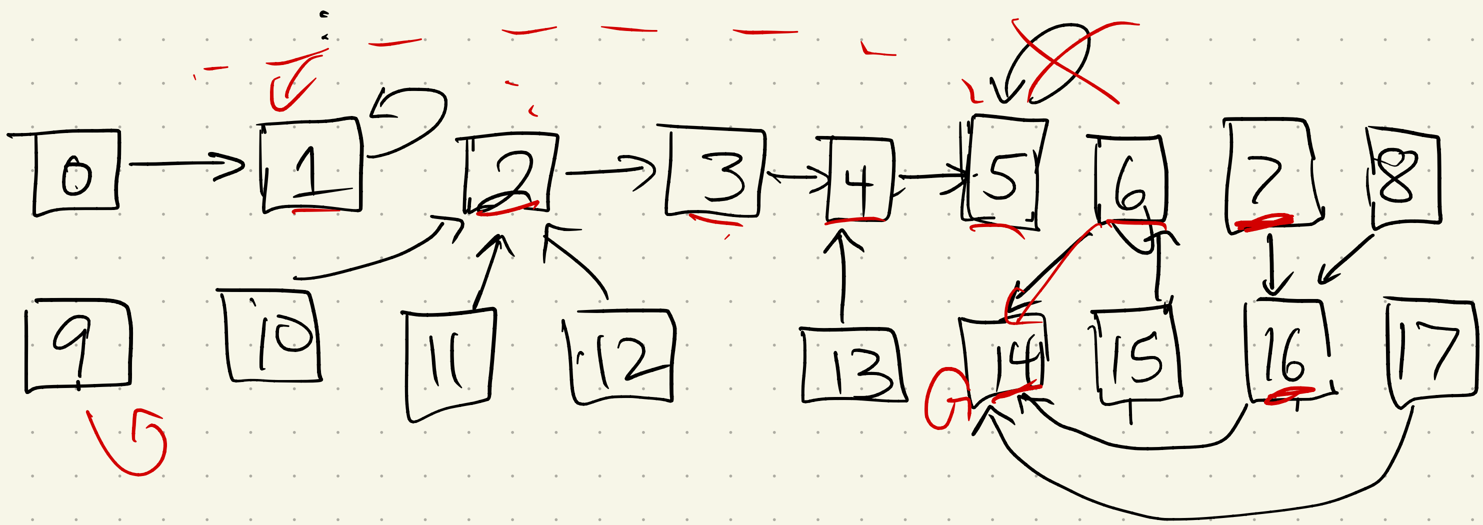
makeset(x) ✓
makeset(y) ✓
makeset(z) ✓
union(x, z) ✓
makeset(a) ✓
makeset(b) ✓
union(a, x)
union(b, y)
makeset(c)
union(z, b)



Implementation:

Don't actually need ptrs! Keep a "parent" array:

index	p[]	v[] value
0	1	a
1	1	b
2	3	c
3	4	d
4	5	e
5	~~5~~ ← new root	f
6	14	g
7	16	h
8	16	i
9	9	j
10	2	k
11	2	l
⋮		



Implementation:

$O(n)$

Find (x):

(while not root)

while $(p[x] \mathrel{!}= x)$

$\quad x = p[x]$

~~return~~ x

Union (x, y):

$\bar{x} = \text{find}(x)$

$\bar{y} = \text{find}(y)$

if $(\bar{x} \mathrel{!}= \bar{y})$

$\quad p[\bar{x}] = y$

$O(1)$

Still some flexibility:

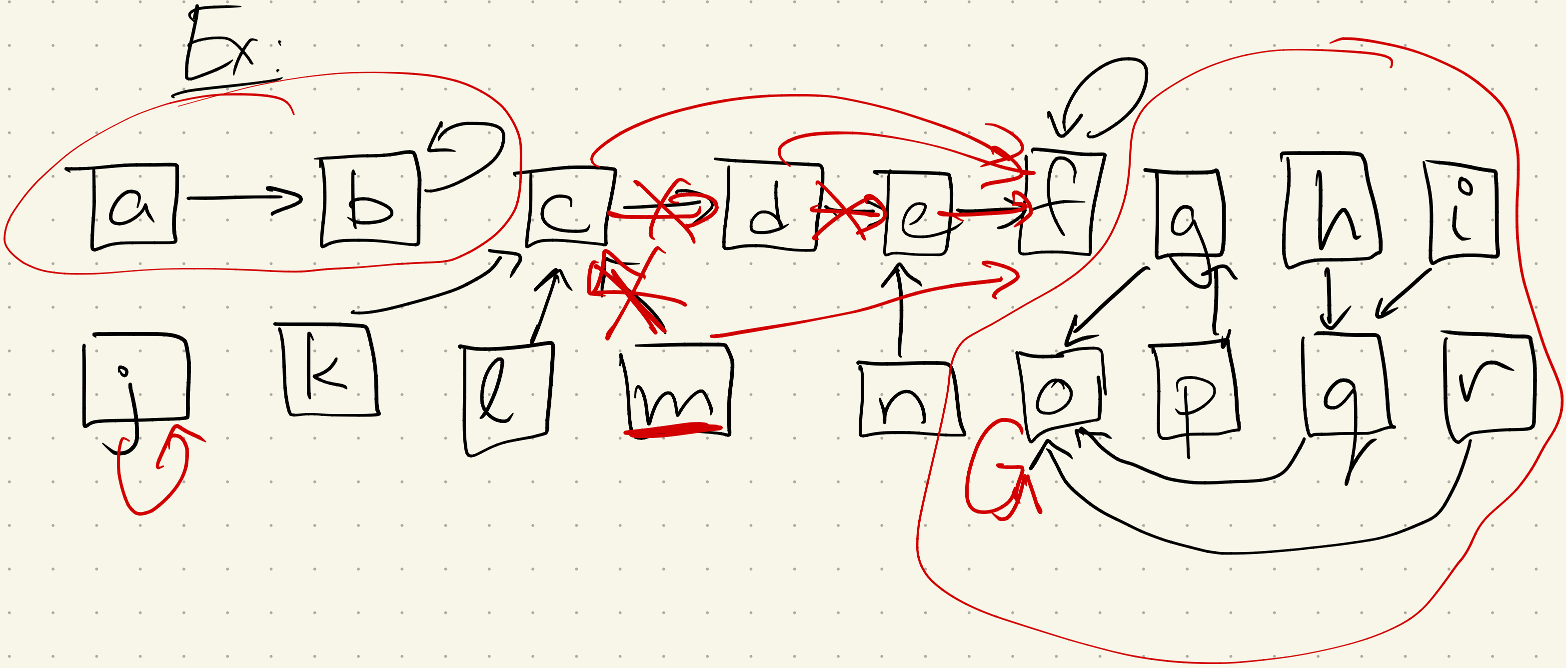
① Need to decide which root becomes the root of new set: ie: union(a,h)

union by rank

② Can also use "path compression": try to point as many things to the root as possible, so later queries get faster.

ie: find(m)

Ex:



1 Union by rank:

(introduced several places)

Each time you union, make smaller tree tree's root the child of larger one.

$\boxed{x \mid 0}$

How?

- Each node will have a "rank" field, initialized to 0.
- In a union:

  If one's rank is smaller:

  smaller "points" to larger

  If both are equal:

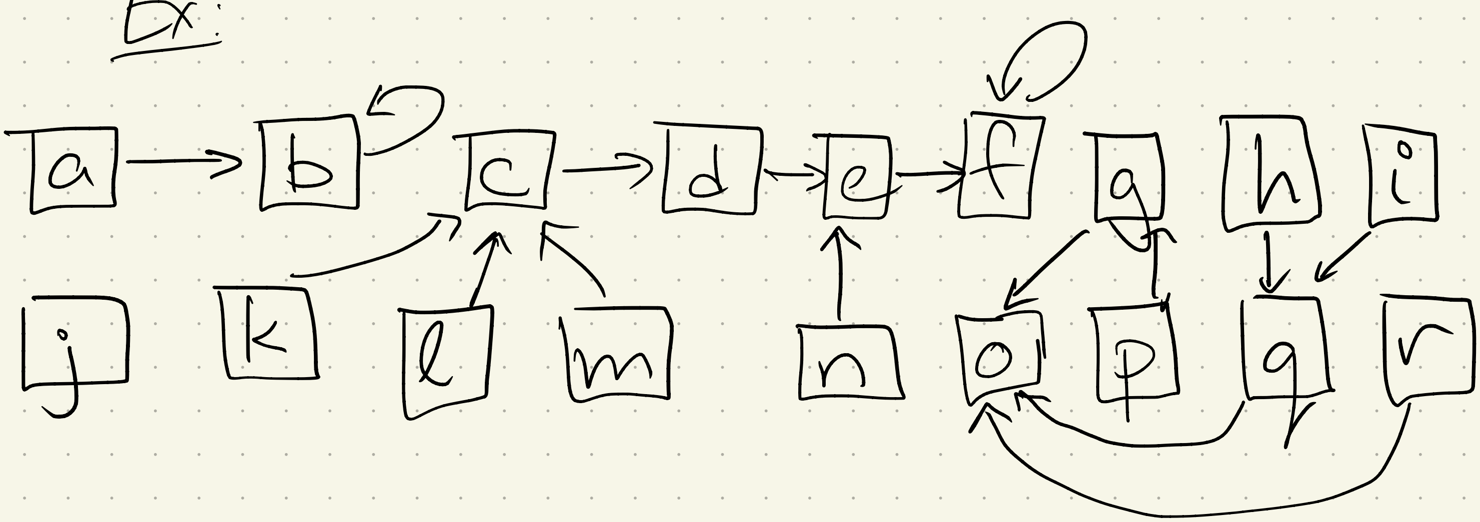
  point one to other, + increment new root's rank

(2) Path compression:

During each find(x) make every node on the path from x to the root point to the root:

So: find(c):

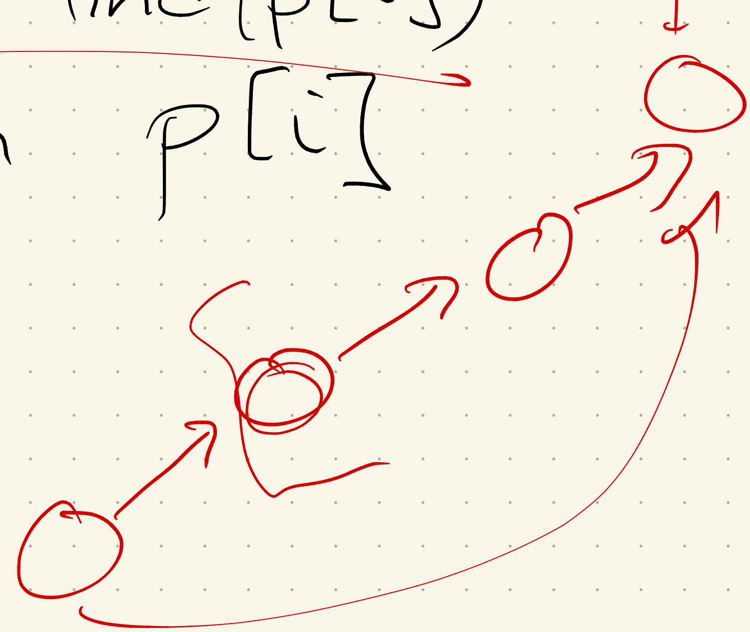
Ex:



Implementation:

find(i) {
  if p[i] == -1
    return i
  else
    ~~return find(p[i])~~
}

p[i] = find(p[i])
return p[i]



No change to rank!

Formalizing the improvement

Amortized Analysis:

Worst case here:

Still $O(\log n)$!

Why?

Might get tree of height $\log n$

Amortized Analysis:

However, if we do <u>many</u> find (or unions), things get faster.

<u>Ex:</u> find(x) $\leftarrow O(\log n)$

find(x) $\leftarrow O(1)$

So: looking for average runtime of one operation, if doing many of them.

UF: size n

m finds

(Assume $m >> n$)

Thm: Any m find or union operations run in time $O((n+m)\log^* n)$

??

Amortized cost of each:

$O(\log^* n)$

Actually $O(n\alpha^{-1}(m))$

not here!

tiny

(Next time)

$\log_2^* n$:= the number of times you apply the $\log_2$ until the result is ≤ 1.

$$= \begin{cases} 1 & \text{if } n \leq 2 \\ 1 + \log_2^*(\log_2 n) & \text{otherwise} \end{cases}$$

n	$\log^* n$
$(0, 1]$	0
$(1, 2]$	1
$(2, 2^2]$	2
$(4, 16] = (2^2, 2^{2^2}]$	3
$(16, 2^{16}] = (2^{2^2}, 2^{2^{2^2}}]$	4
$(2^{16}, 2^{(2^{16})}] = (2^{2^{2^2}}, 2^{2^{2^{2^2}}}]$	5
⋮	

$2^{16} \approx 65{,}000$

(tiny)

Facts we need:

- Once a node stops being a root, it will never be a root again.

  Why? consider unions + finds

  find: only changes parents, stops at root

  union: one root becomes a child - can be path compressed, but not a root

- Once not a root, a node's rank never changes.

  Why? Well, when does rank get changed?

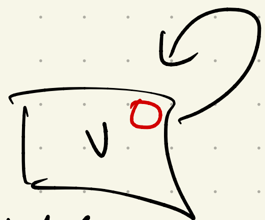
  not in find

  in union, only changes the end root

- Ranks are increasing in any leaf-to-root path.

Proof: induction on time (i.e # of ops)

base case: Singleton

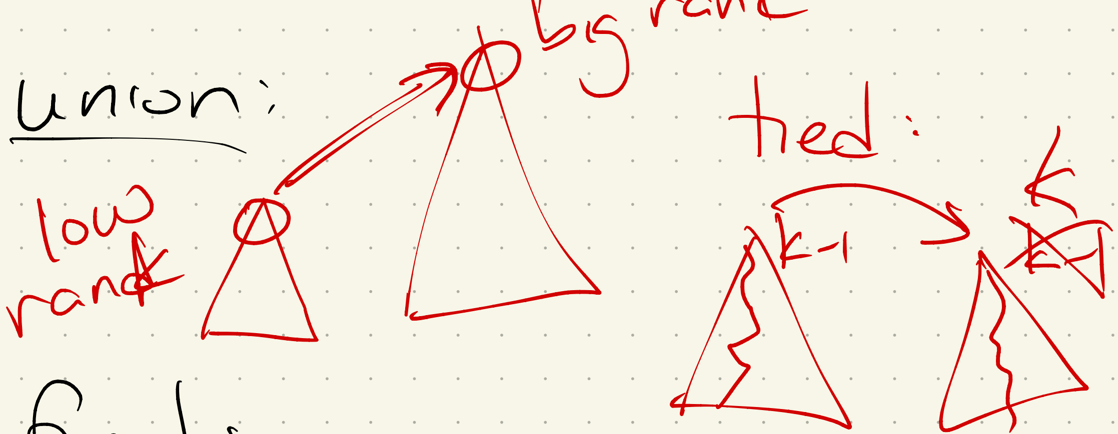


Ind step: consider t^{th} operation = either:

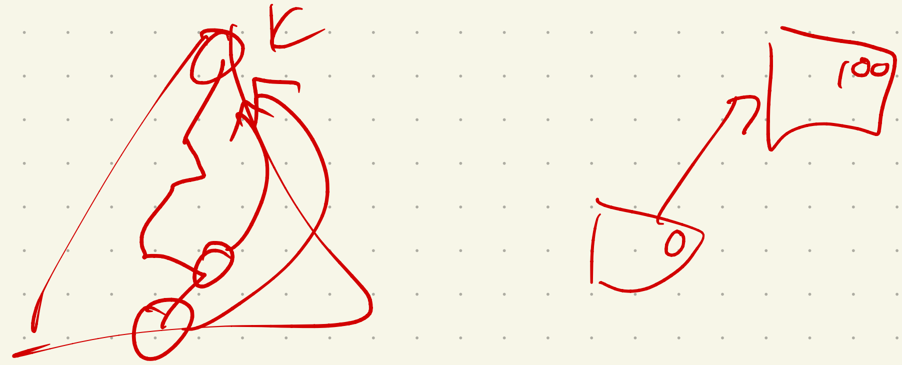
makeset:

no other root to leaf paths change

union:



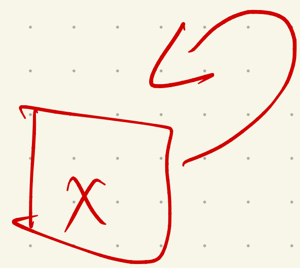
find:



Lemma: When a node gets rank k, there are $\geq 2^k$ items in its tree.

Proof: induction on rank:

$r=0$: $2^0 = 1$



Now assume true for anything $< r$, & consider the first time rank $= r$:

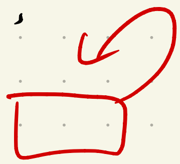
must be a union, with two roots that have rank $r-1$

By IH, those each have $\geq 2^{r-1}$ there are 2 of them.

total # $\geq 2 \cdot 2^{r-1} = 2^r$ ☒

Lemma: For any r, there are at most $(\le)$ $n/2^r$ objects with rank r through entire execution.

Proof: More induction!

$r=0$: rank 0: 

n elements: $\frac{n}{2^0} = n$

$r>0$: If a node v has rank r:

we will "charge" it to the two nodes u & v of rank $r-1$ at time of union.

After union, neither can *ever* make another rank r node.

So: if $\frac{n}{2^{r-1}}$ at rank $r-1$,

then it takes 2 of rank $r-1$ to make one of rank r: $\frac{n}{2^{r-1}} \cdot \frac{1}{2} = \frac{n}{2^r}$

Side note:

Worst case $\log n$:

$\frac{n}{2^r}$ at rank r.

$\Rightarrow$ highest rank? $\log n$

$\frac{n}{2}/2/2/2$

(And so tree height can't be larger)

Back to the $\log_2^* n$ stuff:

<u>Define</u> $\text{Tower}(i) = 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}} \Big\}$ height i

so $\log_2^*(\text{Tower}(i)) = i$

<u>Define</u>: $\text{Block}(i) =$

$$[\text{Tower}(i-1)+1, \text{Tower}(i)]$$

$\text{Block}(0) = [0,1]$ (just b/c)

$\text{Block}(1) = [2,2]$

$\text{Block}(2) = [3,4]$

$\text{Block}(3) = [5,16]$

$\text{Block}(4) = [17, 65536]$

$\text{Block}(5) = [65536, 2^{65536}]$

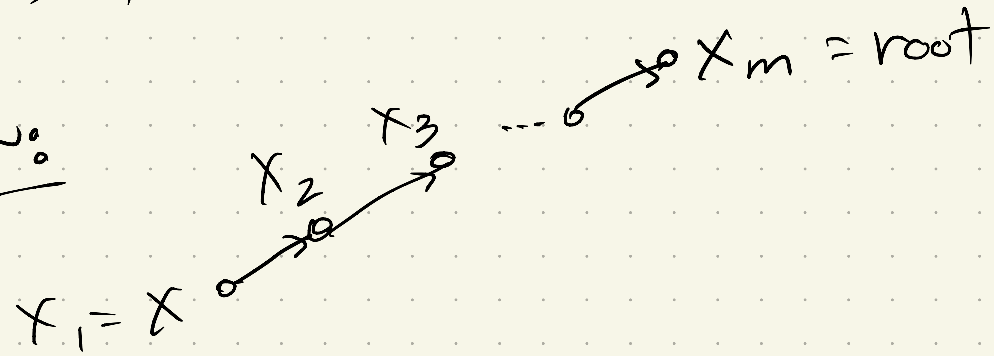
$\vdots$

<u>Now</u>: We know runtime of find(x) = length of x to root path:

Let our path π =

$x = x_1,\ p(x) = x_2,\ p(x_2) = x_3, \ldots, x_m = \text{root}$

<u>π:</u>

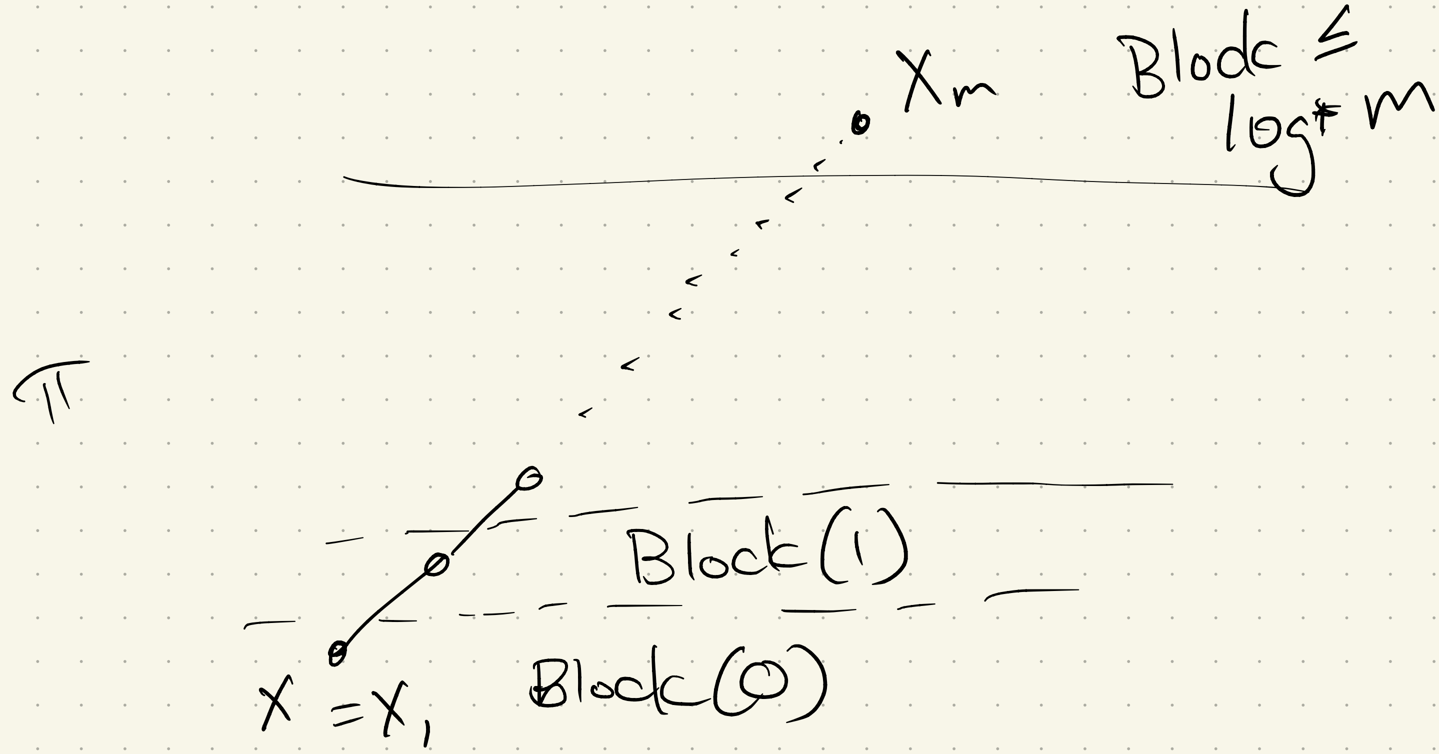


Say a node y is in i^{th} block if $\text{rank}(y) \in \text{Block}(i)$

In UF, max rank of <u>any</u> node is $\log n$.

(So only have $\log^* n$ blocks total.)

In these Blocks:



When we move $x_k \to p(x_k)$, could stay in a block (an internal jump)

or move to higher block (a jump between blocks)

Lemma: If x is an element in Block(i), at most $|\text{Block}(i)|$ finds can pass through it until it moves to Block($i+1$).

pf: What happens with each find?
path compression!

Lemma: At most $\frac{n}{\text{Tower}(i)}$ nodes have rank in Block(i) over entire algorithm.

Pf: For rank r, know

$\leq \frac{n}{2^r}$ elements

at that rank.

$$\text{Block}(i) = [\text{Tower}(i-1)+1, \text{Tower}(i)]$$

so:

$$\sum_{k \in \text{Block}(i)} \frac{n}{2^k}$$

$$= \sum_{k=} \frac{n}{2^k}$$

$$=$$

Finally:

The number of internal jumps in i^{th} block is $O(n)$ (over entire set of m finds).

pf:
- x in Block(i) can have $|\text{Block}(i)|$ internal jumps
- $|\text{Block}(i)| \leq \dfrac{n}{\text{Tower}(i)}$

So # internal jumps $\leq$

Thm: m operations on n elements in U-F take $O((m+n)\log_2^* n)$ total time.

pf: Either an operation is $O(1)$, or its runtime is $\approx$ (# internal jumps) + (# jumps b/t blocks)

internal jumps:

jumps b/t blocks: