Amortized Analysis: Union-Find 0925

We saw it reported in Sedgewick & Wayne's slides that:

- logically treating the collection of sets as a forest
- actually representing the forest as an array (of parent pointers).

- implementing union-by-rank and path-compression yields $\Theta(\alpha(m,n))$ amortized running time!

ops → m; # elements → n

$\alpha(m,n)$ is "inverse Ackerman's" function which grows so slowly that if m, n are $\leq \underbrace{\text{\# atoms in universe}}_{10^{80}}$

then $\alpha(m,n) \leq 4$

I.e., amortized running time of $\Theta(\alpha(m,n))$ is, for all practical purposes, like $\Theta(1)$.

It takes a bit of work to show the $\Theta(\alpha(m,n))$ bound. We will show a different but similar bound.

Defn: $\lg^{(i)} n = \begin{cases} n & \text{if } i = 0 \\ \lg\left(\lg^{(i-1)} n\right) & i > 0 \text{ and } \lg^{(i-1)} n > 0 \\ \text{undefined} & i > 0 \text{ and } \lg^{(i-1)} n < 0 \\ & \text{or } \lg^{(i-1)} \text{ is undefined.} \end{cases}$

Defn: $\lg^* n = \min\{ i \geq 0, \lg^{(i)} n \leq 1 \}$

$\lg^* n$ is inverse of repeated exponentiation.

2^{65536} is way more than the number of atoms in the universe

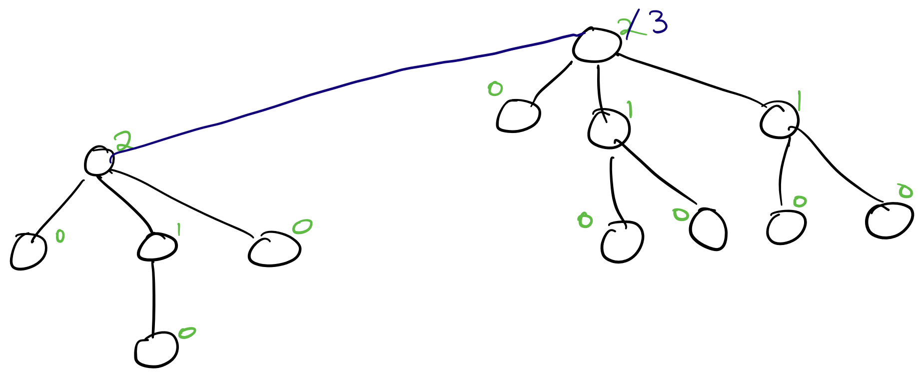
$$\lg 2^{65536} = 65536$$
$$\lg 65536 = 16$$
$$\lg 16 = 4$$
$$\lg 4 = 2$$
$$\lg 2 = 1$$
$$\lg 1 = 0$$

How many times do we apply lg to 2^{65536} till we get to ≤ 1?

5

$\therefore \lg^* 2^{65536} = 5$

Indeed, $\lg^* n \leq 5$ $\forall$ integers we will usually encounter in our work, so proving that an alg runs in $\lg^* n$ amortized time means practically it runs "like constant time"



- rank of a singleton tree is 0
- rank after union op is
  - same if one tree has smaller rank than other (smaller ranked tree becomes child)
  - inc by 1 if trees have same rank.

Easy to show:

Cormen, Leiserson, Rivest & Stein, Algorithms

Lemma 22.3 (CLRS) $\forall$ tree roots x,

$$\text{size}(x) \geq 2^{\text{rank}(x)}$$

Proof: use induction on number of Link operations, where Link is linking root of lower rank tree to x (x is parent - why?)

Exercise for the student.

Lemma 22.2 (CLRS)

rank(x) starts at 0, can only increase while x is a root, and does not change once

x becomes a child.

ranks only go up as you go up a tree.

Lemma 22.4 (CLRS)

$\forall$ integer $r \geq 0$, $\exists \leq \frac{n}{2^r}$ nodes of rank r.

Proof: Fix r.

Suppose that, in the course of things, whenever a root X gets rank r, X paints all the nodes in its tree X-coloured

$\therefore$ by Lemma 22.3, $\geq 2^r$ nodes are painted each time a root gets rank r.

Can a node a be painted twice?

No - will never paint a node twice.

Since no node can be painted twice,

- there are $\leq n$ painted nodes,
- Each colour-class has size $\geq 2^r$

$\therefore \exists \leq \frac{n}{2^r}$ colour classes

$\therefore \leq \frac{n}{2^r}$ nodes ever get rank r. $\blacksquare$

Corollary: $\forall$ nodes have rank $\leq \lfloor \lg n \rfloor$.

Let us consider a sequence of operations...

m' ops:

MakeSet(30), Makeset(.), ... Makeset(50), $\underbrace{\text{union}(30, 50)}_{\text{Find}(30),\ \text{Find}(50),\ \text{Link}(r_1, r_2)}$, Find(20), -

replace $\rightarrow$

Union with this.

$\leftarrow m$ is new # ops in sequence

What does that do to m', the number of ops? no more than triples it, to become $m \leq 3m'$.

If we can show the sequence runs in $O(m \lg^* n)$ time, then it runs in $O(m' \lg^* n)$ time.

So we will consider our sequence as being of operations

MakeSet(-), Find(-), Link(-,-)

Theorem: A sequence of m MakeSet, Find, Link ops performed using path compression and union-by-rank runs in worst-case $O(m \lg^* n)$ time.

Proof

- We assess charges to each operation corresponding to actual cost of each operation.

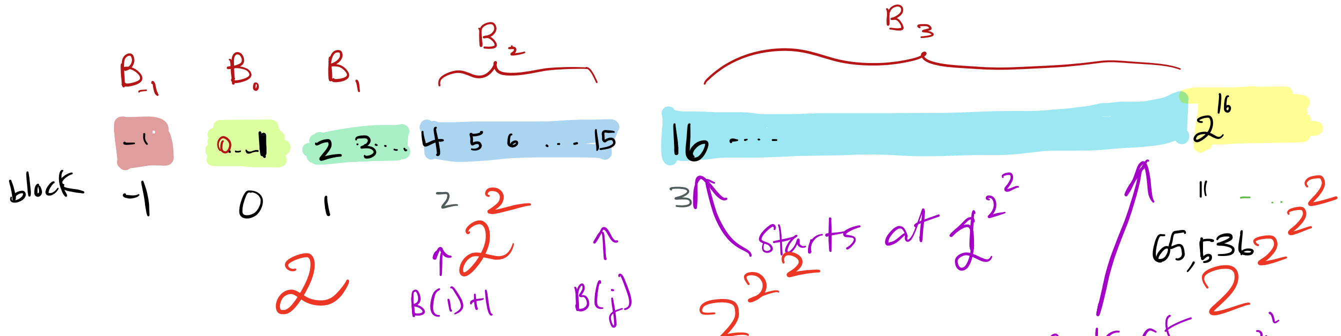
MakeSet — one charge per op'n

Link — one charge per op'n.

For Find:

- note that number of charges = number of parent pointers altered to point to a new parent of ~~same or~~ greater rank than old parent

ranks will be considered to fall into $\boxed{\text{Blocks}}$



We will now use an Accounting Method

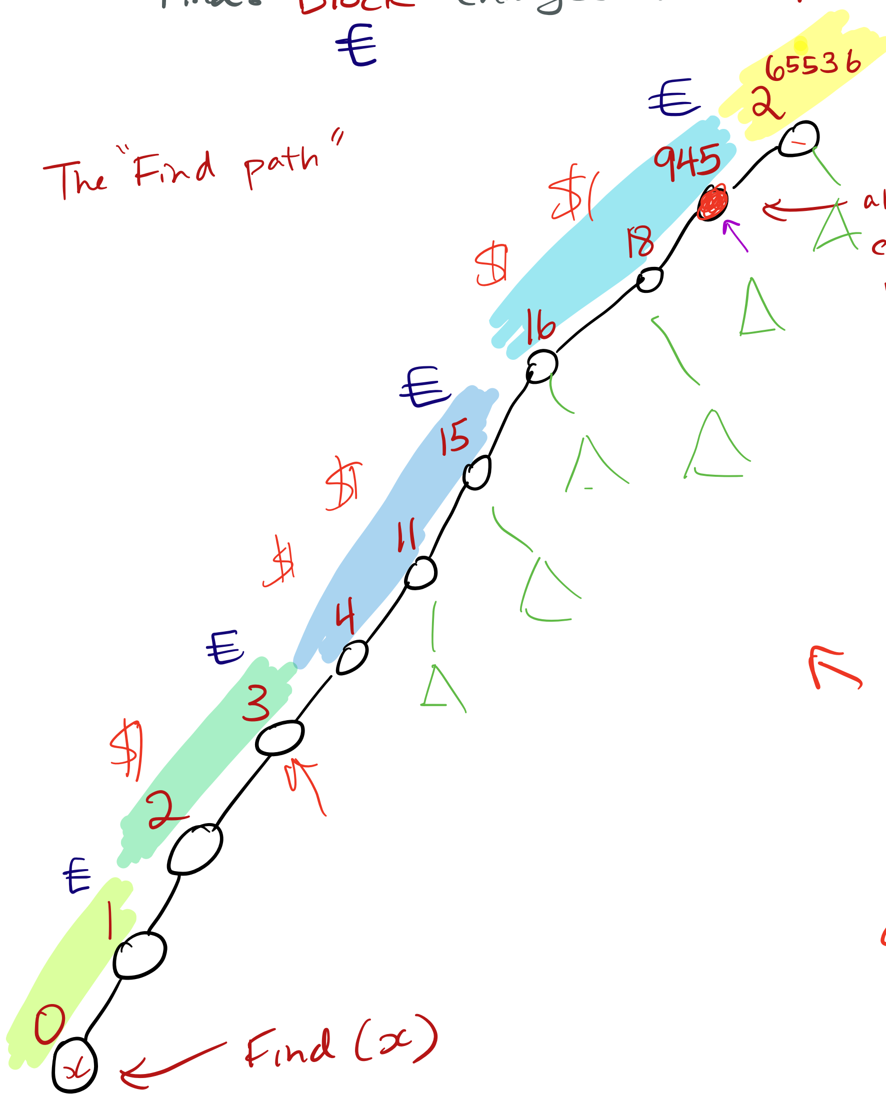
- all work is charged to the "account" of work done during the course of m operations, Makeset(-), Find(-), Link(-,-) where n of the ops are makeset(-).

- a "charge" will count for any constant amount of work.

- Makeset(-) — one charge
- Link(-,-) — one charge

Find (_)

There are 2 kinds of charges within a Find: Block charges € and Path $ charges.



always a block charge to child of root, even if root is in same block (eg rank= 946)

$\exists \leq \lg^* n$ blocks containing all possible ranks.

Claim: Once a node other than a root or its child are assessed a Block charge it will never again be charged a path charge. $

Proof: If not a root or its child when it gets a block charge, then its parent has rank in a higher block, and that gap will never diminish (parent may change, but will only have same or higher rank).

Total # of charges in all the FindSet operations:

€

1. For each Find, number of Block charges is

$\leq 1 + \lg^* n$ — Why?

$\therefore$ total is $O(m \lg^* n)$

2. We ask, for a given node x over all the Finds, how many times can x be charged a path charge? $

- only while x's parent is in same block
- we only need to attend to the Find ops that give x a new parent, because otherwise x is a child of a root and is getting €, not $
- can only do this $\underbrace{B(j) - B(j-1) - 1}_{\text{size of block containing } x\text{'s rank.}}$ times

Recall – a nodes rank is "frozen" once it becomes a child of another node, and it never gets a \$ (path charge) when it is a root.

∴ it is sufficient to look at every non-root node at the end of the run and add up the sizes of the Blocks their ranks belong to.

$N(j)$ = number of nodes with ranks in Block(j)

$$N(j) \le \sum_{r=B(j-1)+1}^{B(j)} \frac{n}{2^r} = \frac{n}{2^{B(j-1)+1}} + \frac{n}{2^{B(j-1)+2}} + \dots$$

for $j=0$, $N(0) = \frac{n}{2^0} + \frac{n}{2^1} = \frac{3n}{2} = \frac{3n}{2B(0)}$

for $j \ge 1$, $N(j) \le \frac{n}{2^{B(j-1)+1}} \cdot \underbrace{\sum_{i=0}^{\infty} \frac{1}{2^i}}_{2}$

$$\le n \sum_{i=0}^{\infty} \frac{1}{2^i \, 2^{B(j-1)+1}}$$

$$= \frac{n}{2^{B(j-1)+1}} + \frac{n}{2^{B(j-1)+2}} + \dots$$

$$\le \frac{n}{2^{B(j-1)}} = \frac{n}{B(j)} \le \frac{3}{2} \cdot \frac{n}{B(j)}$$

Hence $N(j) \le \dfrac{3n}{2B(j)}$

Let $P(n)$ = number of path changes

$$P(n) \le \sum_{j=0}^{\lg^* n - 1} \underbrace{\frac{3n}{2B(j)}}_{\text{upper bound on number of nodes with rank} \in B(j)} \underbrace{\big(B(j) - B(j-1)\big)}_{\text{upper bound on number of ranks the nodes parent can have}}$$

upper bound on number of nodes with rank $\in B(j)$

upper bound on number of ranks the nodes parent can have

- each path change comes with a new parent-rank within the block.

$$\le \frac{3n}{2} \lg^* n$$

$\therefore$ Total # of path changes \$ is $\le \frac{3}{2} n \lg^* n$

Total # of block changes € is $\le m \lg^* n$

Also, $n \le m$, so total # charges is $O(m \lg^* n)$

■

Corollary: A sequence of m MakeSet(_) Union(_,_) Find() operations, n of which are MakeSet(_), can be performed on the disjoint-set-forest implementation of Union-Find (using union-by-rank and path compression) in worst case $O(\lg^* n)$ amortized time.

Going back to that mysterious page...

Hence $N(j) \le \dfrac{3n}{2B(j)}$

Let $P(n)$ = number of path changes

$$P(n) \le \sum_{j=0}^{\lg^* n - 1} \underbrace{\frac{3n}{2B(j)}}_{\text{upper bound on number of nodes with rank } \in B(j)} \underbrace{\big(B(j) - B(j-1)\big)}_{\text{upper bound on number of ranks the nodes parent can have}}$$

# of different blocks

– each path change comes with a new parent-rank within the block.

$$= \frac{3n}{2} \lg^* n$$

∴ Total # of path changes ($) is $\le \frac{3}{2} n \lg^* n$

Total # of block changes (€) is $\le m \lg^* n$

Also, $n \le m$, so total # charges is $O(m \lg^* n)$

∎

How many blocks can there be for n-element forest?

rank r is in block $\lg^* r$, for

$r = 0, 1, \ldots, \lfloor \lg n \rfloor$ ← $\lfloor \lg n \rfloor$ is maximum rank.

The highest numbered block is

$$\lg^*(\lg n) = \lg^* n - 1.$$