Algorithm Design and Analysis

Union-Find (More Amortized Analysis!)

Reminder

- **Midterm One is next Tuesday at 7:00pm**
- If you 100% can not make this for a legitimate reason, email us or post on Piazza **by the end of today**.

Roadmap for today

- Design the ***Union-Find*** data structure for the ***disjoint sets problem***
- Practice ***potential functions*** by analyzing Union-Find

Motivation: Kruskal's Algorithm

Review (Minimum Spanning Tree): A spanning tree of an undirected graph with the least total (edge) cost of all possible spanning trees

Review (Kruskal's Algorithm): For each edge (u, v) in sorted order by cost, add the edge to the spanning tree if u and v are not connected.

How do we do that part??

The disjoint-sets problem

Problem (Disjoint Sets): We want to support the following API:

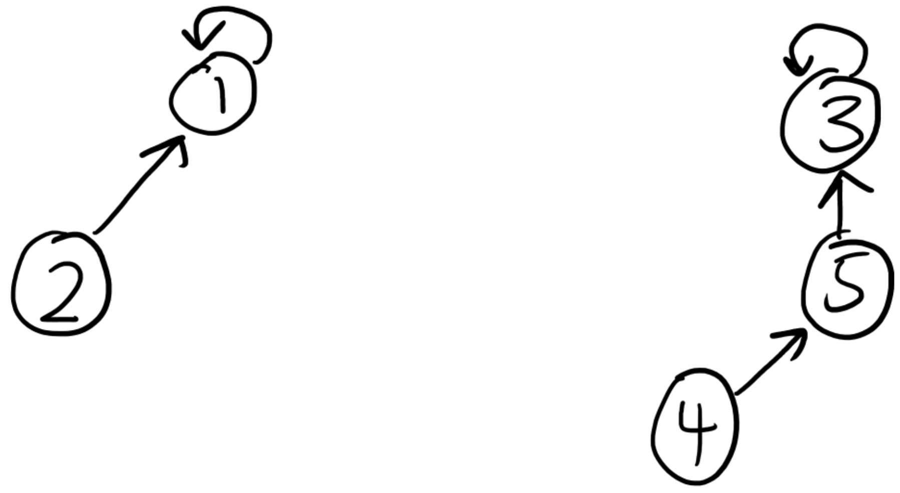
- **MakeSet**$(\boldsymbol{x})$: Create a set consisting of the single element $\{x\}$
- **Find**$(\boldsymbol{x})$: Return the *representative element* of the set containing x
- **Union**$(\boldsymbol{x}, \boldsymbol{y})$: Merge the two sets $S_x \ni x$ and $S_y \ni y$ into a single set.

Simple but inefficient #1: Maintain a representative for each element. Union loops over every element and updates the ID of the representative: **costs $\boldsymbol{O}(\boldsymbol{n})$**. Find **costs $\boldsymbol{O}(1)$**.

Simple but inefficient #2: Maintain a graph with an adjacency list. Union just adds a new edge: **costs $\boldsymbol{O}(1)$**. But find must search the entire connected component: **costs $\boldsymbol{O}(\boldsymbol{n})$**.

The disjoint-set forest data structure

- ***Key idea***: Represent the sets as **trees**. Use the roots of the trees as the representative element.
- Representation: Store a parent pointer for each node. Roots have no parent (by convention, $p(x) = x$ for roots).



Implementation (basic version)

- **MakeSet**(x):

  Create x

  set $p(x) \leftarrow x$

- **Find**(x):

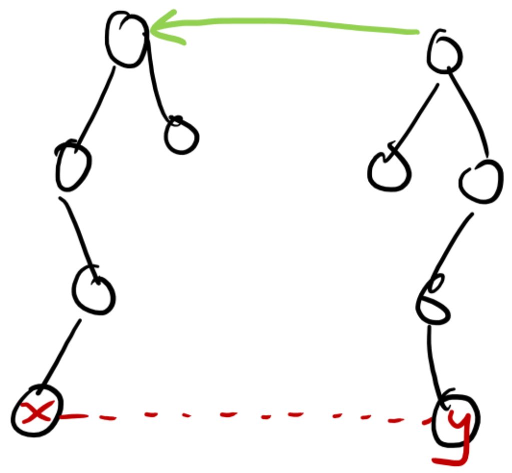
  Walk up parent chain until root.

- **Union**(x, y):

  Link(Find(x), Find(y))

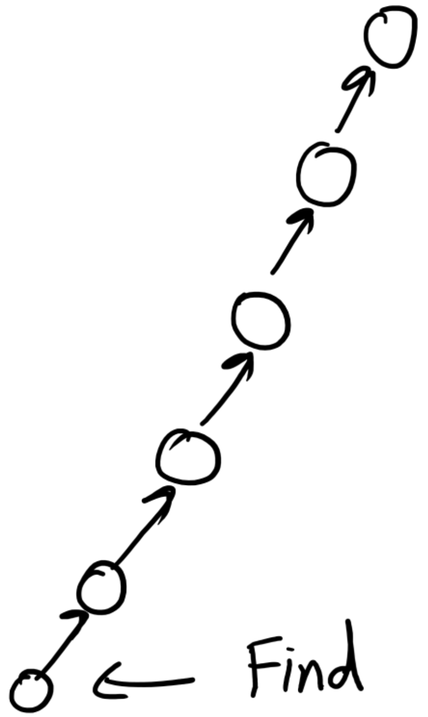
- **Link**(x, y)

  $p(y) \leftarrow x$



Performance

Theorem: Let n be the current number of elements in the sets (i.e., the number of MakeSet operations performed so far). There exists inputs for which every find costs $\Theta(n)$.



Making Union better?

- The bad performance was caused by long chains of nodes…
- Can we just… not do that?

Idea (Union-by-size): When performing a Union, make the smaller tree a child of the larger tree. If they're the same size, then pick arbitrarily.

- We should store an extra field $s(x)$ that knows the size of the trees
- $s(x)$ is the size of the tree rooted at x (we don't care about non-roots)

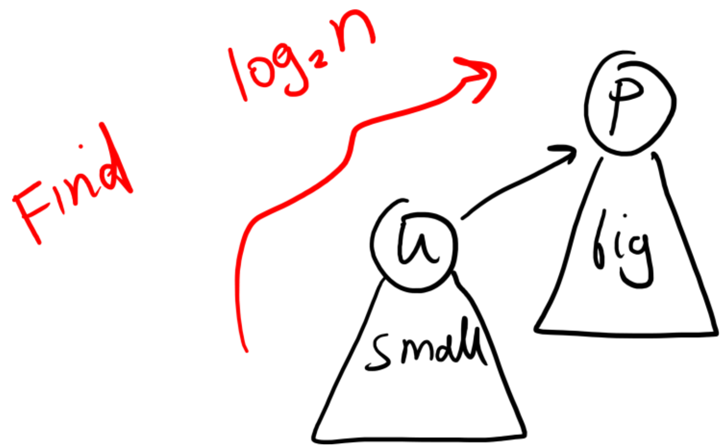
Union-by-size implementation

- **Link(x, y):**

```
    If $s(x) < s(y)$ then swap$(x, y)$
    $p(y) \leftarrow x$
    $s(x) \leftarrow s(x) + s(y)$
```

Performance of union-by-size

Theorem: Let n be the current number of elements in the sets. Using *union-by-size*, every Link operation costs $O(1)$ and every Find operation costs $O(\log n)$ in the **worst-case**.

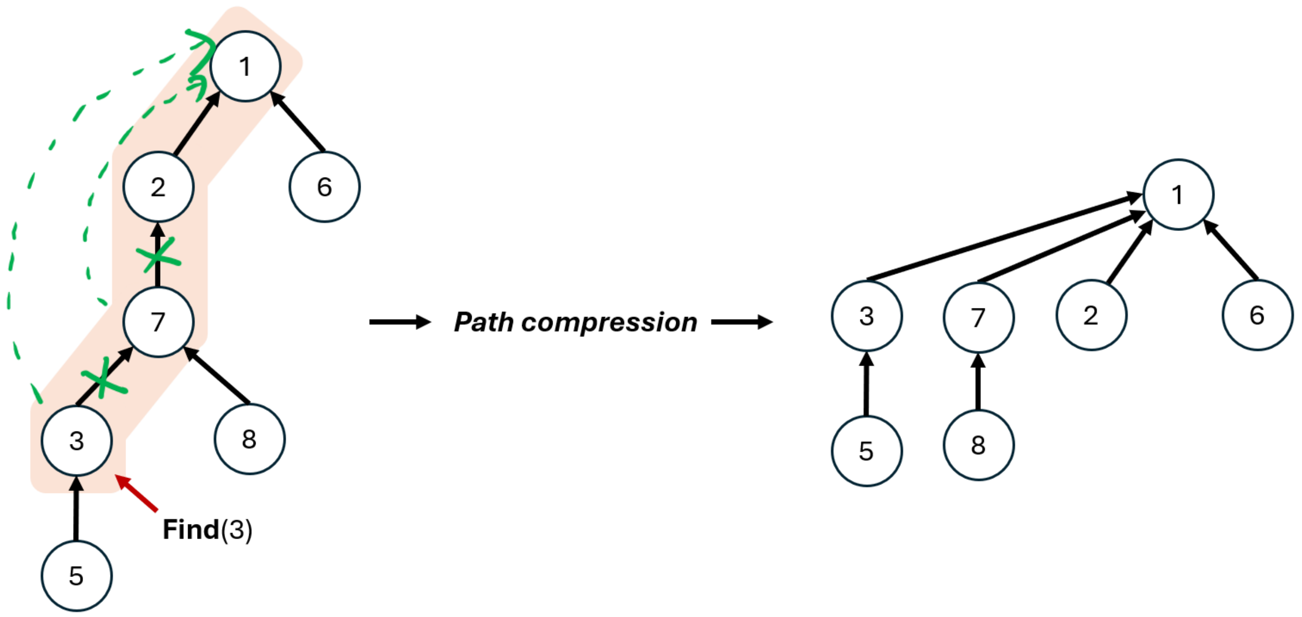


$\text{size}(p) > 2\ \text{size}(u)$

Another improvement

- We just made Union better. Can we instead/also make Find better?

Idea (Path compression): When performing a Find, point every node along the path at its current root/representative element.



Path compression implementation

- **Find**(x):

```
if $p(x) \neq x$:
    set $\underline{p(x)} \leftarrow$ Find$(p(x))$
return $p(x)$
```

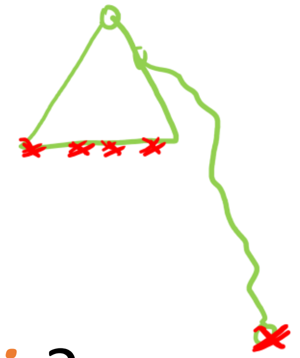
Cost model for amortization

- To avoid arbitrary constants in the analysis, we will once again work in a simplified cost model. All our analyses will be asymptotically valid in the word RAM up to constant factors.
  - **MakeSet** costs 1
  - **Link** costs 1
  - **Find** costs **number of nodes touched**

- **Goal**: ***Amortized costs*** of $O(\log n)$ for each operation

Performance of path compression

Theorem: Let n be the current number of elements in the sets. Using *path compression* (but not union-by-size), in our cost model, the ***amortized cost*** of MakeSet is 1, Link is $(1 + \log n)$, and Find is $(2 + \log n)$.



- **Observation**: ***Balance*** is what matters
- Balanced trees are always fast, imbalanced trees are slow
- How do we measure how balanced a tree is at a ***per-node basis***?

Balanced or imbalanced?

Definition (heavy/light): Given a node u and its parent p, call a node:

- ***Heavy*** if $\text{size}(u) > \frac{1}{2}\,\text{size}(p)$, i.e., u contains a majority of p's descendants
- ***Light*** if $\text{size}(u) \leq \frac{1}{2}\text{size}(p)$, i.e., u contains at most half of p's descendants

- Root is neither heavy nor light (it has no parent)
- In a perfectly balanced tree, every node is light (except root)
- In a chain (the worst-balanced tree), every node is heavy (except root)

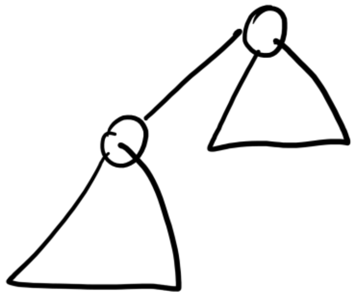
& leaf.

Balanced or imbalanced?

Lemma (light lemma): On any root-to-leaf path in any tree of n nodes, there are at most $\log n$ light nodes.

Definition (Light):

$$\mathrm{size}(u) \le \frac{1}{2}\mathrm{size}(p),$$



Same proof as before!

Reaching your potential

- Find costs #nodes touched = $(1 + \#\text{heavy} + \#\text{light})$.
- We know that $\#\text{light} \leq \log n$ **(light lemma)**
- So, Find costs at most $(1 + \log n + \#\text{heavy})$.
- We want to define a ***potential function*** that will save up and **pay for the cost of touching the heavy nodes**

$$\mathbf{ac} = \underbrace{1 + \log n + \#\text{heavy}}_{\textit{Actual cost}} + \underbrace{\Delta\Phi}_{\textit{Change in potential}}$$

want this to be $\approx -\#\text{heavy}$

A *potential* idea

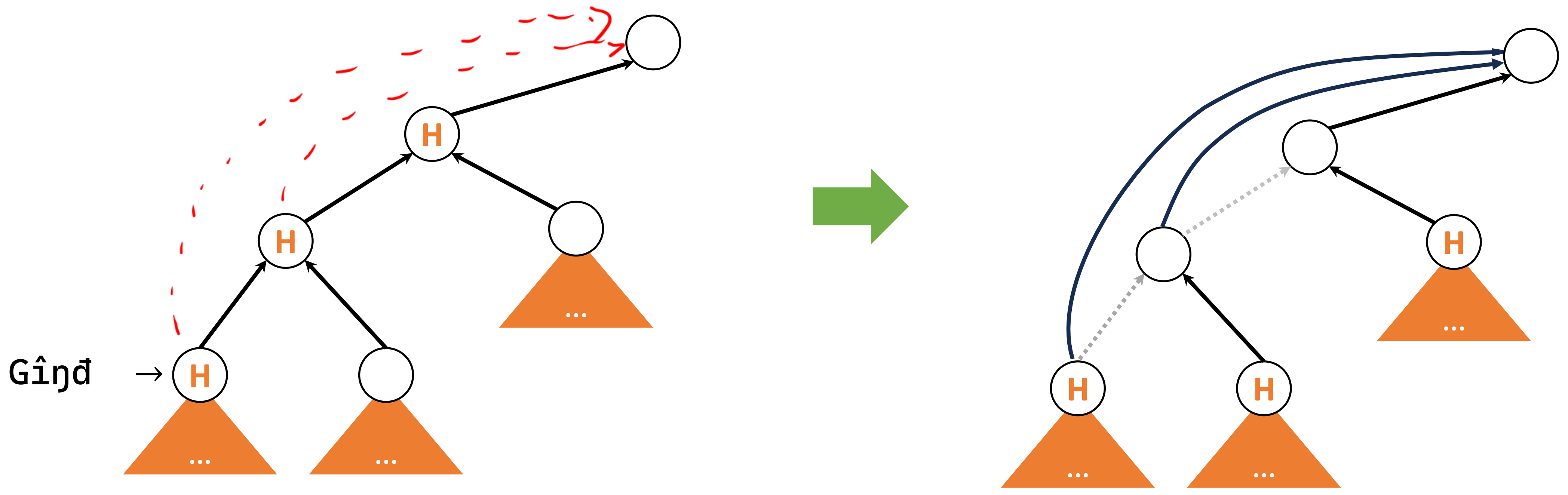
- ***Observation***: A node can have ***at most one heavy child***
- *Potential* potential function idea

$$\Phi(F) = \text{\# heavy nodes}$$

- ***Problem***: Can we prove that the number of heavy nodes decreases?
- ***Exercise****:* Draw a scenario where a path compression operation does not decrease the number of heavy nodes.

Too many heavy nodes!

- ***Exercise:*** Draw a scenario where a path compression operation does not decrease the number of heavy nodes.



Refining the potential

- ***Observation***: A node can have ***at most one heavy child*** ***but*** moving it does not necessarily reduce the number of heavy children (a sibling may become heavy in its place)
- However, is there a maximum number of times this can happen?
- Moving the heavy child ***at least halves*** the size of the subtree!
- Therefore, the subtree of node x with size $s(x)$ can have its heavy child moved at most $\underline{\log_2(\text{size}(x))}$ times!

The balance potential

- Define our potential function to be:

$$\Phi(F) = \sum_{u \in F} \log(\mathrm{size}(u))$$

all nodes $\longrightarrow$

- **Nice properties**:
  - Initially zero (all trees start at size 1)
  - Always non-negative
  - Increases when we perform a Link
  - Decreases when we perform a Find

$\left.\begin{array}{l}\text{Initially zero}\\ \text{Always non-negative}\end{array}\right\}$ no debt $\Phi(S_m) \geq \Phi(S_0)$

$\left.\begin{array}{l}\text{Increases when we perform a Link}\\ \text{Decreases when we perform a Find}\end{array}\right\}$ Links save up $$$ to pay for Finds

Analysis of MakeSet

$$\Phi = \sum \log(\text{size}(u))$$

Lemma (cost of MakeSet): MakeSet does not change $\Phi(F)$

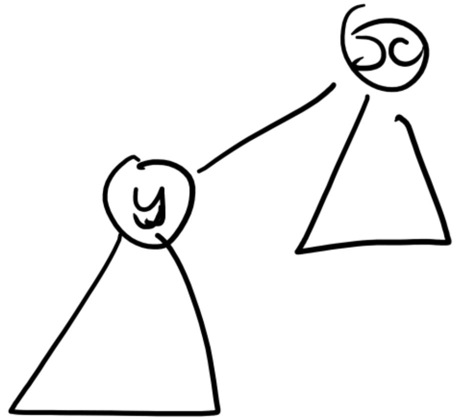
$$\Delta \Phi = \log(1) = 0$$

Corollary: The MakeSet operation has an amortized cost of 1

Analysis of Link

$$\Phi = \sum \log(\mathrm{size}(u))$$

Lemma (cost of Link): A link operation at most increases $\Phi(F)$ by $\log n$



$$\Delta\Phi = +\log(\mathrm{size}'(x)) - \log(\mathrm{size}(x))$$

$$\le \log(n) - \log(\mathrm{size}(x))$$

$$\le \log(n)$$

$$ac = \mathrm{actual} + \Delta\Phi$$

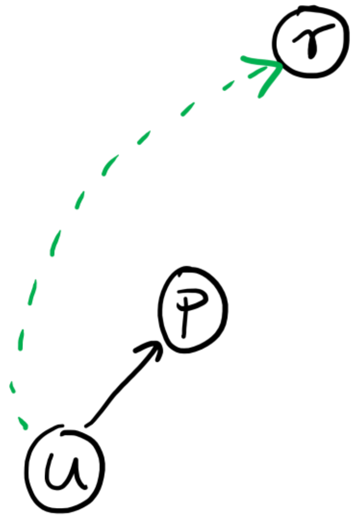
Corollary: A link operation has amortized cost at most $1 + \log n$

Analysis of Find

$$\Phi = \sum \log_2(\text{size}(u))$$

Lemma: A Find operation decreases $\Phi(F)$ by at least $\#\text{heavy nodes} - 1$

- Consider ***heavy nodes*** u with parent p (other than r) on the **Find** path



$$\text{size}'(p) = \text{size}(p) - \text{size}(u)$$
$$< \tfrac{1}{2}\text{size}(p)$$

$$\Delta\Phi_{(\text{at node } p)}$$
$$= \log(\text{size}'(p)) - \log(\text{size}(p))$$
$$< \log\left(\tfrac{1}{2}\text{size}(p)\right) - \log(\text{size}(p)) = -1$$

Add this up for all heavy u

Analysis of Find

Corollary (cost of Find): Find has amortized cost at most $(2 + \log n)$

$$\text{a.c.}_{\text{FIND}} \le \underbrace{1 + \cancel{\#\text{heavy}} + \#\text{light}}_{\text{actual cost}} \underbrace{- (\cancel{\#\text{heavy}} - 1)}_{\Delta\Phi}$$

$$\le 2 + \#\text{light}$$

$$\le 2 + \log n$$

Summary of Union-Find

$\alpha(n)$

- ***Union-Find with union by size***:
  - ***Link***: $O(1)$
  - ***Find***: $O(\log n)$ **worst-case**
- ***Union-Find with path compression***:
  - ***Link***: $O(\log n)$ **amortized**
  - ***Find***: $O(\log n)$ **amortized**
- ***Union-Find with both! (Not proven in this class)***:
  - ***Link***: $O(\alpha(n))$ **amortized**
  - ***Find***: $O(\alpha(n))$ **amortized**
  - $\Omega(\alpha(n))$ is also a lower bound so this is optimal!

(Text book, CLRS)