

Disjoint-Set Forests

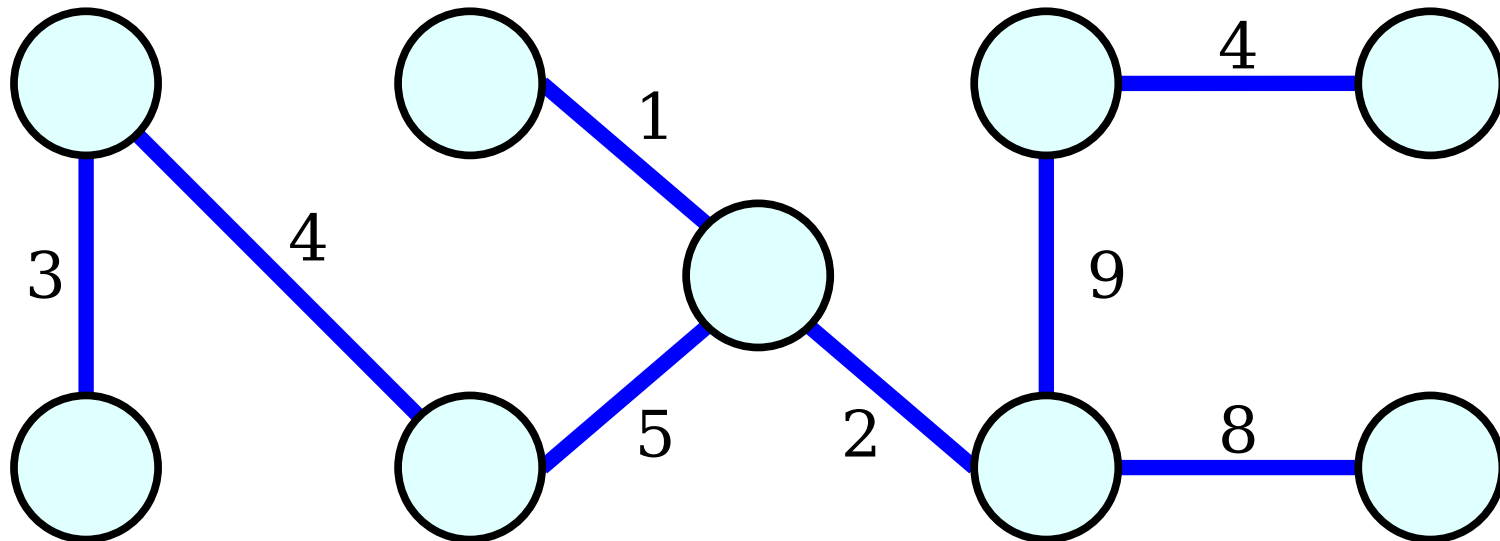
Outline for Today

- ***Incremental Connectivity***
 - Finding connected nodes as a graph changes.
- ***Disjoint-Set Forests***
 - A surprisingly simple and subtle data structure.
- ***Analyzing Disjoint-Set Forests***
 - A clever, nuanced analysis with a surprising result.

Incremental Dynamic Connectivity

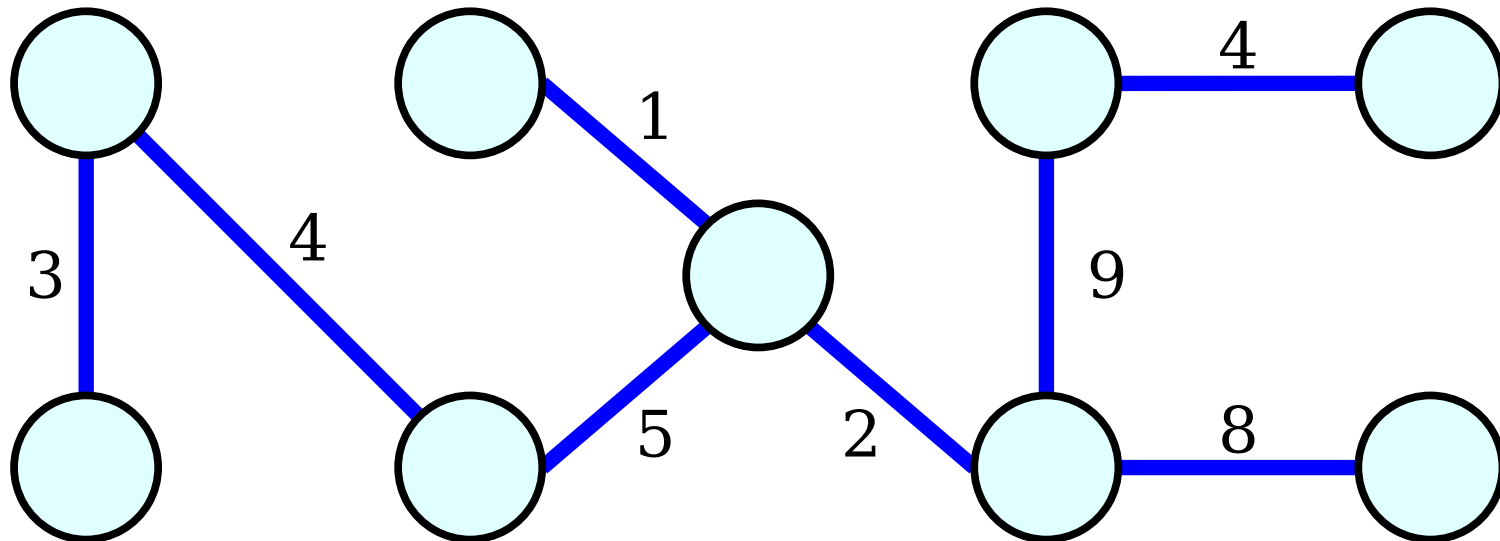
Kruskal's Algorithm

- ***Kruskal's Algorithm*** finds an MST of a graph. It works as follows:
 - Remove all edges from the graph and sort them from lowest to highest.
 - Repeatedly insert edges back into the graph, as long as their endpoints aren't already reachable from each other.



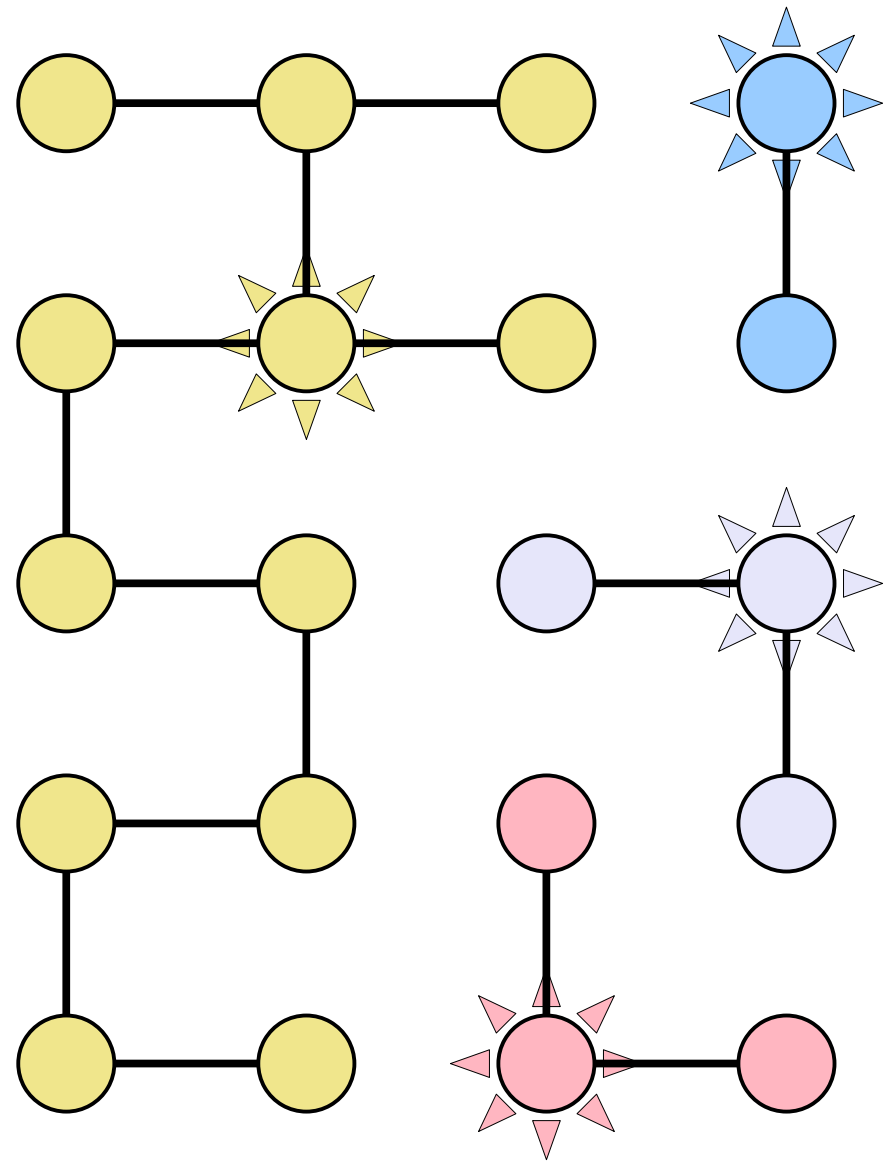
Incremental Connectivity

- Kruskal's algorithm needs a data structure that solves *incremental dynamic connectivity*:
 - answer questions about which nodes are reachable
 - in a graph that is changing
 - via edges being added in.
- **Question:** How efficiently can we do this?



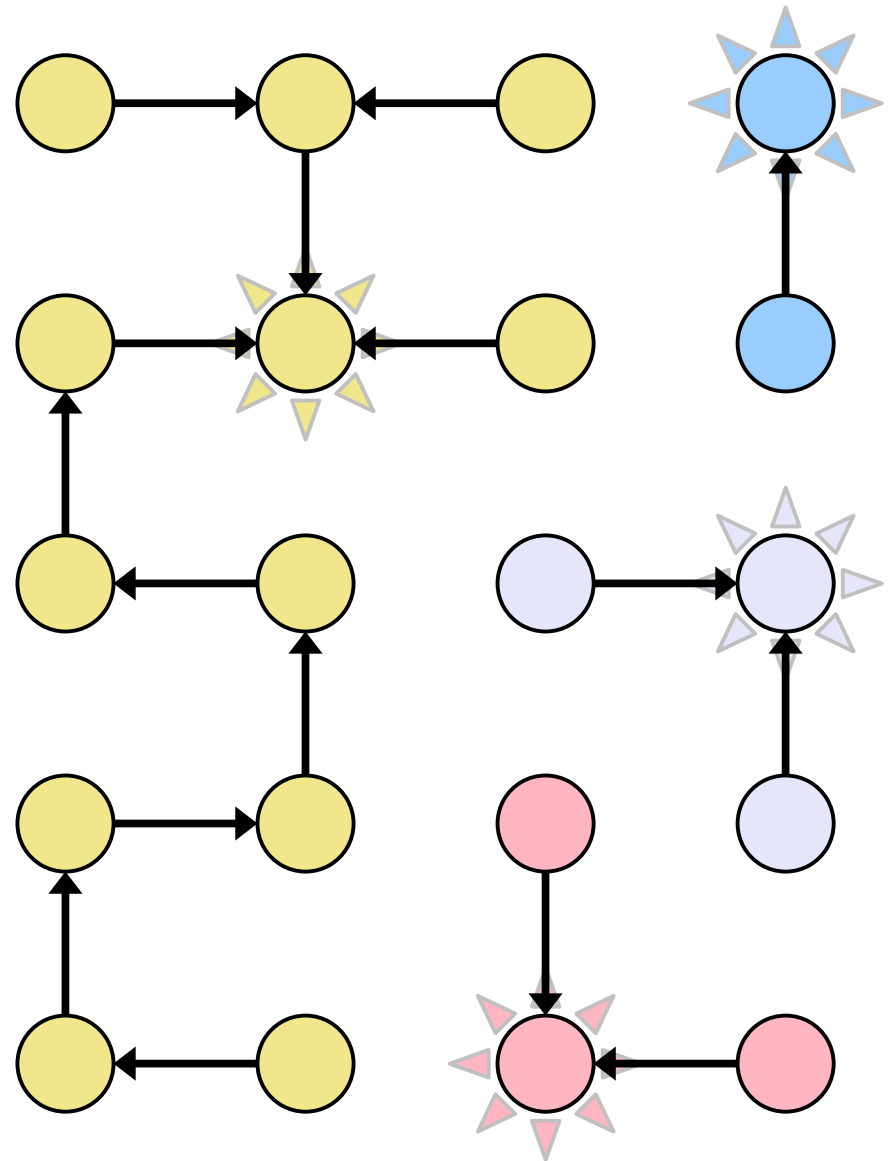
Representatives

- **Idea:** Assign a **representative** to each CC in the graph.
- To see if two nodes are in the same CC, check if they have the same representative.
- To link together two different CCs, change the representative of all the nodes in one CC to be the representative of the other CC.



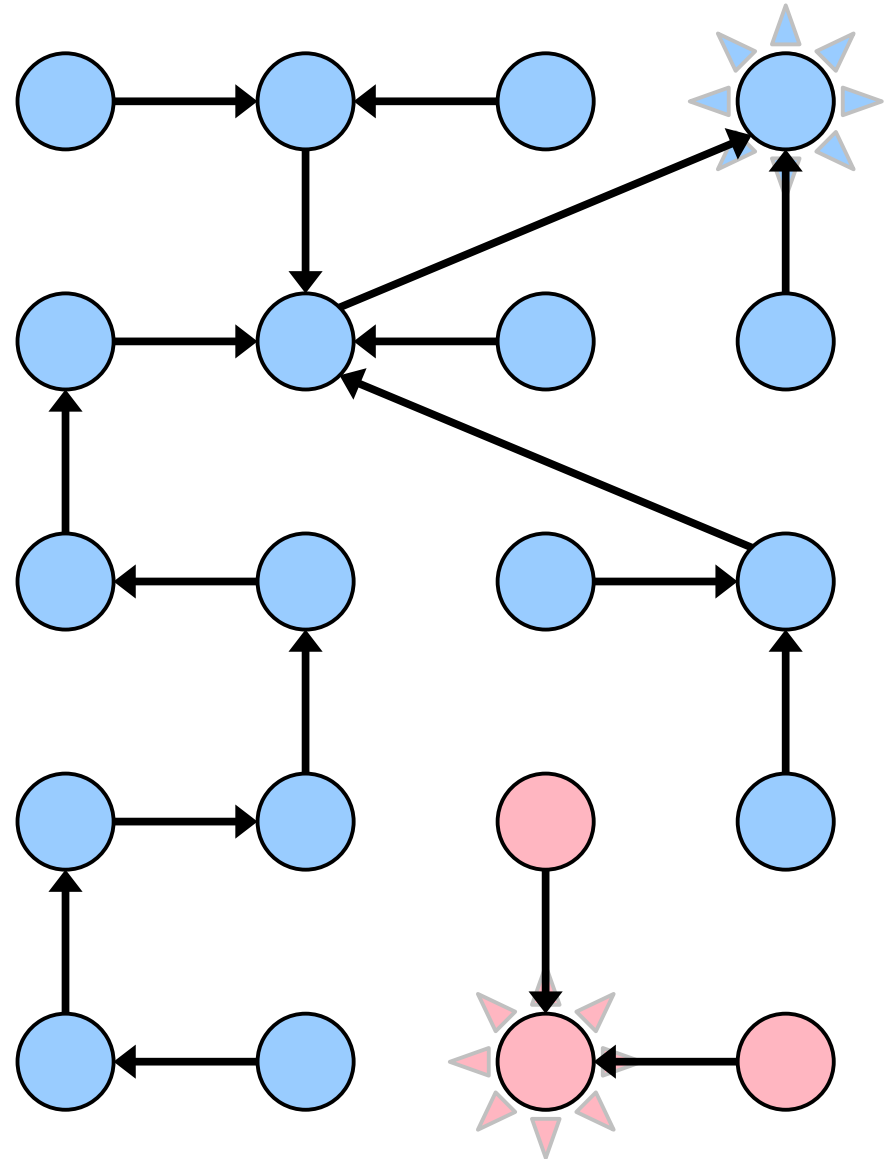
Representatives

- Here's how we'll implement this idea.
 - Each node has a **parent pointer**.
 - Representatives' parent pointers are null.
 - Other nodes' parent pointers form chains leading to the representative.
- Although the original graph is undirected, parent pointers are directed.
- This data structure is called a **disjoint-set forest**.



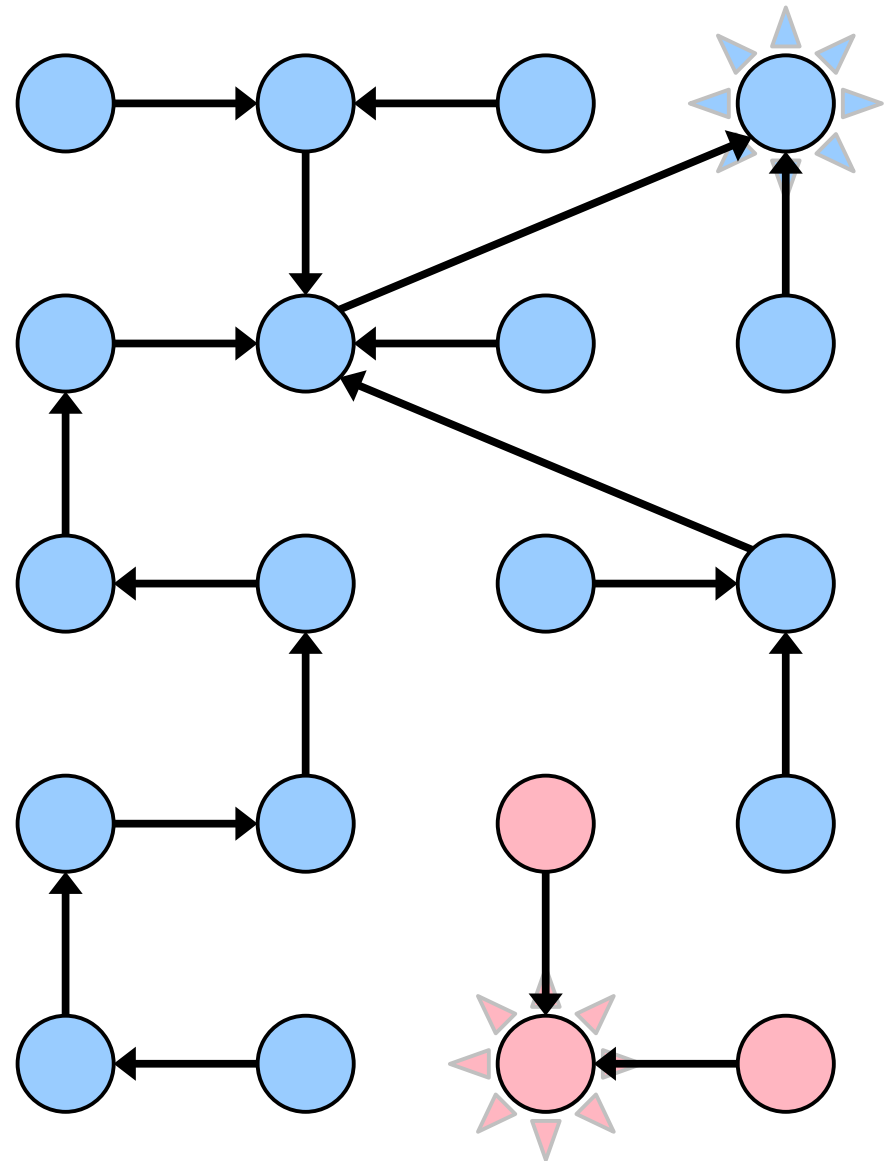
Representatives

- We'll support two operations.
- **find**(x) returns x 's representative. It works by following parent pointers until we hit the representative.
- **union**(x, y) merges the clusters containing x and y . It works by finding x and y 's representatives. If they aren't equal, it assigns one of those representatives the other as a parent.



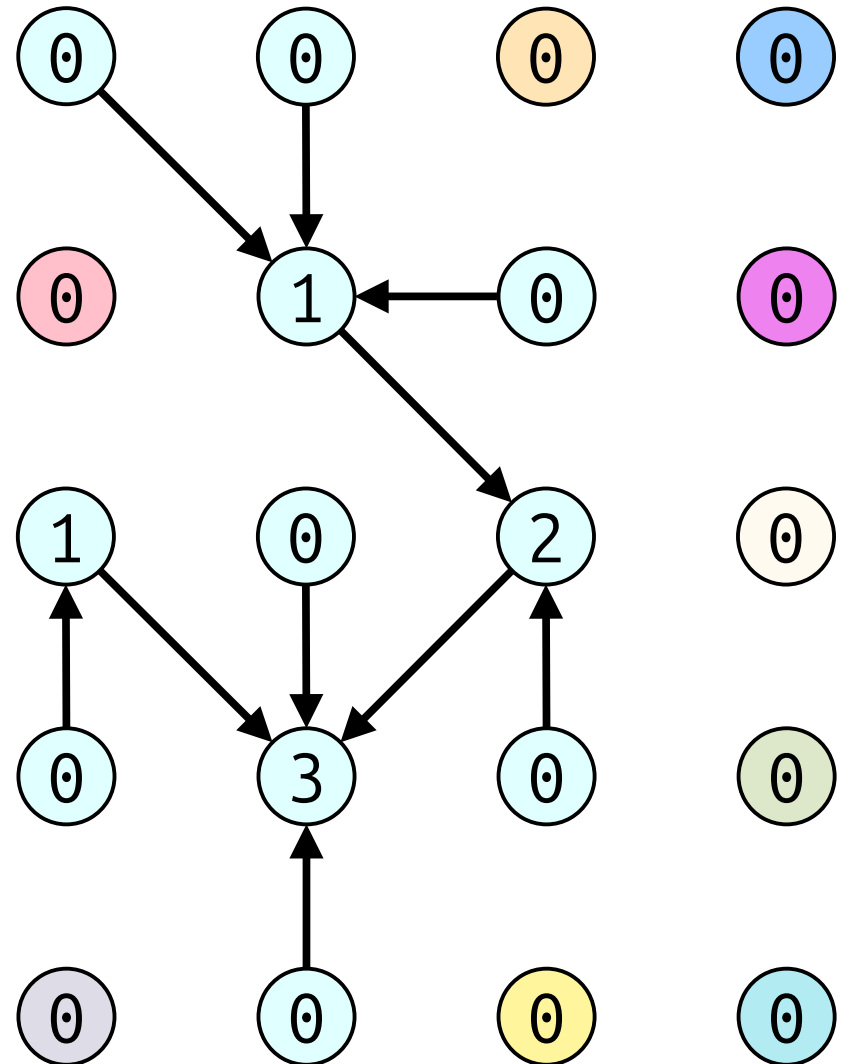
Representatives

- Unfortunately, this system can be very slow.
- If we aren't careful with how we link trees, the cost of a *find* or *union* can grow to $\Theta(n)$, where n is the number of nodes in the graph.
- ***Can we do better?***



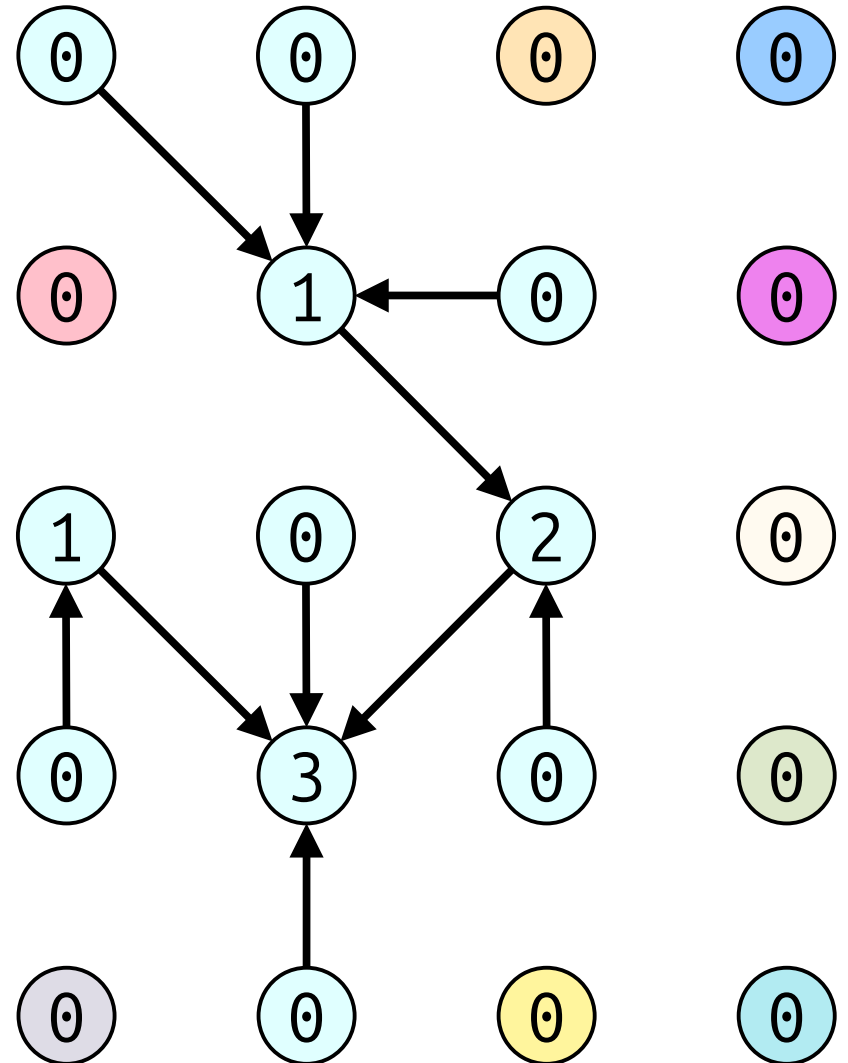
Union-By-Rank

- Assign each node a **rank**, initially 0.
- When linking two representatives x and y :
 - If one representative has a lower rank than the other, set its parent to the other.
 - Otherwise, arbitrarily set x 's parent to y , then increment y 's rank.
- This keeps the lengths of parent chains low.



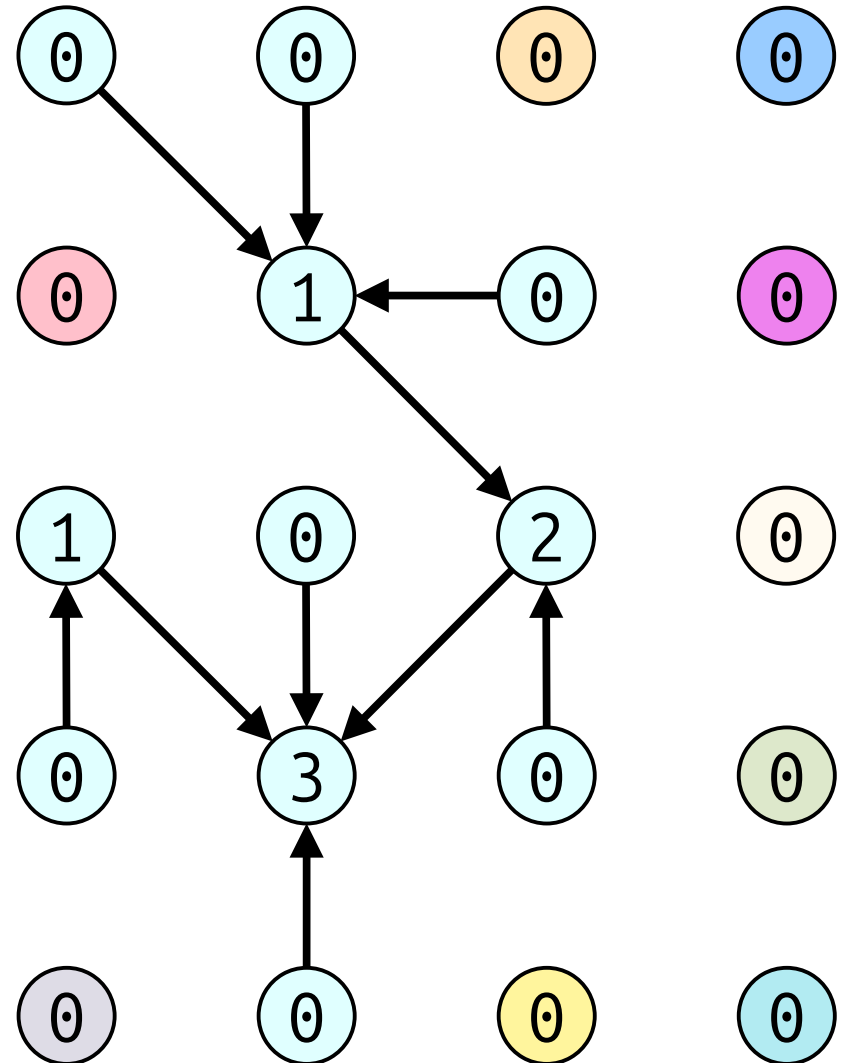
Union-By-Rank

- **Lemma:** A node of rank r has at least one child of rank 0, at least one child of rank 1, ..., and at least one child of rank $r - 1$.
- **Proof:** Induction!
 - Rank 0 nodes have no children.
 - A node v of rank $r + 1$, at the time its rank increased, was a tree of rank r that got another tree of rank r as a child.
 - By the IH v already had children of ranks 0, 1, 2, ..., $r - 1$. Now it also has a child of rank r . ■



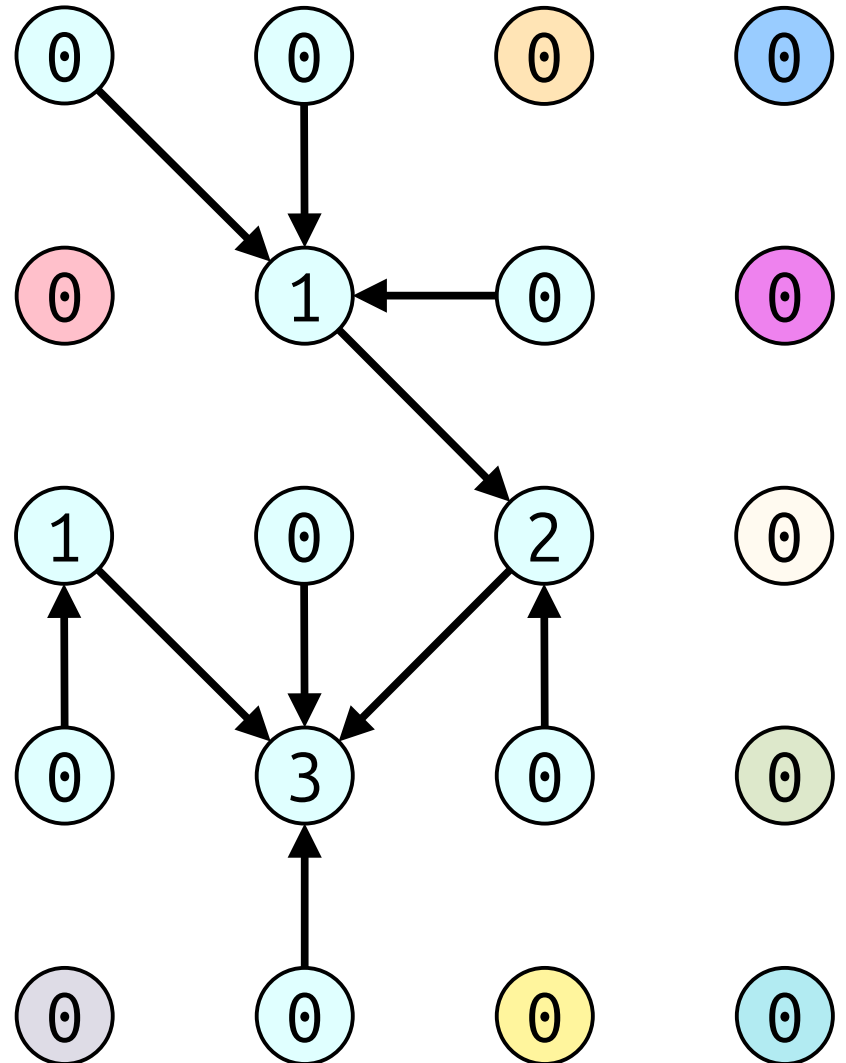
Union-By-Rank

- Our lemma tells us that the “simplest” tree whose root has rank r is a binomial tree of order r .
- A nice consequence of this is that all trees in a forest of n nodes have height $O(\log n)$, so each *union* and *find* takes time $O(\log n)$.
- ***Can we do better?***



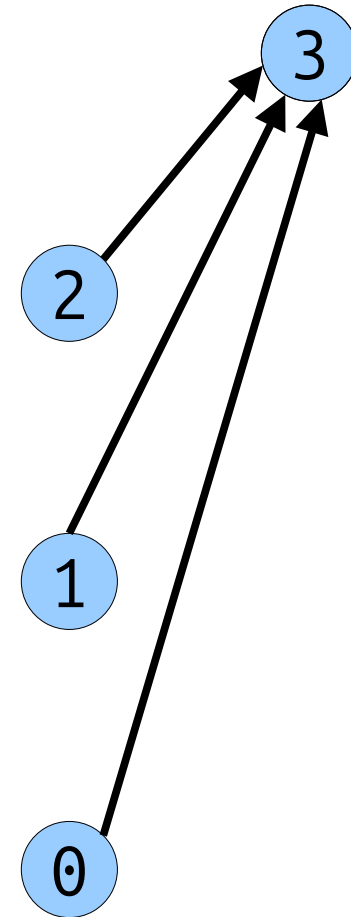
An Observation

- Suppose we call *find*(x) multiple times.
- Each time we do that, we may have to traverse a chain of $O(\log n)$ nodes to find its representative.
- Do we really need to scan things so many times?



Path Compression

- ***Path compression*** is an optimization on the ***find*** operation.
- After figuring out x 's representative, change the parent pointers of all of x 's ancestors to point directly to x 's representative.
- This makes it a lot faster to find representatives across multiple operations.



Path Compression

- The resulting code for our data structure is surprisingly simple:

```
Node* find(Node* source) {
    if (source->parent == nullptr) return source;

    /* Path compression: update parent before returning. */
    source->parent = find(source->parent);
    return source->parent;
}

void doUnion(Node* one, Node* two) {
    /* Find the representatives. */
    one = find(one);
    two = find(two);

    if (one == two) return;
    if (one->rank > two->rank) swap(one, two);

    /* Link and update ranks if needed. */
    one->parent = two;
    if (one->rank == two->rank) two->rank++;
}
```

- Now, all we have to do is analyze the runtime.

Time-Out for Announcements!

Please evaluate this course on Axess.

Your feedback makes a difference!

Final Exam Logistics

- Our final exam is on ***Saturday, June 6th*** from ***3:30PM - 6:30PM*** right here in STLC 114.
- Exam is cumulative. All topics from lectures and the problem sets are fair game.
- The exam is closed-book, closed-computer, and limited-note. You can bring one double-sided 8.5" × 11" sheet of notes with you.
- A practice exam (last year's final) is on the course website.
- Students with OAE accommodations: we think we've reached out to all of you to coordinate an alternate exam time. If not, ping us ASAP!

Final Exam Logistics

- We will be proctoring the final exam this weekend.
- What you need to know:
 - ***Bring your Stanford ID*** to the exam.
 - ***Sit in your assigned seat***. We'll have a seating chart on the projector when you arrive.
 - All personal belongings will need to be at the front of the exam room.
 - No electronic devices are permitted. That includes headphones, watches, etc.
- Thanks for your patience and understanding.

Back to CS166!

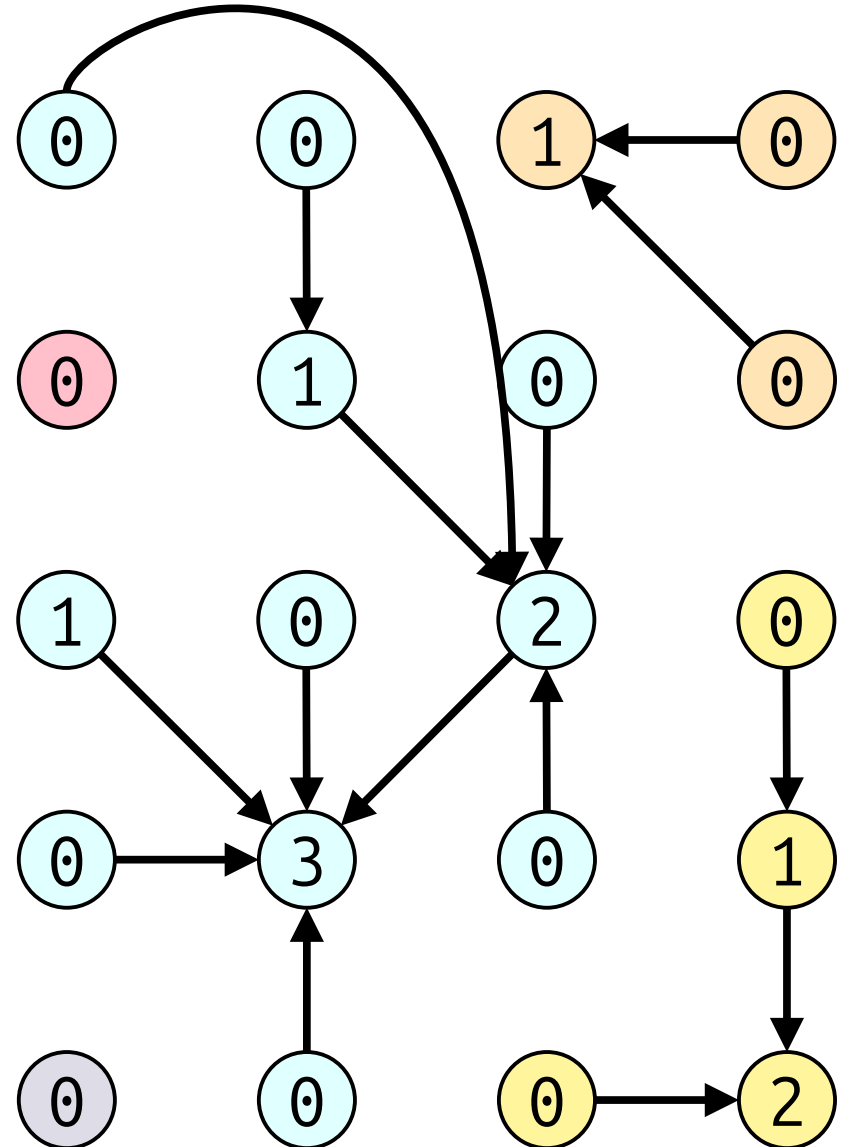
Analyzing Disjoint-Set Forests

History

- The analysis of union-by-rank plus path compression has a long history.
- For a while, its actual efficiency was an open problem!
- In 1979 Tarjan proved a tight upper bound on the runtime using a clever and nuanced analysis, and provided a matching lower bound.
- In 2003 Seidel and Sharir arrived at the same upper bound using a totally different technique.
- Both analyses require a careful analysis of the costs of the operations and ends in a very surprising result.
- The analysis I'll share comes from Seidel and Sharir and is based on [***this set of lecture slides***](#) from an algorithms course at Princeton.

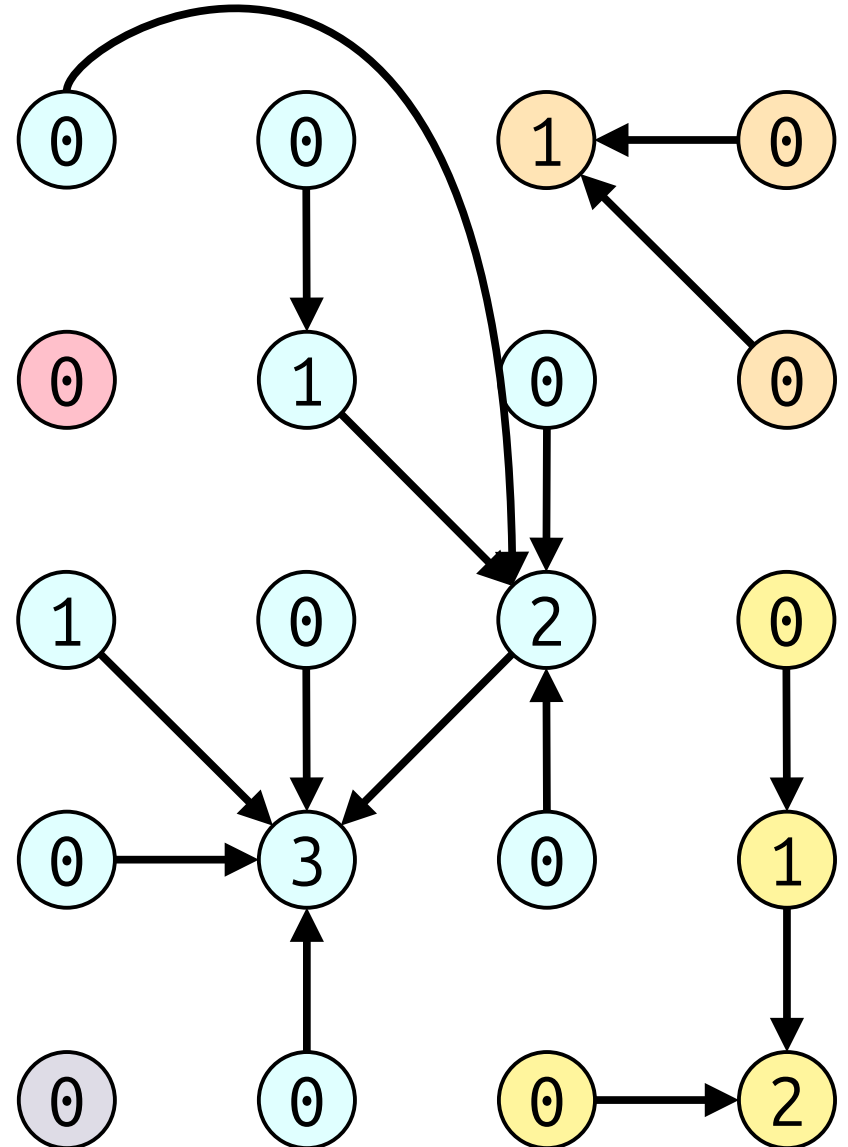
Some Simplifying Assumptions

- Imagine that we're given a forest formed by union-by-rank but *without* any compression.
- We then allow for path compression along arbitrary paths from some node x to some ancestor y .



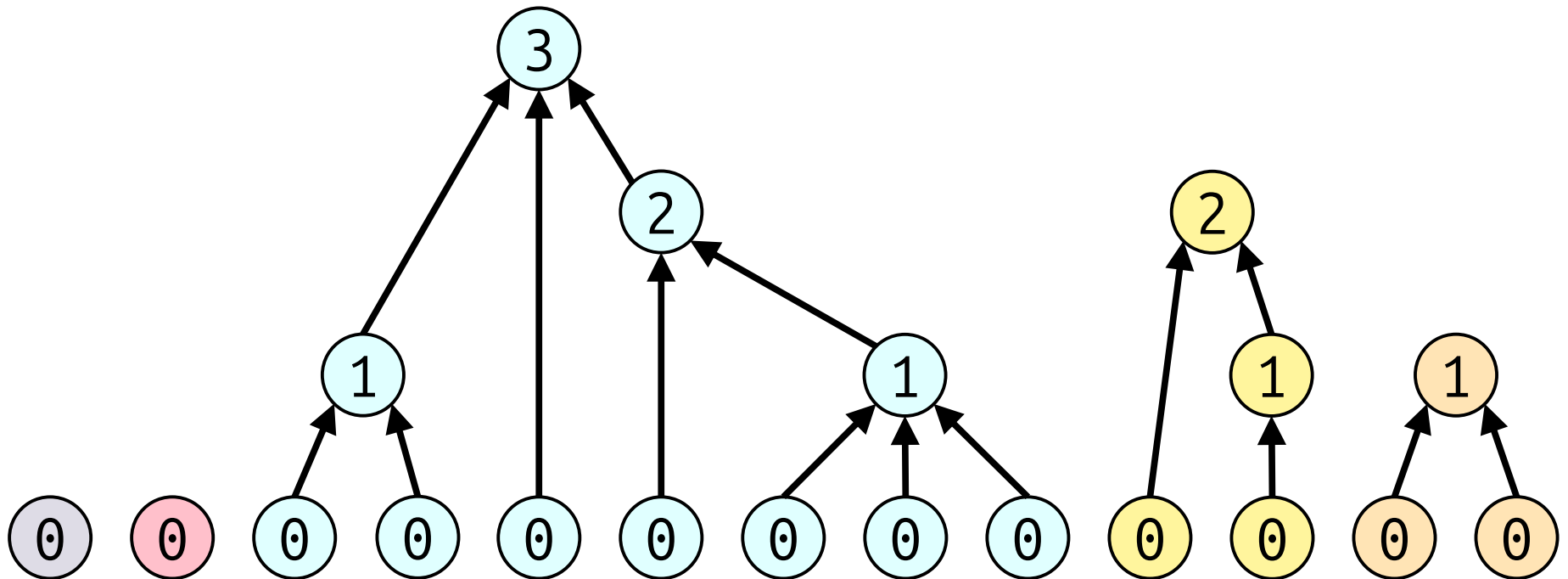
Some Simplifying Assumptions

- **Claim:** This simplification doesn't change the analysis.
- It's a good exercise to think through why this is.
- Having a forest that only gets compressed makes the analysis *much* simpler.



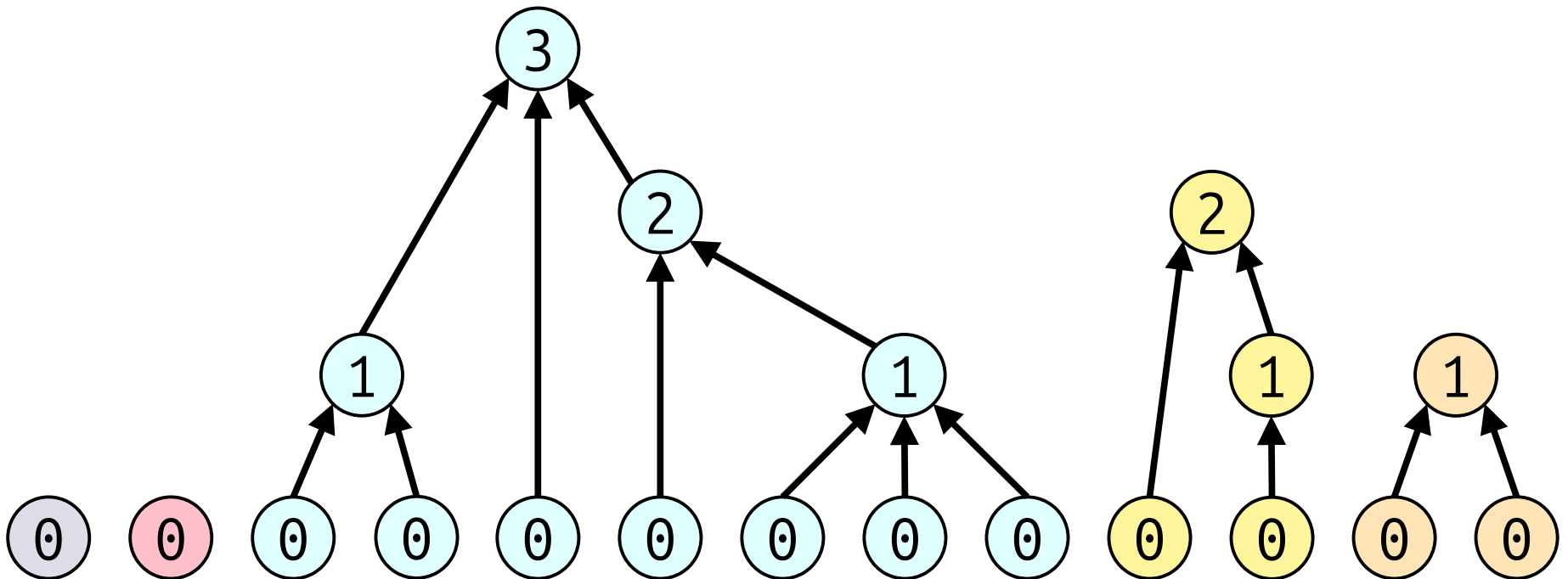
Our Analysis

- Some notation we'll use throughout this analysis:
 - Let n be the number of nodes in the disjoint-set forest.
 - Let m be the number of operations performed.
 - Let r be the maximum rank of any node in the forest. (We know $r = O(\log n)$, but could be lower.)
- In practice, we'll have $m = \Omega(n)$, and we'll assume this in our analysis.



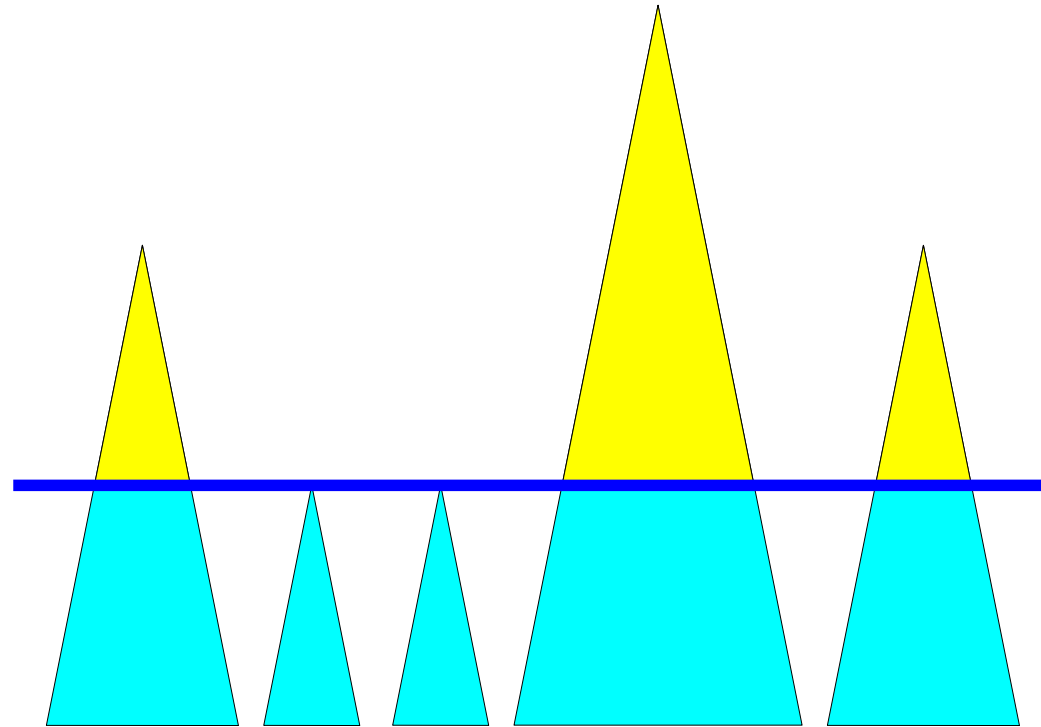
A Starting Analysis

- **Lemma:** The number of pointer changes is at most $m + n \cdot r / 2$.
- **Proof Sketch:** Consider nodes of zero and nonzero rank.
 - **Nodes of rank 0:** A node of rank 0 only has its parent change if it is the start node of a compress. There are m compresses, so these pointers change at most m times.
 - **Nodes of nonzero rank:** When a parent changes, the new parent's rank is bigger than the old parent's rank, so a node's parent's rank can increase at most r times. There are at most $n / 2$ nodes of nonzero rank. This gives a bound of $n \cdot r / 2$.



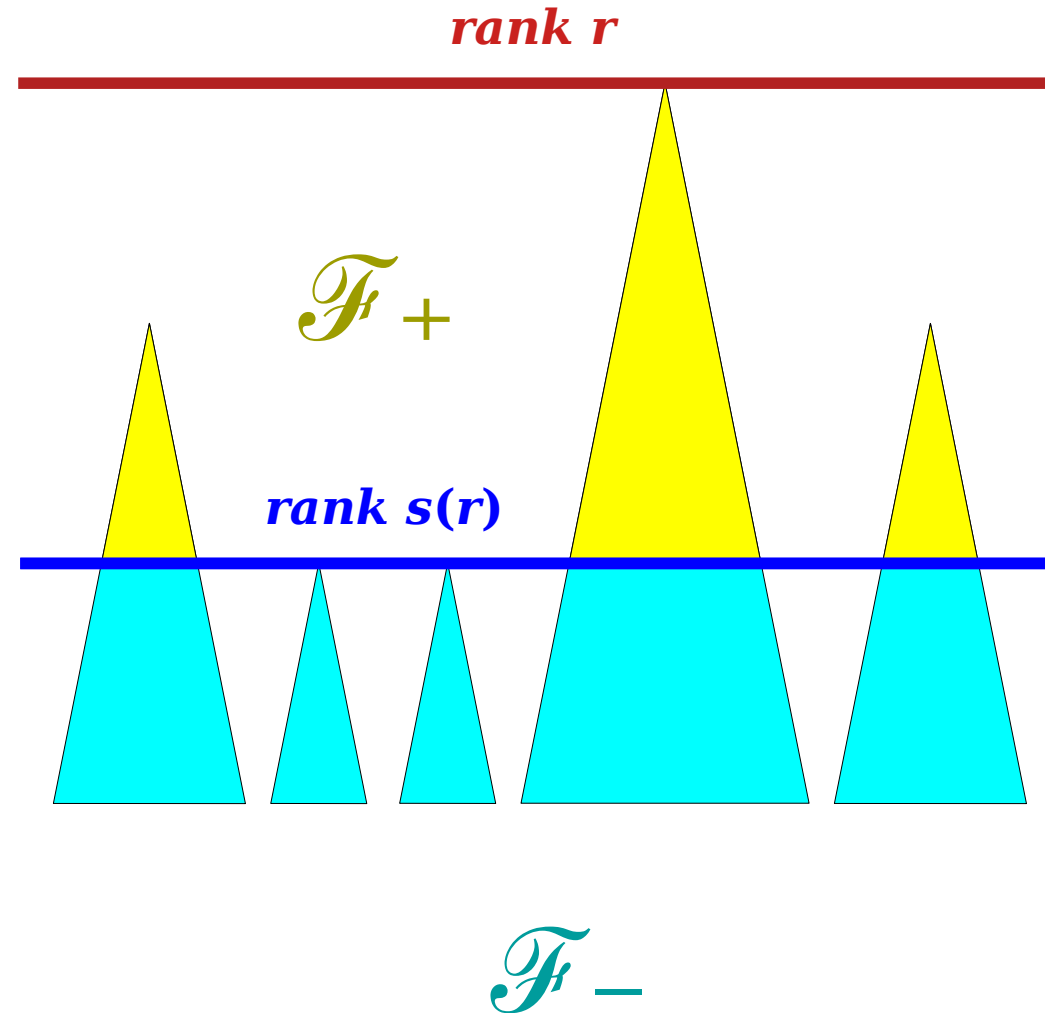
A Starting Analysis

- Our starting analysis is weak.
 - Compressing a path impacts other nodes not on that path.
 - Nodes with high starting rank can't have their parents change too many times.
- These effects work differently in different parts of the tree.
 - The first effect is more pronounced at the bottom of the forest.
 - The second effect is more pronounced at the top.
- **Idea:** Split the forest into a “top forest” and “bottom forest,” and analyze the costs in each forest separately.



Forest Slicing

- As before, let r be the maximum rank in \mathcal{F} .
- Suppose that, somehow, we pick a rank $s(r)$ as a separating rank.
- Then, split our forest \mathcal{F} into two forests:
 - \mathcal{F}_- consists of all nodes of rank $s(r)$ or below.
 - \mathcal{F}_+ consists of all nodes of rank above $s(r)$.
- **Goal:** Split the cost of compressions across \mathcal{F}_- and \mathcal{F}_+ .

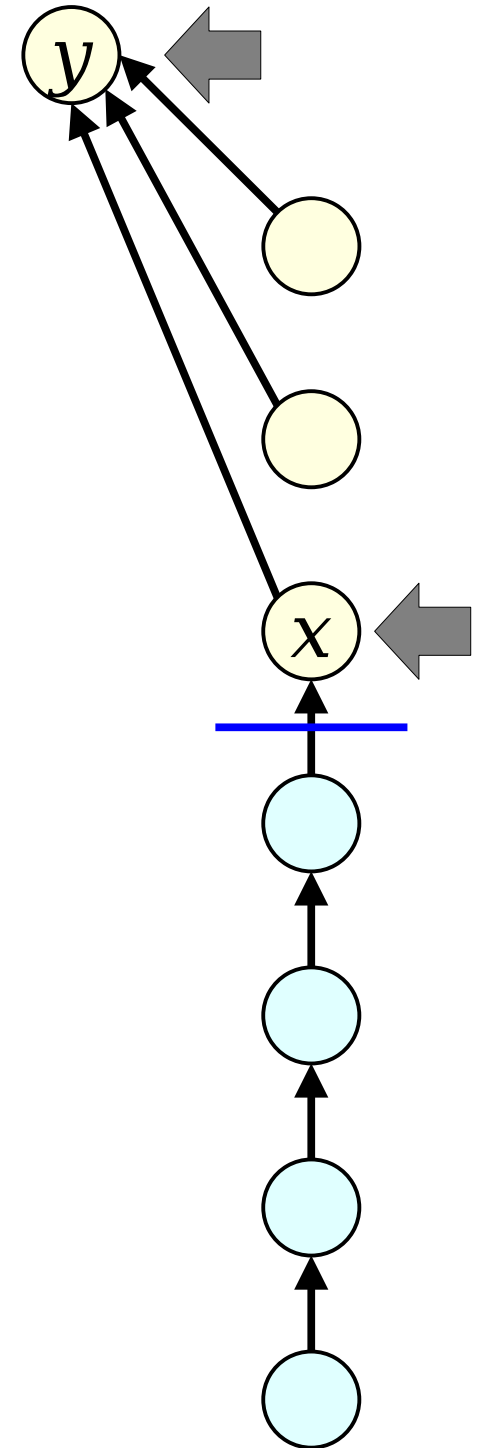


Some Terminology

- Let $C(m, n, r)$ be the maximum number of pointer changes that can be made if there are m compresses, there are n nodes, and the maximum rank is r .
- Using this notation, our earlier result is that
$$C(m, n, r) \leq m + n \cdot r / 2.$$
- **Question:** What is $C(m, n, 0)$? What's $C(m, n, 1)$?
- **Goal:** Split \mathcal{F} into \mathcal{F}_+ and \mathcal{F}_- , find a way to write a recurrence relation for $C(m, n, r)$, then solve the recurrence to get a tight bound on the cost of any series of **unions** and **finds**.

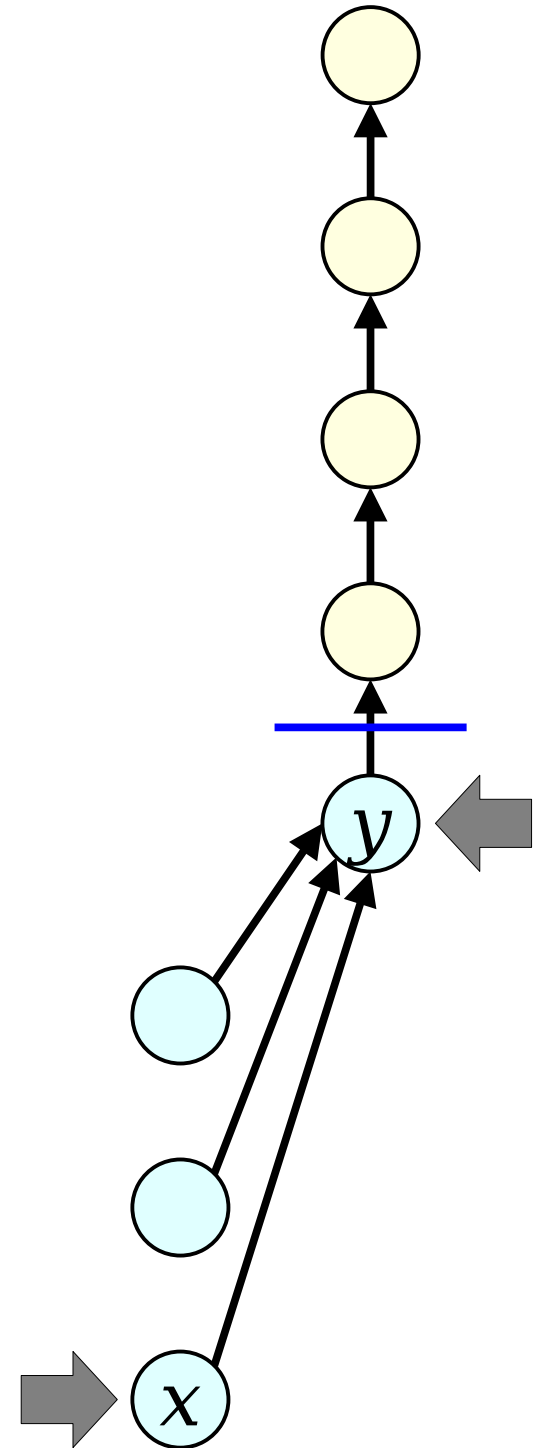
Forest Slicing

- Focus on any one compression from x to y . Let's see how it interacts with \mathcal{F}_- and \mathcal{F}_+ .
- **Case 1:** Both x and y are in \mathcal{F}_+ .
- Then the compression is purely within \mathcal{F}_+ .



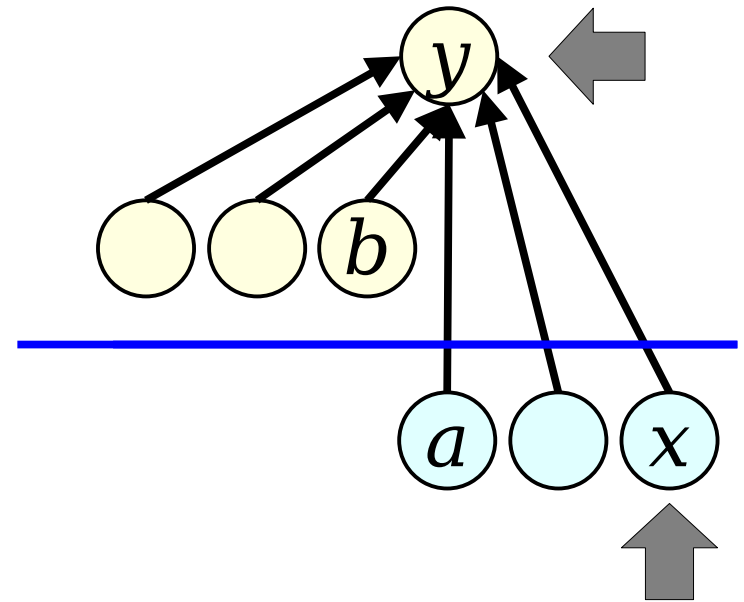
Forest Slicing

- **Case 2:** x and y are both in \mathcal{F}_- .
- Then the compression is purely within \mathcal{F}_- .



Forest Slicing

- **Case 3:** x is in \mathcal{F}_- and y is in \mathcal{F}_+ .
- We compress from b to y , purely in \mathcal{F}_+ .
- a , whose parent was already in \mathcal{F}_+ , gets a new parent in \mathcal{F}_+ .
- Every node from x (inclusive) and a (exclusive), whose parents were in \mathcal{F}_- , now has a parent in \mathcal{F}_+ .



Forest Slicing

Case 3: x is in \mathcal{F}_- and y is in \mathcal{F}_+ .

- We compress from b to y , purely in \mathcal{F}_+ .
- a , whose parent was already in \mathcal{F}_+ , gets a new parent in \mathcal{F}_+ .
- Every node from x (inclusive) and a (exclusive), whose parents were in \mathcal{F}_- , now has a parent in \mathcal{F}_+ .

Recursively handle this when processing \mathcal{F}_+ .

Happens once per compression from \mathcal{F}_- to \mathcal{F}_+ .

Happens once per non-root node in \mathcal{F}_- , counting across all compressions

Putting It All Together

- **Claim:** The cost of *all* the compressions performed in \mathcal{F} is bounded by the following:
 - The cost of some compressions purely in \mathcal{F}_- .
 - The cost of some compressions purely in \mathcal{F}_+ .
 - This includes compressions originally in \mathcal{F}_+ , plus the “tops” of compressions from \mathcal{F}_- to \mathcal{F}_+ .
 - The number of compresses from \mathcal{F}_- to \mathcal{F}_+ .
 - This accounts for changing the parents of nodes in \mathcal{F}_- whose parents are already in \mathcal{F}_+ .
 - The number of nodes in \mathcal{F}_- with parents in \mathcal{F}_- .
 - Each of these nodes may get a parent in \mathcal{F}_+ for the first time once.

$$C(m, n, r) \leq \dots$$

Cost of compressions purely in \mathcal{F}_+ .	$C(m_+, n_+, r)$
Cost of compressions purely in \mathcal{F}_- .	$C(m - m_+, n, s(r))$
The number of compresses from \mathcal{F}_- to \mathcal{F}_+	m_+
Number of nodes in \mathcal{F}_- with parents in \mathcal{F}_- .	$n - n_+ \cdot s(r)$

The Recurrence

- Putting it all together:

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + C(m_+, n_+, r) + m_+ + n - n_+ \cdot s(r).$$

- Now, all we need to do is solve this.



- Don't panic! This is indeed tricky, but it's not as bad as it looks.

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + \\ C(m_+, n_+, r) + \\ m_+ + n - n_+ \cdot s(r).$$

- **Recall:** $C(m, n, r) \leq m + n \cdot r / 2.$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + \\ m_+ + n_+ \cdot r / 2 + \\ m_+ + n - n_+ \cdot s(r).$$

- **Recall:** $C(m, n, r) \leq m + n \cdot r / 2$.

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + 2m_+ + n_+ \cdot r / 2 + n - n_+ \cdot s(r).$$

- **Recall:** $C(m, n, r) \leq m + n \cdot r / 2$.

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + 2m_+ + n + n_+ \cdot (r / 2 - s(r)).$$

- **Recall:** $C(m, n, r) \leq m + n \cdot r / 2.$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + 2m_+ + n + n_+ \cdot (r / 2 - s(r)).$$

- ***Clever Decision:*** Set $s(r) = r / 2$.

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, r / 2) + 2m_+ + n.$$

- ***Clever Decision:*** Set $s(r) = r / 2$.

$$C(m, n, r) \leq C(m - m_+, n, r/2) + 2m_+ + n$$

Starting with m ...

...take some of m away...

... and add it into the total.

$$C(m, n, r) \leq 2(\text{4 green circles}) + n$$

$$2(\text{2 pink circles}) + n$$

$$2(\text{3 light blue circles}) + n$$

$$2(\text{1 grey circle}) + n$$

...

$$\leq 2m + \underline{\hspace{2cm}}$$

$$C(m, n, r) \leq C(m - m_+, n, r/2) + 2m_+ + n$$

How many layers
can this recursion
have?

$$C(m, n, r) \leq 2(\text{4 green circles}) + n$$

$$2(\text{2 pink circles}) + n$$

$$2(\text{3 light blue circles}) + n$$

$$2(\text{1 grey circle}) + n$$

...

$$\leq 2m + n \lg r$$

Where We Are

- We've just proven that

$$C(m, n, r) \leq 2m + n \lg r.$$

- The maximum rank in an n -node forest is $r = O(\lg n)$.
- This gives a bound of **$O(m + n \log \log n)$** for any series of operations.
- That's a lot better than the $O(m \log n)$ we started with – and it's just due to better accounting, rather than a fundamental reenvisioning of the data structure.
- Is this a tight bound, or can we do better?

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + \\ C(m_+, n_+, r) + \\ m_+ + n - n_+ \cdot s(r).$$

- ~~**Recall:** $C(m, n, r) \leq m + n \cdot r / 2.$~~
- **Recall:** $C(m, n, r) \leq 2m + n \lg r.$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + 2m_+ + n_+ \cdot \lg r + m_+ + n - n_+ \cdot s(r).$$

- ~~**Recall:** $C(m, n, r) \leq m + n \cdot r / 2.$~~
- **Recall:** $C(m, n, r) \leq 2m + n \lg r.$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + 3m_+ + n_+ \cdot \lg r + n - n_+ \cdot s(r).$$

- ~~**Recall:** $C(m, n, r) \leq m + n \cdot r / 2.$~~
- **Recall:** $C(m, n, r) \leq 2m + n \lg r.$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + 3m_+ + n + n_+ \cdot (\lg r - s(r)).$$

- ~~**Recall:** $C(m, n, r) \leq m + n \cdot r / 2.$~~
- **Recall:** $C(m, n, r) \leq 2m + n \lg r.$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, \lg r) + 3m_+ + n$$

- ***Clever Decision:*** Set $s(r) = \lg r$.

$$C(m, n, r) \leq C(m - m_+, n, \lg r) + 3m_+ + n$$

Starting with m ...

...take some of m away...

... and add it into the total.

$$C(m, n, r) \leq 3(\text{4 green circles}) + n$$

$$3(\text{2 pink circles}) + n$$

$$3(\text{3 light blue circles}) + n$$

$$3(\text{1 grey circle}) + n$$

...

$$\leq 3m + \underline{\hspace{2cm}}$$

$$C(m, n, r) \leq C(m - m_+, n, \lg r) + 3m_+ + n$$

How many layers
can this recursion
have?

$$C(m, n, r) \leq 3(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array}) + n$$

$$3(\bullet \bullet \bullet) + n$$

$$3(\begin{array}{c} \bullet \\ \bullet \end{array}) + n$$

$$3(\bullet) + n$$

...

$$\leq 3m + n \lg^* r$$

Where We Are

- We've just proven that

$$C(m, n, r) \leq 3m + n \lg^* r.$$

- Since $r = O(\log n)$ this gives a bound of $O(m + n \log^* n)$ for any series of operations.
- That's a *substantial* improvement over our previous bound – and all we did was feed the analysis back into itself!
- *Can we do better?*

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + \\ C(m_+, n_+, r) + \\ m_+ + n - n_+ \cdot s(r).$$

- ~~**Recall:** $C(m, n, r) \leq m + n \cdot r / 2.$~~
- ~~**Recall:** $C(m, n, r) \leq 2m + n \lg r.$~~
- **Recall:** $C(m, n, r) \leq 3m + n \lg^* r.$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + 3m_+ + n_+ \cdot \lg^* r + m_+ + n - n_+ \cdot s(r).$$

- ~~**Recall:** $C(m, n, r) \leq m + n \cdot r / 2.$~~
- ~~**Recall:** $C(m, n, r) \leq 2m + n \lg r.$~~
- **Recall:** $C(m, n, r) \leq 3m + n \lg^* r.$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + 4m_+ + n_+ \cdot \lg^* r + n - n_+ \cdot s(r).$$

- ~~**Recall:** $C(m, n, r) \leq m + n \cdot r / 2.$~~
- ~~**Recall:** $C(m, n, r) \leq 2m + n \lg r.$~~
- **Recall:** $C(m, n, r) \leq 3m + n \lg^* r.$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + 4m_+ + n + n_+ \cdot (\lg^* r - s(r)).$$

- ~~**Recall:** $C(m, n, r) \leq m + n \cdot r / 2.$~~
- ~~**Recall:** $C(m, n, r) \leq 2m + n \lg r.$~~
- **Recall:** $C(m, n, r) \leq 3m + n \lg^* r.$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, \lg^* r) + 4m_+ + n$$

- ***Clever Decision:*** Set $s(r) = \lg^* r$.

$$C(m, n, r) \leq C(m - m_+, n, \lg^* r) + 4m_+ + n$$

How many layers
can this recursion
have?

$$C(m, n, r) \leq 4(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array}) + n$$

$$4(\bullet \bullet \bullet) + n$$

$$4(\begin{array}{c} \bullet \\ \bullet \end{array}) + n$$

$$4(\bullet) + n$$

...

$$\leq 4m + n \lg^{**} r$$

Where We Are

- We've just proven that

$$C(m, n, r) \leq 4m + n \lg^{**} r.$$

- This gives a bound of $O(m + n \log^{**} n)$ for any series of operations.
- The \log^{**} function grows *incredibly* slowly – this is a substantial improvement on our previous bound!
 - Great question to ponder – what is the smallest n for which $\log^{**} n = 10$?
- What happens if we feed this back into itself?

If we start with this bound on $C(m, n, r)$ then we get this bound on $C(m, n, r)$:
$m + n \cdot r / 2$	$2m + n \lg r$
$2m + n \lg r$	$3m + n \lg^* r$
$3m + n \lg^* r$	$4m + n \lg^{**} r$
$4m + n \lg^{**} r$	$5m + n \lg^{***} r$
$km + n \lg^{*(k-2)} r$	$(k+1)m + n \lg^{*(k-1)} r$

Interpreting This Result

- We now have a family of bounds on the cost of doing m operations on an n -node forest with max rank r :

$$2m + n \lg r$$

$$3m + n \lg^* r$$

$$4m + n \lg^{**} r$$

$$5m + n \lg^{***} r$$

- These all have the form

$$(k + 2)m + n \lg^{*(k)} r.$$

- There's some point at which making k larger makes that first term larger without decreasing the second term.
- What is it?

The Ackermann Inverse

- The **Ackermann inverse function**, denoted $\alpha(z)$, is defined as follows:

$$\alpha(z) = \min\{ k \in \mathbb{N} \mid \log^{*(k)} z \leq k \}$$

- Intuitively, this counts how many times you have to put stars on $\log^{***...***} z$ before you hit the number of stars in the tower.
- This function grows more slowly than anything in the iterated logarithm family – and that should give you a sense of just how slowly this function grows!
- **Worthwhile Activity:** find the smallest natural numbers where α produces 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10.

The Ackermann Inverse

- We define $\alpha(z)$ as

$$\alpha(z) = \min\{ k \in \mathbb{N} \mid \log^{*(k)} z \leq k \}$$

- We have a bound of

$$(k + 2)m + n \log^{*(k)} r.$$

- Picking $k = \alpha(r) = \alpha(\log n)$ gives this bound:

$$O((m + n)\alpha(\log n)).$$

- This is *essentially* “ $O(m + n)$,” because that α term is a constant for any input that could ever be fed in with the resources we know about in the universe. But technically speaking it’s superlinear. 😊

A Tighter Analysis

- By being a bit more clever with the analysis, we can tighten the bound as follows.
- Define $\alpha(m, n)$ as
$$\alpha(m, n) = \min\{ k \in \mathbb{N} \mid \log^{*(k)} m/n \leq \log n \}.$$
- Then the cost of m operations on an n -element forest can be shown to be $O(m\alpha(m, n))$, a slight improvement over what we just did here.

Major Ideas for Today

- Iterated functions generalize the idea of “how many times can you divide by two before you run out of things?”
- Iterated logarithms are a family of very slowly-growing functions, each of which grows more slowly than the previous one.
- The Ackermann inverse function grows slower than any number of iterated logarithms and essentially count what level of iteration is needed to clear a number.

That's a wrap!

A Huge Word of Thanks



Appendix: Deriving the Recurrence

$$C(m, n, r) \leq \dots$$

Cost of compressions
purely in \mathcal{F}_+ .

$$C(m_+, n_+, r)$$

Cost of compressions
purely in \mathcal{F}_- .

The “tops” of all compressions
running from \mathcal{F}_- to \mathcal{F}_+ are
handled in this bunch.

The number of
compresses from
 \mathcal{F}_- to \mathcal{F}_+

Let m_+ be the number of
compressions charged to \mathcal{F}_+ ,
including both compressions
purely within \mathcal{F}_+ and the “tops”
of compressions crossing
from \mathcal{F}_- to \mathcal{F}_+ .

Number of nodes in
 \mathcal{F}_- with parents in \mathcal{F}_- .

$$C(m, n, r) \leq \dots$$

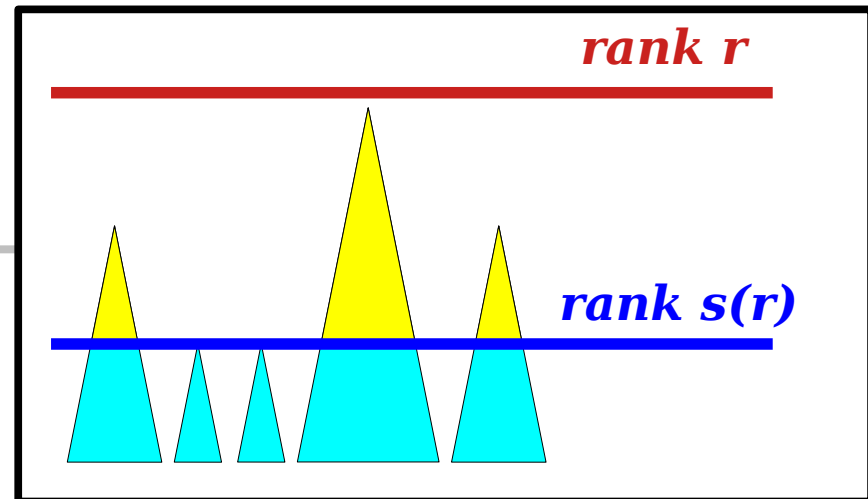
Cost of compressions
purely in \mathcal{F}_+ .

$$C(m_+, n_+, r)$$

Cost of compressions
purely in \mathcal{F}_- .

$$C(m - m_+, n, s(r))$$

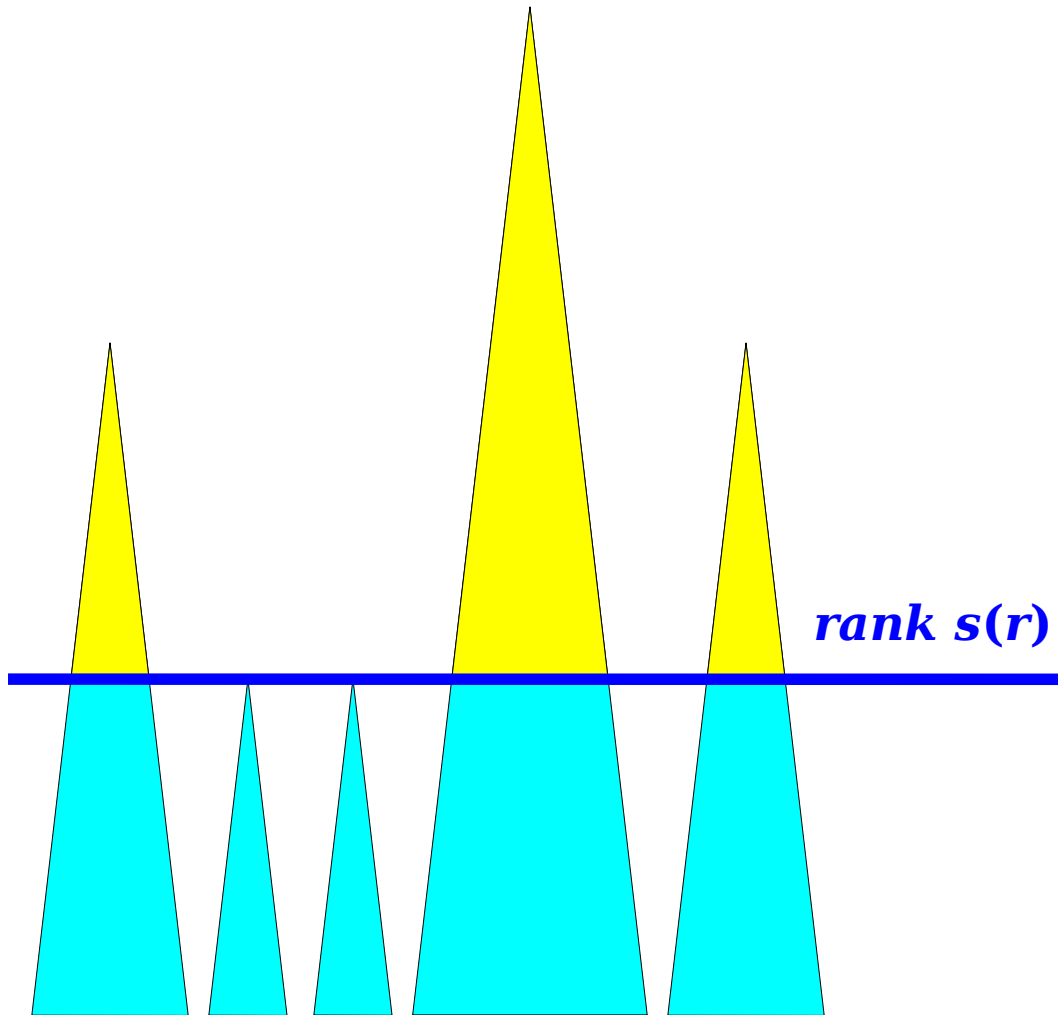
The number of
compresses from
 \mathcal{F}_- to \mathcal{F}_+



Number of nodes in
 \mathcal{F}_- with parents in \mathcal{F}_- .

$$C(m, n, r) \leq \dots$$

Cost of compressions purely in \mathcal{F}_+ .	$C(m_+, n_+, r)$
Cost of compressions purely in \mathcal{F}_- .	$C(m - m_+, n, s(r))$
The number of compresses from \mathcal{F}_- to \mathcal{F}_+	m_+
Number of nodes in \mathcal{F}_- with parents in \mathcal{F}_- .	(Since this includes all compresses from \mathcal{F}_- to \mathcal{F}_+).



Every node in \mathcal{F}_+ has rank $s(r) + 1$ or greater.

Each rank- k node has children of ranks $0, 1, 2, \dots, k - 1$.

So every node in \mathcal{F}_+ has at least $s(r) + 1$ children in \mathcal{F}_- .

There are n total nodes, and n_+ of them are in \mathcal{F}_+ .

Nodes in \mathcal{F}_- : $n - n_+$.

Nodes in \mathcal{F}_- whose parents are in \mathcal{F}_+ : $n_+ \cdot (s(r) + 1)$

Nodes in \mathcal{F}_- with parents in \mathcal{F}_- : $n - n_+ \cdot (s(r) + 2)$.

Number of nodes in \mathcal{F}_- with parents in \mathcal{F}_- .

$$n - n_+ \cdot (s(r) + 2)$$

$$C(m, n, r) \leq \dots$$

Cost of compressions purely in \mathcal{F}_+ .	$C(m_+, n_+, r)$
Cost of compressions purely in \mathcal{F}_- .	$C(m - m_+, n, s(r))$
The number of compresses from \mathcal{F}_- to \mathcal{F}_+	m_+
Number of nodes in \mathcal{F}_- with parents in \mathcal{F}_- .	$n - n_+ \cdot s(r)$