

Disjoint-Set Forests

Outline for Today

- ***Iterated Functions***
 - Making an implicit idea explicit.
- ***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.

Iterated Functions

Iterated Functions

- Recursive functions work by converting a problem of size n into one or more subproblems of a smaller size.
- How much smaller those subproblems are indicates how many levels of recursion we'll have.
 - $n \rightarrow n - 1$: $\Theta(n)$ levels.
 - $n \rightarrow n/2$: $\Theta(\log n)$ levels

Iterated Functions

- Let f be a function. The ***iterated function of f*** , denoted f^\star is a function defined as follows:

$$f^\star(n) = \begin{cases} 0 & \text{if } f(n) \leq 1 \\ 1 + f^\star(f(n)) & \text{otherwise} \end{cases}$$

- Intuitively, $f^\star(n)$ is (roughly) the number of times you need to apply f to n to reduce it to a sufficiently small constant.
 - If $f(n) \leq 1$, no steps are needed.
 - Otherwise, you need one step to turn n into $f(n)$, then $f^\star(f(n))$ more steps from there.

Iterated Functions

	$f^*(n)$	As seen in...
$f(n) = n - 1$	$\Theta(n)$	Linear search
$f(n) = n/2$	$\Theta(\log n)$	Binary search
$f(n) = n^{1/2}$	$\Theta(\log \log n)$	Rabin's closest pair of points algorithm
$f(n) = \log n$	$\Theta(\log^* n)$	Succinct binary rank

Iterated Logarithms

- **Intuition:** The log function is incredibly effective at shrinking down large quantities.
 - Number of protons in the known universe: $\approx 2^{240}$.
 - $\log^{(0)} 2^{240} = 1,766,847, [\dots 57 \text{ digits } \dots], 292,619,776$
 - $\log^{(1)} 2^{240} = 240$
 - $\log^{(2)} 2^{240} \approx 7.91$
 - $\log^{(3)} 2^{240} \approx 2.98$
 - $\log^{(4)} 2^{240} \approx 1.58$
 - $\log^{(5)} 2^{240} \approx 0.66$
- So $\log^* 2^{240} = 4$.
- The **iterated logarithm of n** , denoted **$\log^* n$** , grows *much* more slowly than $\log n$.

Intuiting $\log^* n$

- What is $\log^* n$ for the value of n shown below?

$$n = 2^{2^{2^{2^{2^{2^{2^{2^{2^{2^{2^{2^{2^{2^2}}}}}}}}}}}}}}$$

- **Answer:** $\log^* n = 16$.
- The value of n is inconceivably large, and yet $\log^* n$ is small enough to hold in your hand. The \log^* function grows very, very slowly!

Iterated Iterates

- What is the value of this expression?

$$\log^{**} \left(2^{2^{2^{2^{2^{2^{2^{2^{2^{2^2}}}}}}}}}} \right)$$

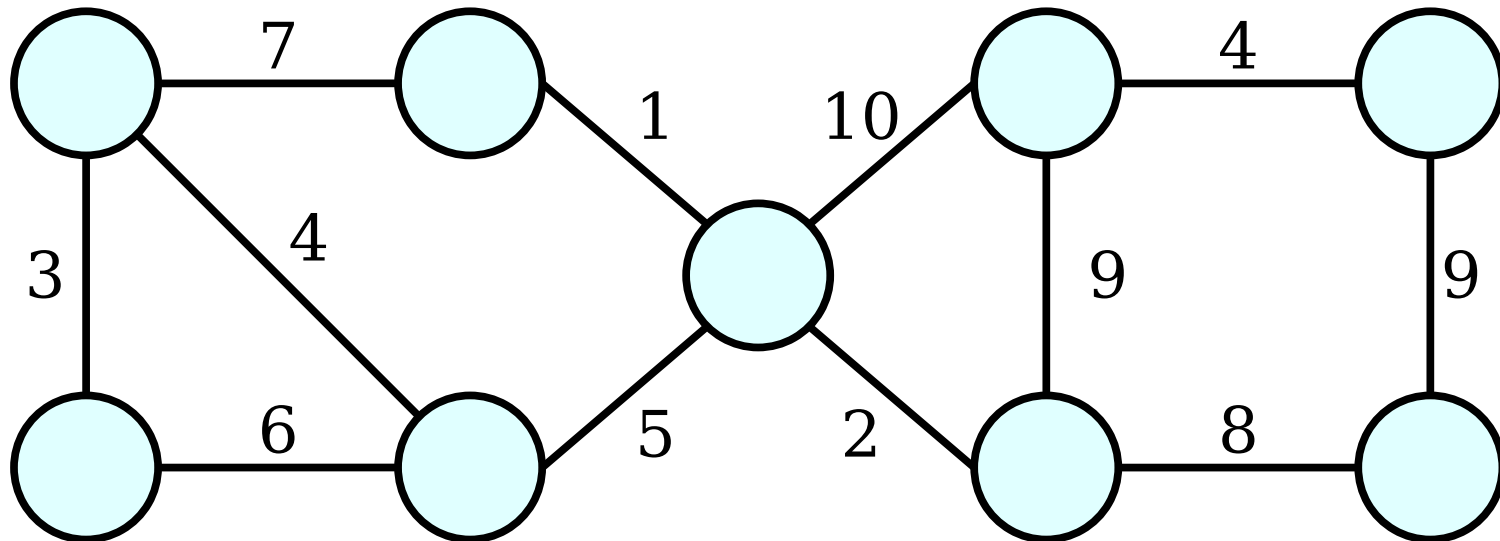
- After taking one \log^* , we're left with $16 = 2^{2^2}$.
- After taking another \log^* , we're left with 2.
- After taking another \log^* , we're left with 0.
- So the above expression evaluates to **2**.
- How big of an input do we need to get $\log^{**} n$ to be 3?

Just how slowly can a function grow?

Incremental Dynamic Connectivity

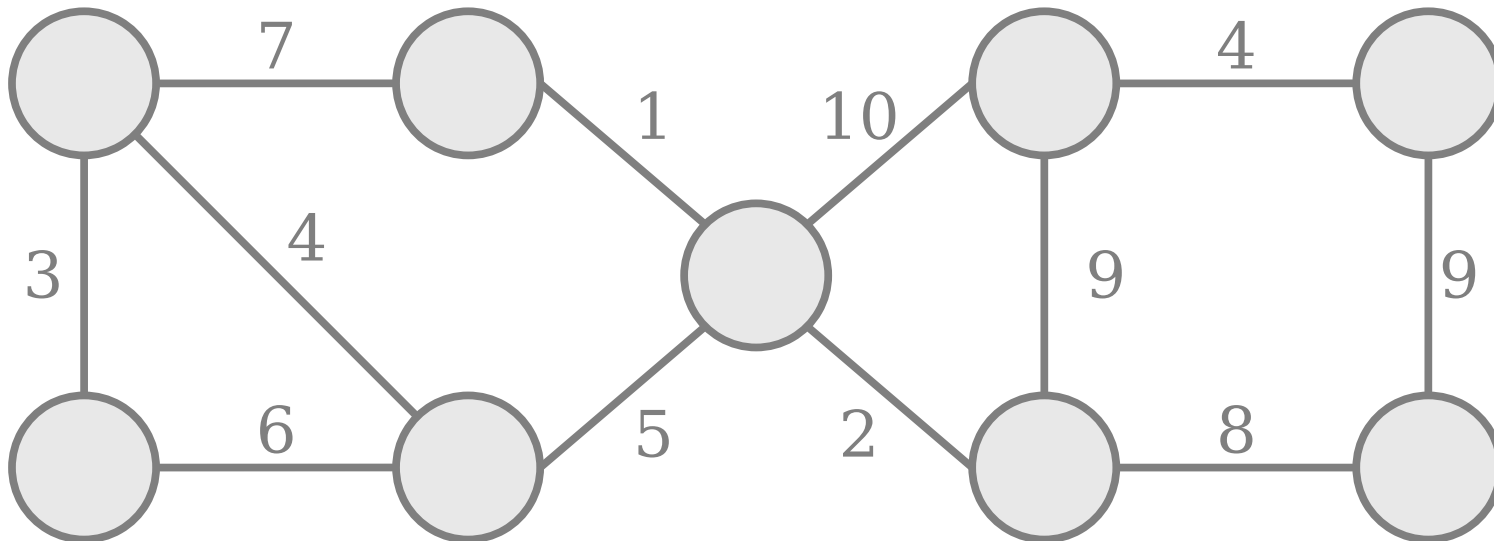
Kruskal's Algorithm

- ***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.



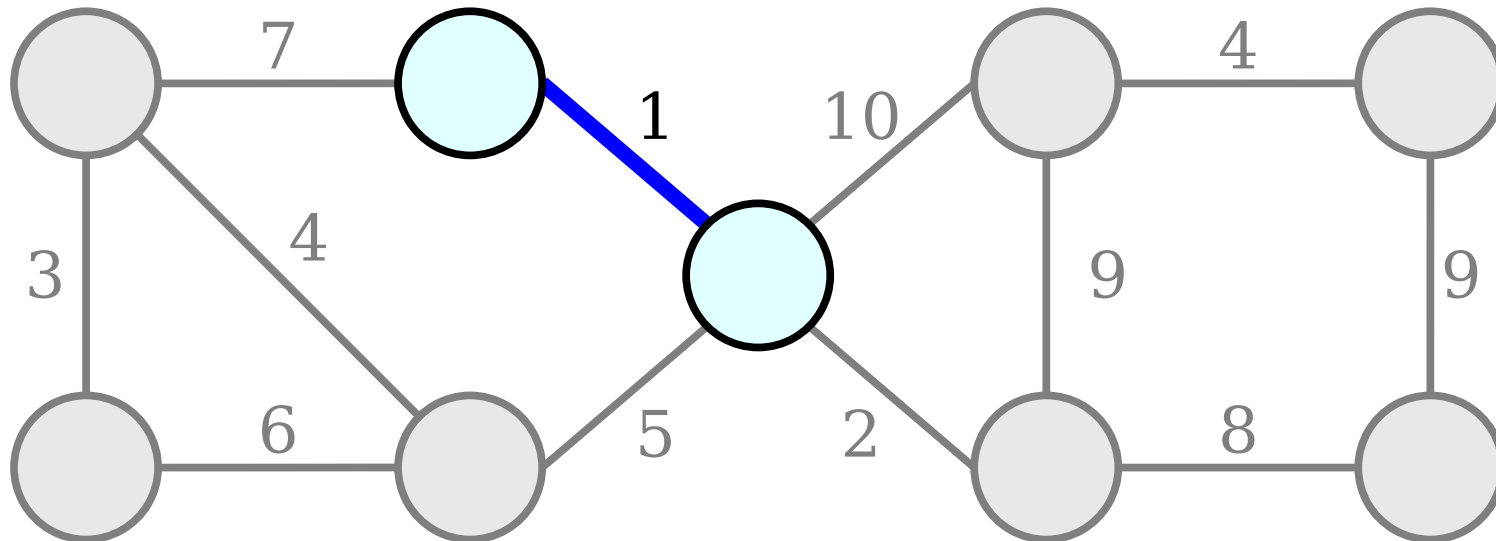
Kruskal's Algorithm

- ***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.



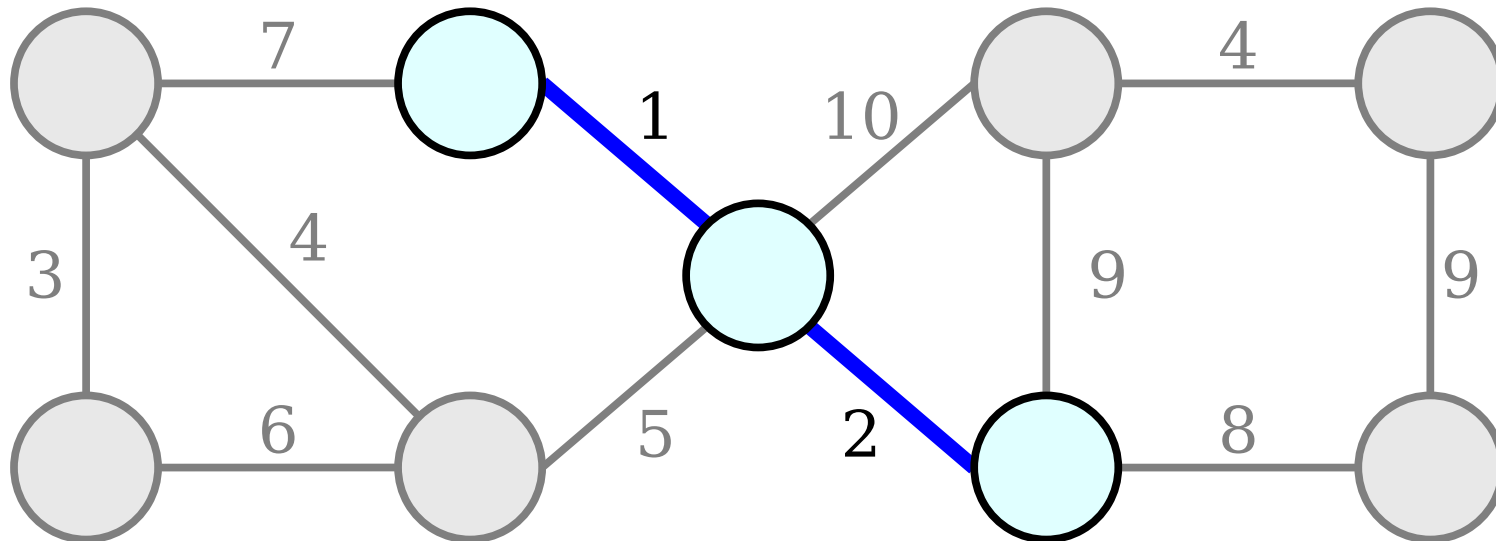
Kruskal's Algorithm

- ***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.



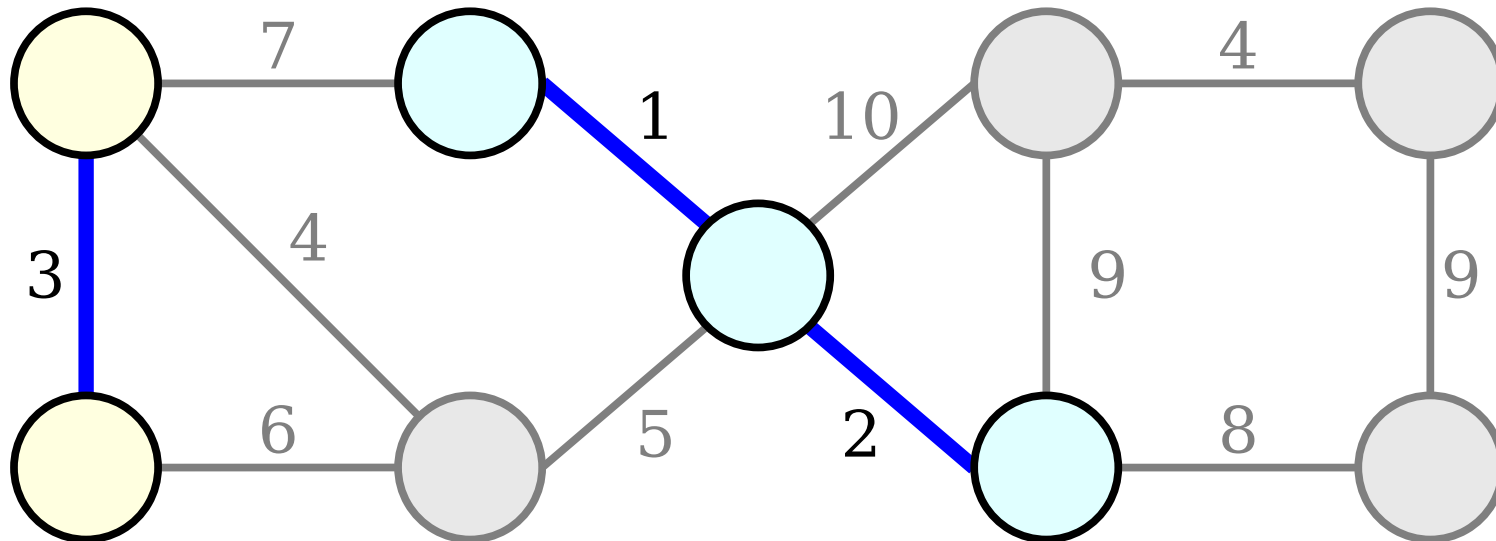
Kruskal's Algorithm

- ***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.



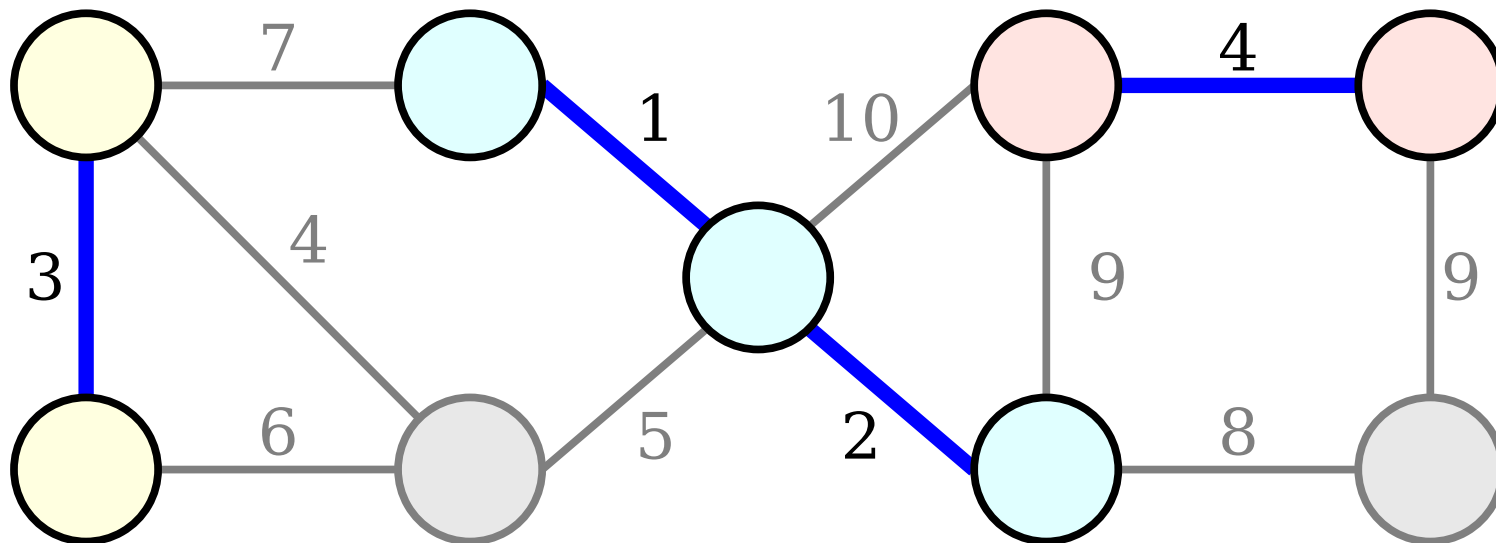
Kruskal's Algorithm

- ***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.



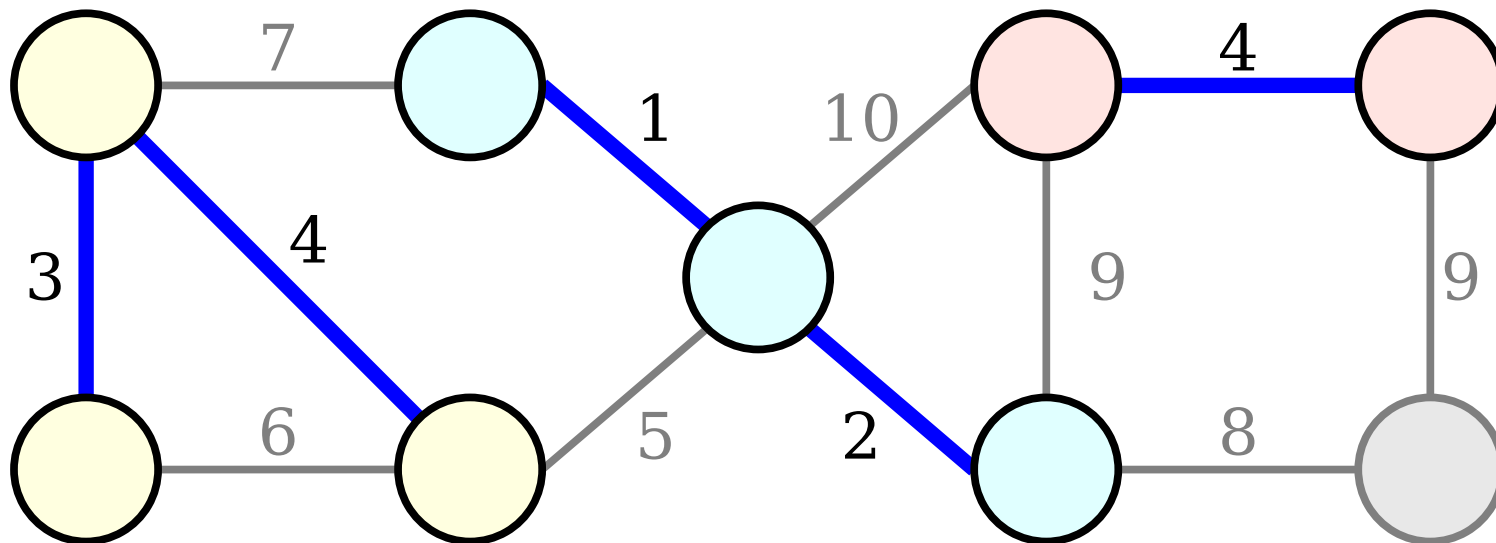
Kruskal's Algorithm

- ***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.



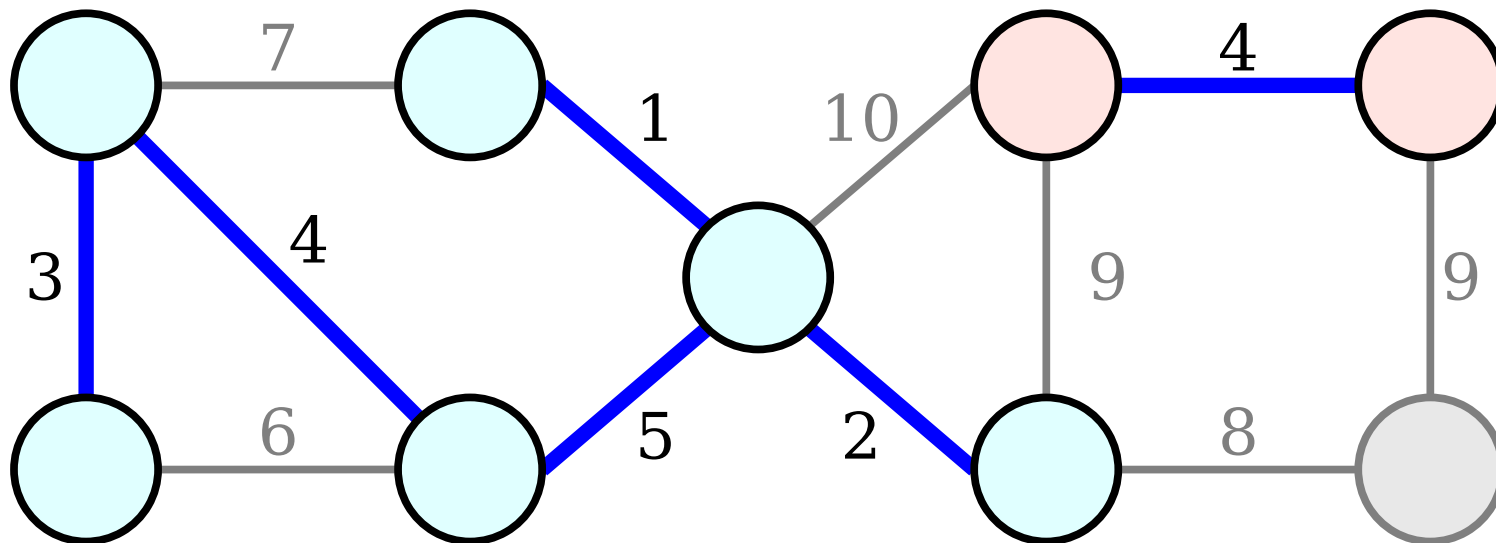
Kruskal's Algorithm

- ***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.



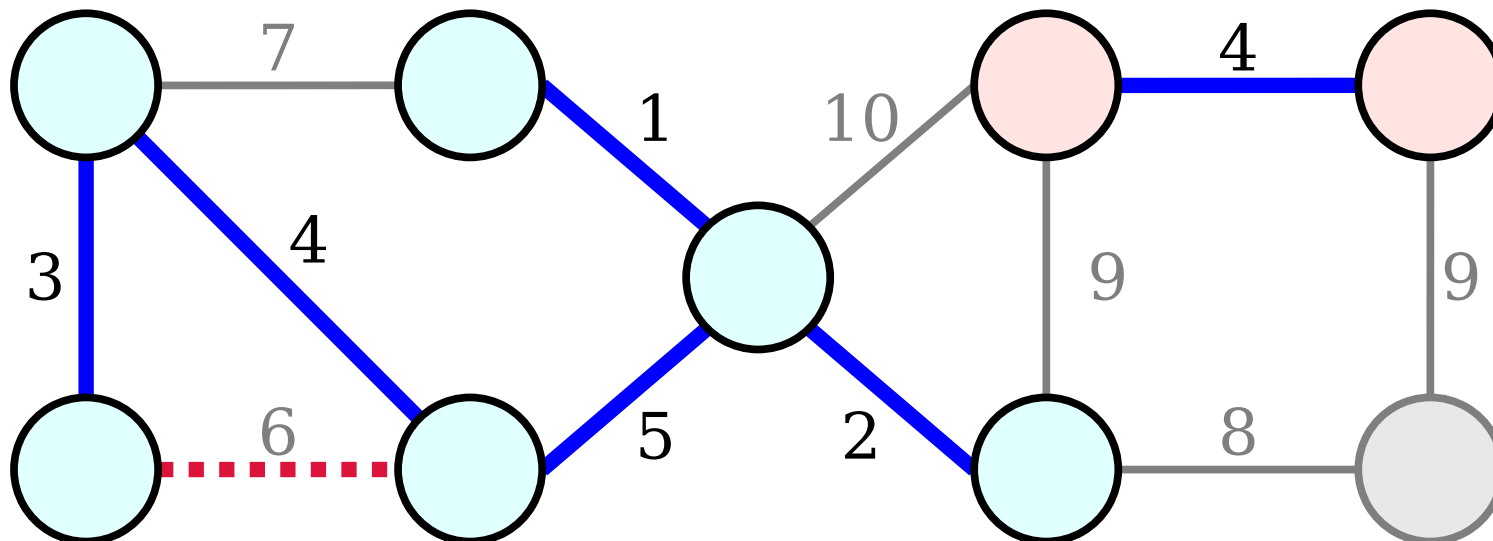
Kruskal's Algorithm

- ***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.



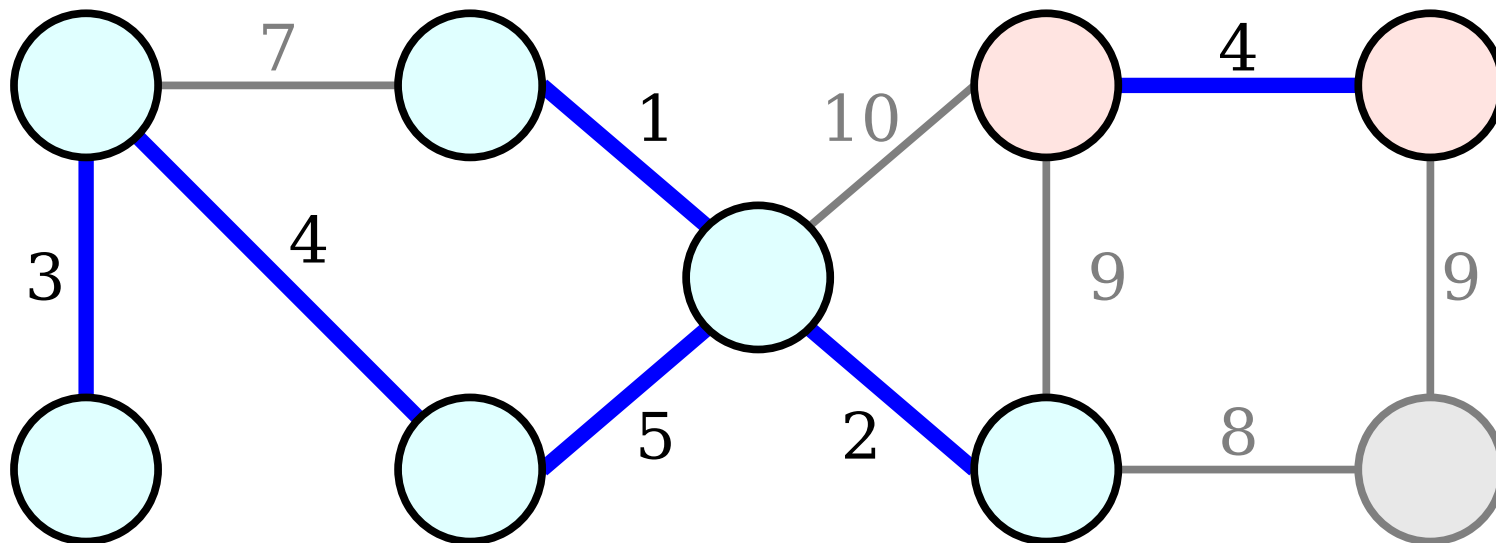
Kruskal's Algorithm

- ***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.



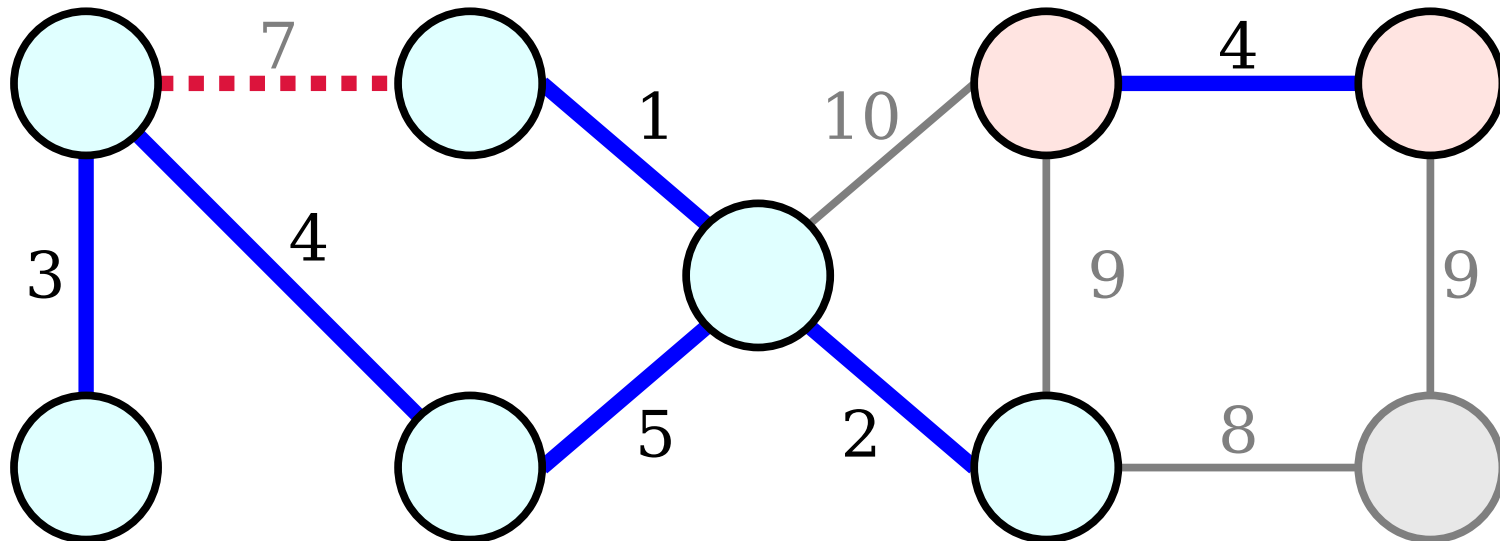
Kruskal's Algorithm

- ***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.



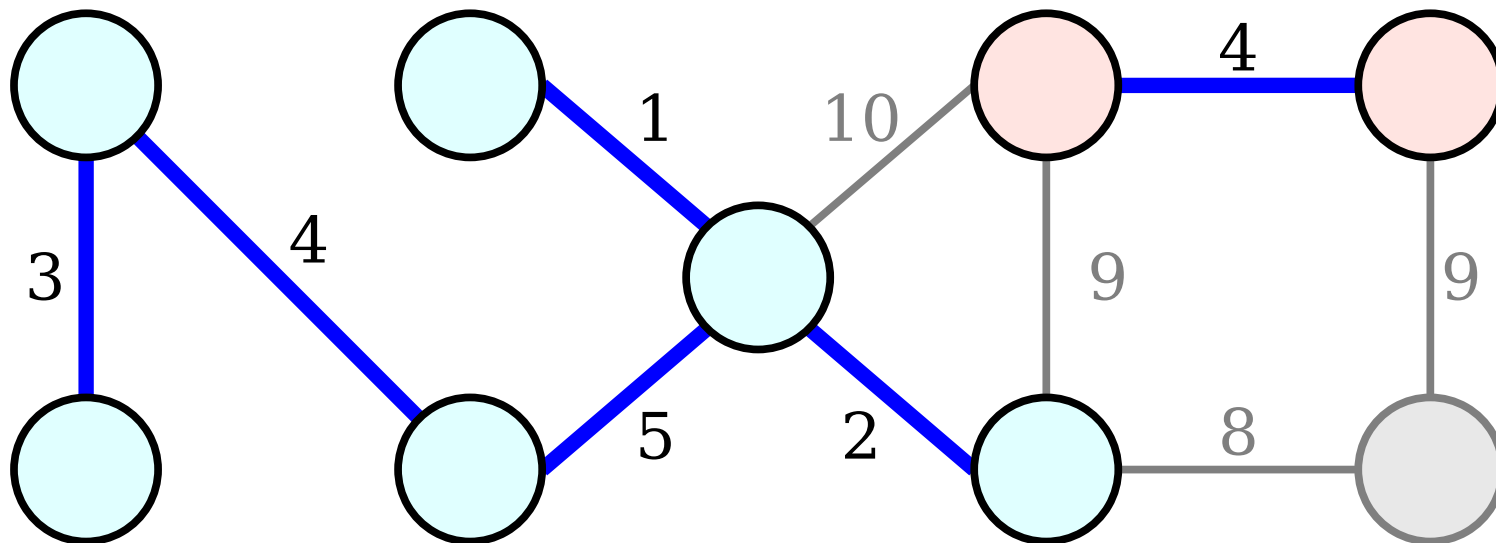
Kruskal's Algorithm

- ***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.



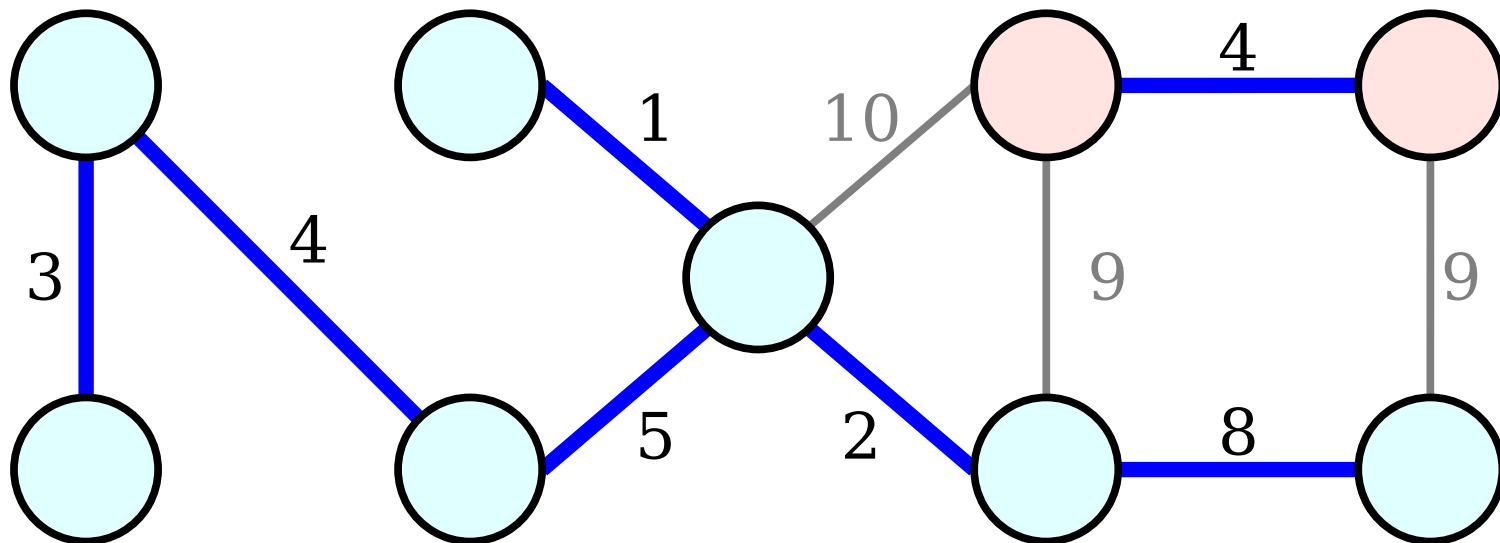
Kruskal's Algorithm

- ***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.



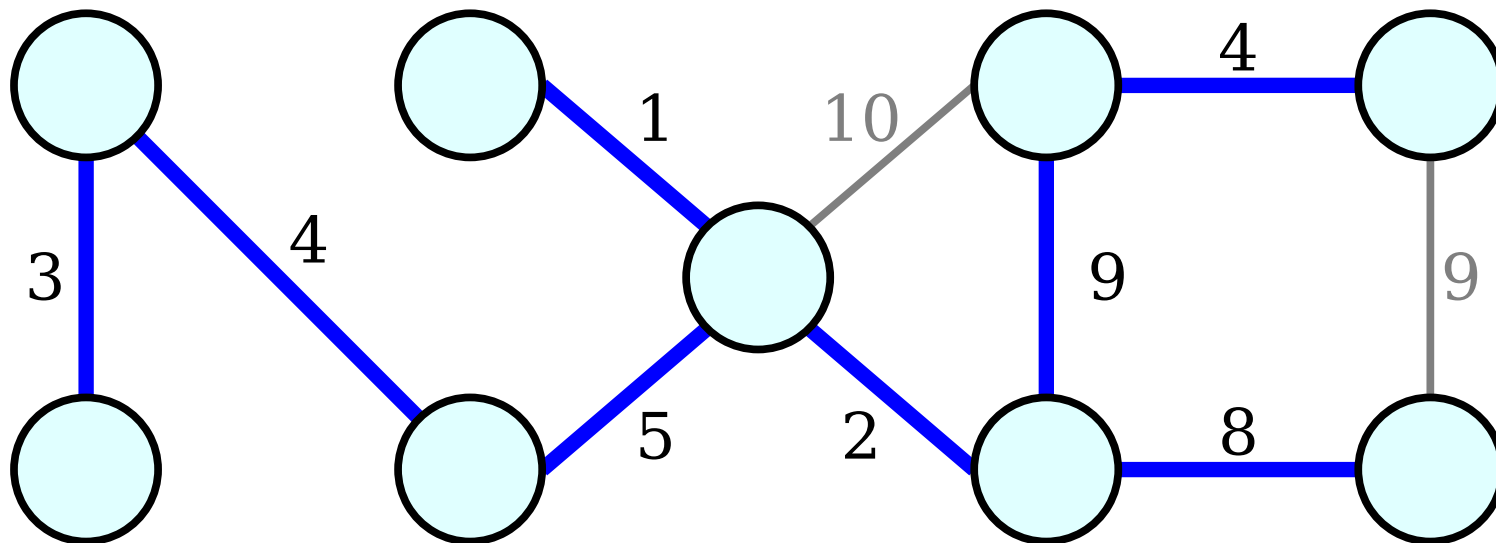
Kruskal's Algorithm

- ***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.



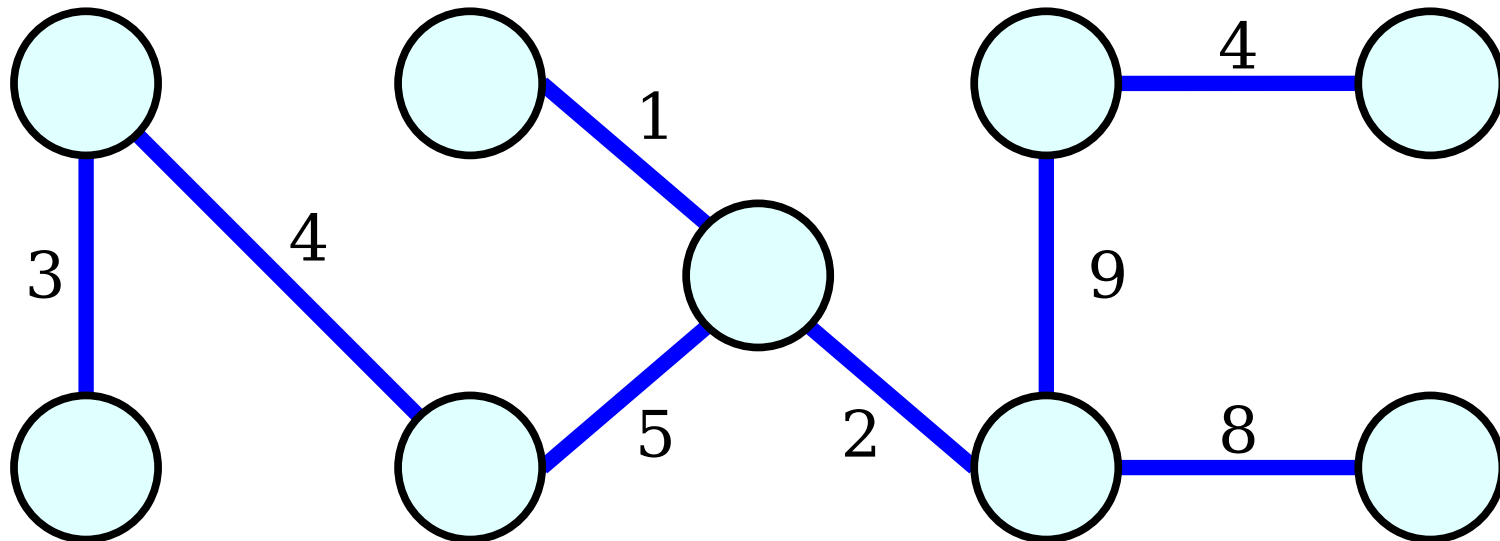
Kruskal's Algorithm

- ***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.



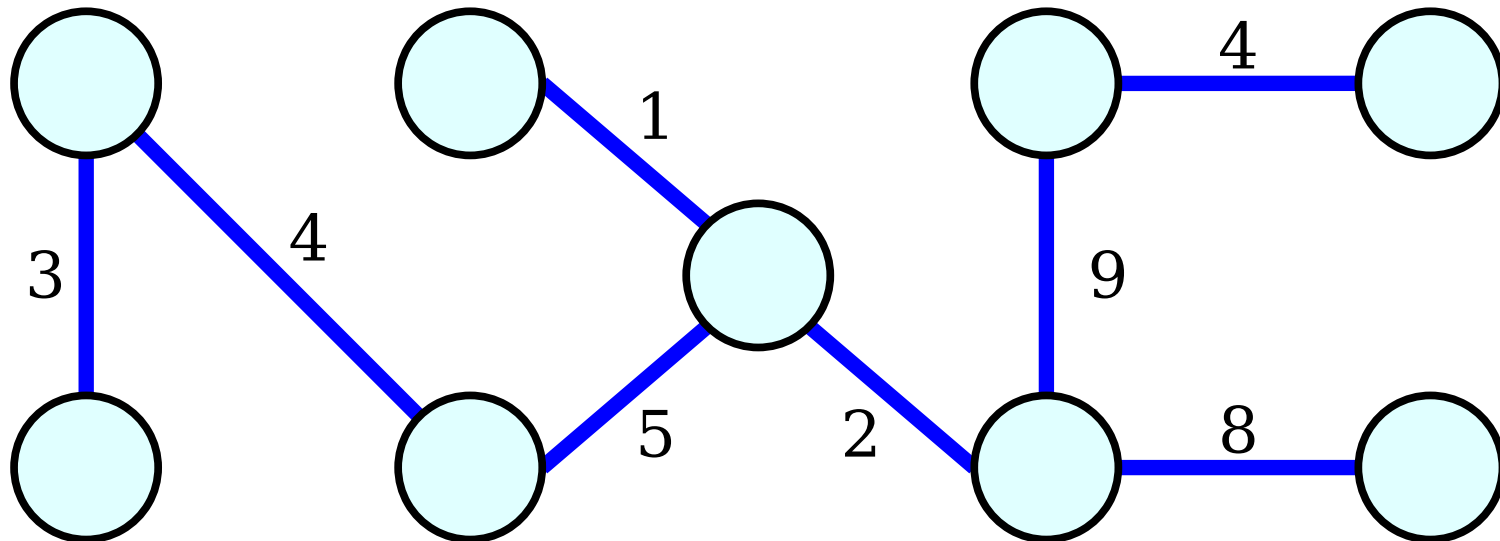
Kruskal's Algorithm

- ***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.



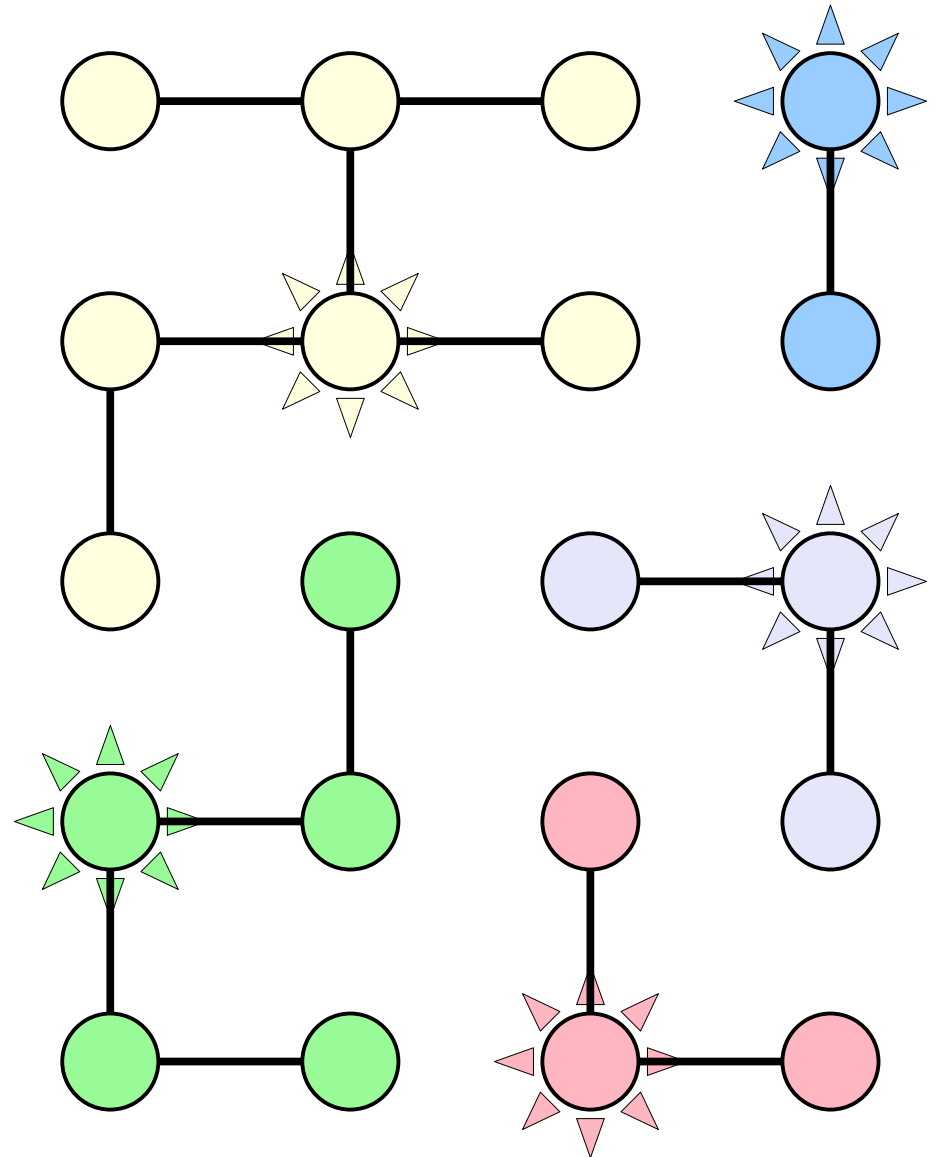
Incremental Connectivity

- Kruskal's algorithm needs a data structure that solves **incremental connectivity**.
 - We begin with an empty graph.
 - We need to be able to add new edges to the graph and check whether arbitrary pairs of nodes are connected.
- **Question:** How efficiently can we do this?



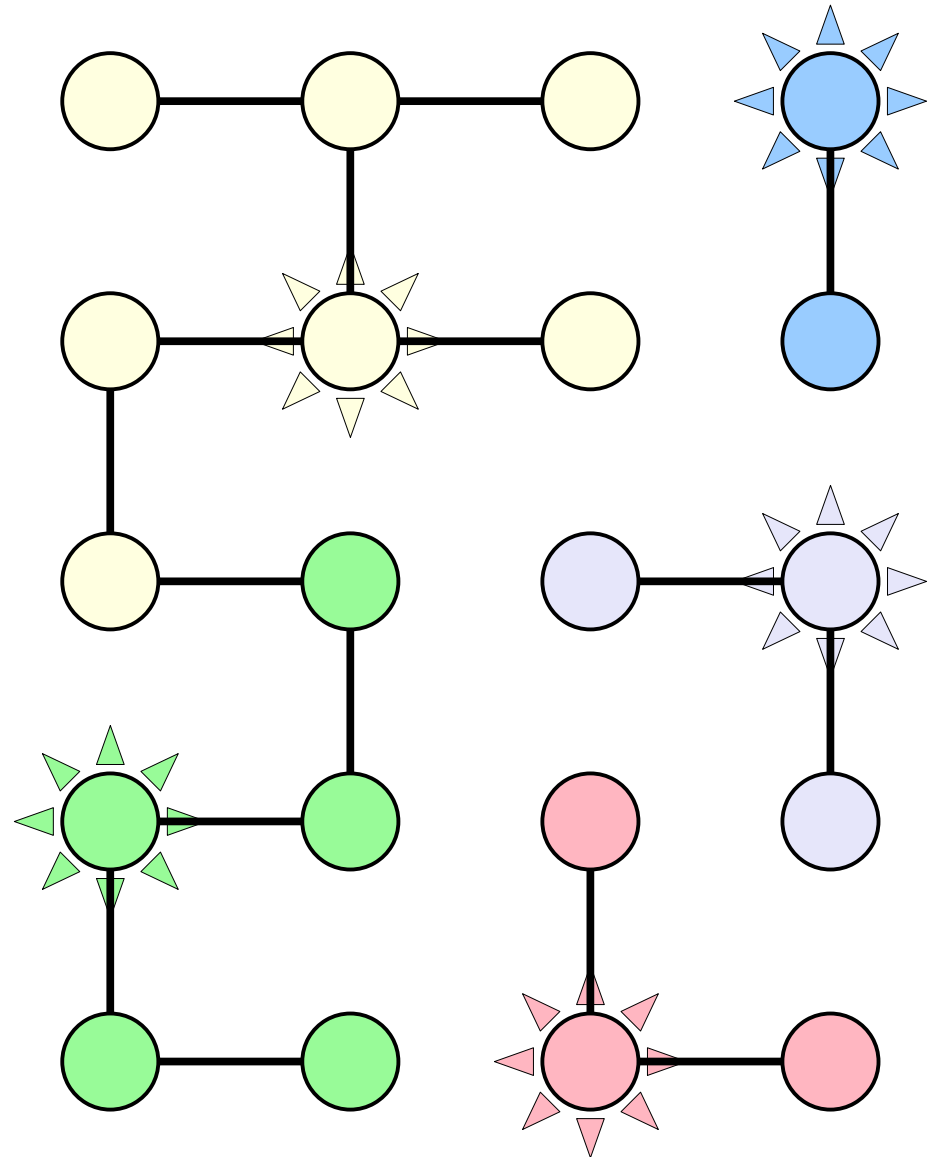
Representatives

- **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.



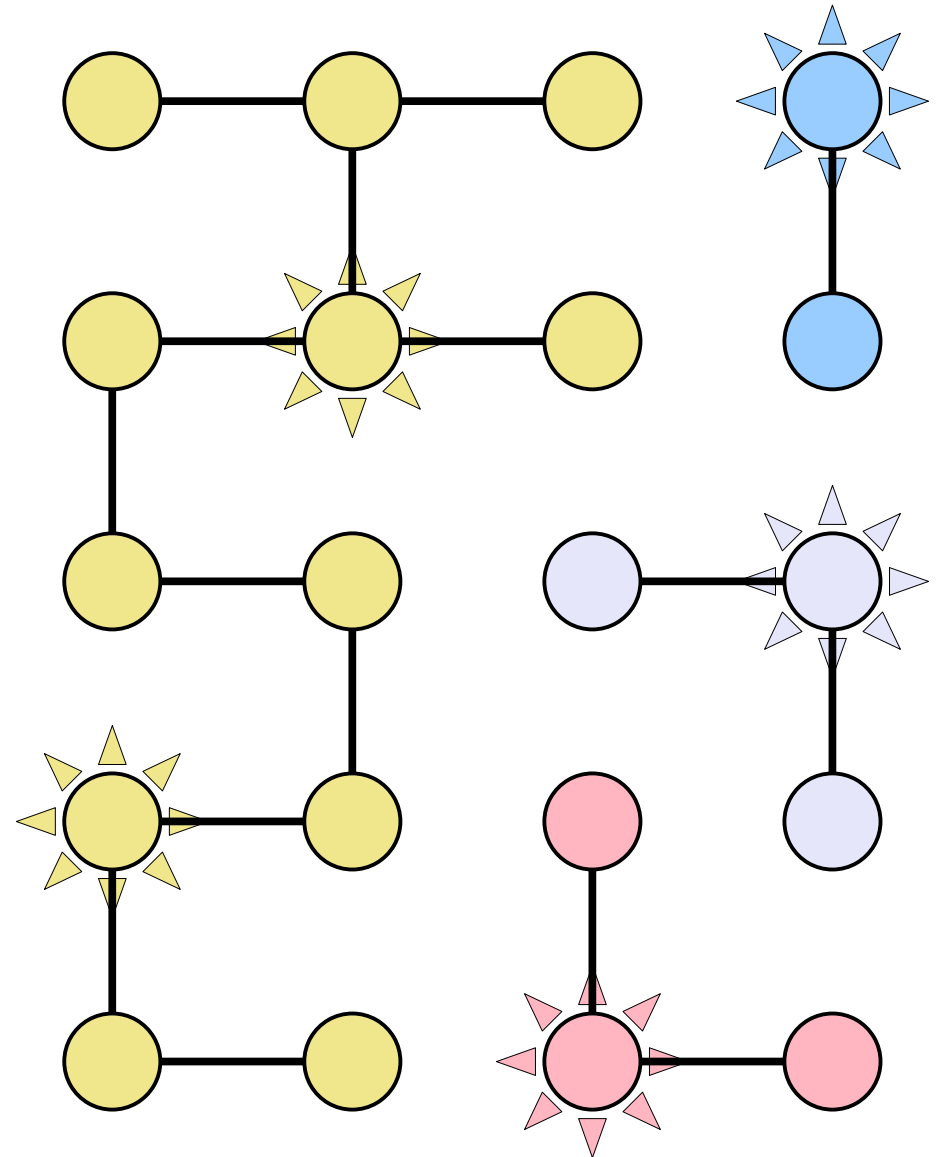
Representatives

- **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.



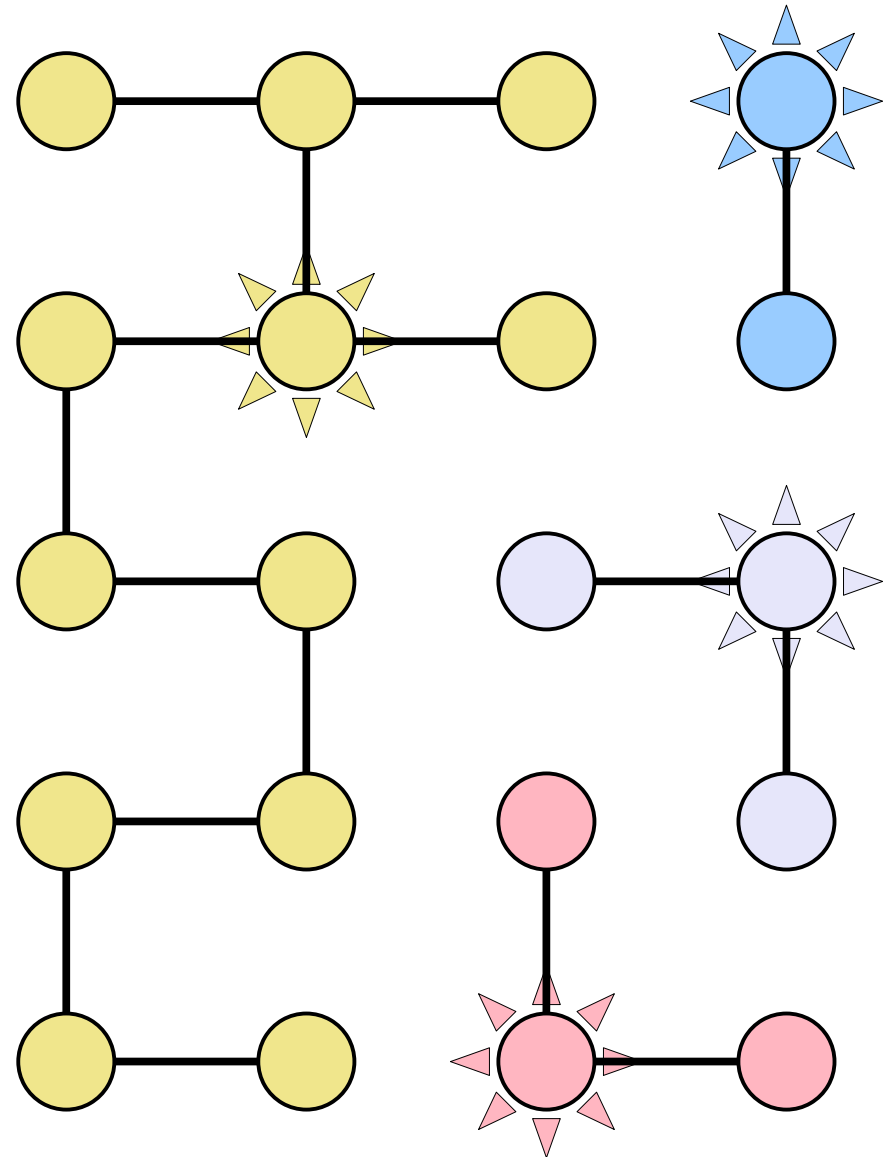
Representatives

- **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.



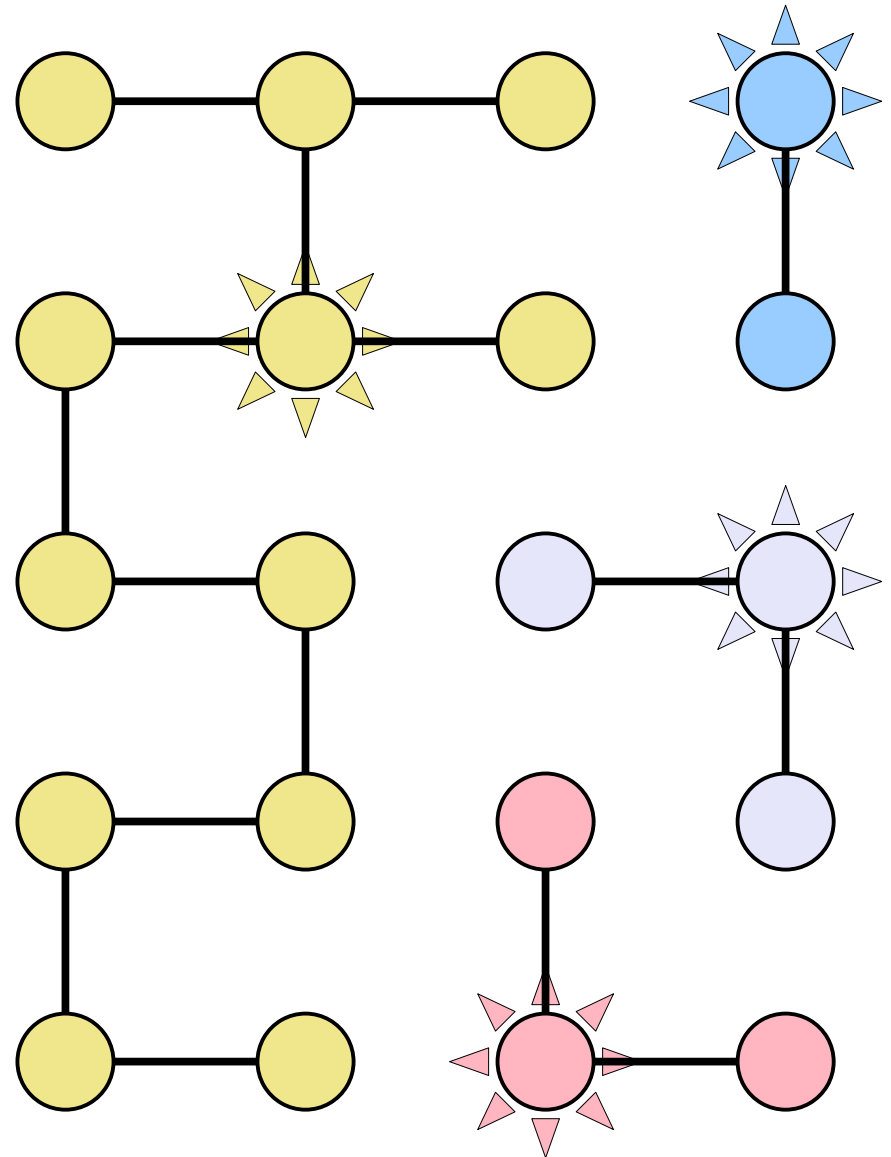
Representatives

- **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.



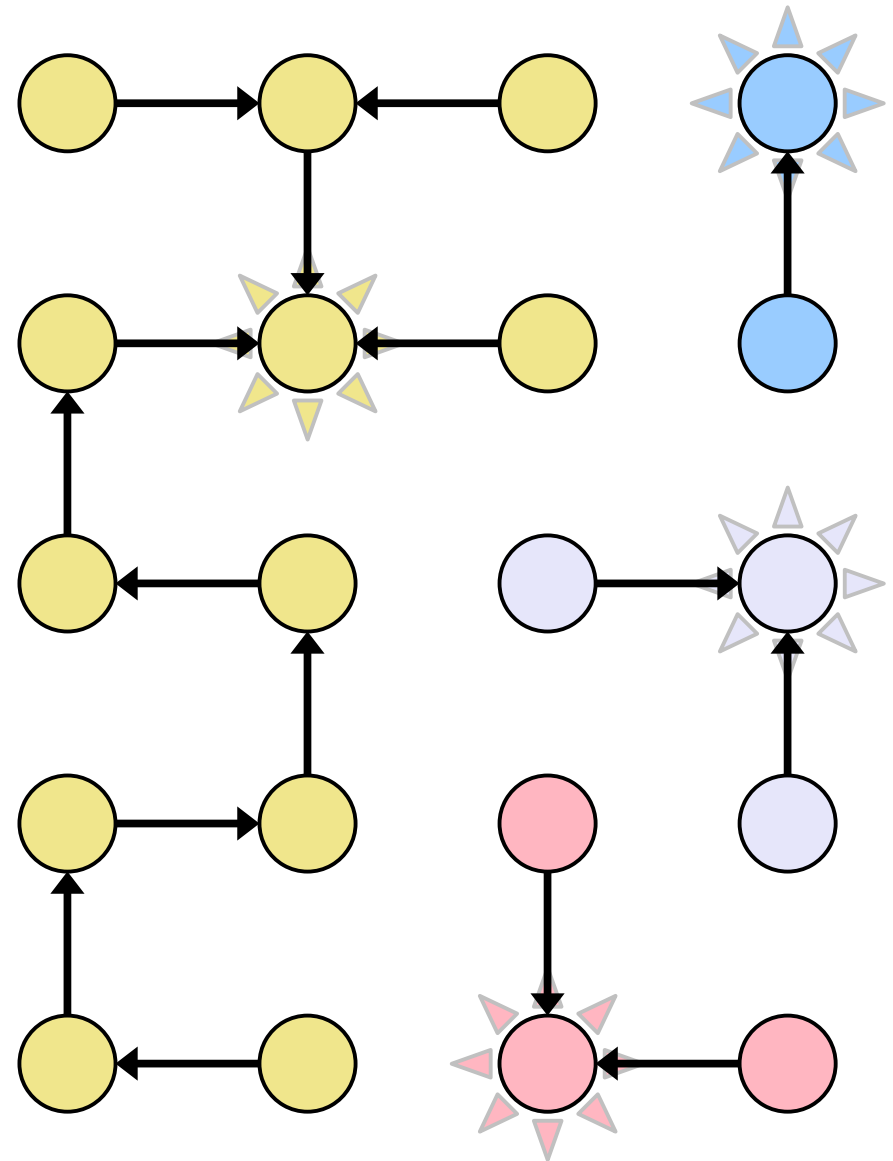
Representatives

- 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**.



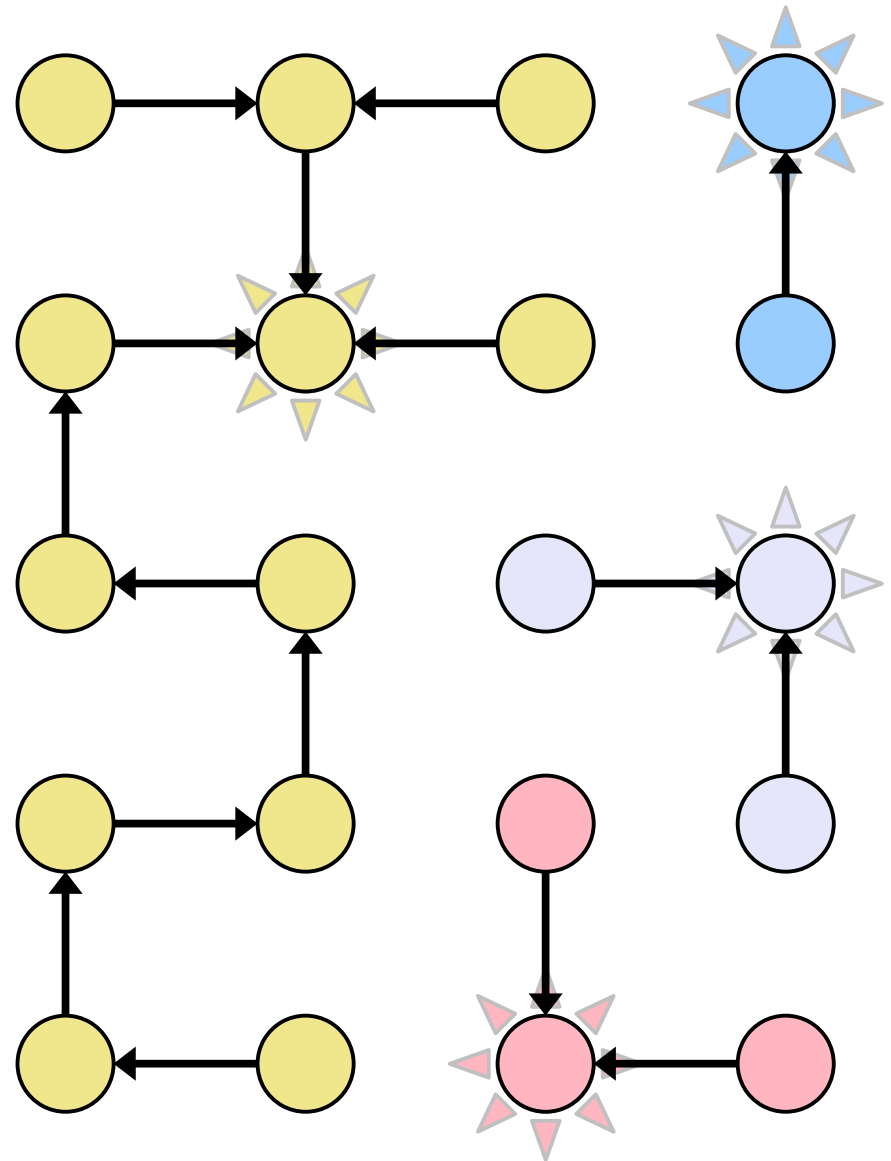
Representatives

- 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**.



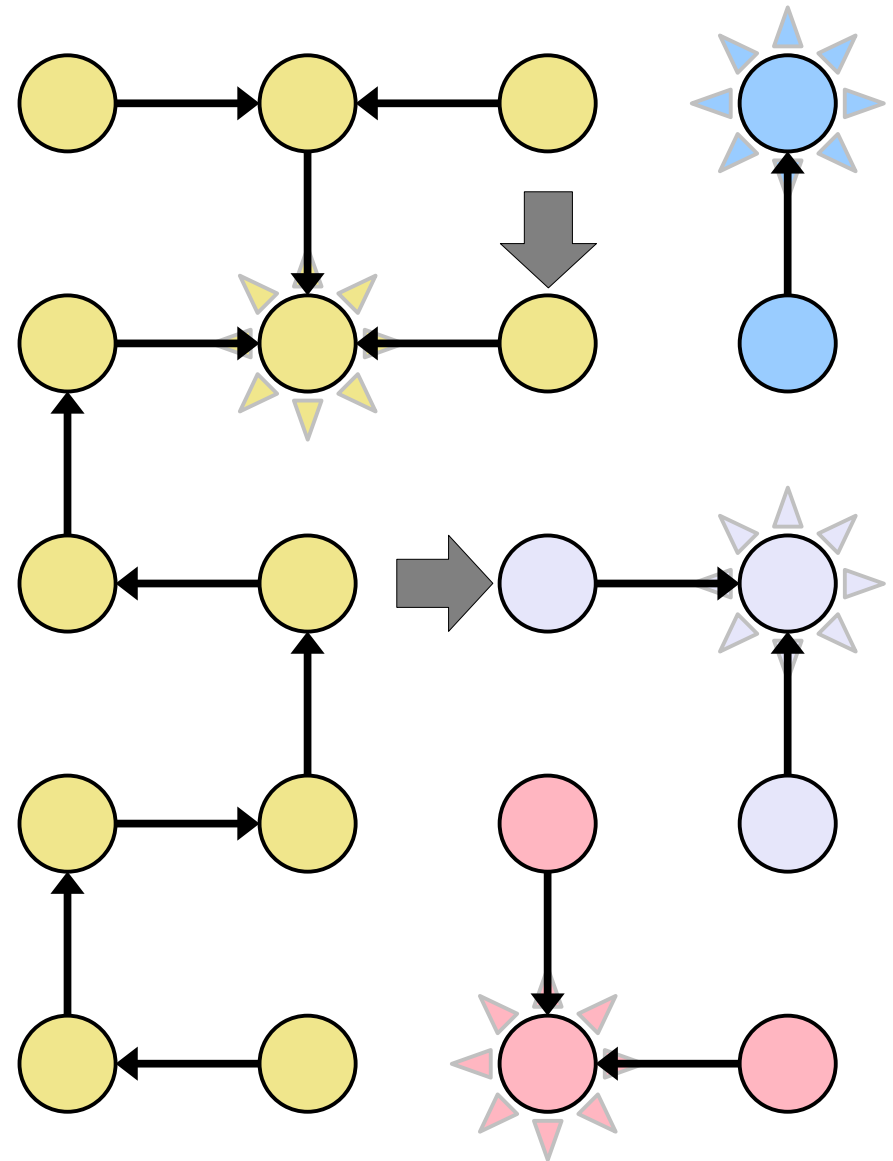
Representatives

- 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.



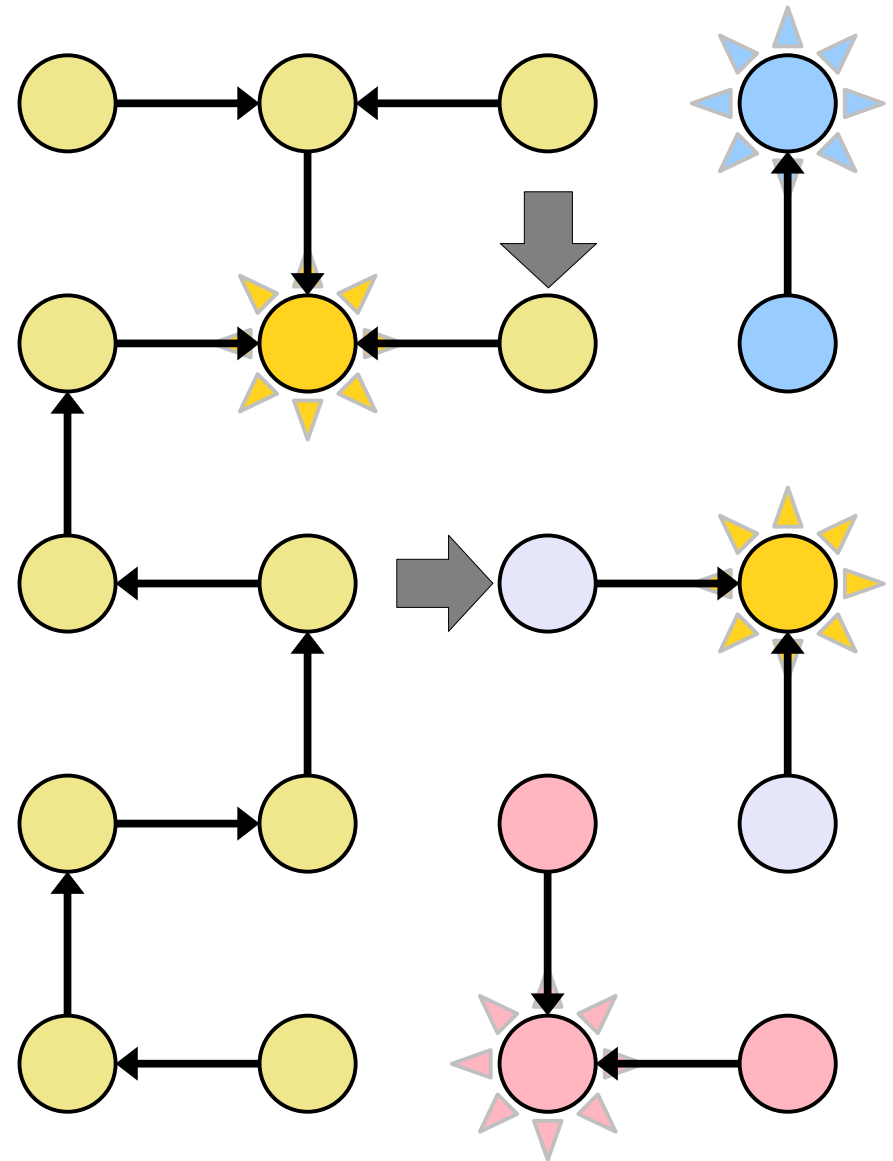
Representatives

- 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.



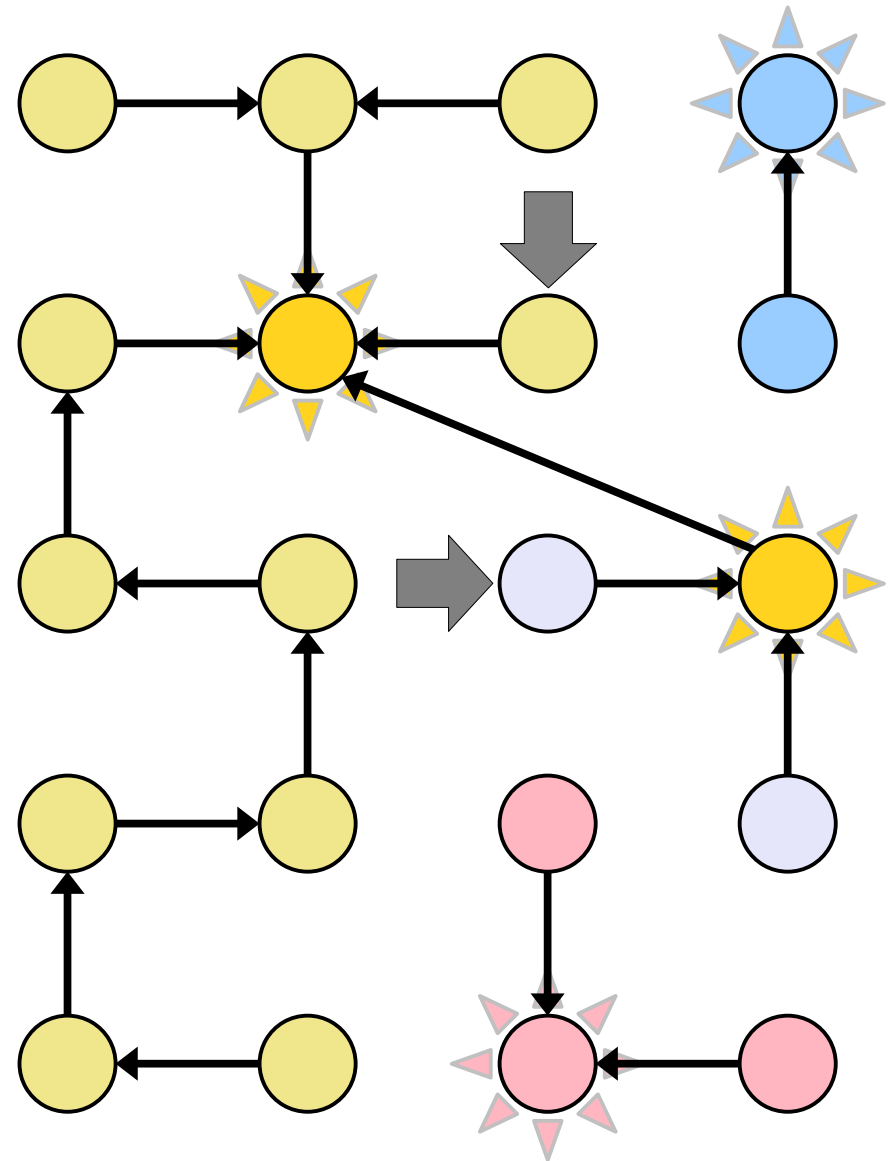
Representatives

- 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.



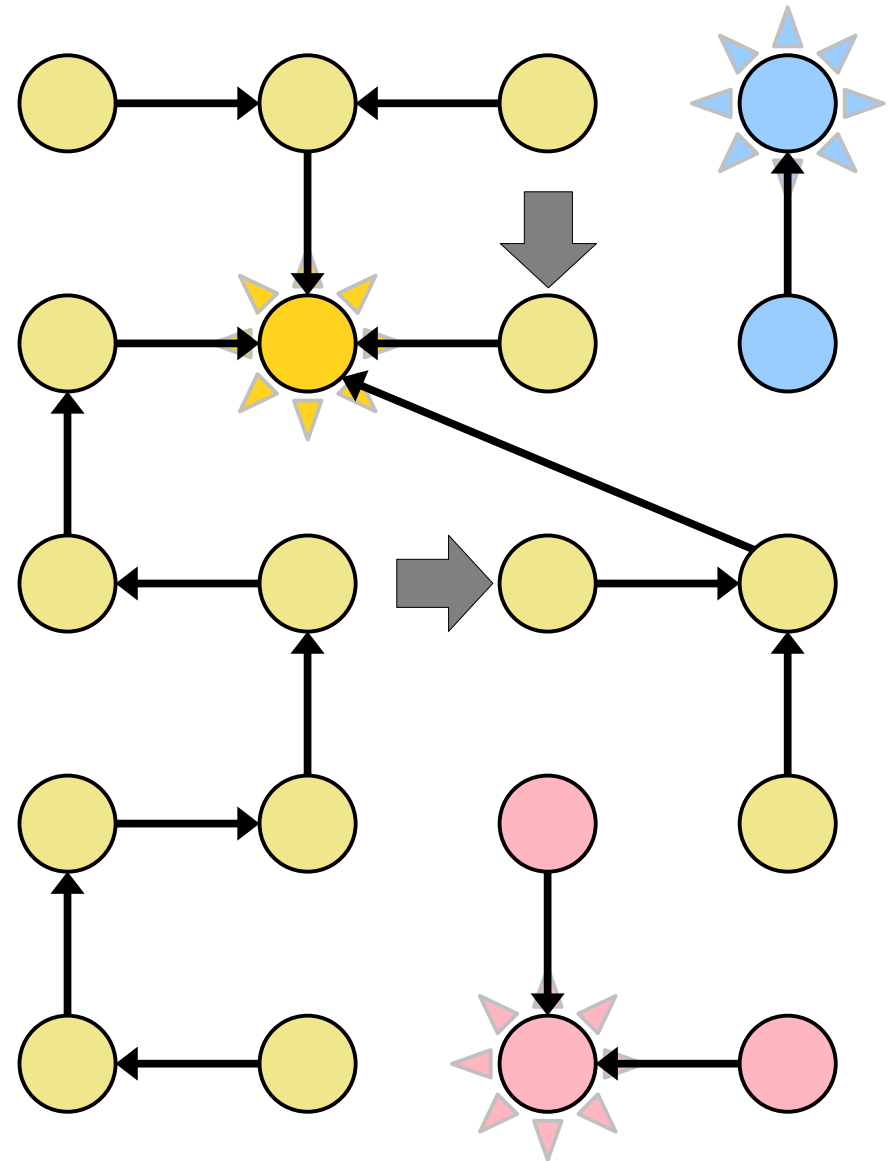
Representatives

- 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.



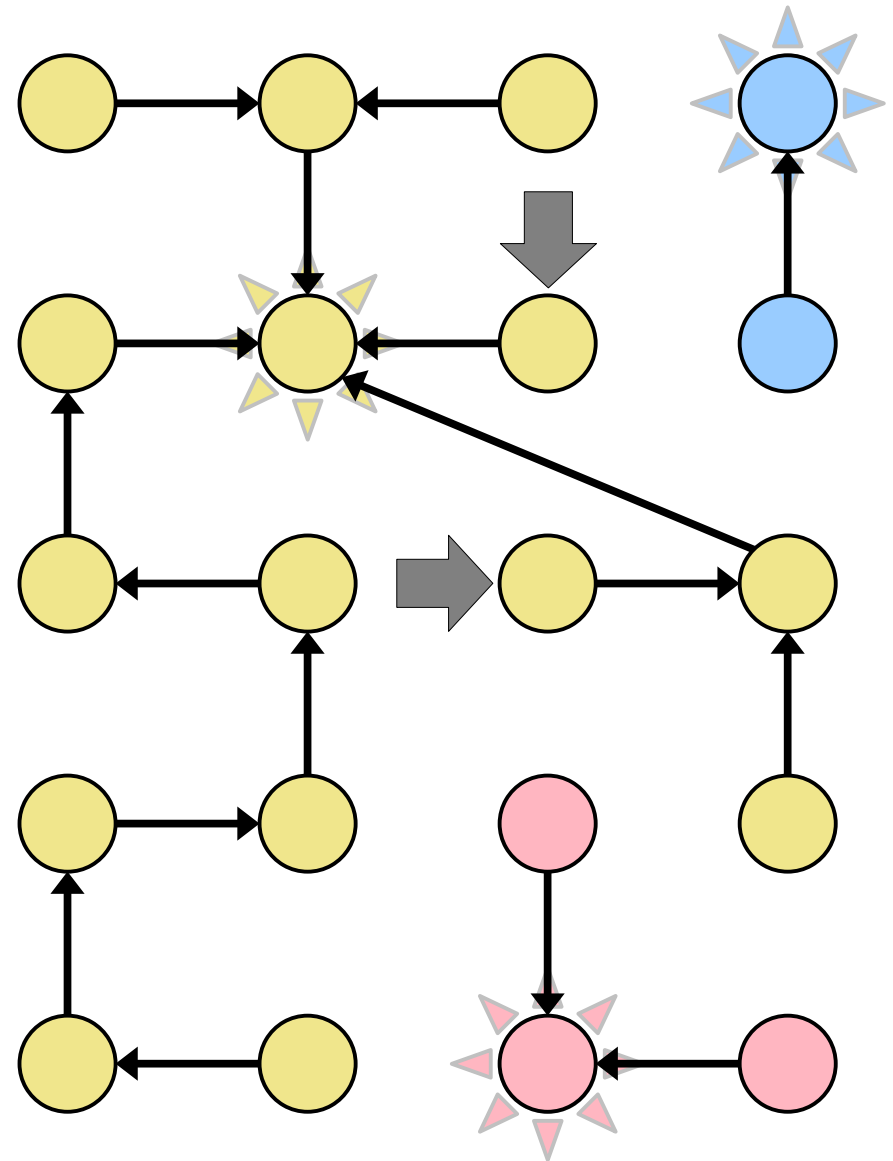
Representatives

- 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.



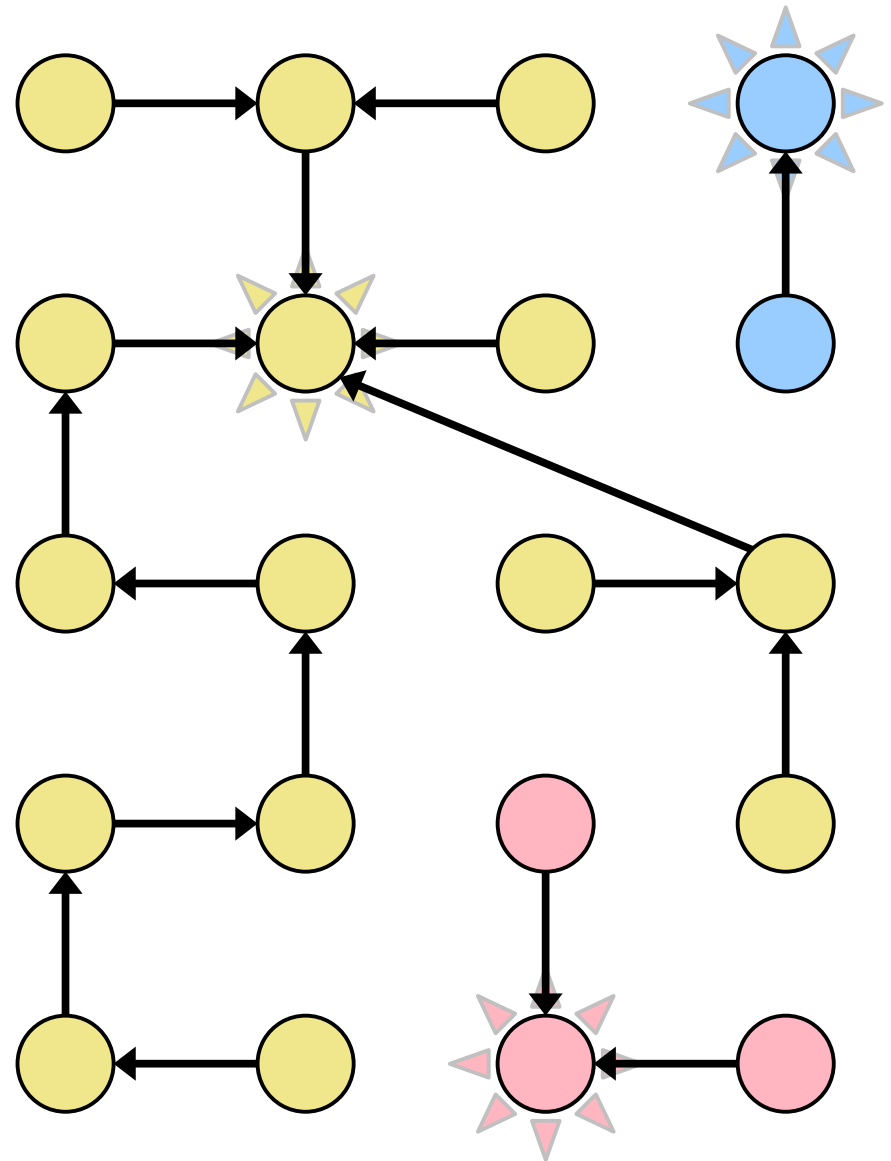
Representatives

- 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.



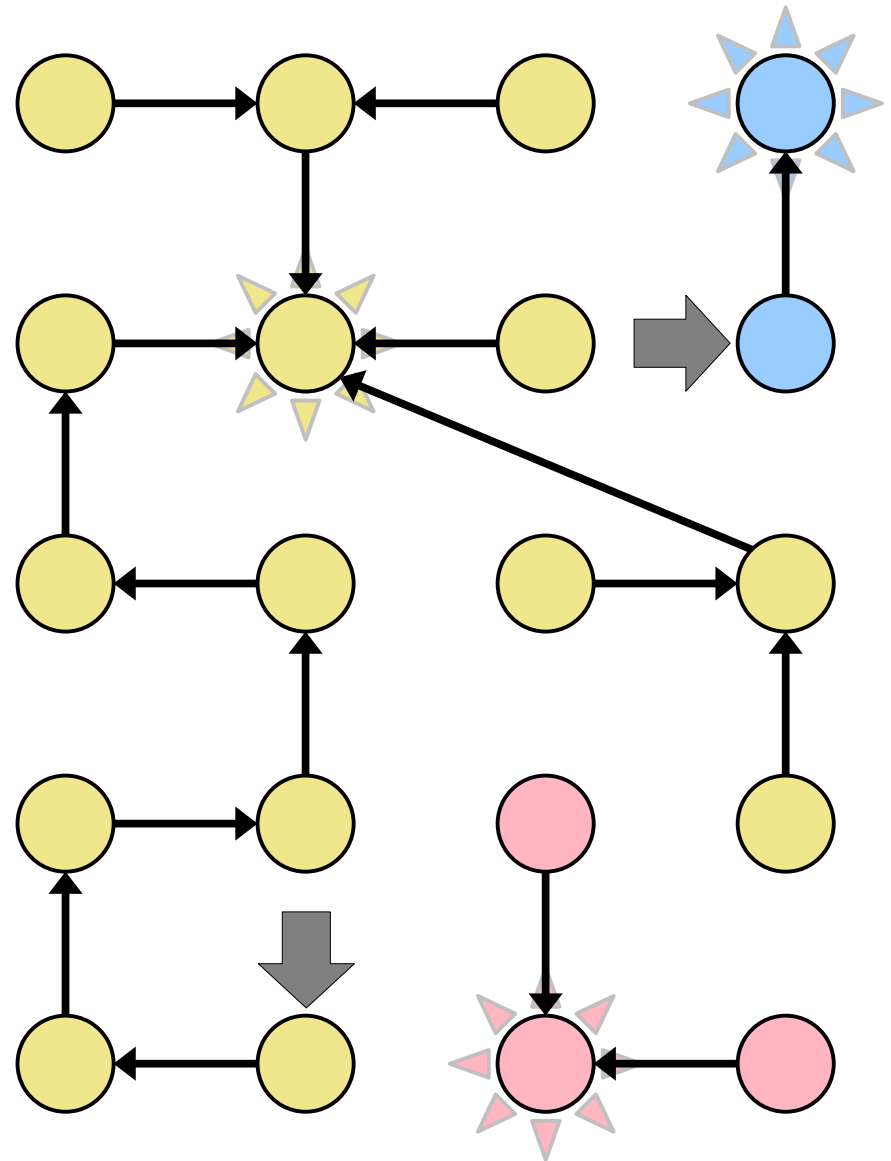
Representatives

- 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.



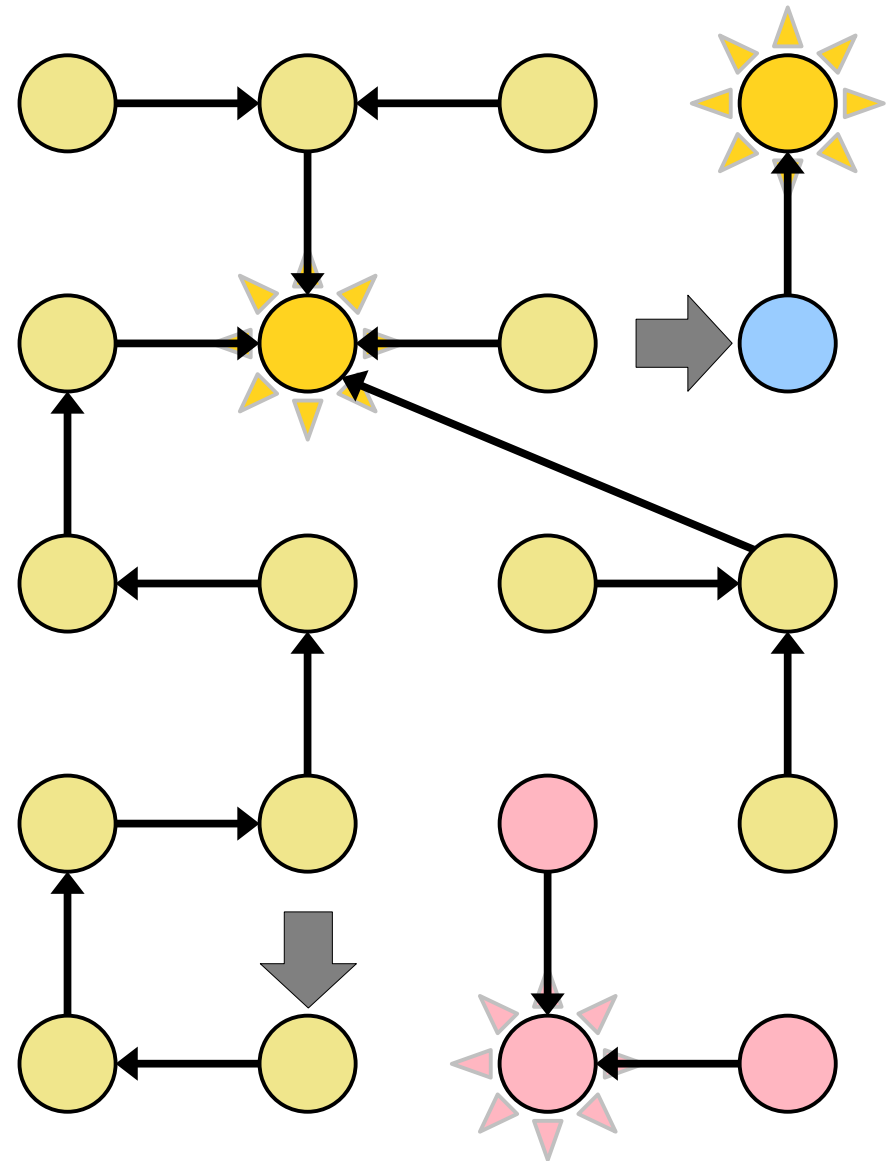
Representatives

- 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.



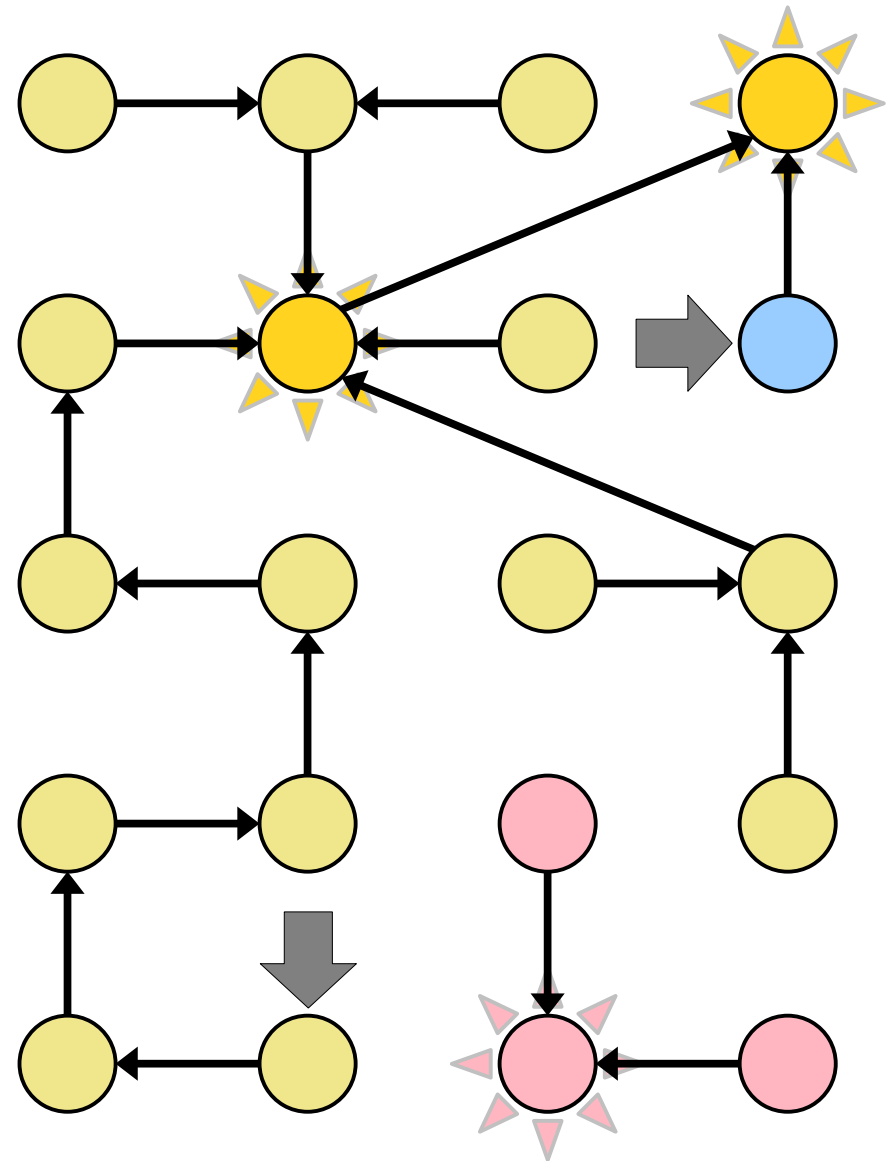
Representatives

- 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.



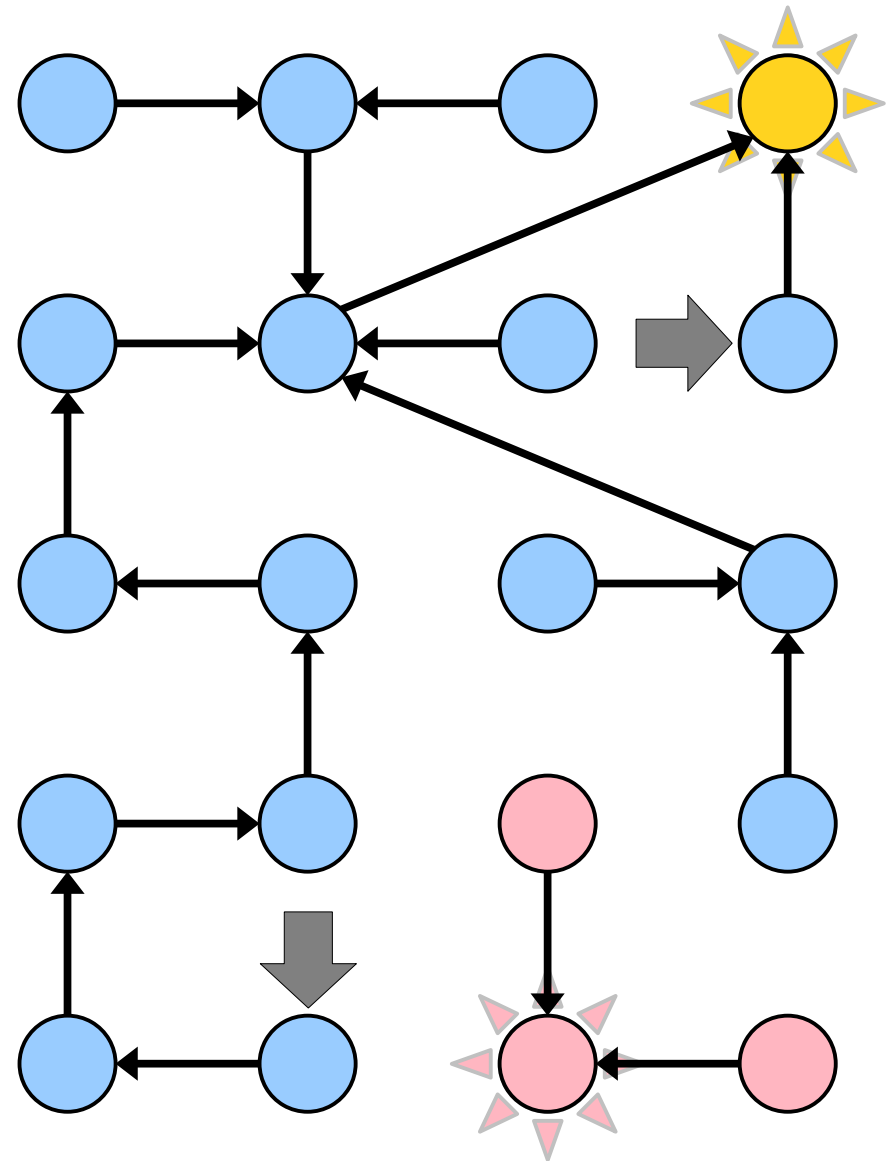
Representatives

- 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.



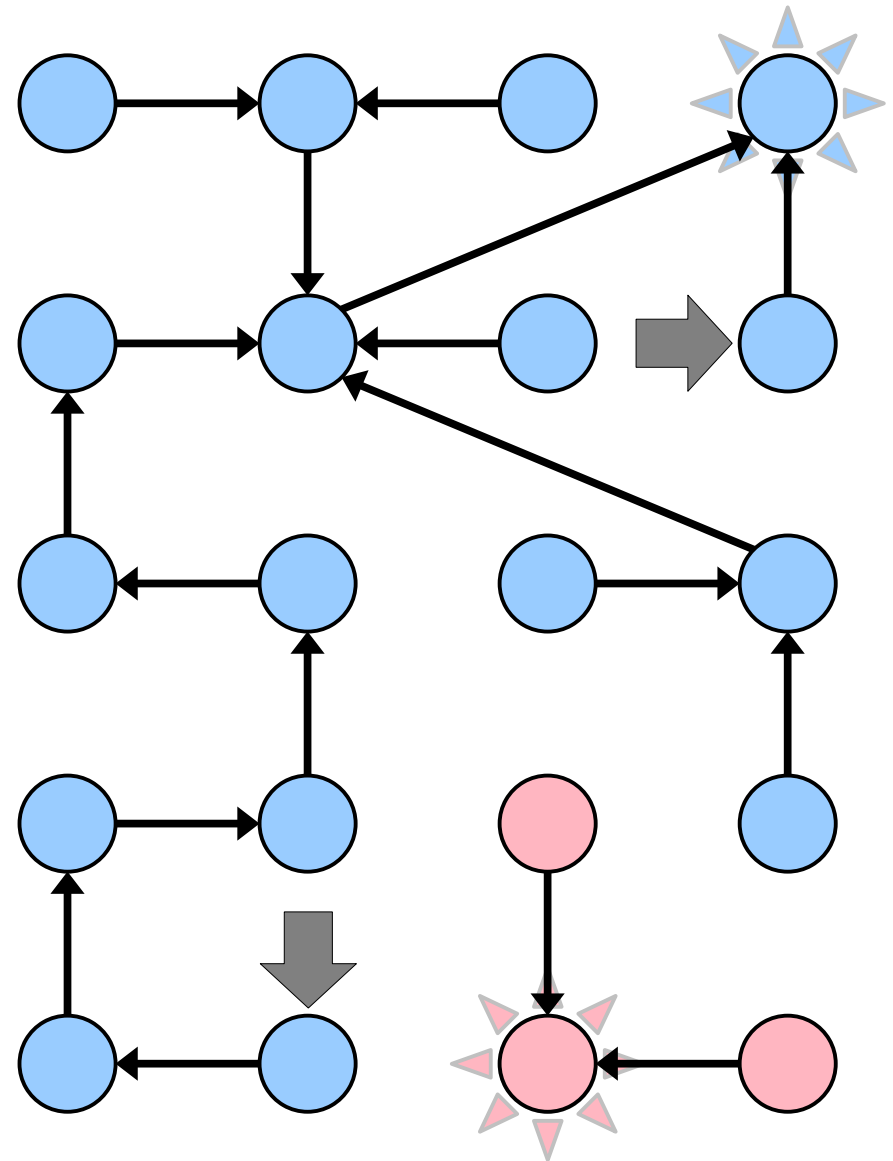
Representatives

- 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.



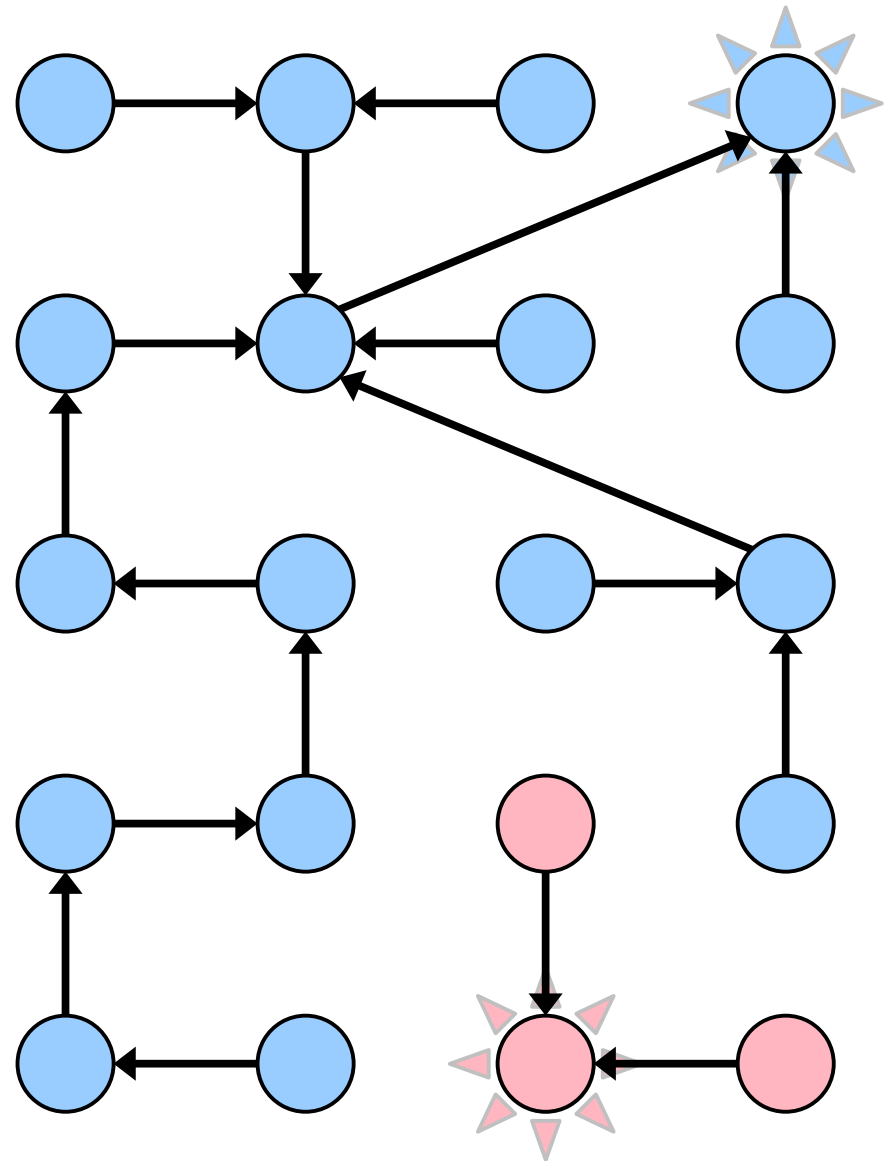
Representatives

- 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.



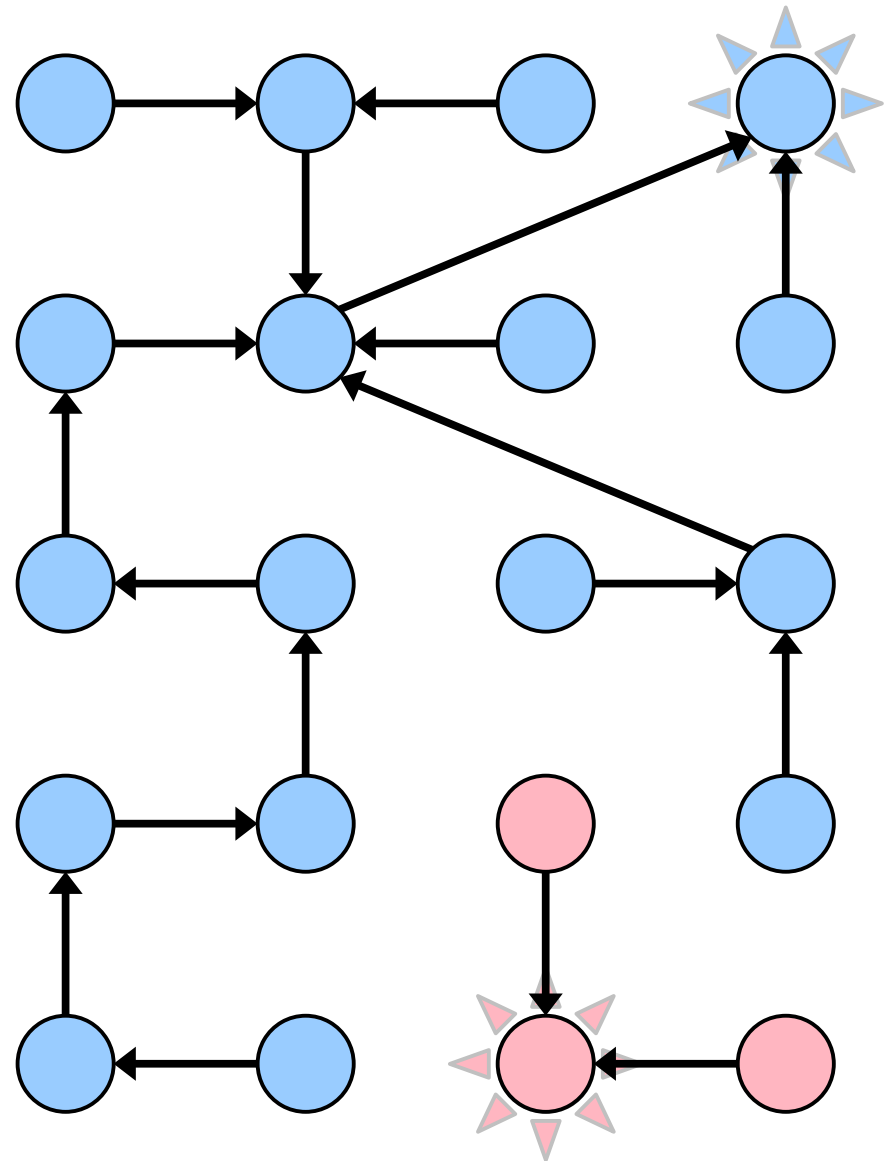
Representatives

- 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.



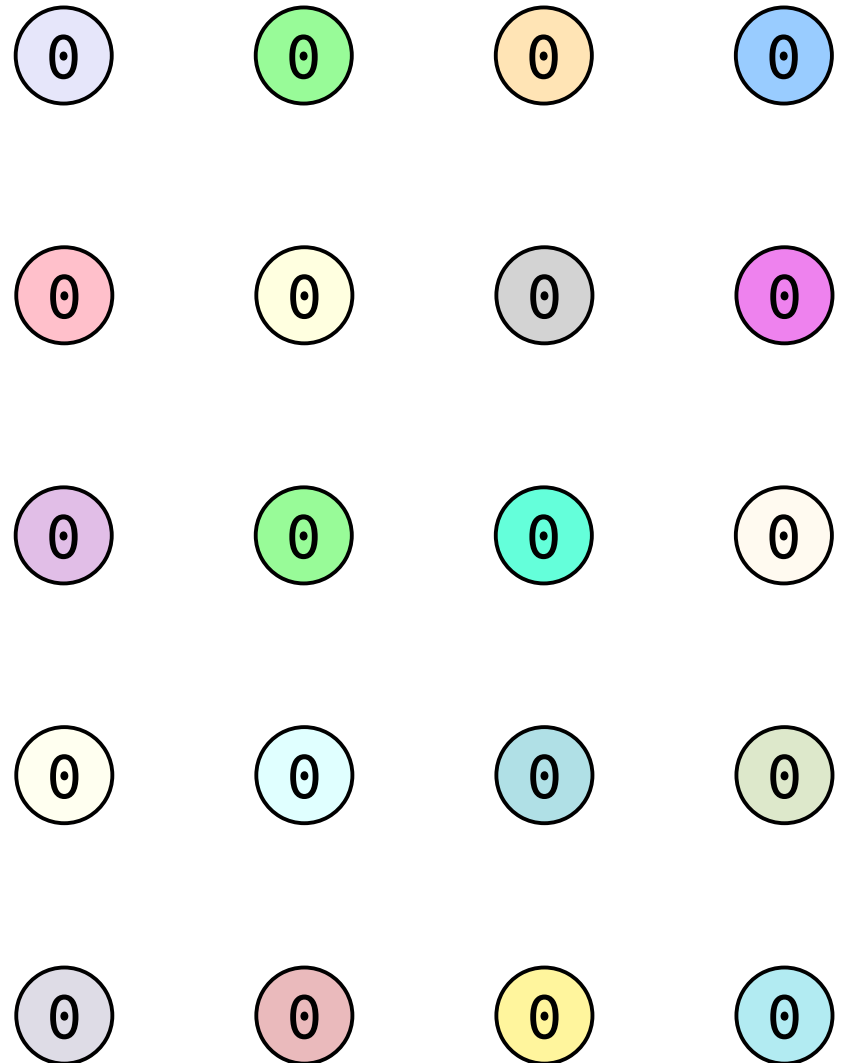
Representatives

- 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?***



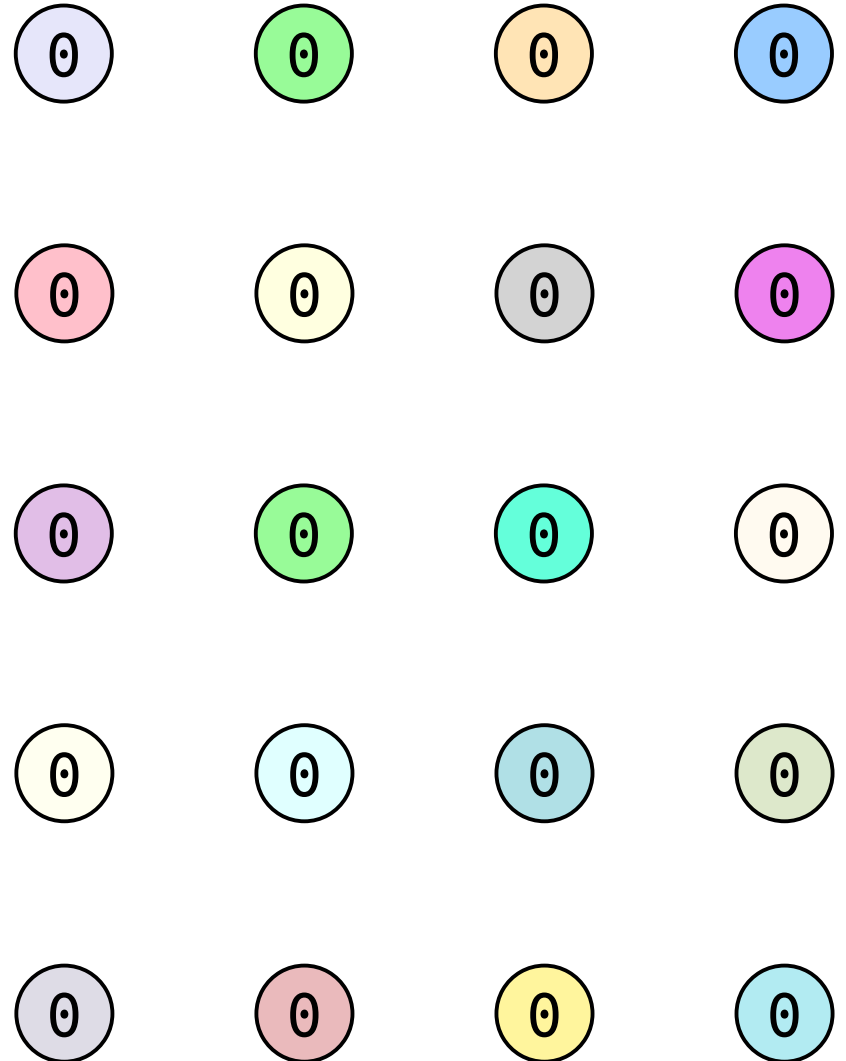
Union-By-Rank

- Assign each node a *rank*, initially 0.



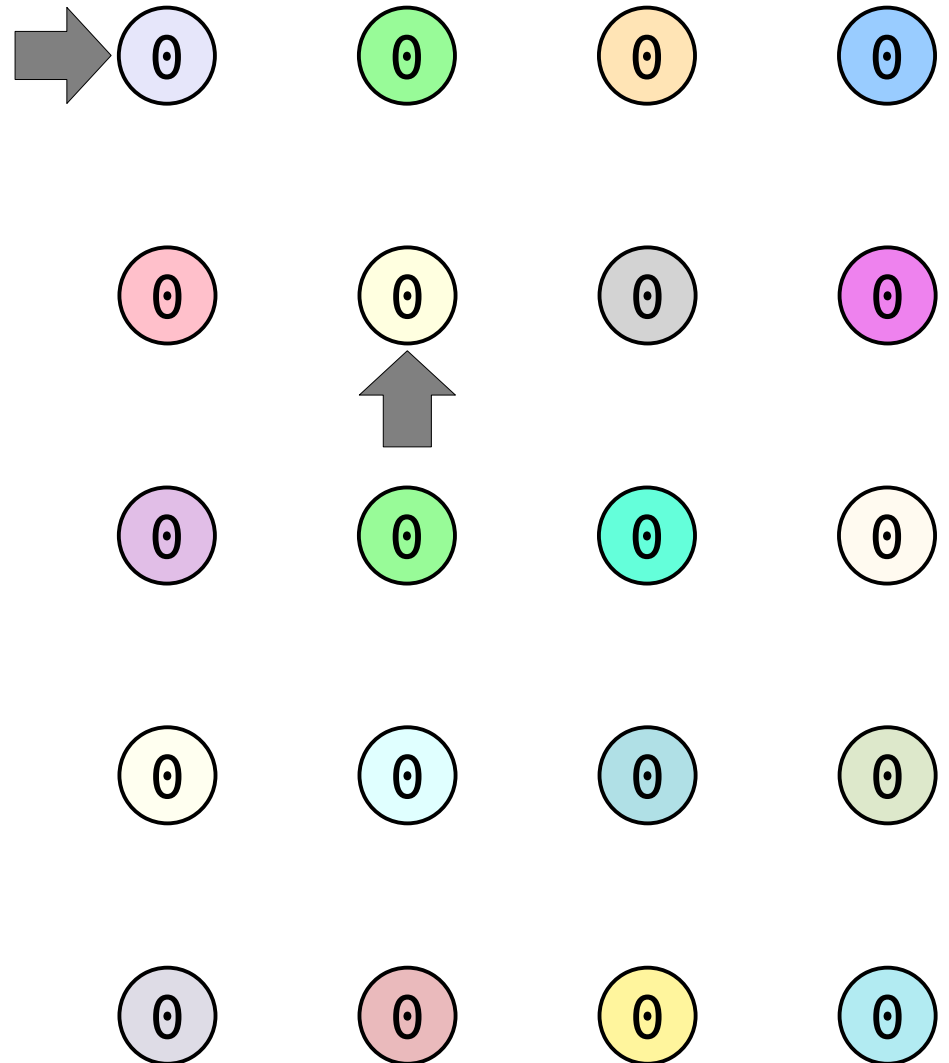
Union-By-Rank

- 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.



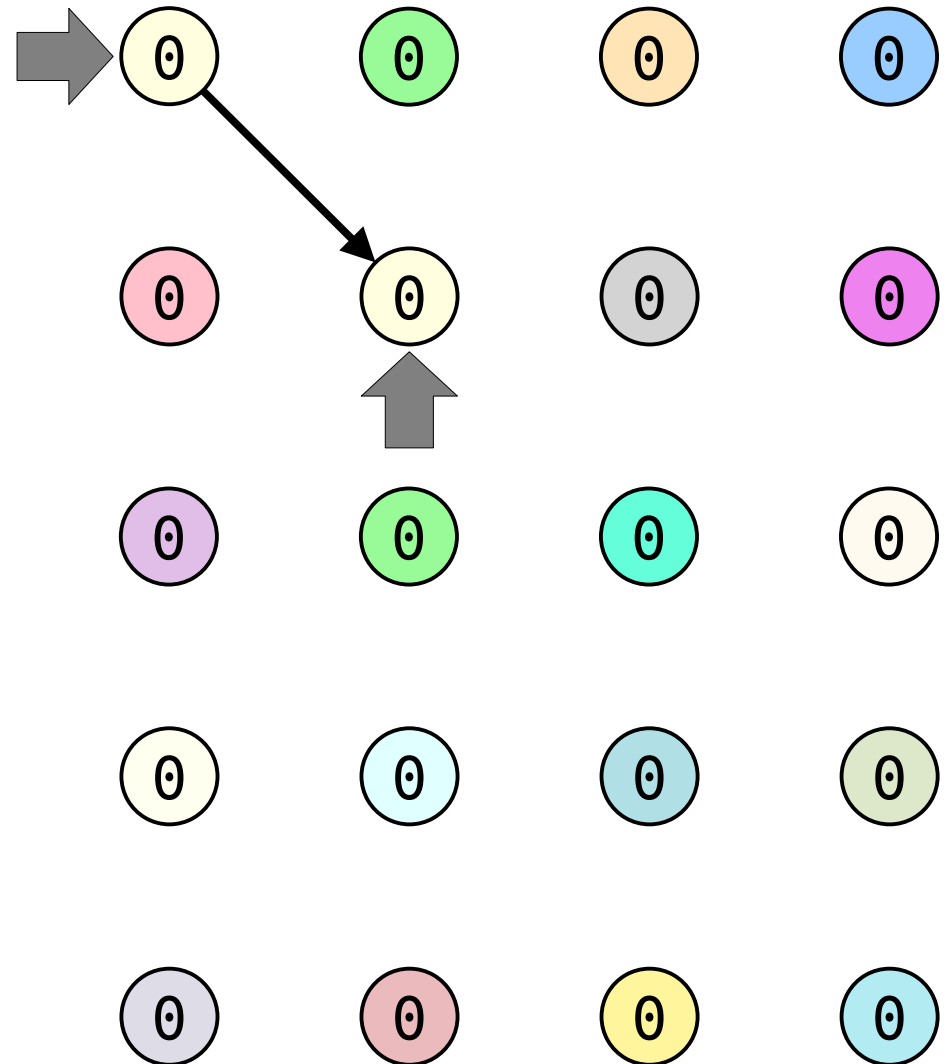
Union-By-Rank

- 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.



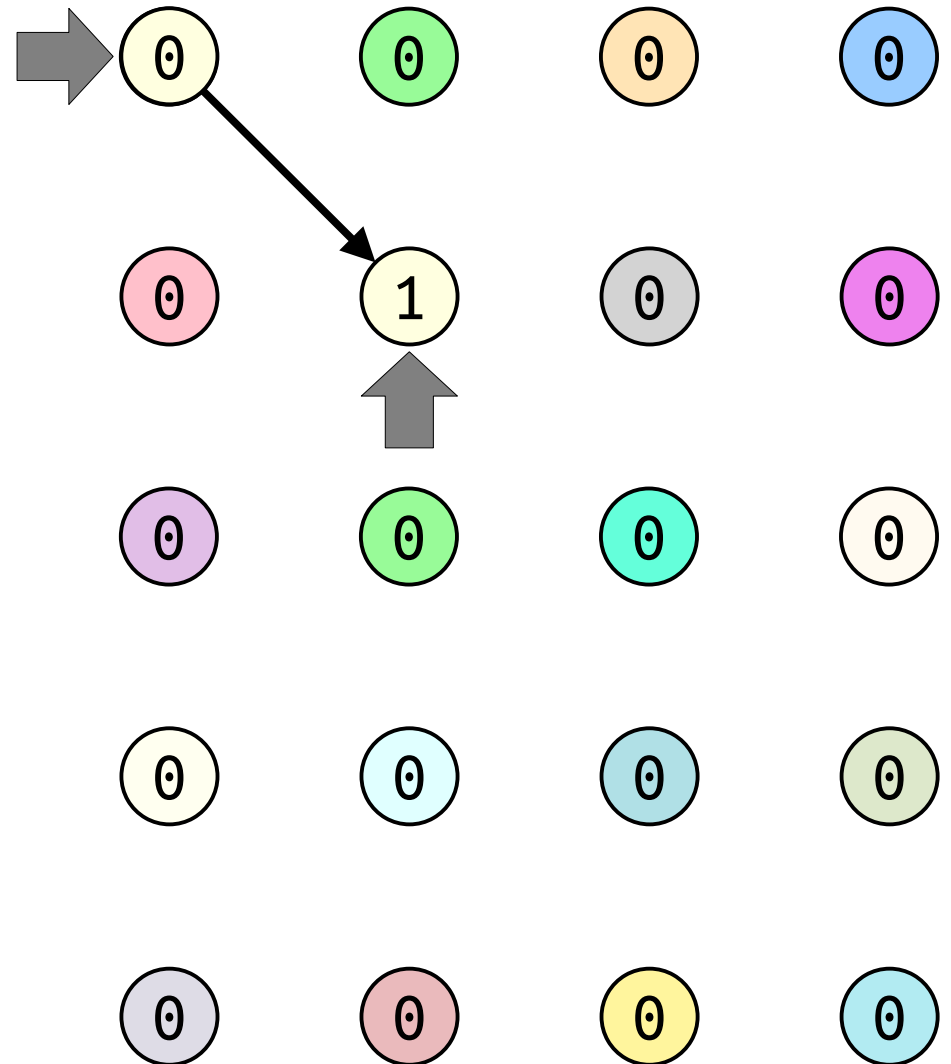
Union-By-Rank

- 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.



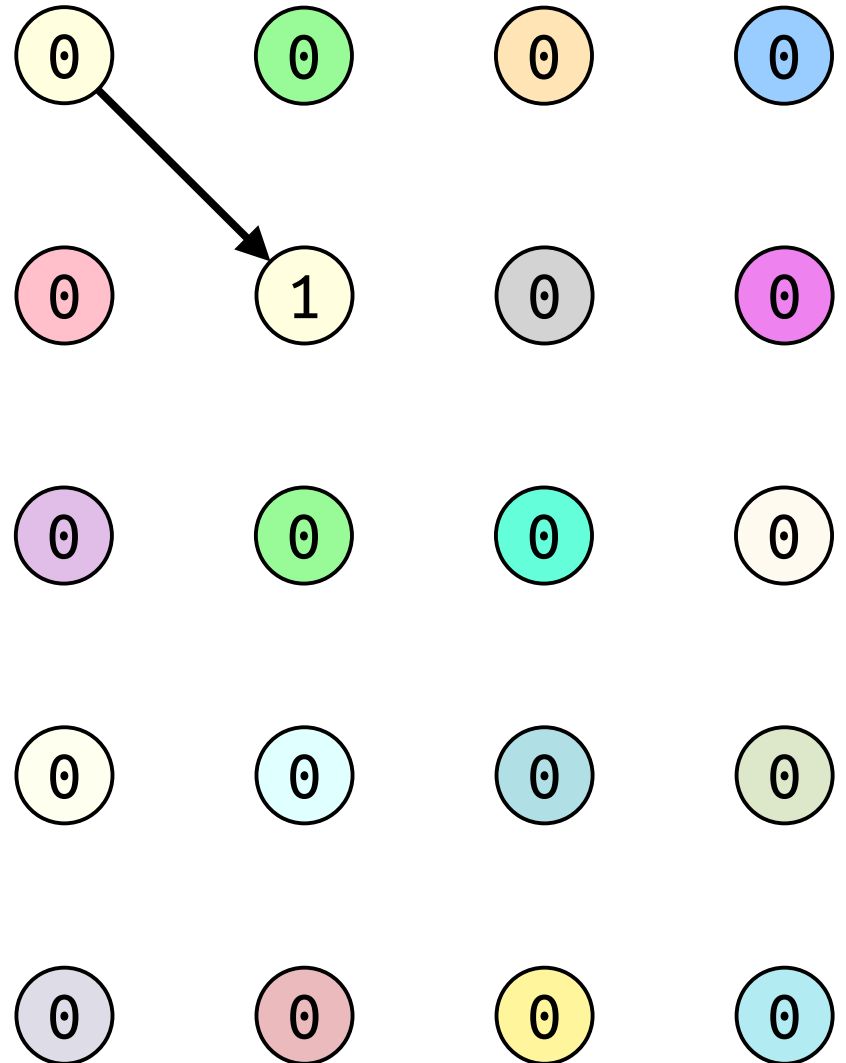
Union-By-Rank

- 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.



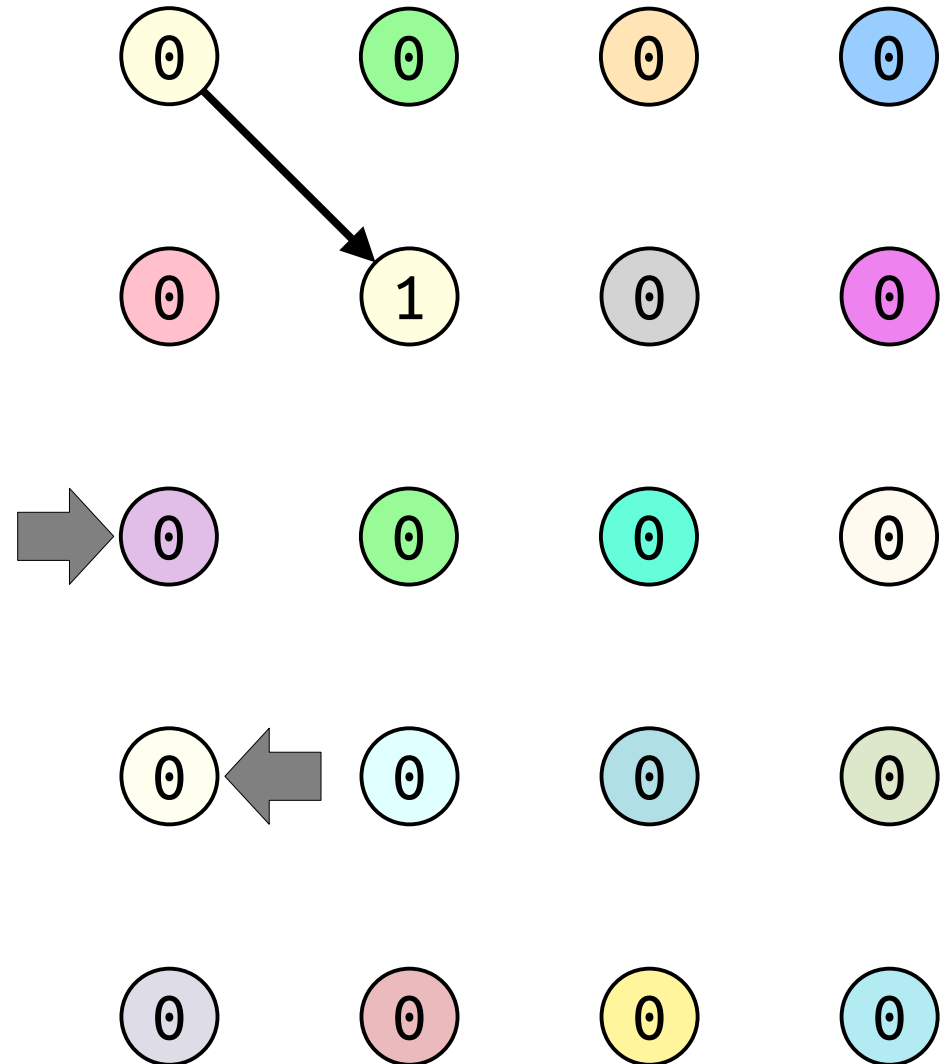
Union-By-Rank

- 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.



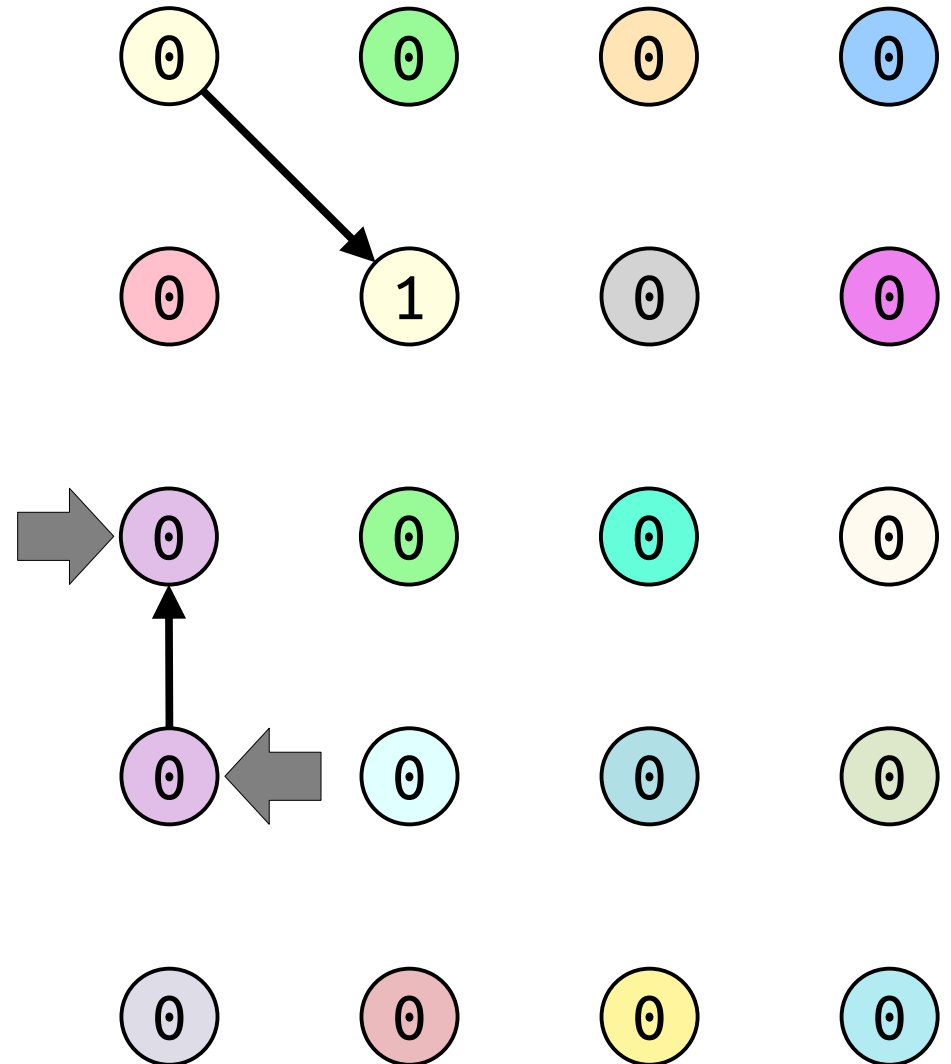
Union-By-Rank

- 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.



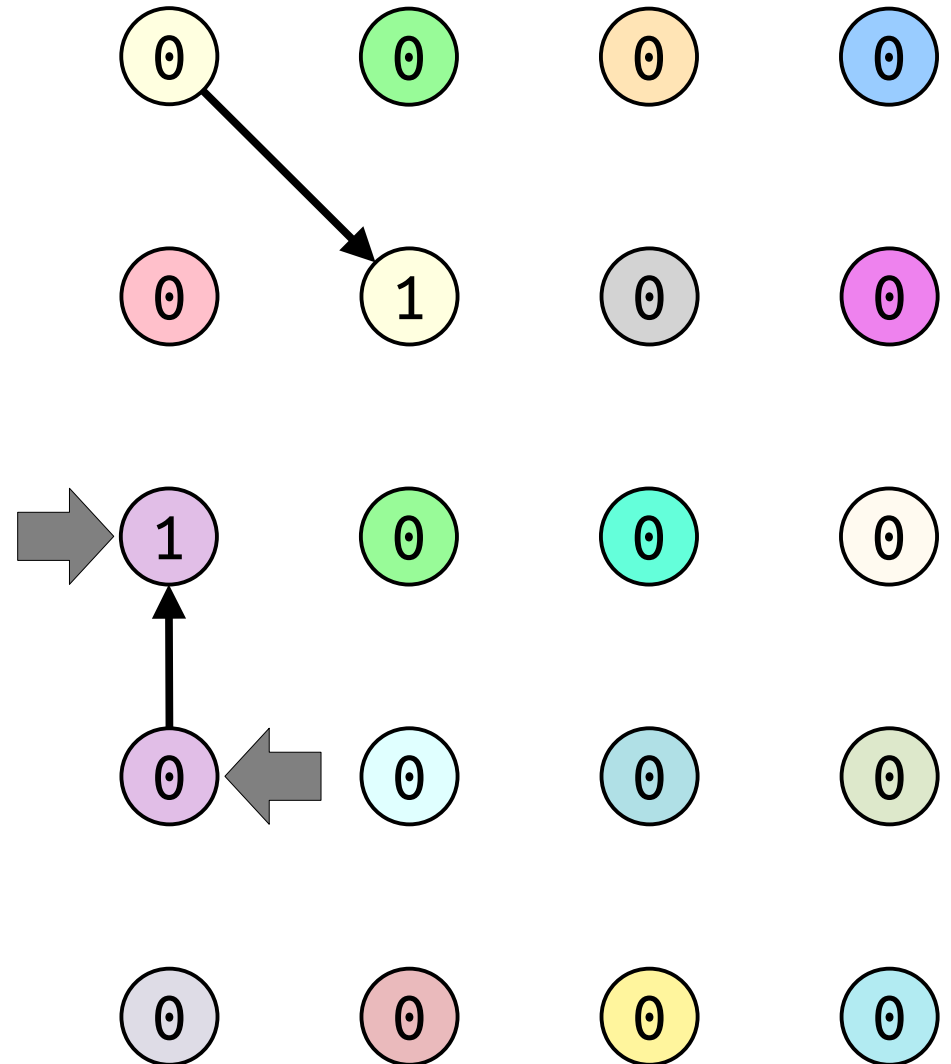
Union-By-Rank

- 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.



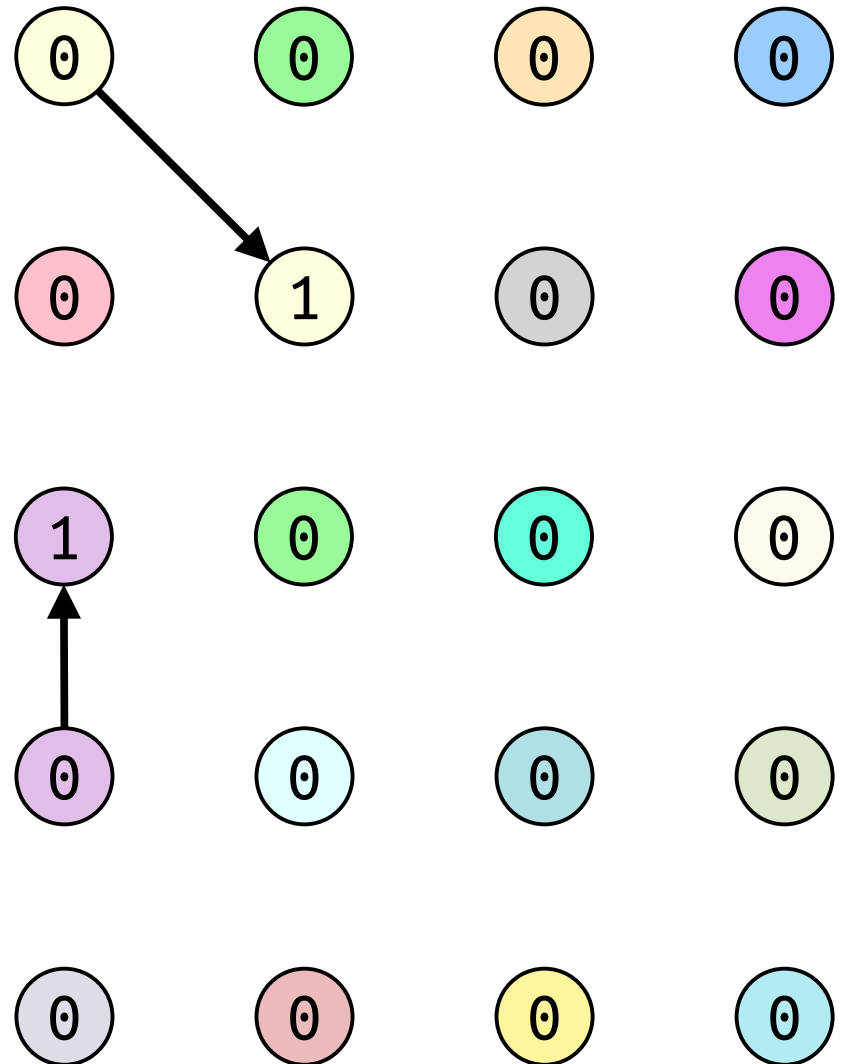
Union-By-Rank

- 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.



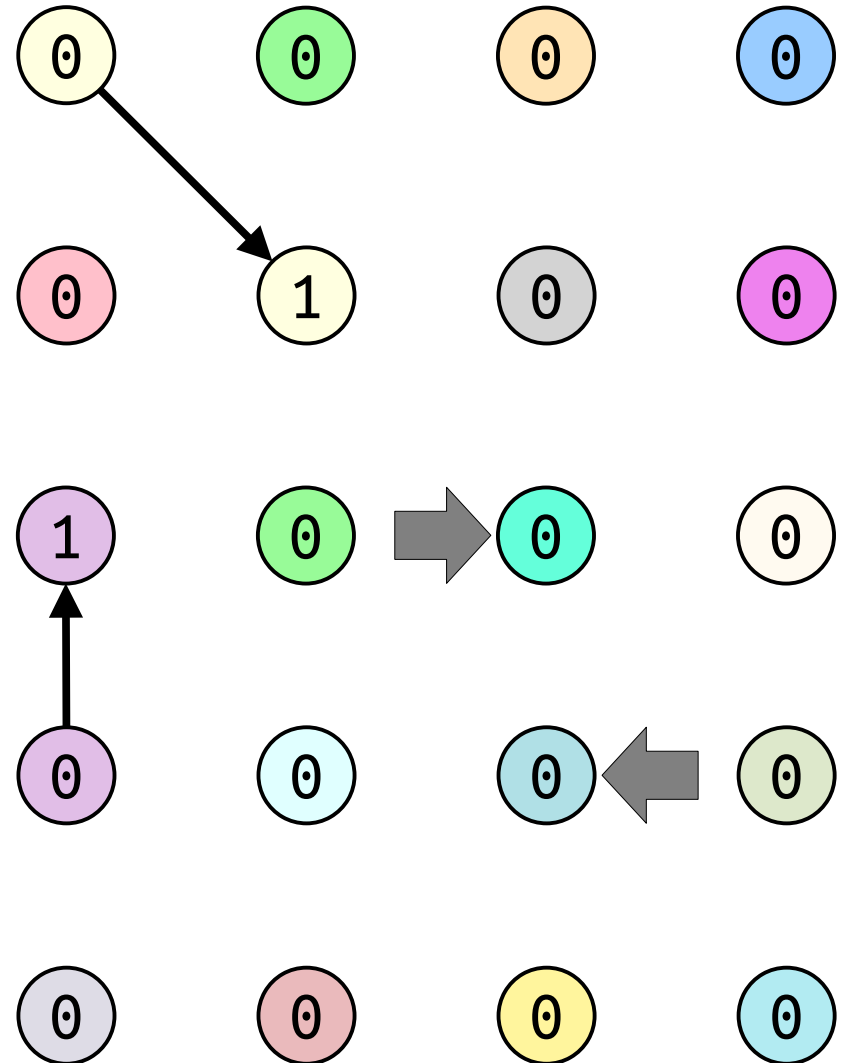
Union-By-Rank

- 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.



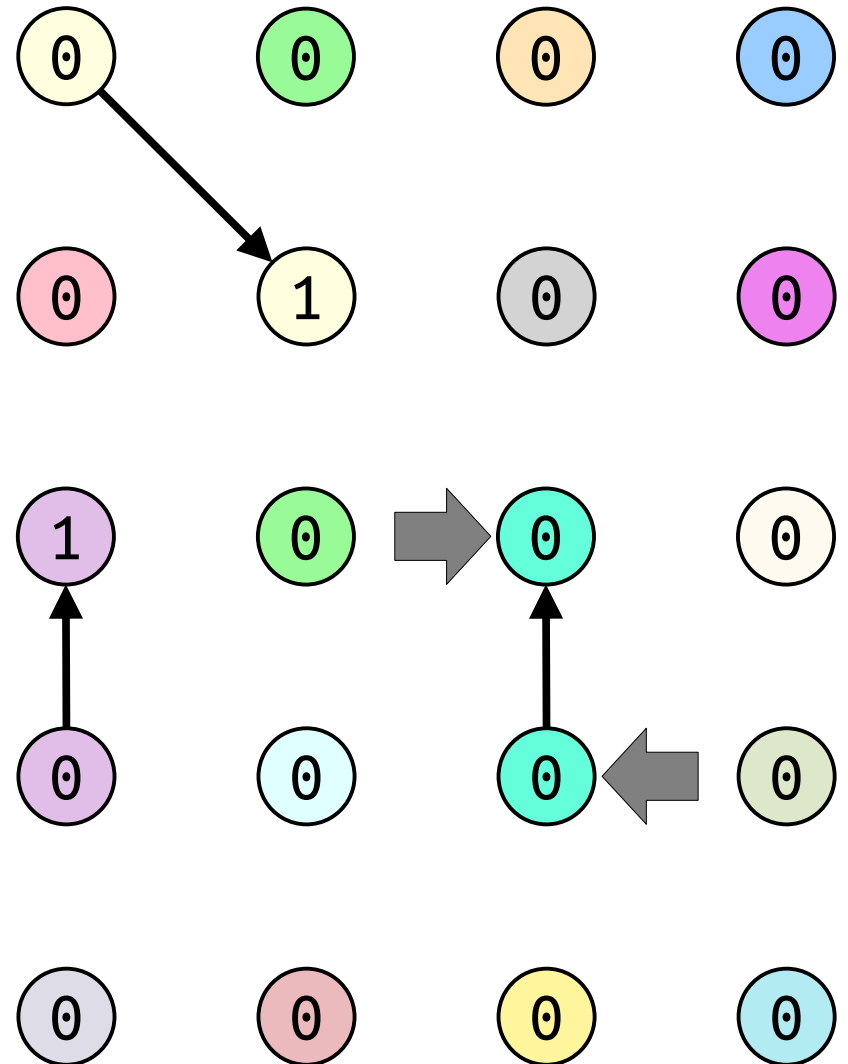
Union-By-Rank

- 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.



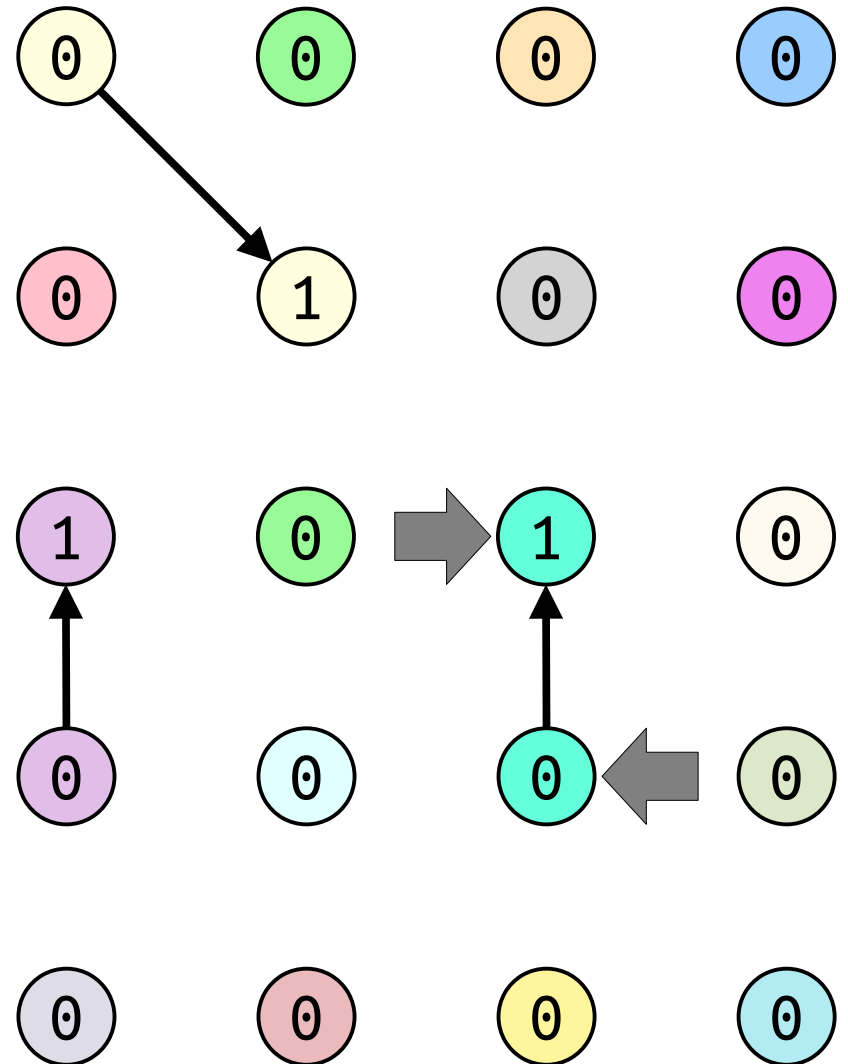
Union-By-Rank

- 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.



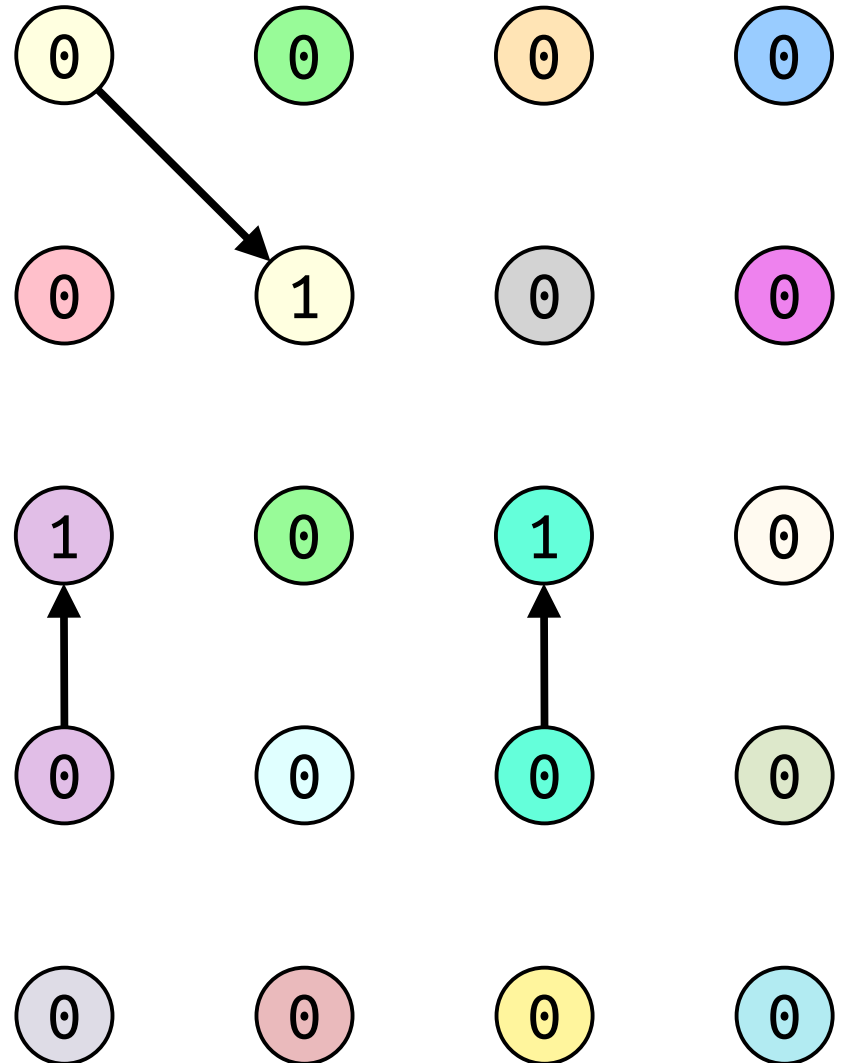
Union-By-Rank

- 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.



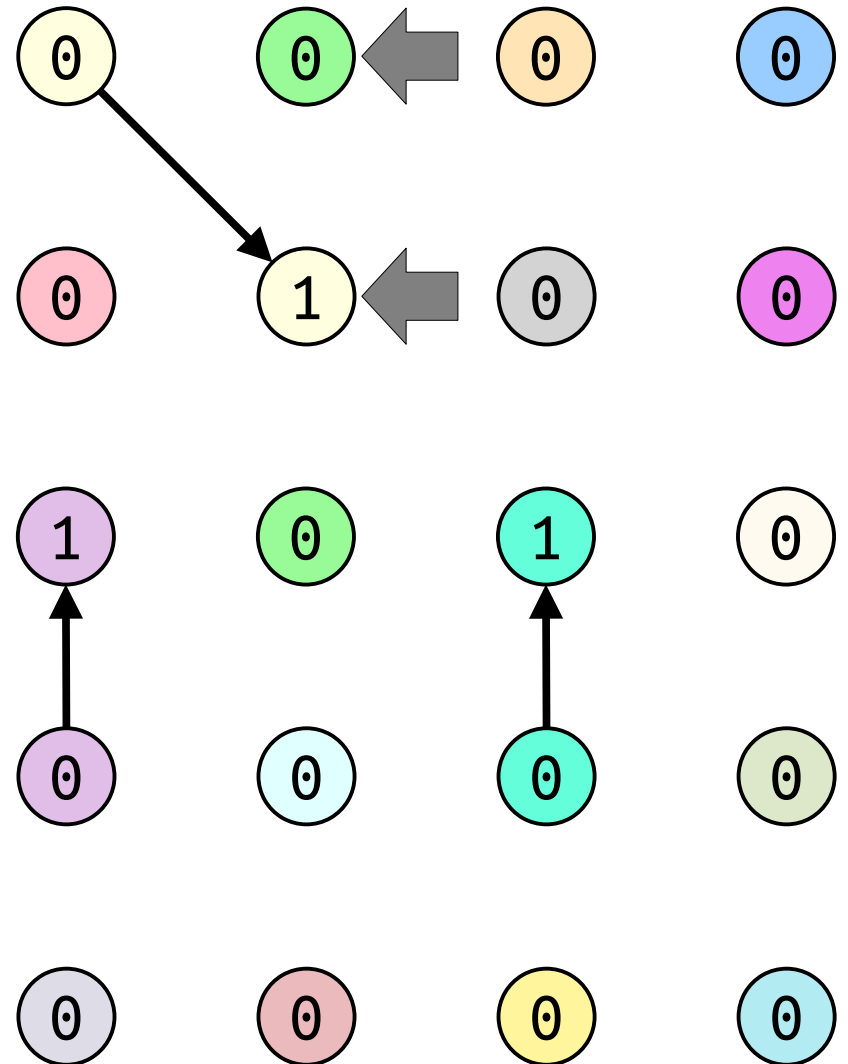
Union-By-Rank

- 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.



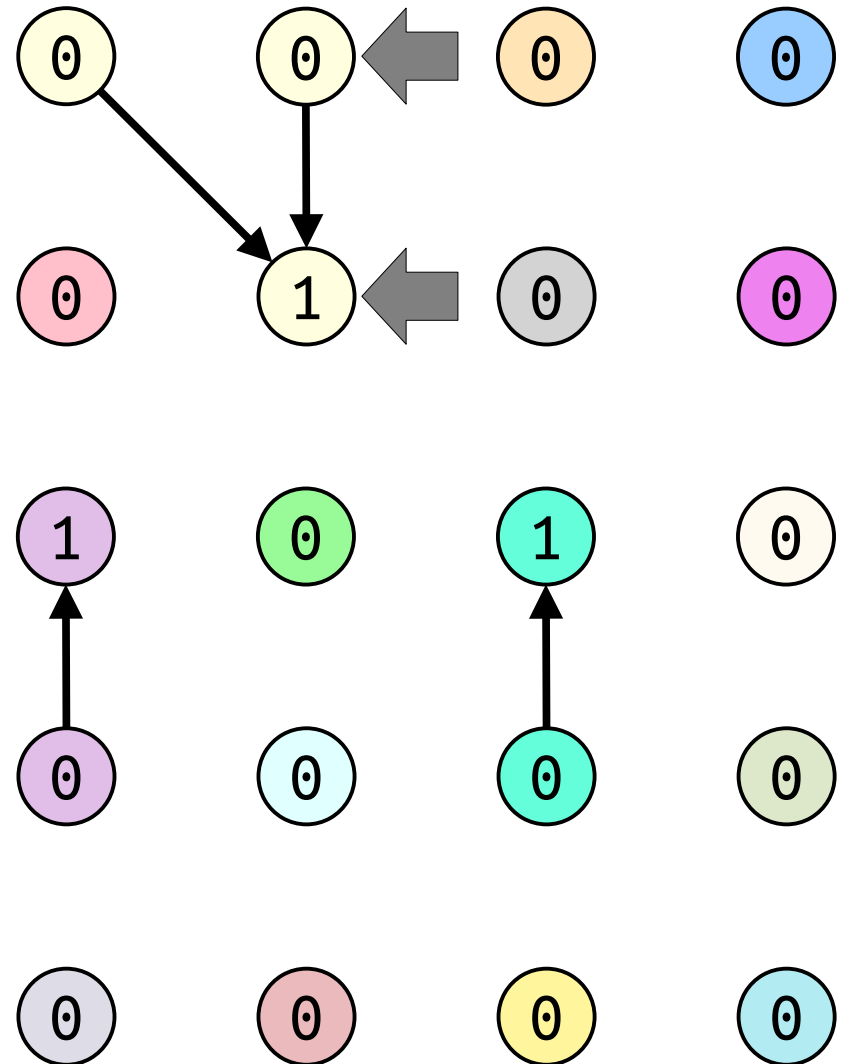
Union-By-Rank

- 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.



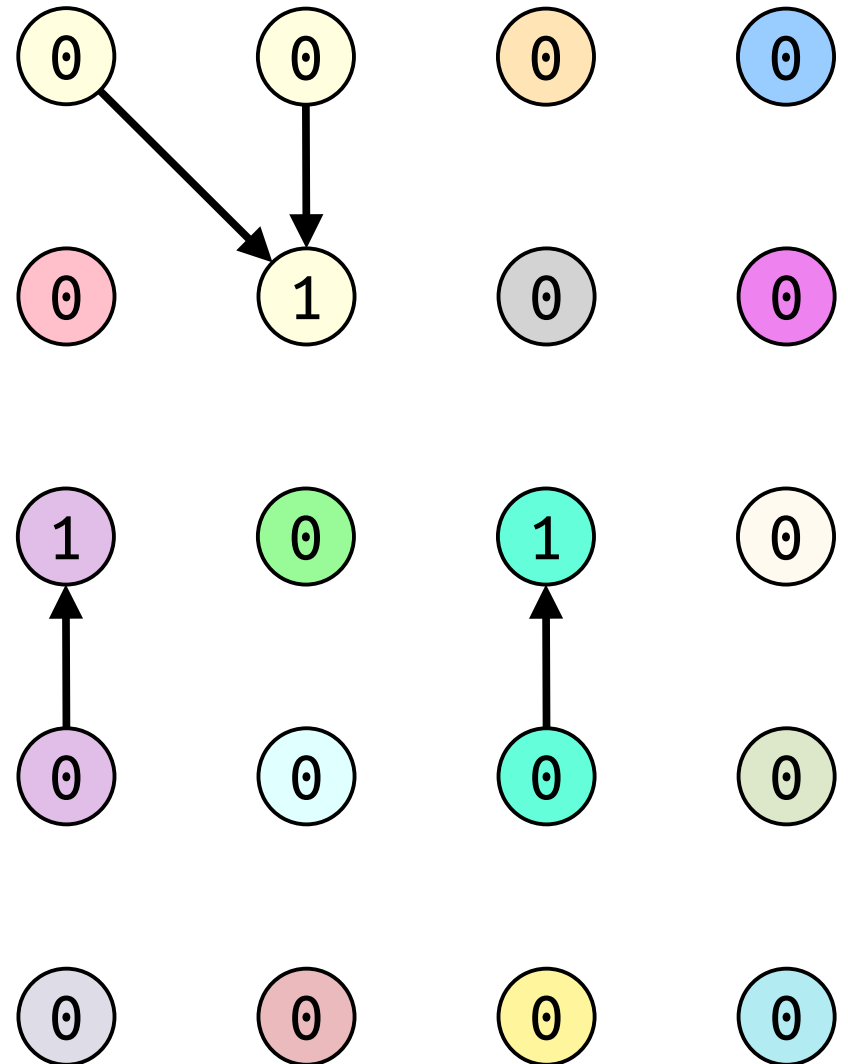
Union-By-Rank

- 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.



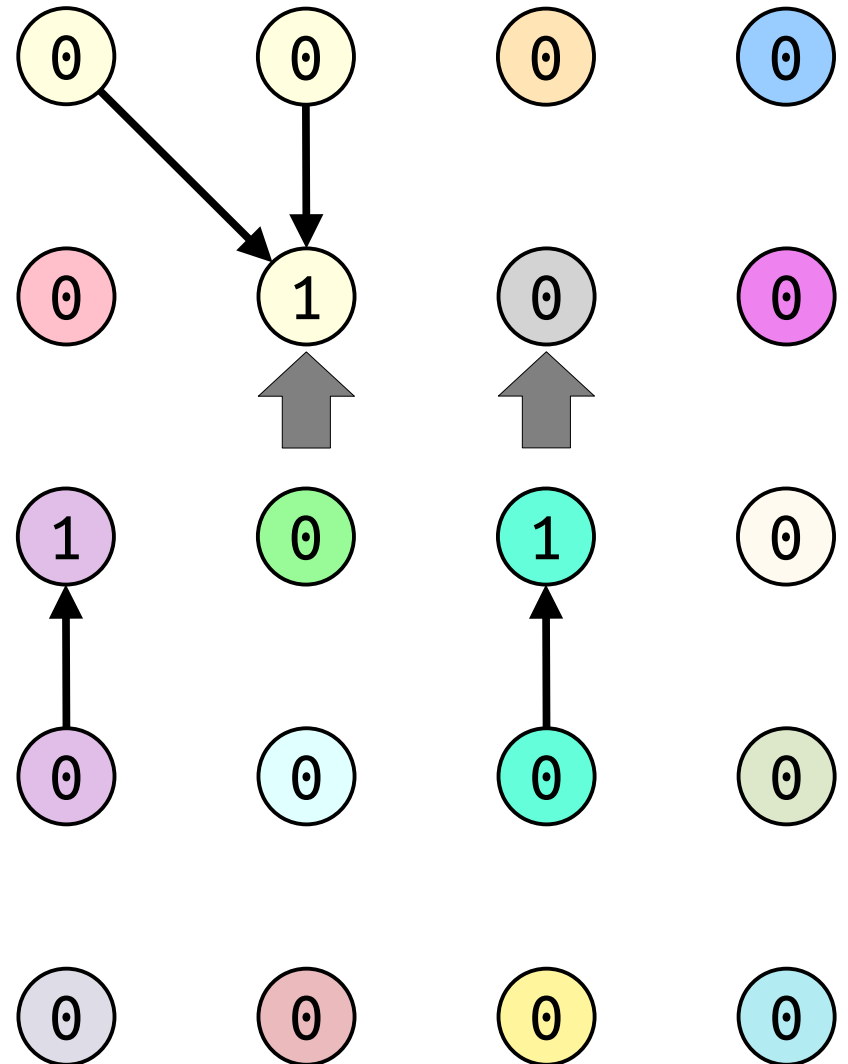
Union-By-Rank

- 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.



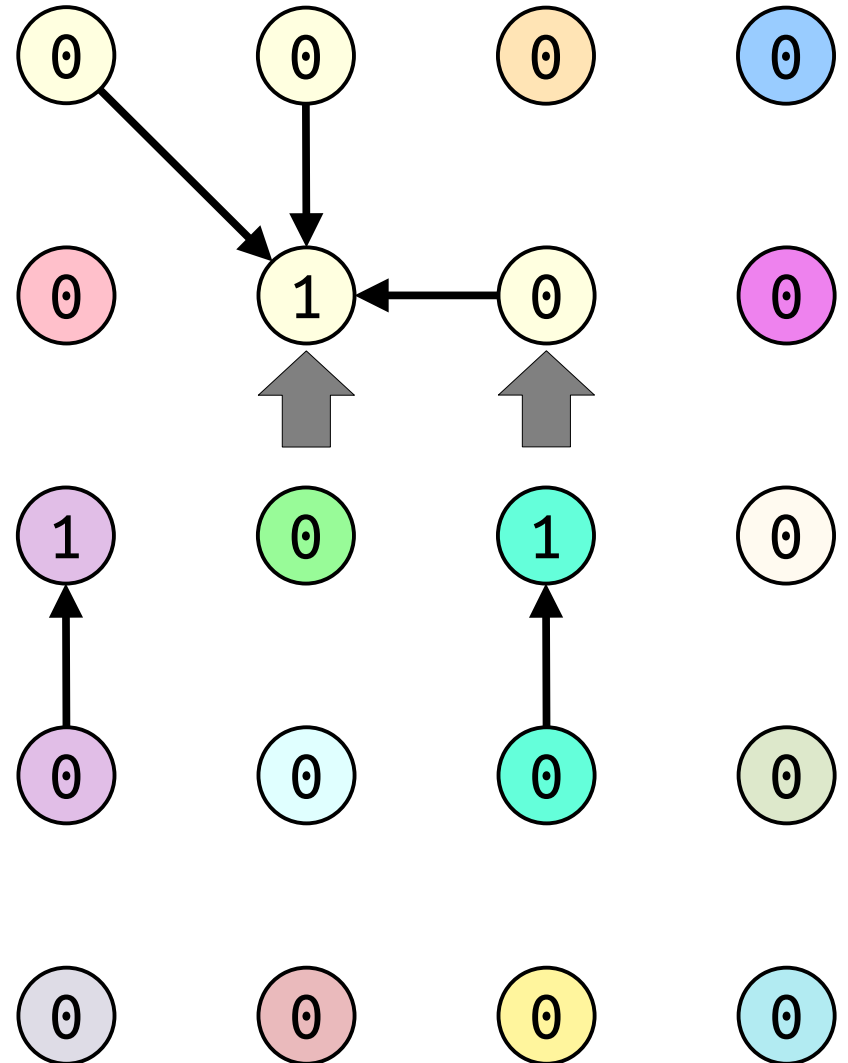
Union-By-Rank

- 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.



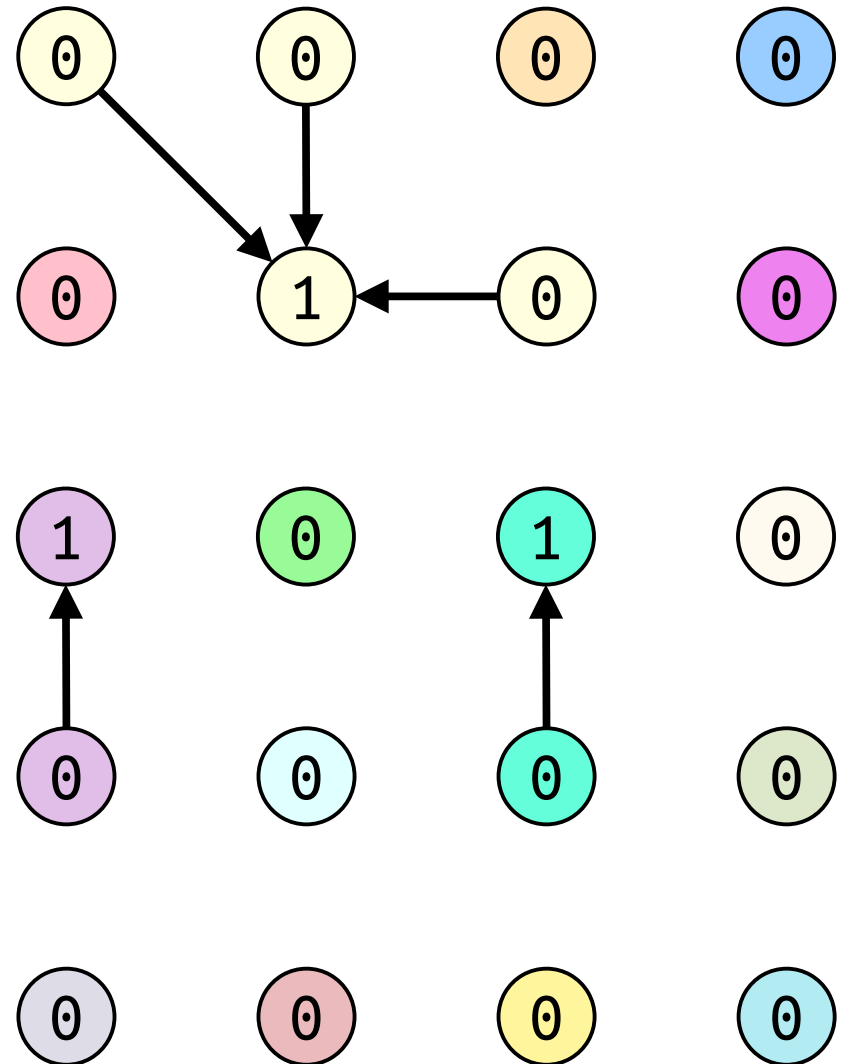
Union-By-Rank

- 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.



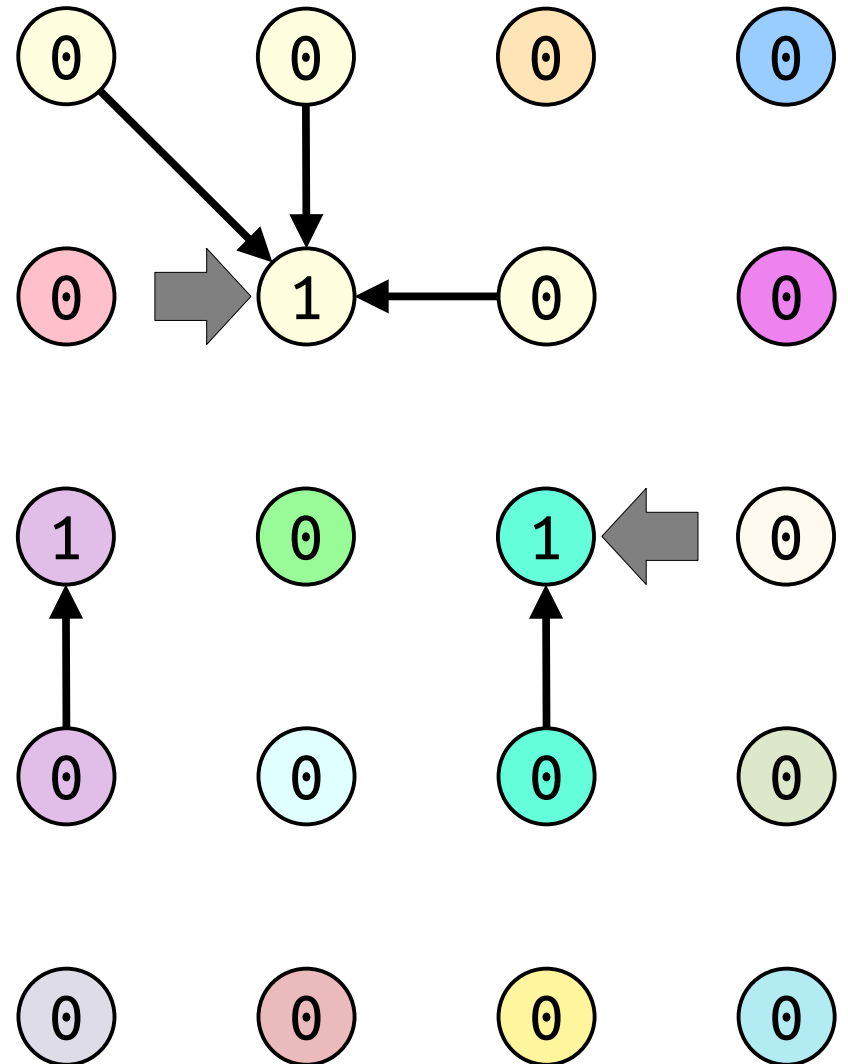
Union-By-Rank

- 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.



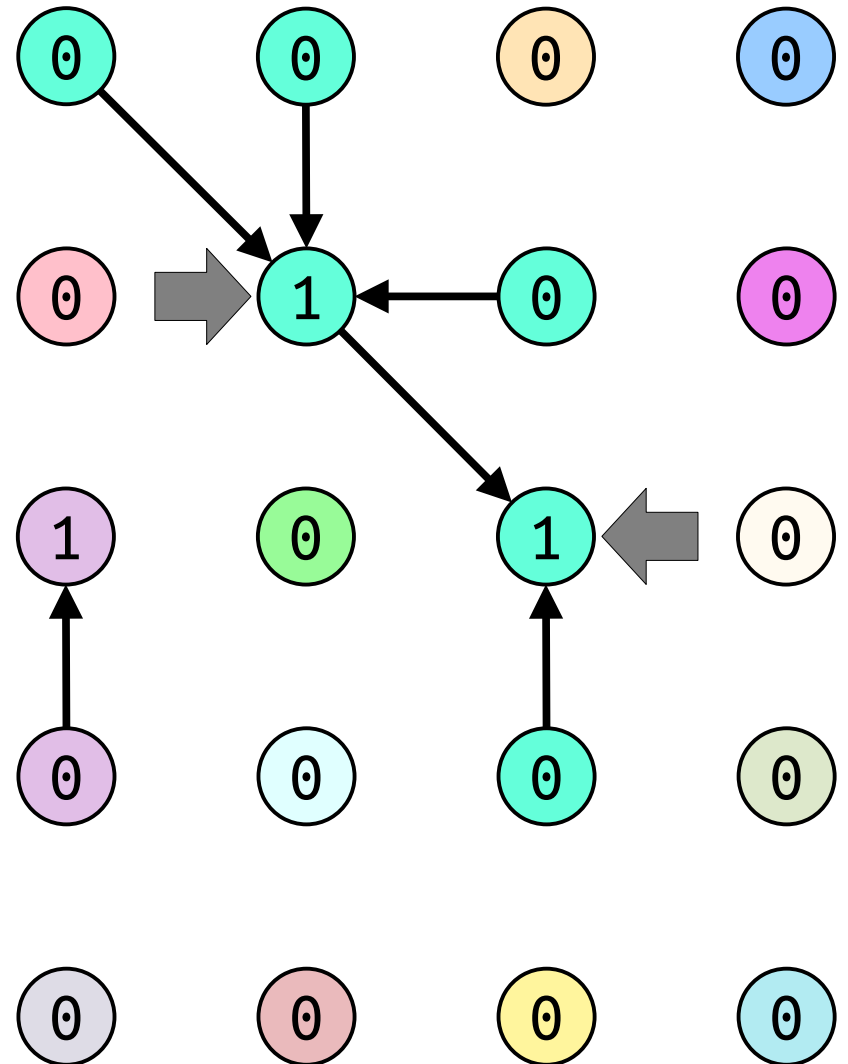
Union-By-Rank

- 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.



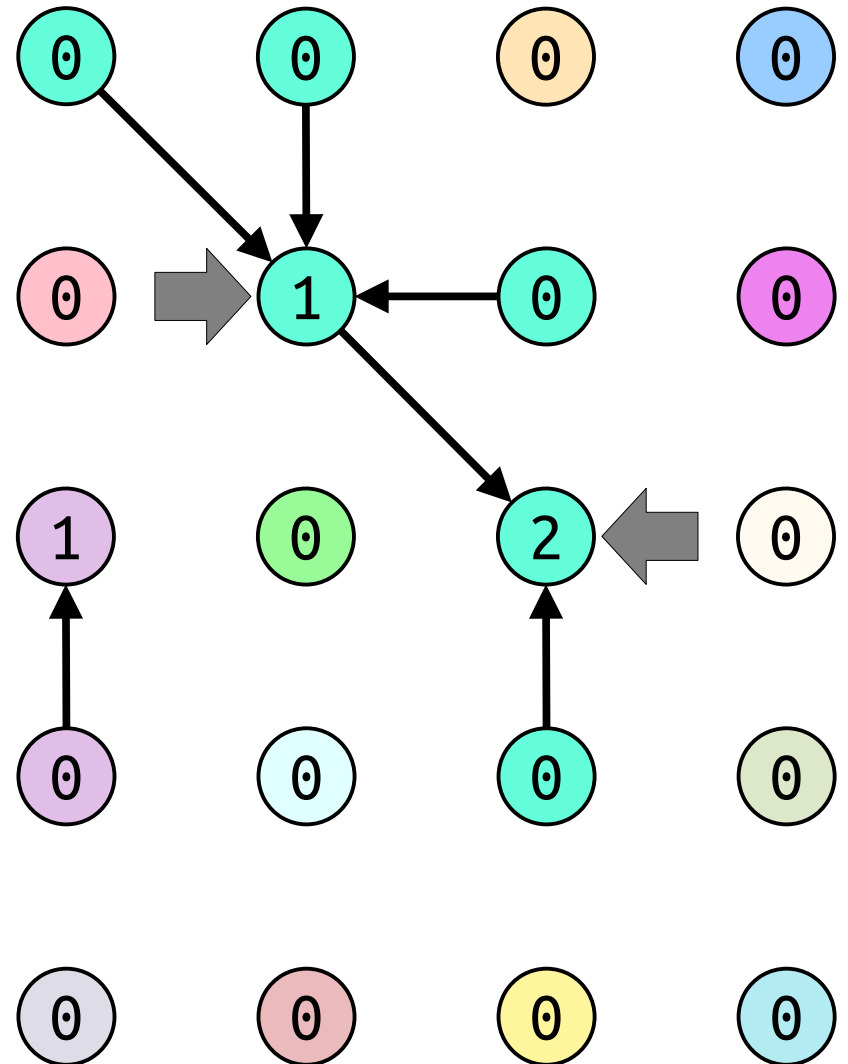
Union-By-Rank

- 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.



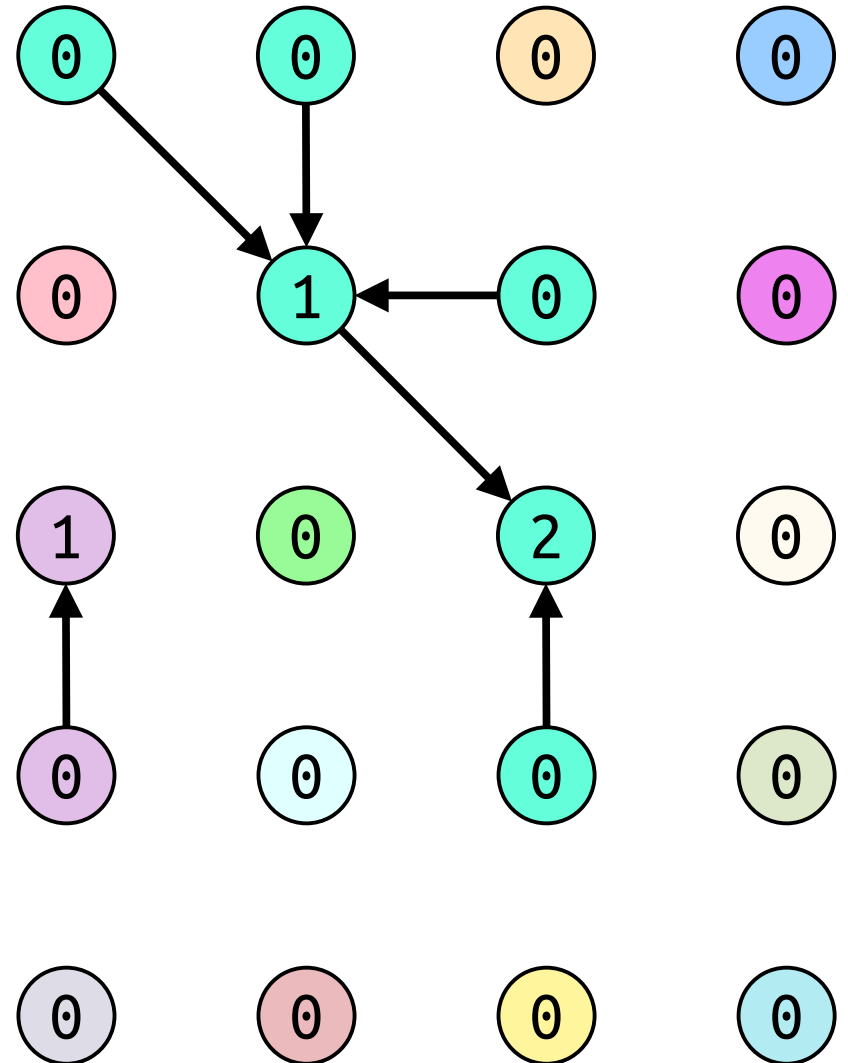
Union-By-Rank

- 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.



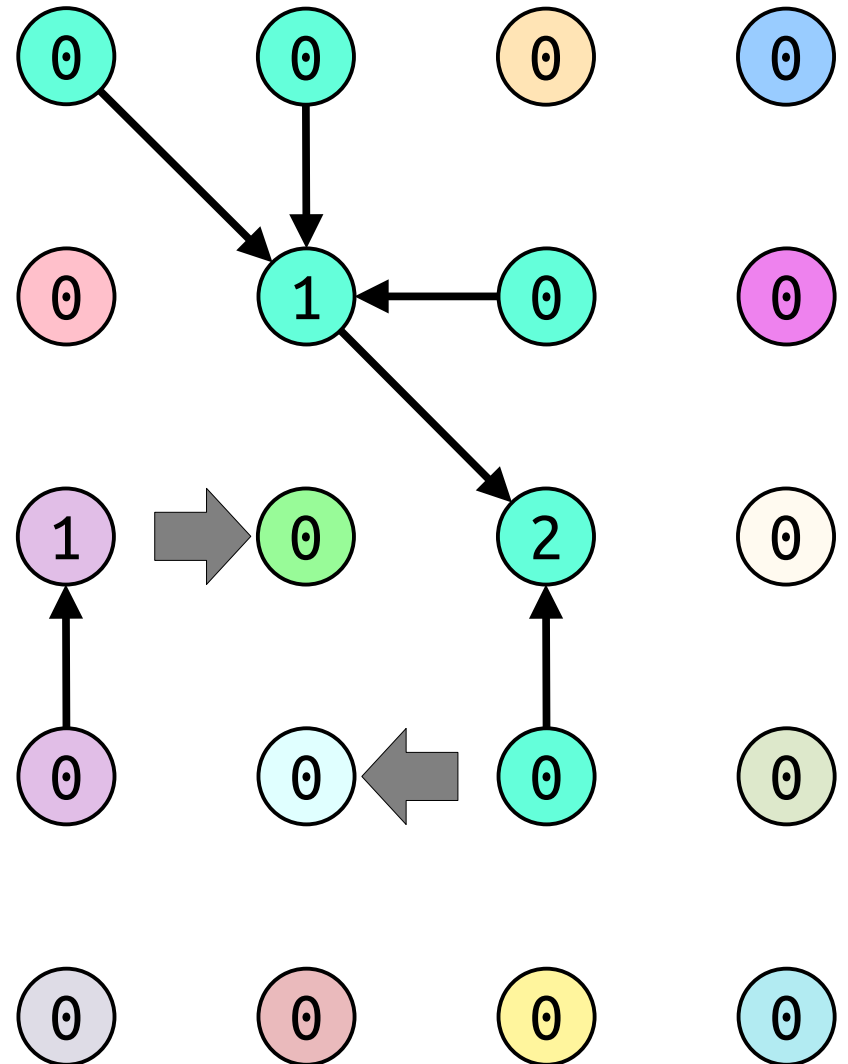
Union-By-Rank

- 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.



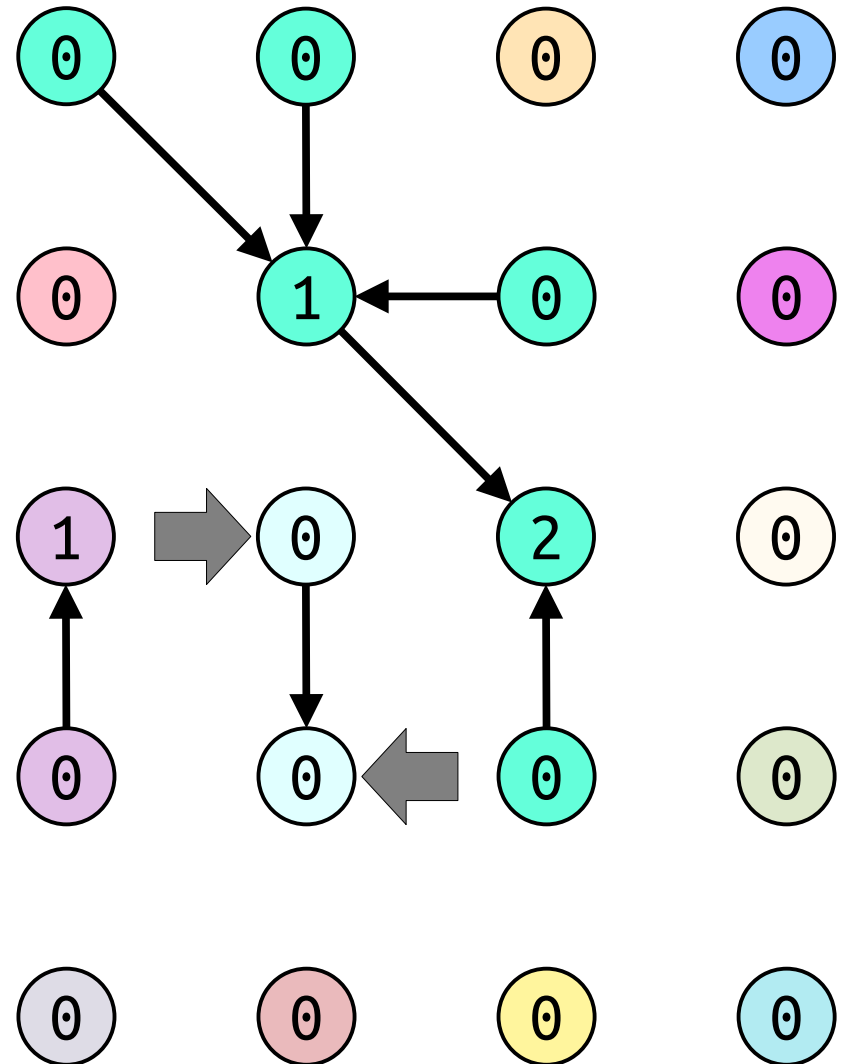
Union-By-Rank

- 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.



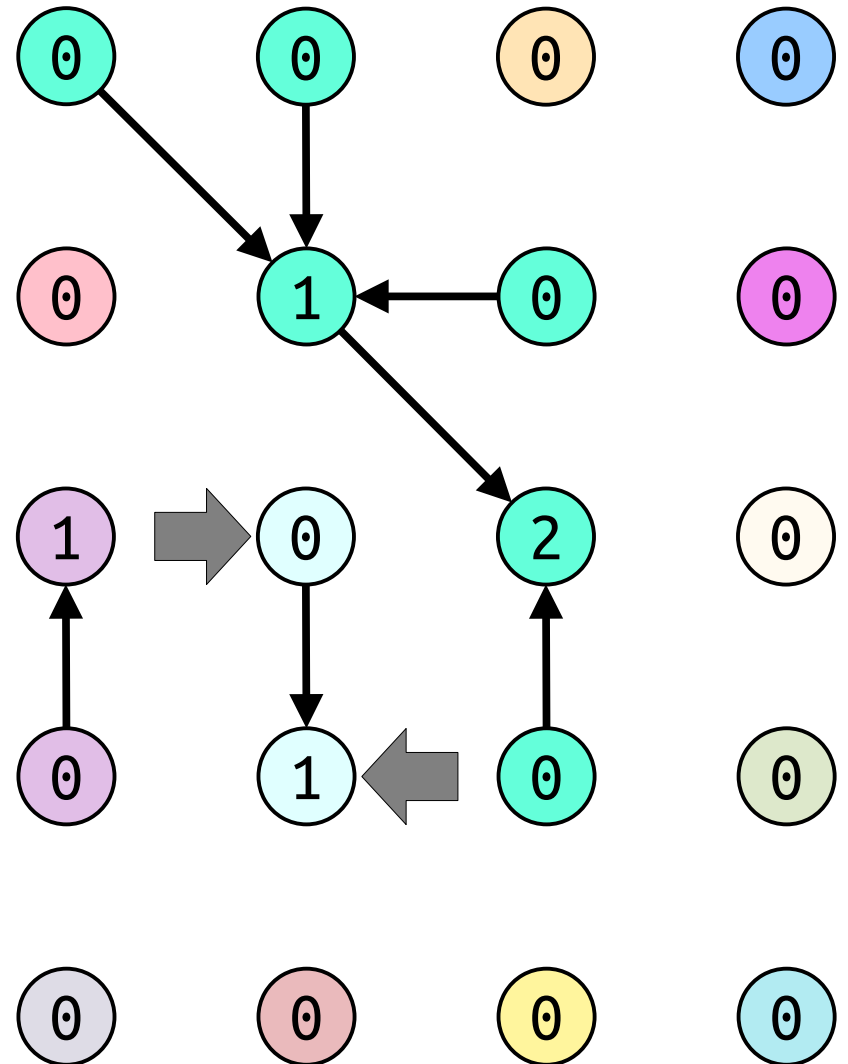
Union-By-Rank

- 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.



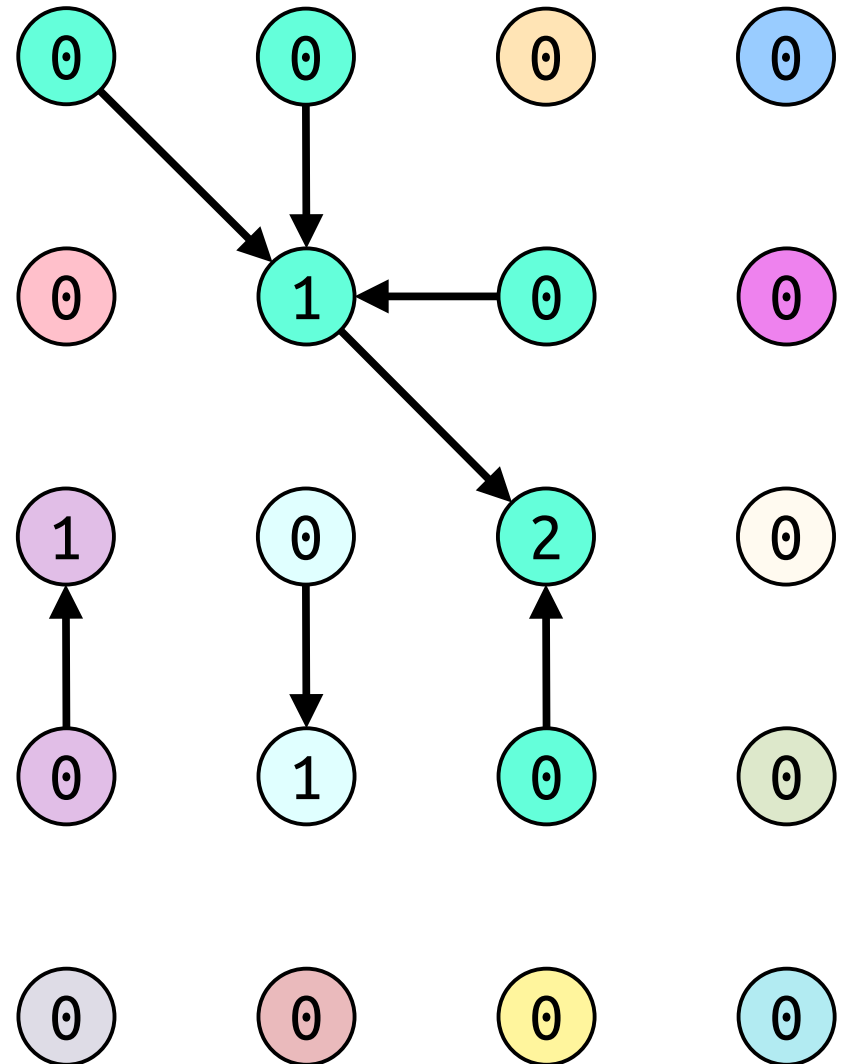
Union-By-Rank

- 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.



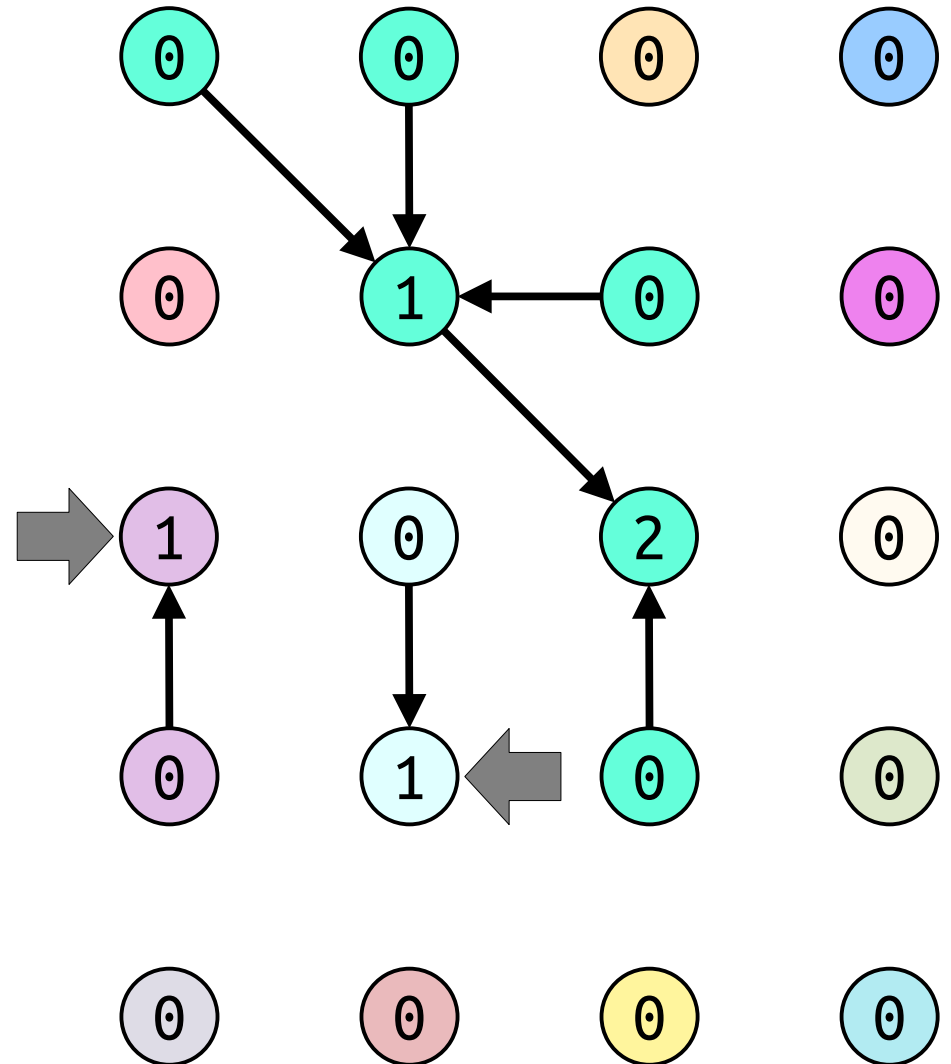
Union-By-Rank

- 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.



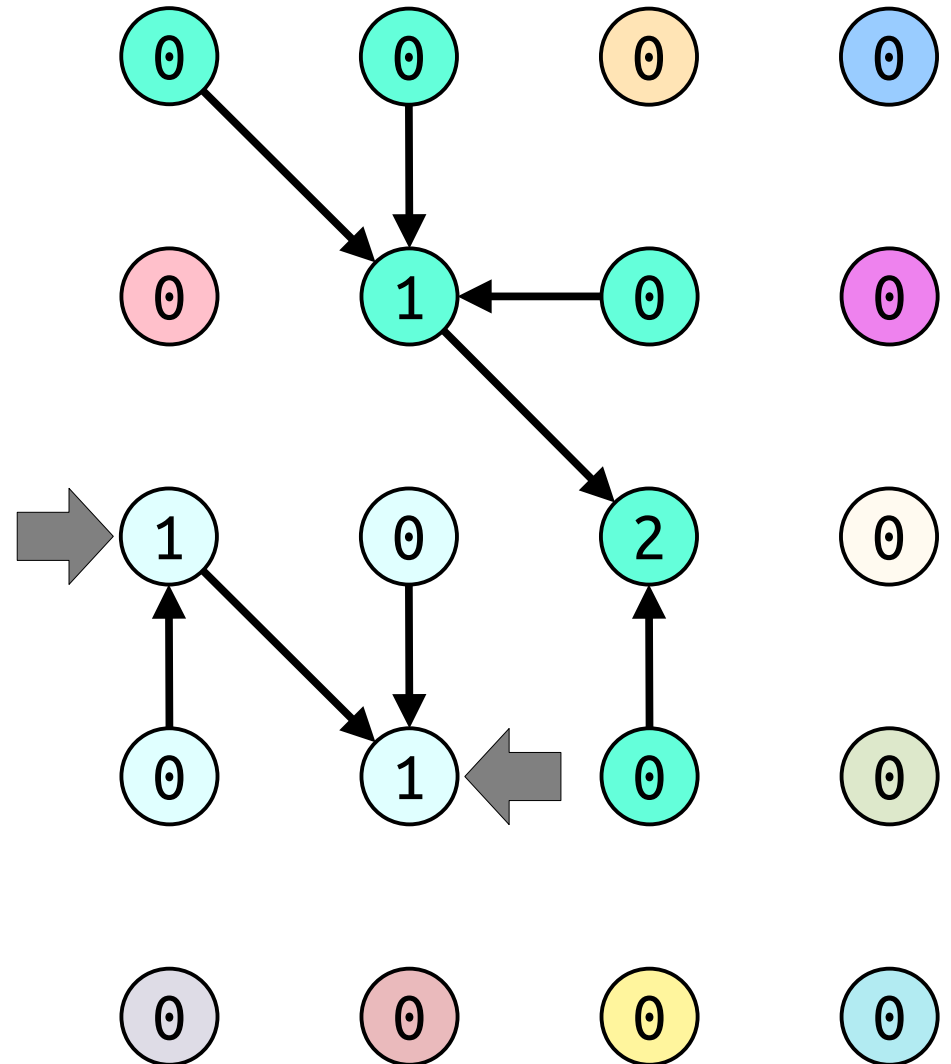
Union-By-Rank

- 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.



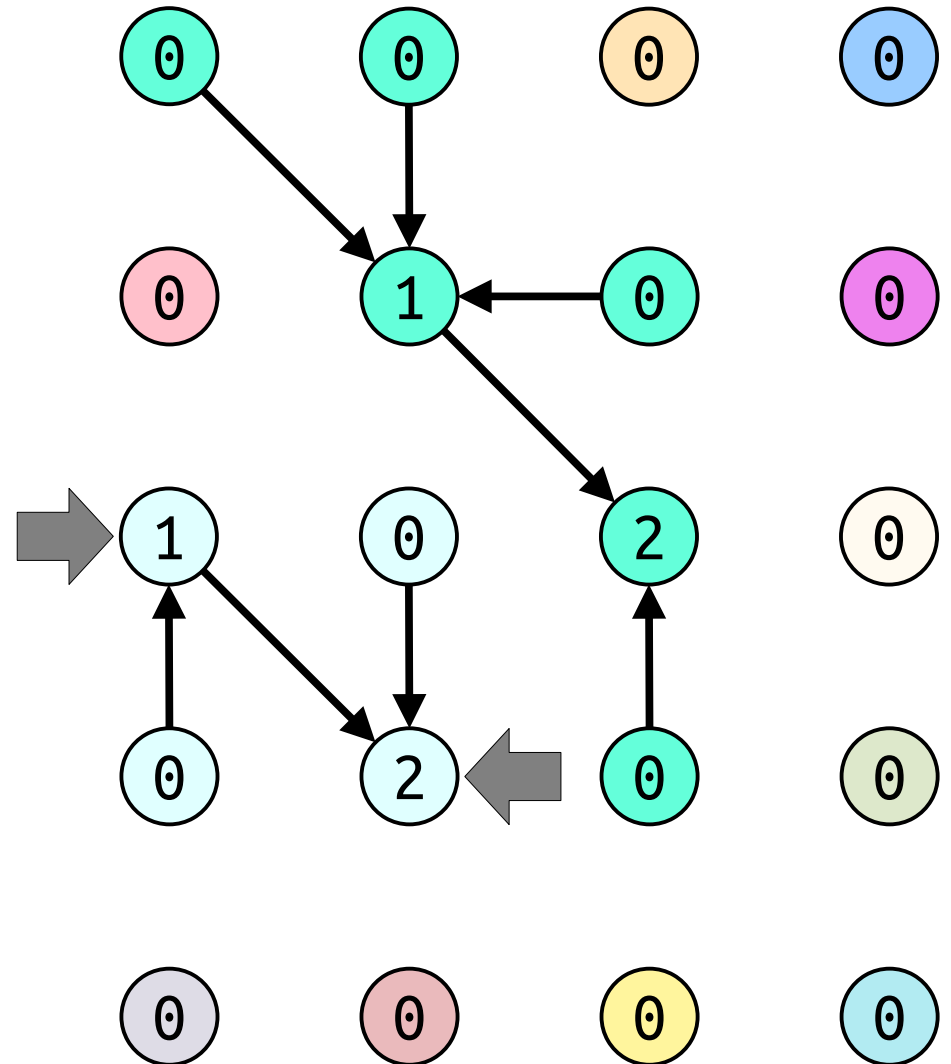
Union-By-Rank

- 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.



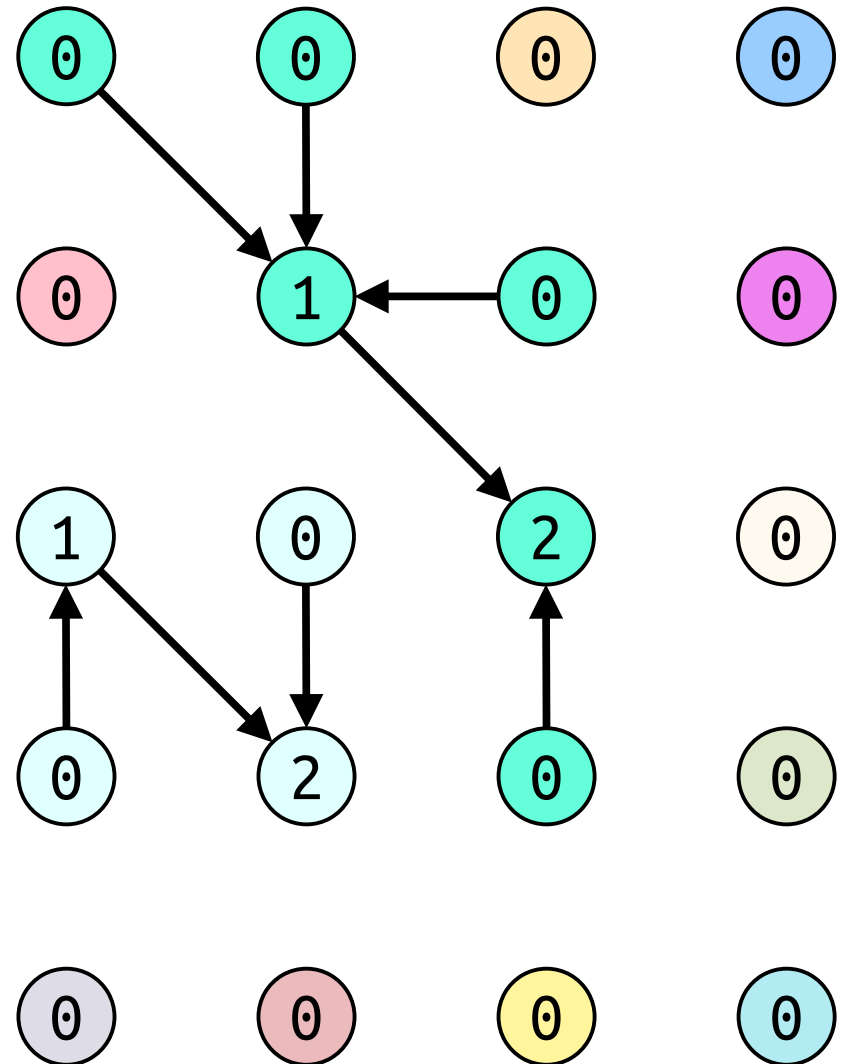
Union-By-Rank

- 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.



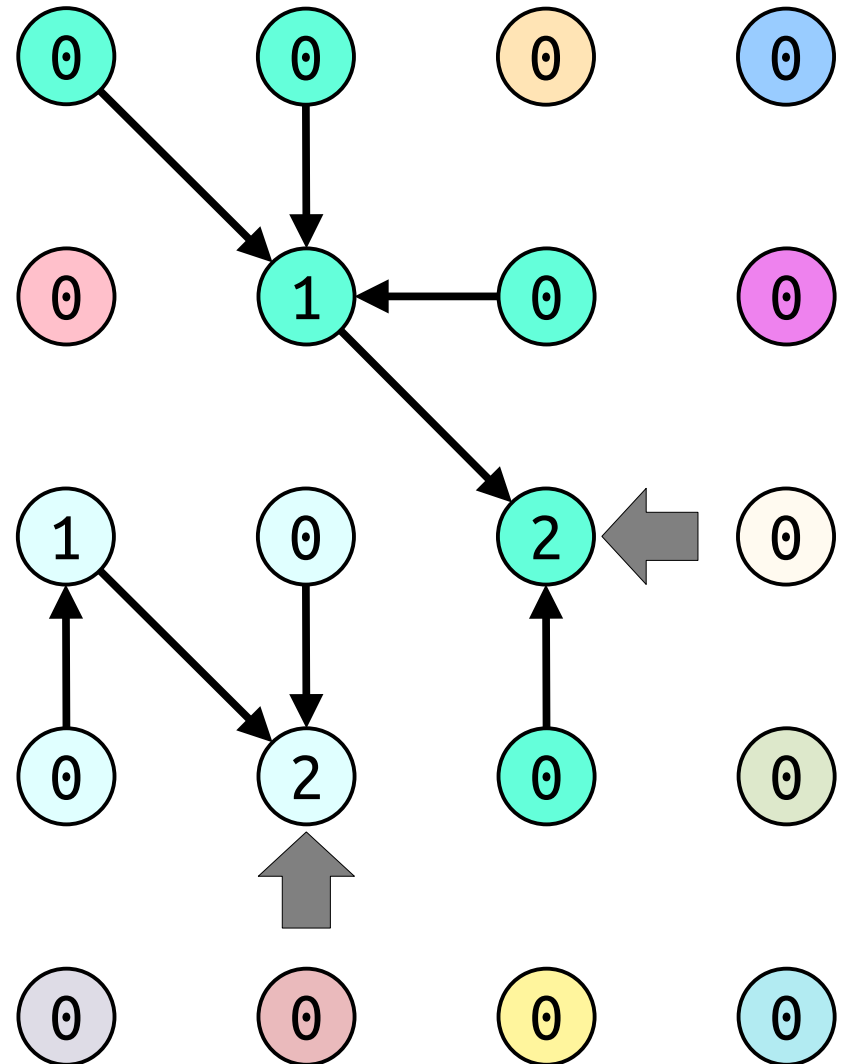
Union-By-Rank

- 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.



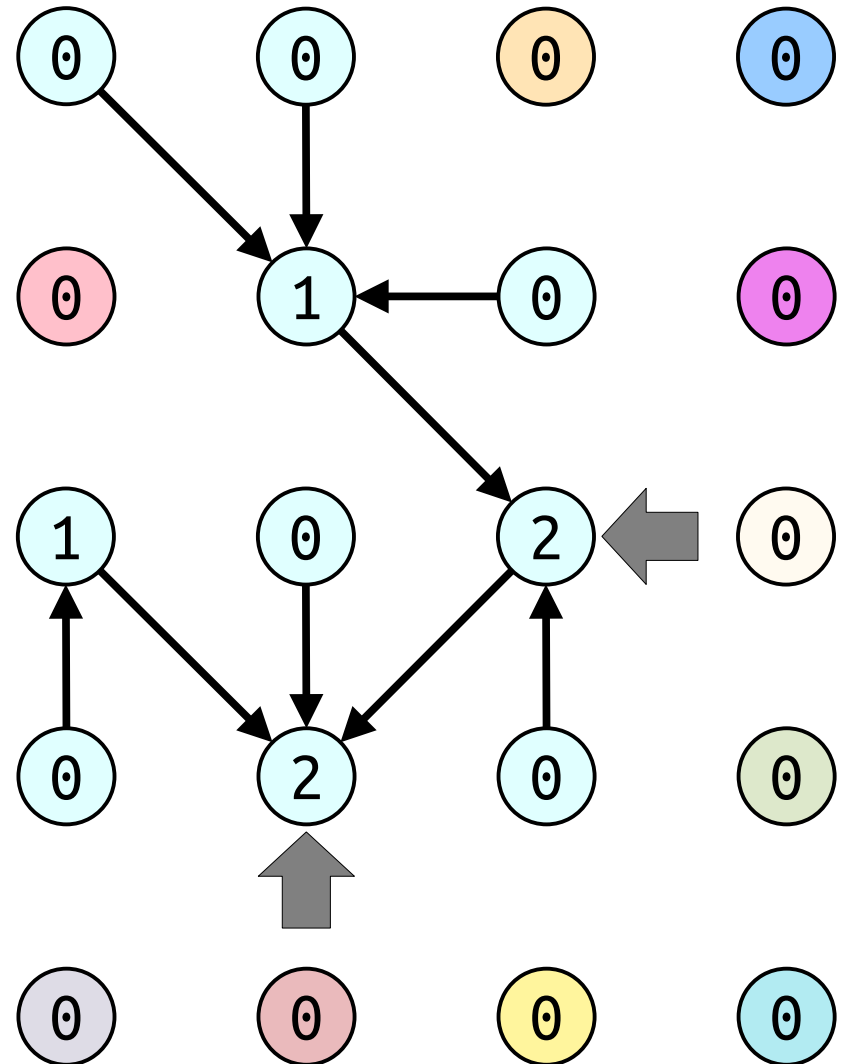
Union-By-Rank

- 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.



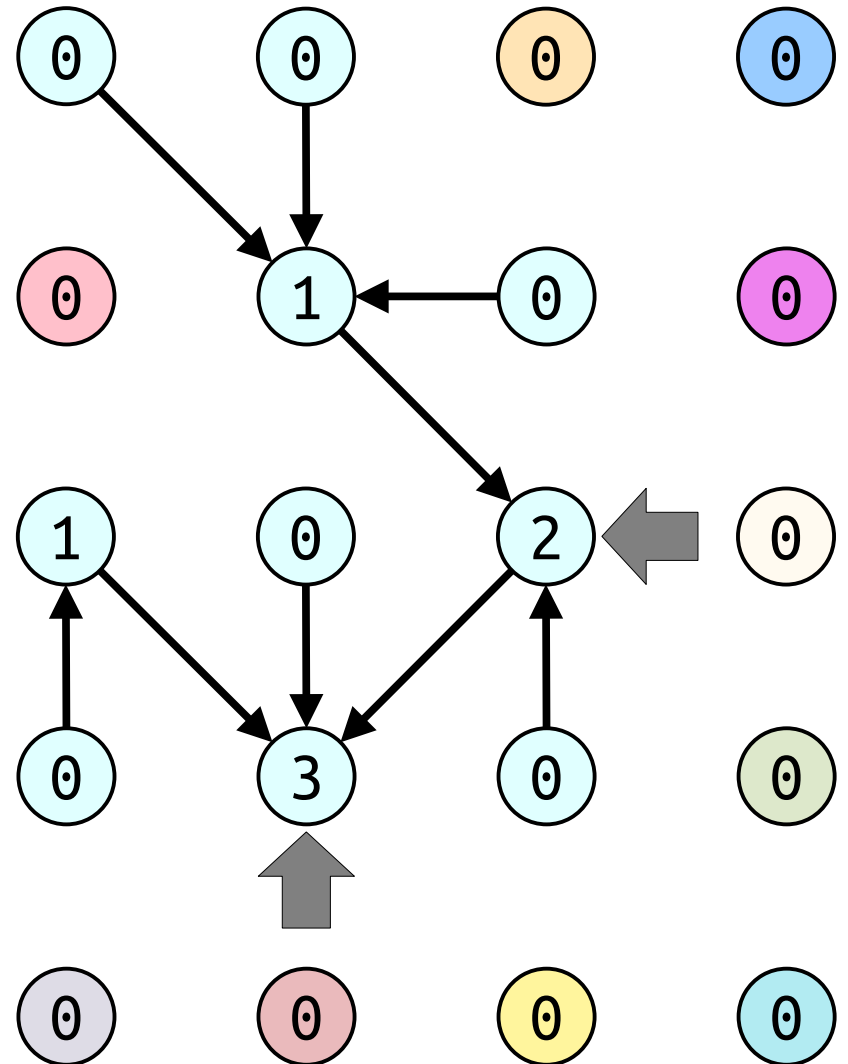
Union-By-Rank

- 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.



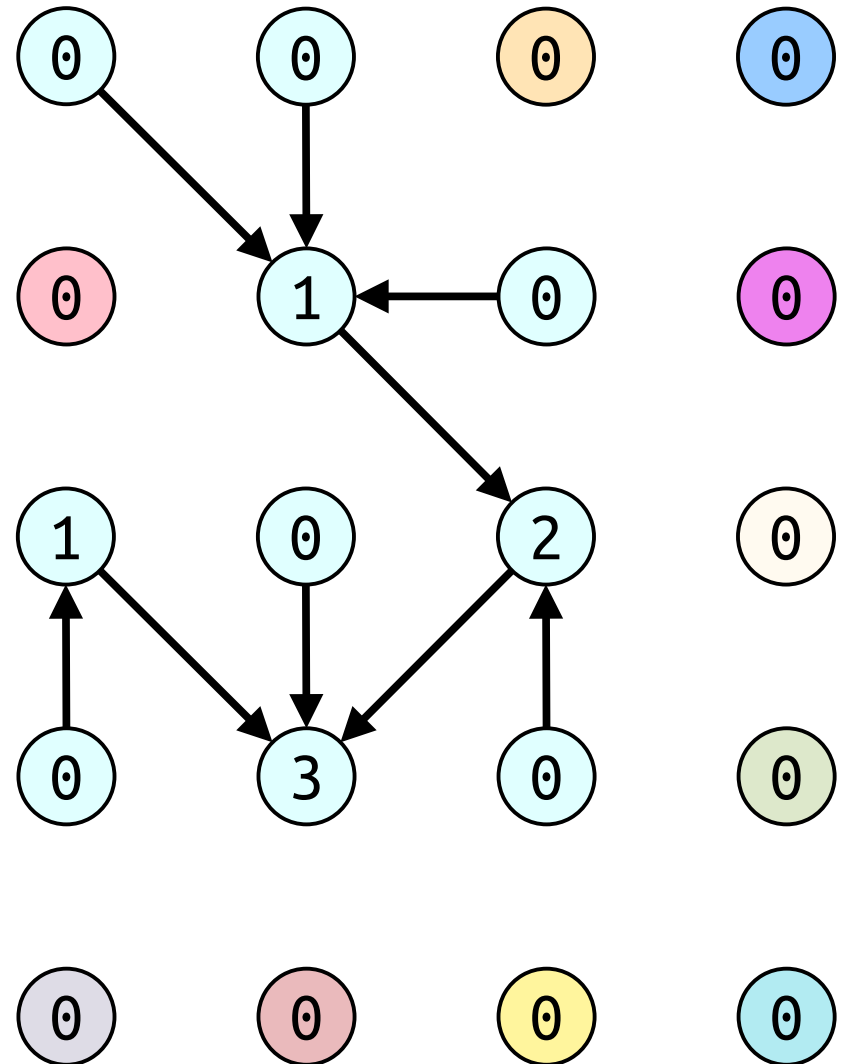
Union-By-Rank

- 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.



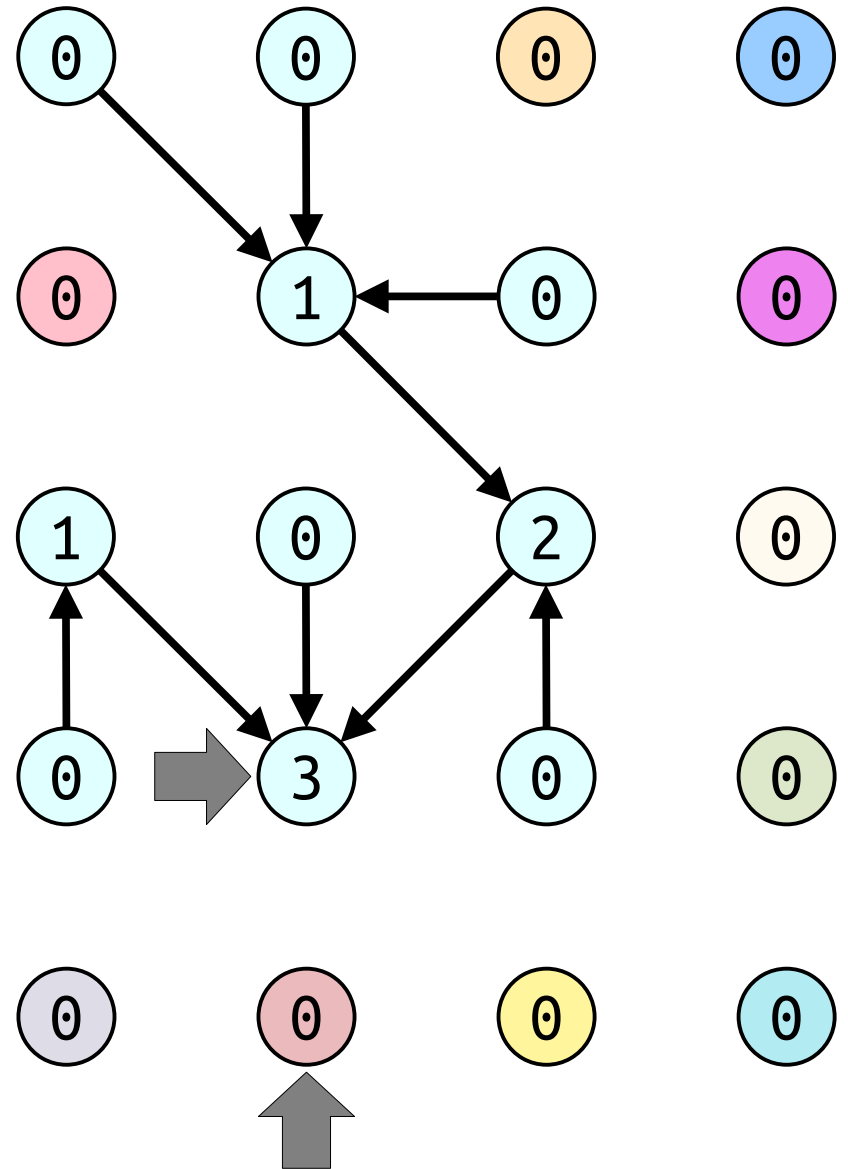
Union-By-Rank

- 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.



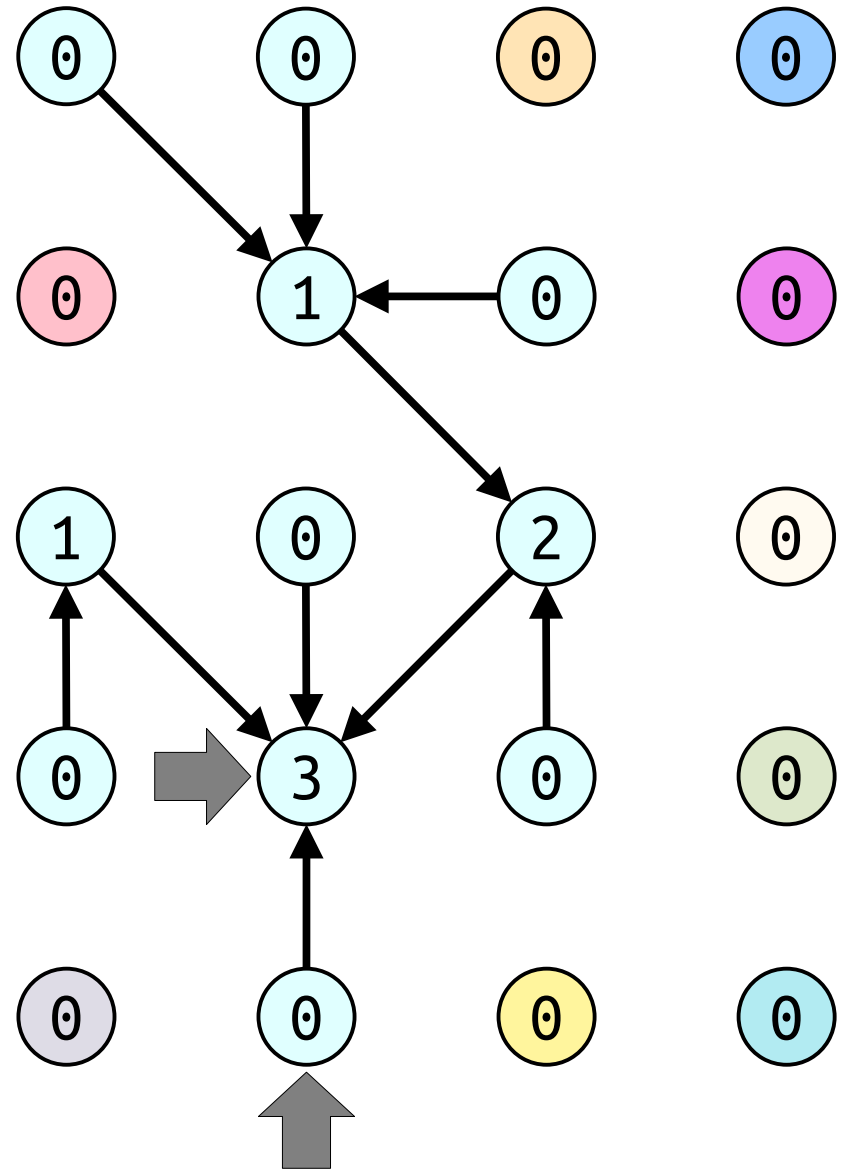
Union-By-Rank

- 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.



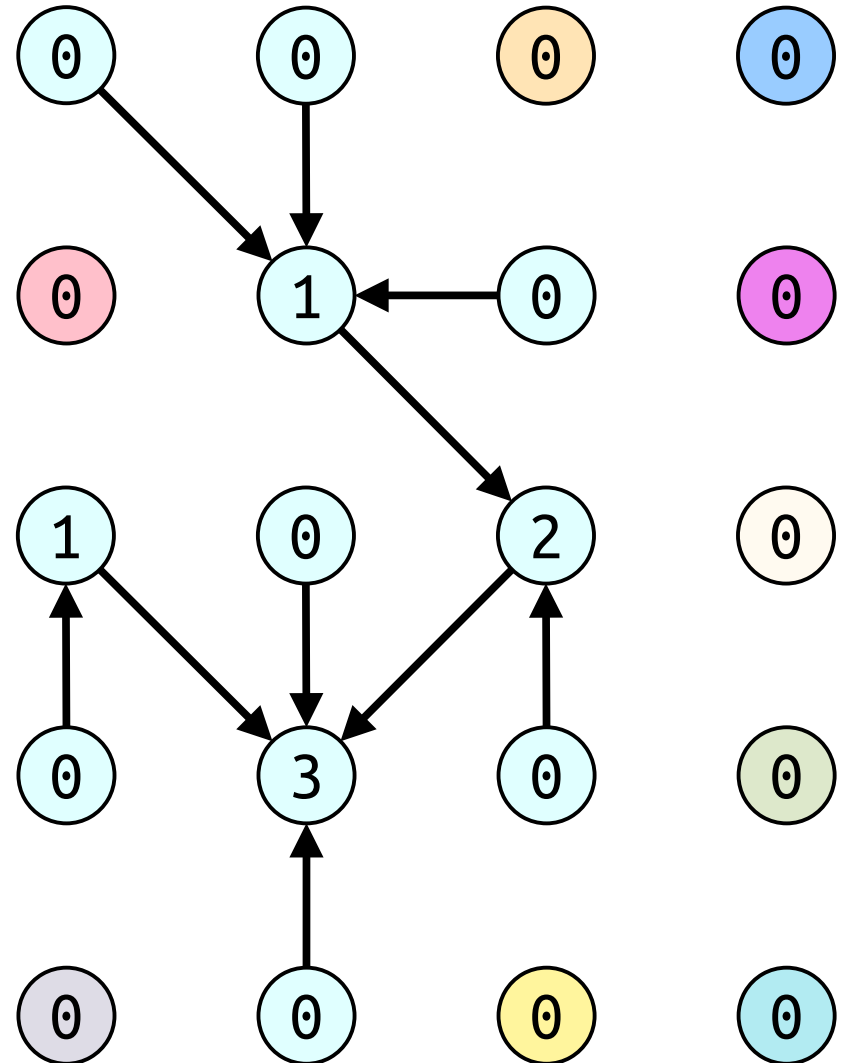
Union-By-Rank

- 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

- 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.



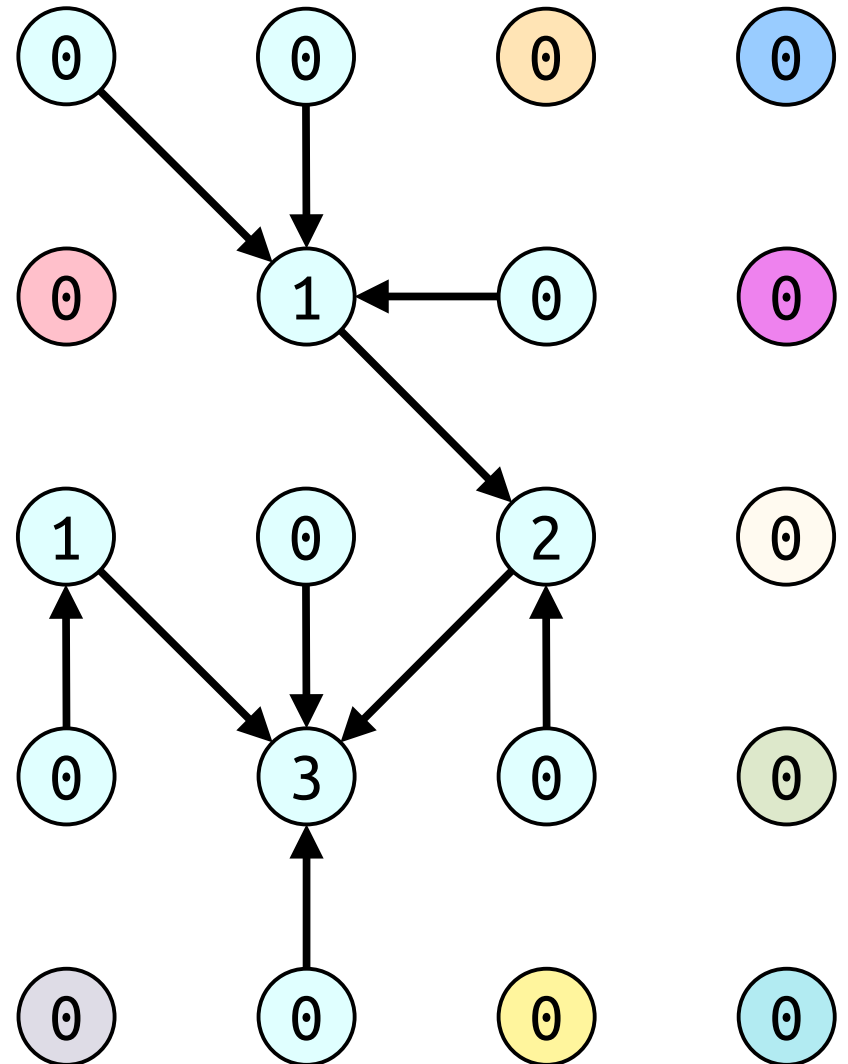
Union-By-Rank

- **Lemma:** A node of rank r has children of ranks $0, 1, 2, \dots$, and $r - 1$.

Why?

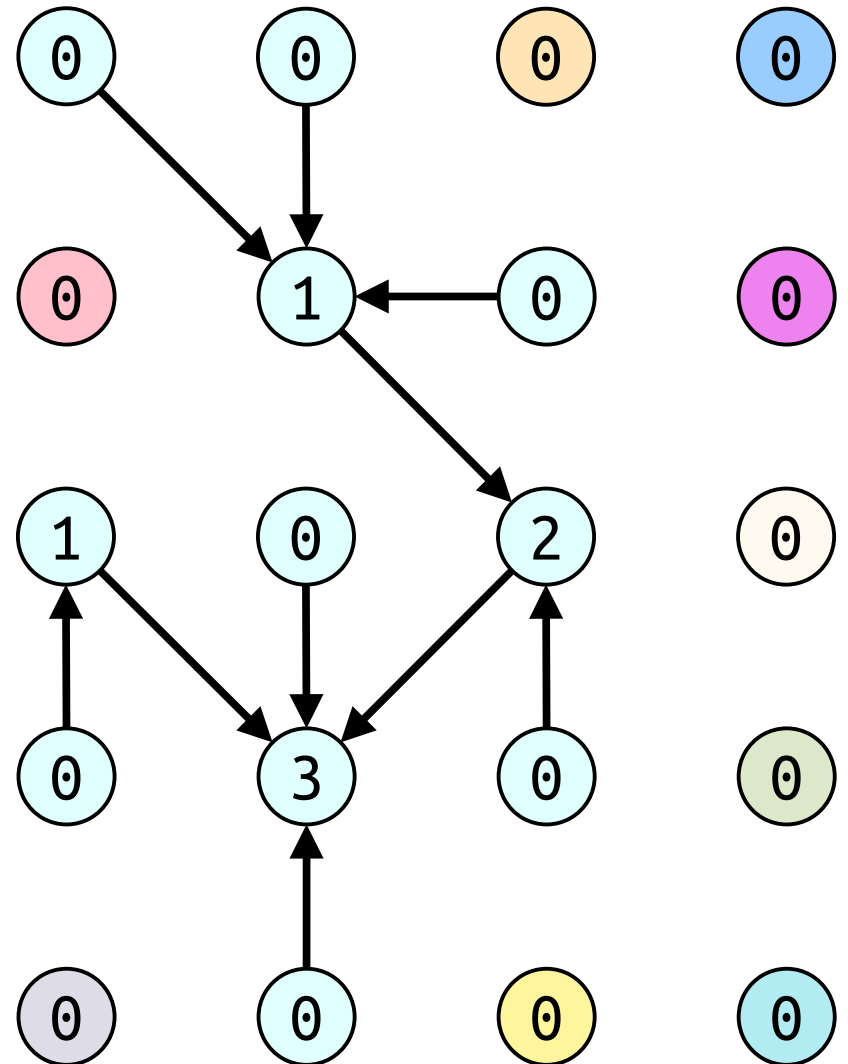
Answer at

<https://pollev.com/cs166spr23>



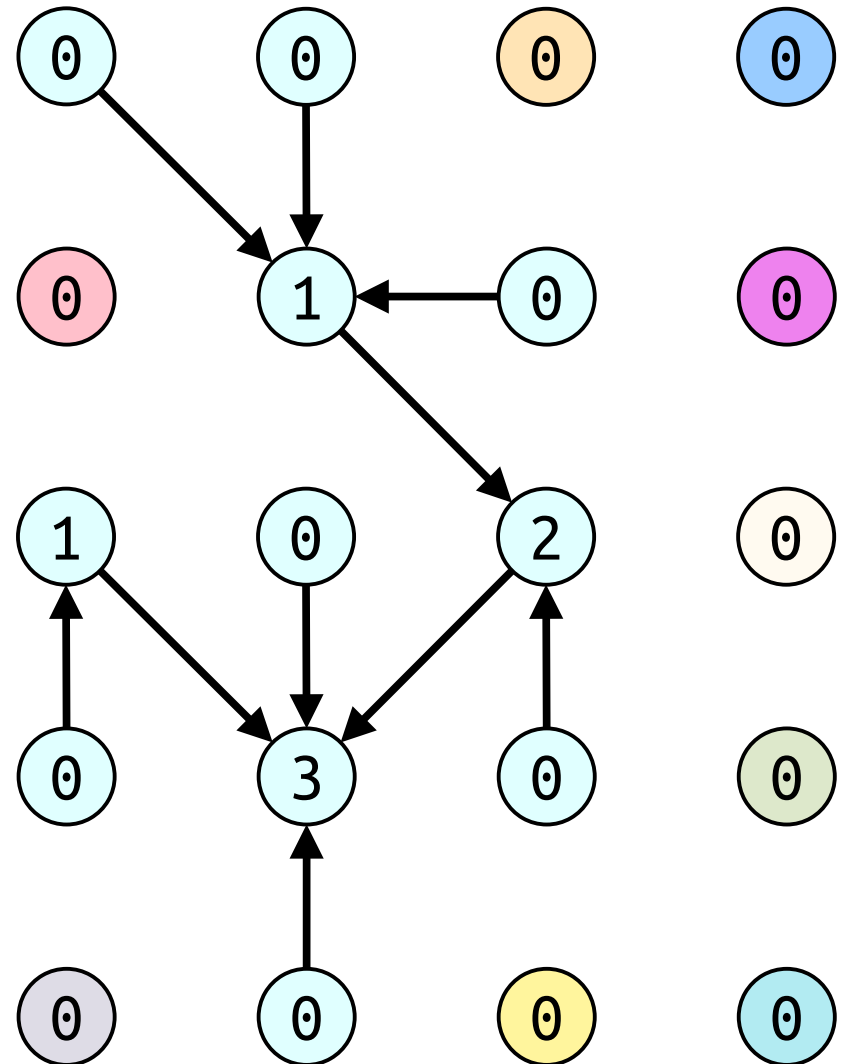
Union-By-Rank

- **Lemma:** A node of rank r has children of ranks $0, 1, 2, \dots,$ and $r - 1$.
- **Proof:** Induction!
 - A node of rank 0 has no children.
 - A node v of rank $r + 1$, at the time its rank was 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 $0, 1, 2, \dots, r - 1$. Now it also has a child of rank r . ■



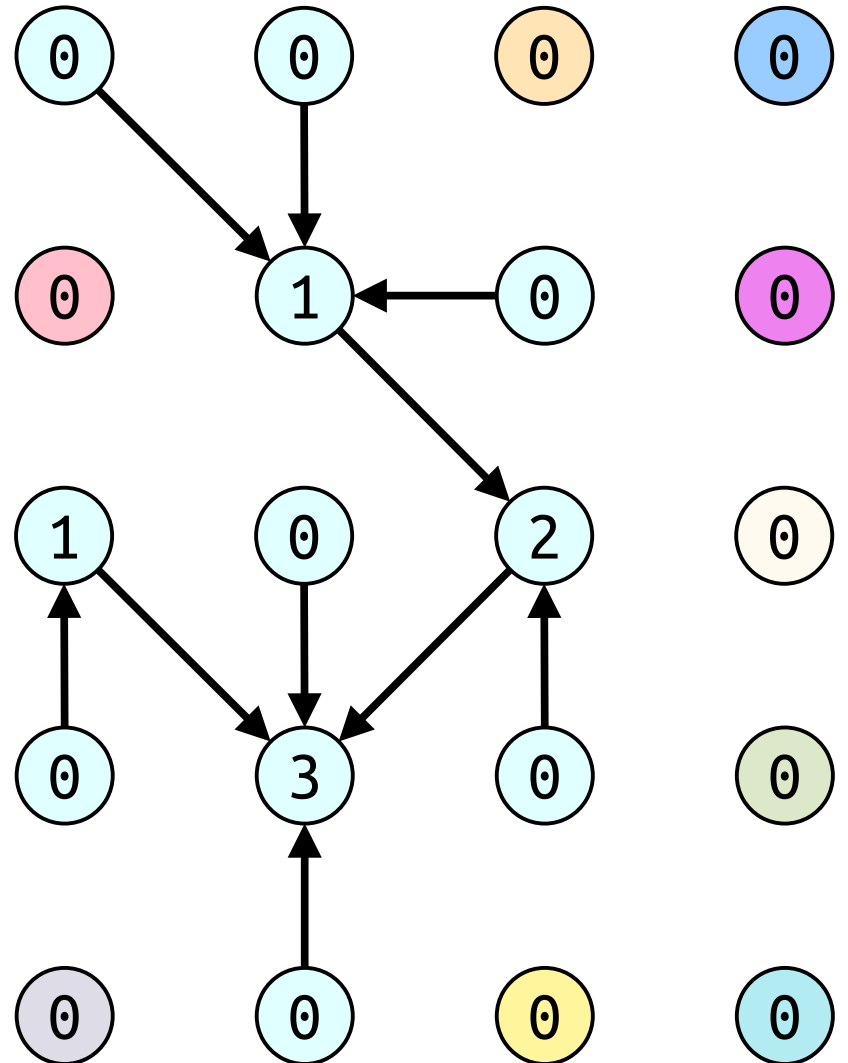
Union-By-Rank

- Our lemma tells us, indirectly, 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?**



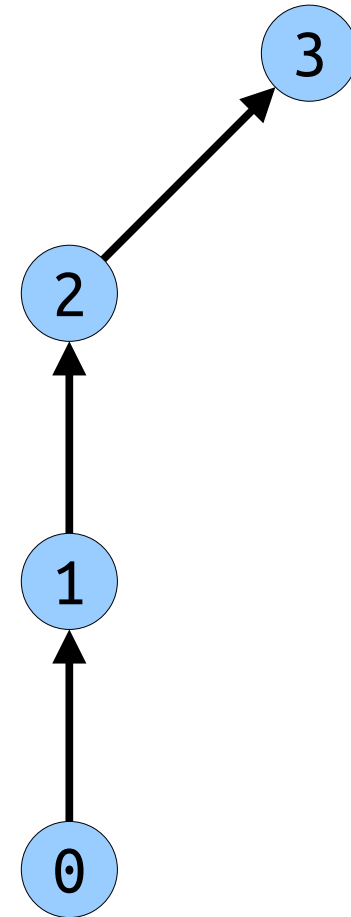
An Observation

- 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?



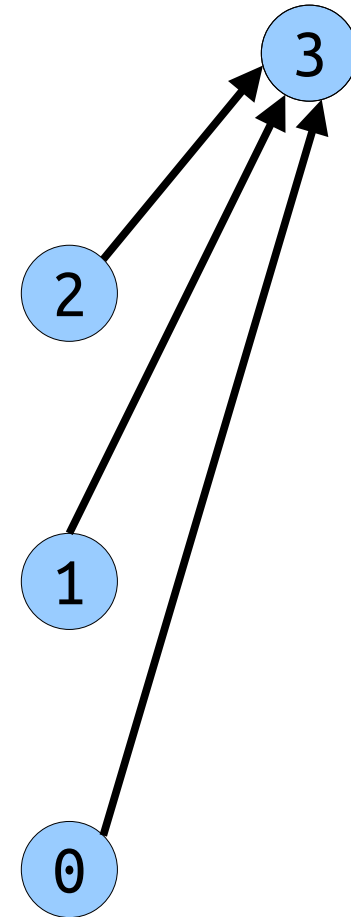
Path Compression

- ***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.



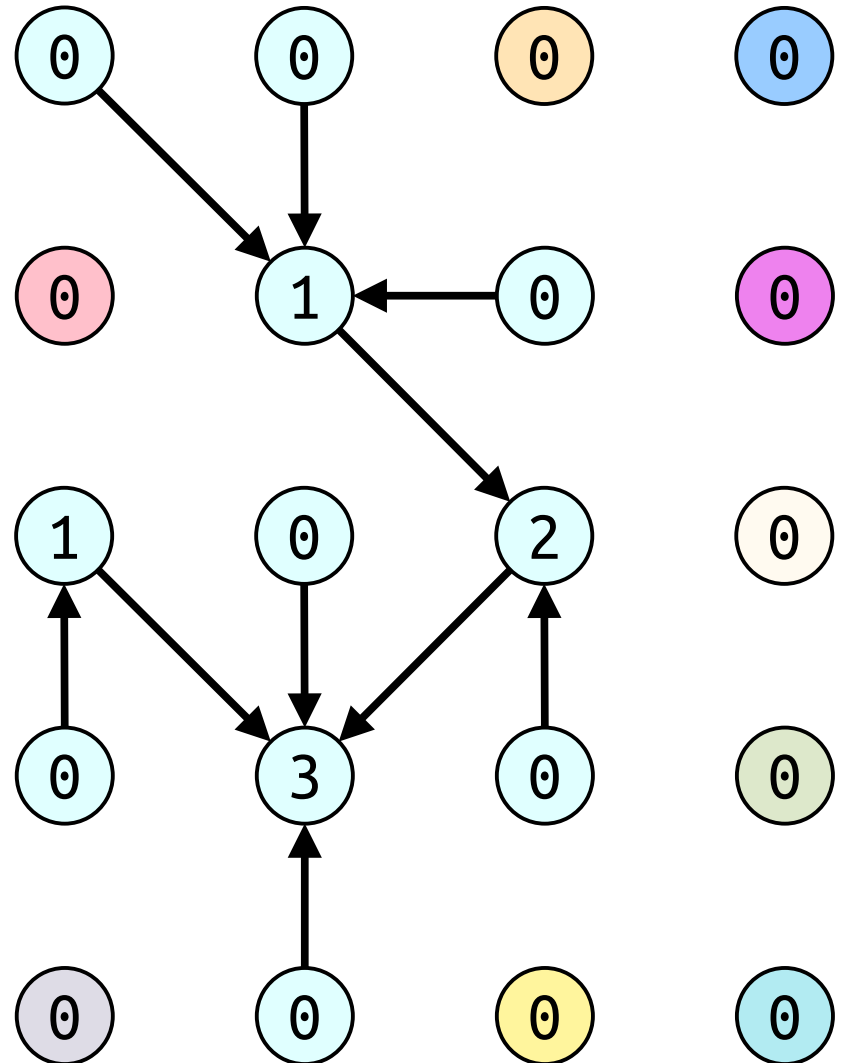
Path Compression

- ***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.



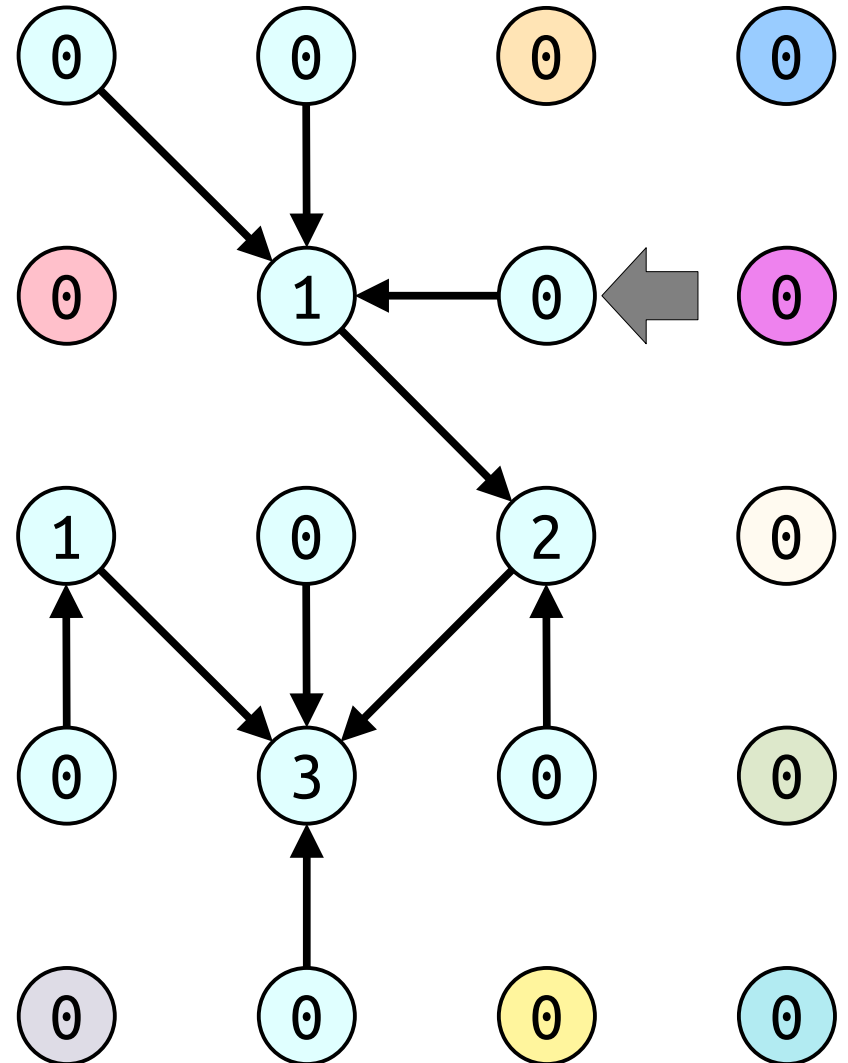
Path Compression

- **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.



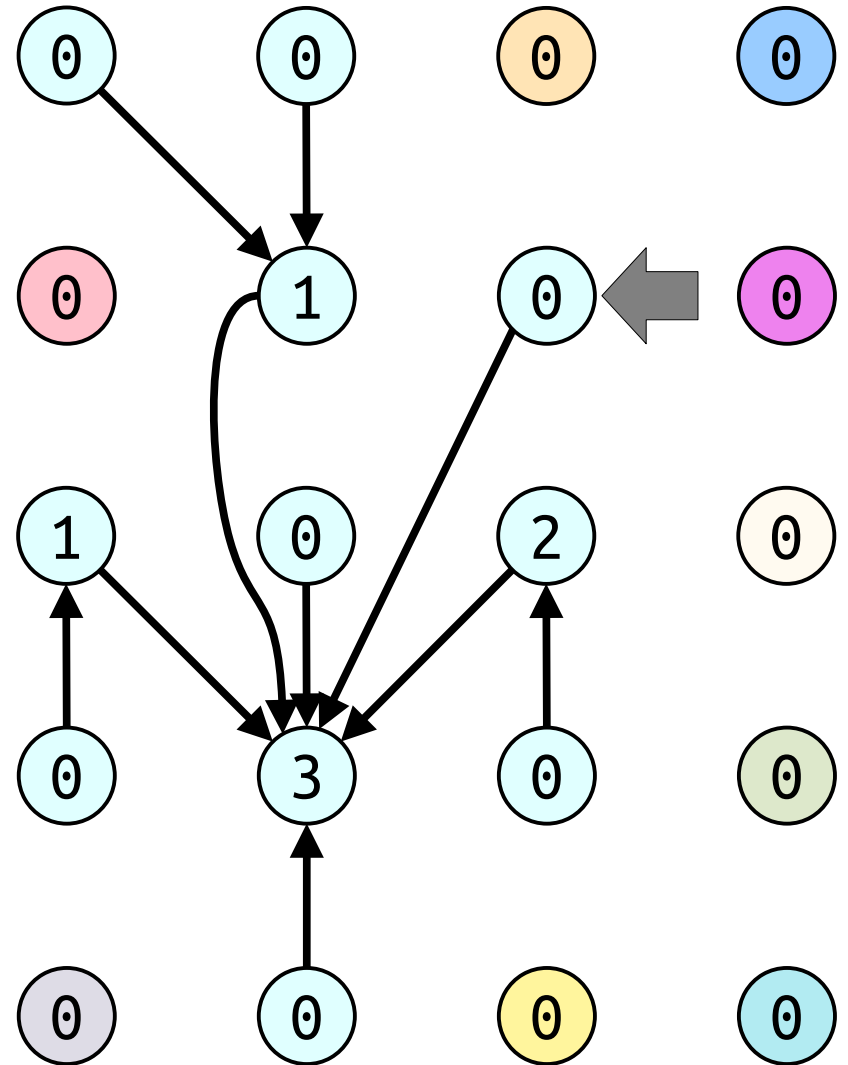
Path Compression

- **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.



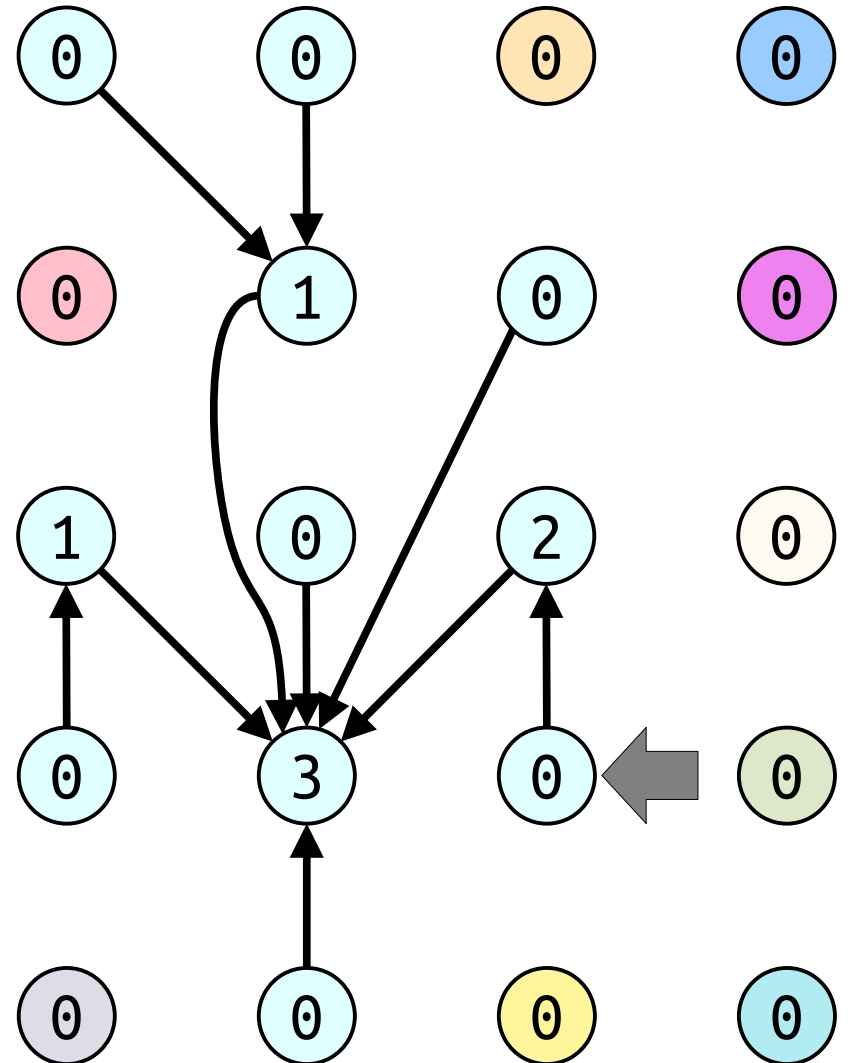
Path Compression

- **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.



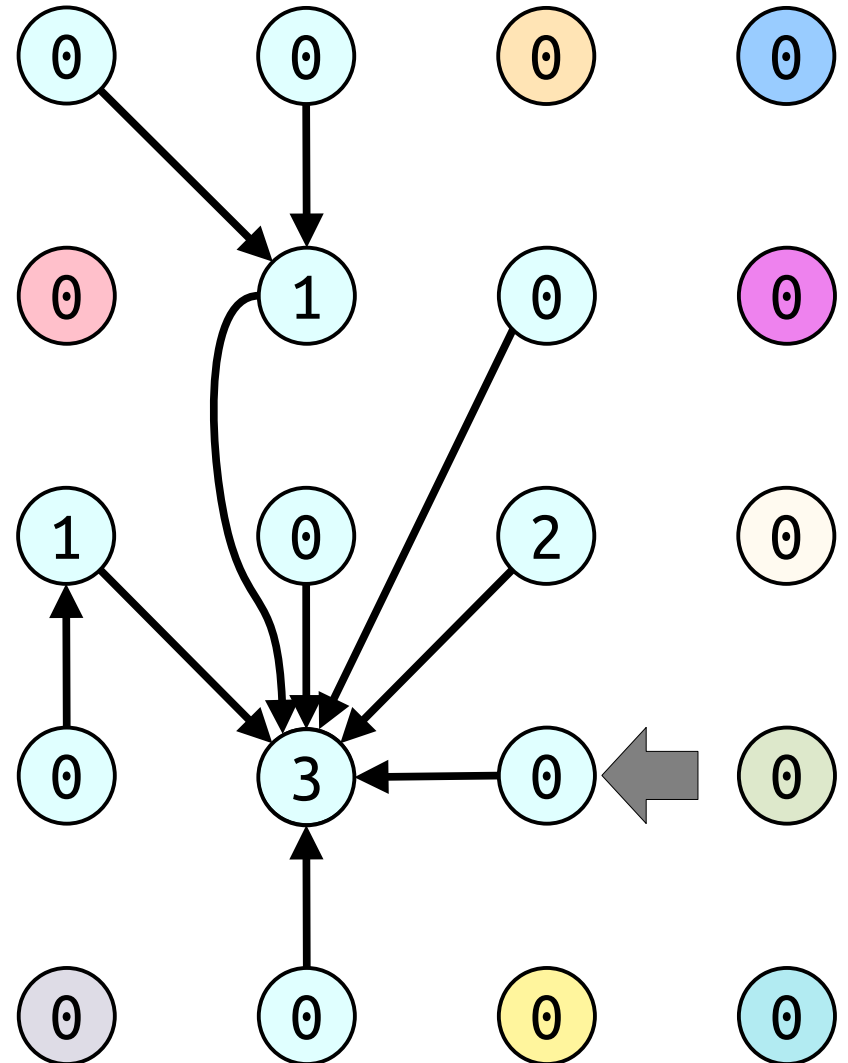
Path Compression

- **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.



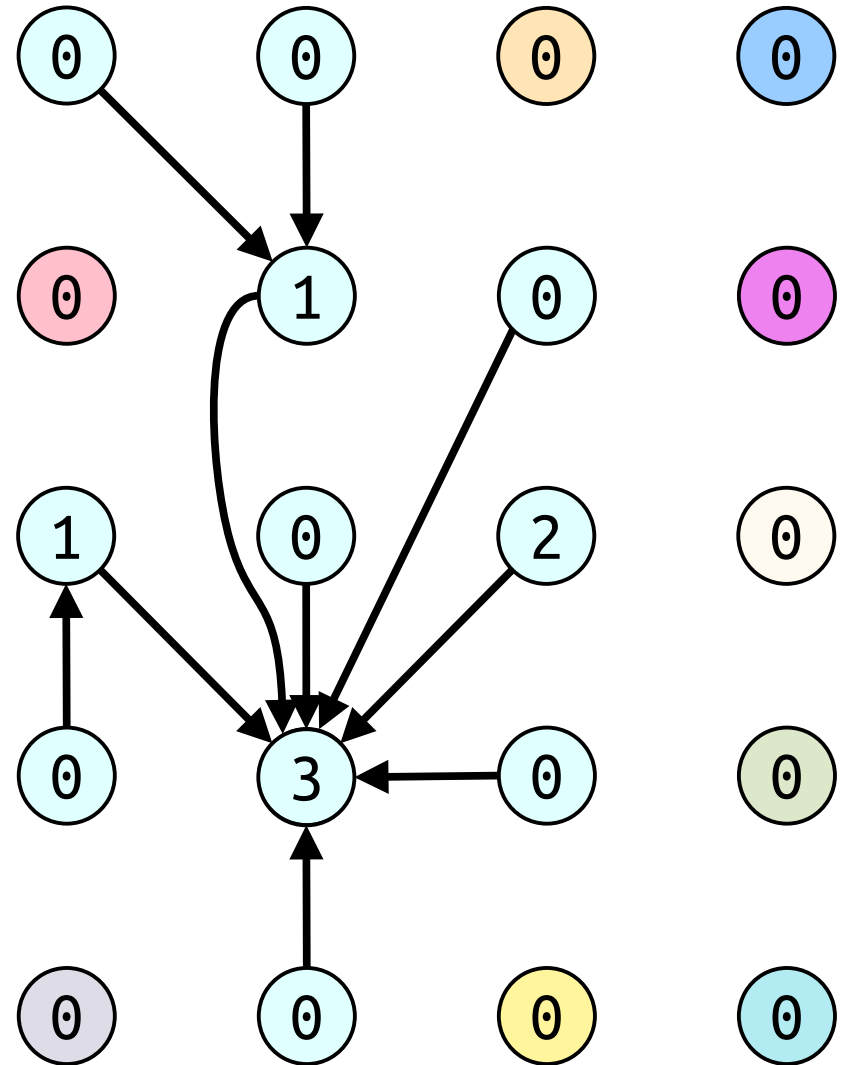
Path Compression

- **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

- ***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->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.

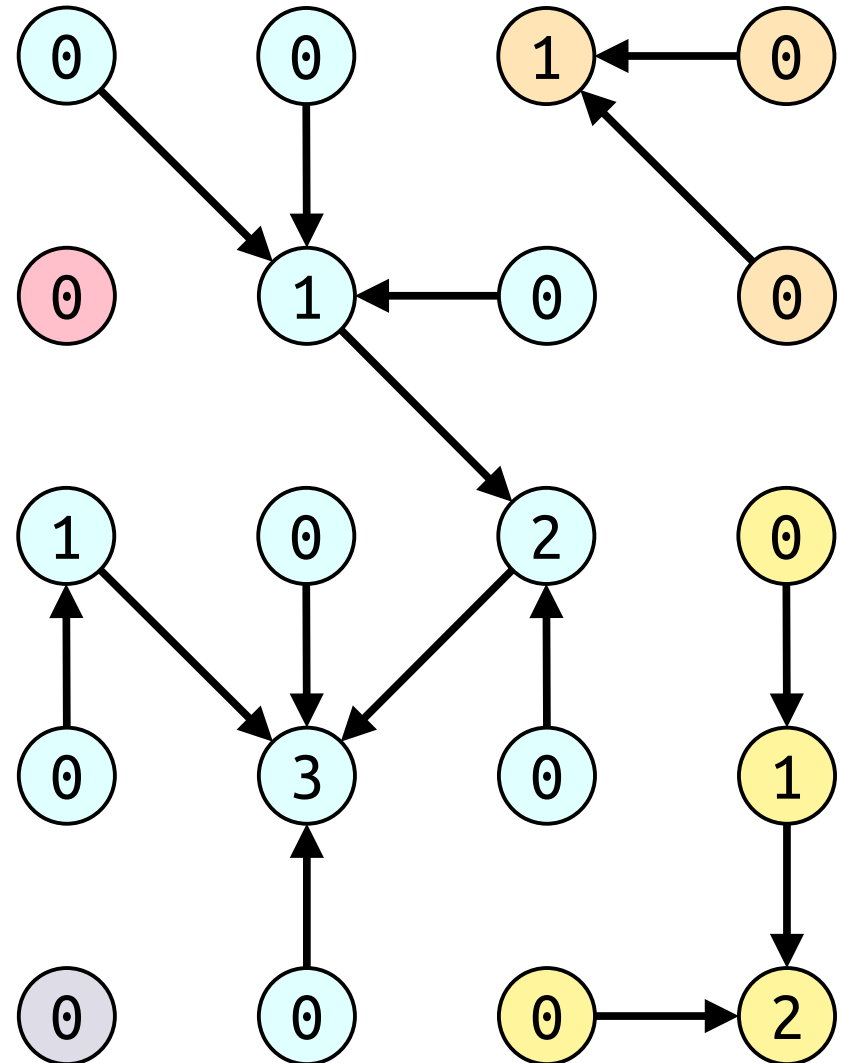
Analyzing Disjoint-Set Forests

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 result 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 Harvard.

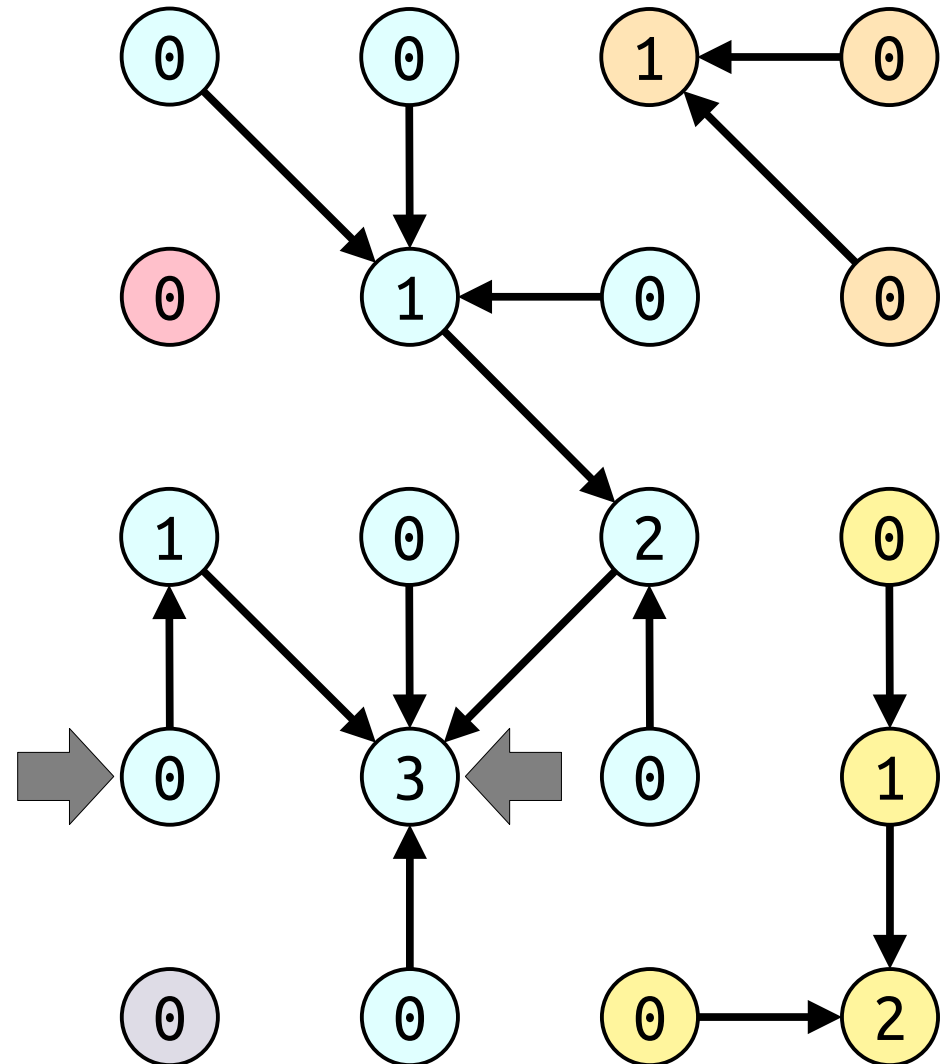
Our Analysis

- We're going to analyze a slightly simplified version of this problem.
- We'll be given a forest \mathcal{F} formed purely from union-by-rank, then do a series of path compressions on it.
 - These don't have to go all the way from a node to its representative.
- Our goal will be to bound the total amount of work done.
- **Great Exercise:** Show that this analysis carries over to the case of interleaved *unions* and *finds*.



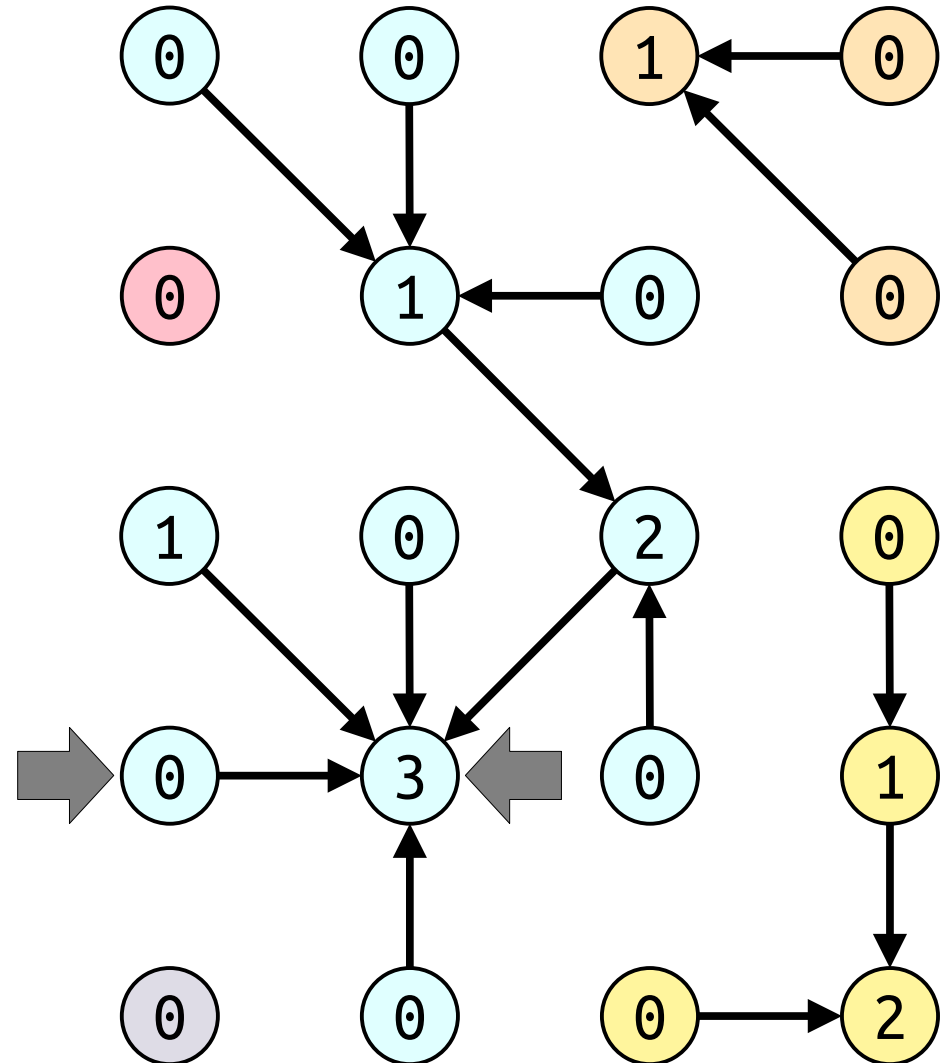
Our Analysis

- We're going to analyze a slightly simplified version of this problem.
- We'll be given a forest \mathcal{F} formed purely from union-by-rank, then do a series of path compressions on it.
 - These don't have to go all the way from a node to its representative.
- Our goal will be to bound the total amount of work done.
- **Great Exercise:** Show that this analysis carries over to the case of interleaved *unions* and *finds*.



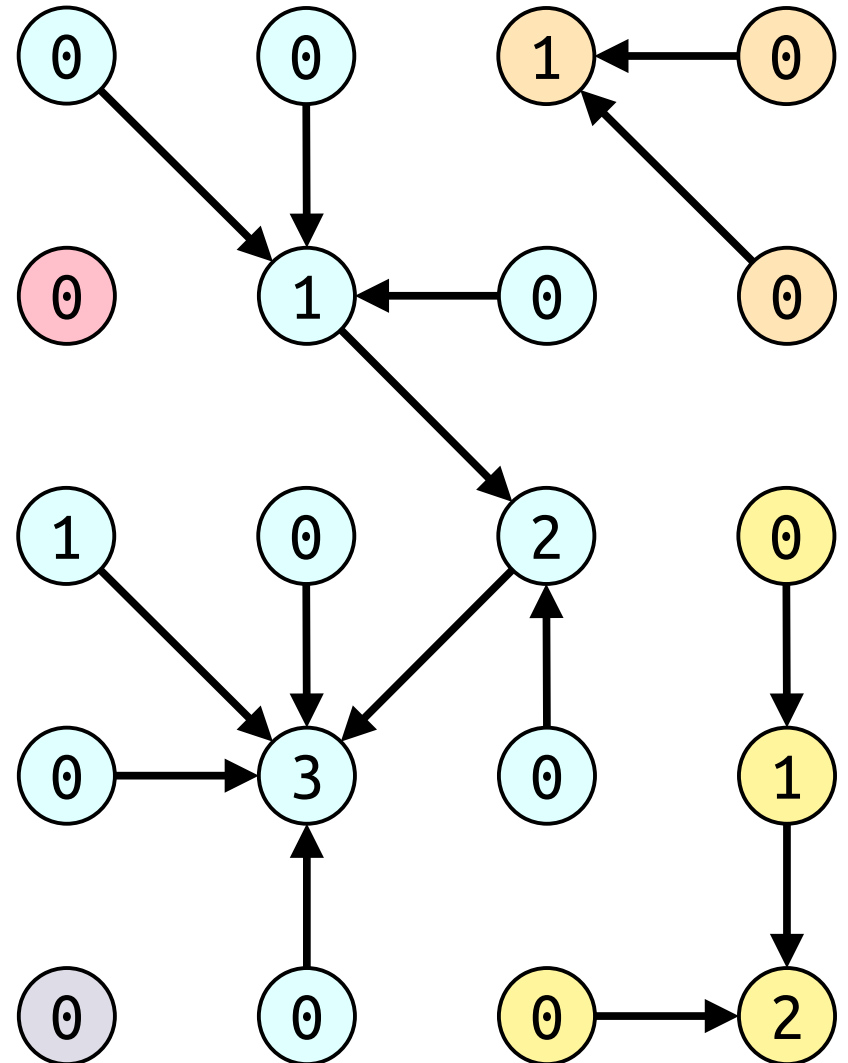
Our Analysis

- We're going to analyze a slightly simplified version of this problem.
- We'll be given a forest \mathcal{F} formed purely from union-by-rank, then do a series of path compressions on it.
 - These don't have to go all the way from a node to its representative.
- Our goal will be to bound the total amount of work done.
- **Great Exercise:** Show that this analysis carries over to the case of interleaved **unions** and **finds**.



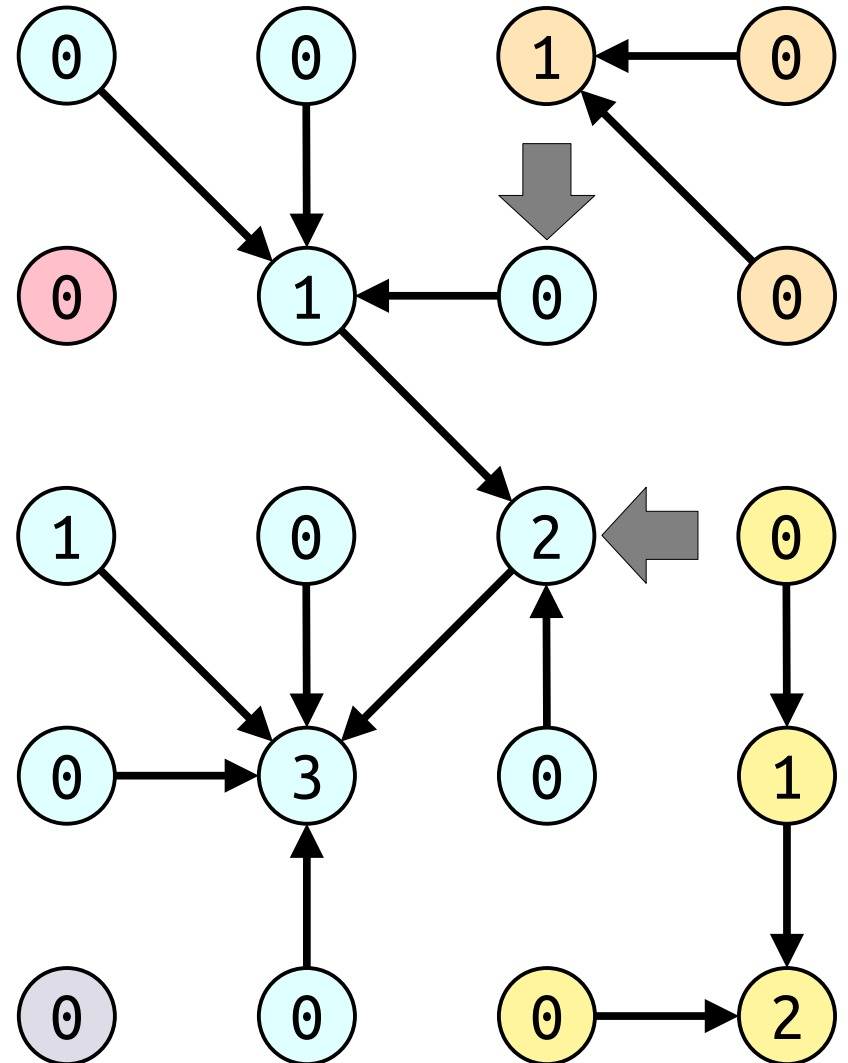
Our Analysis

- We're going to analyze a slightly simplified version of this problem.
- We'll be given a forest \mathcal{F} formed purely from union-by-rank, then do a series of path compressions on it.
 - These don't have to go all the way from a node to its representative.
- Our goal will be to bound the total amount of work done.
- **Great Exercise:** Show that this analysis carries over to the case of interleaved *unions* and *finds*.



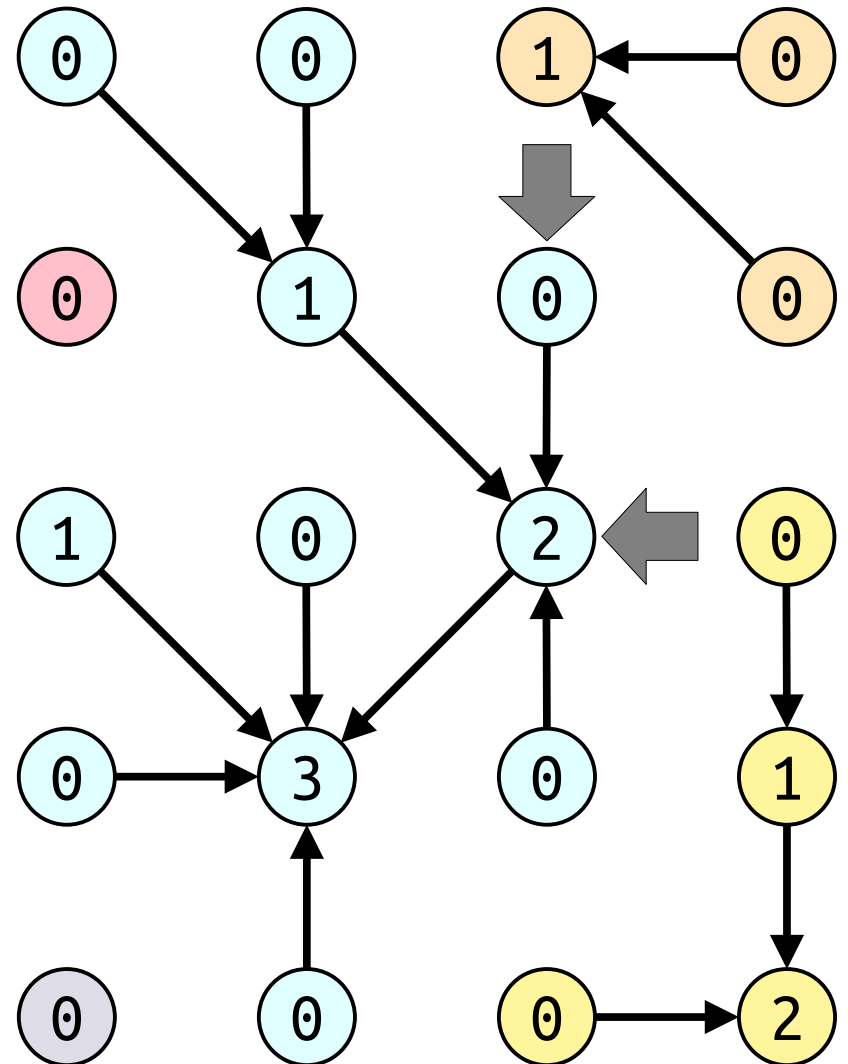
Our Analysis

- We're going to analyze a slightly simplified version of this problem.
- We'll be given a forest \mathcal{F} formed purely from union-by-rank, then do a series of path compressions on it.
 - These don't have to go all the way from a node to its representative.
- Our goal will be to bound the total amount of work done.
- **Great Exercise:** Show that this analysis carries over to the case of interleaved *unions* and *finds*.



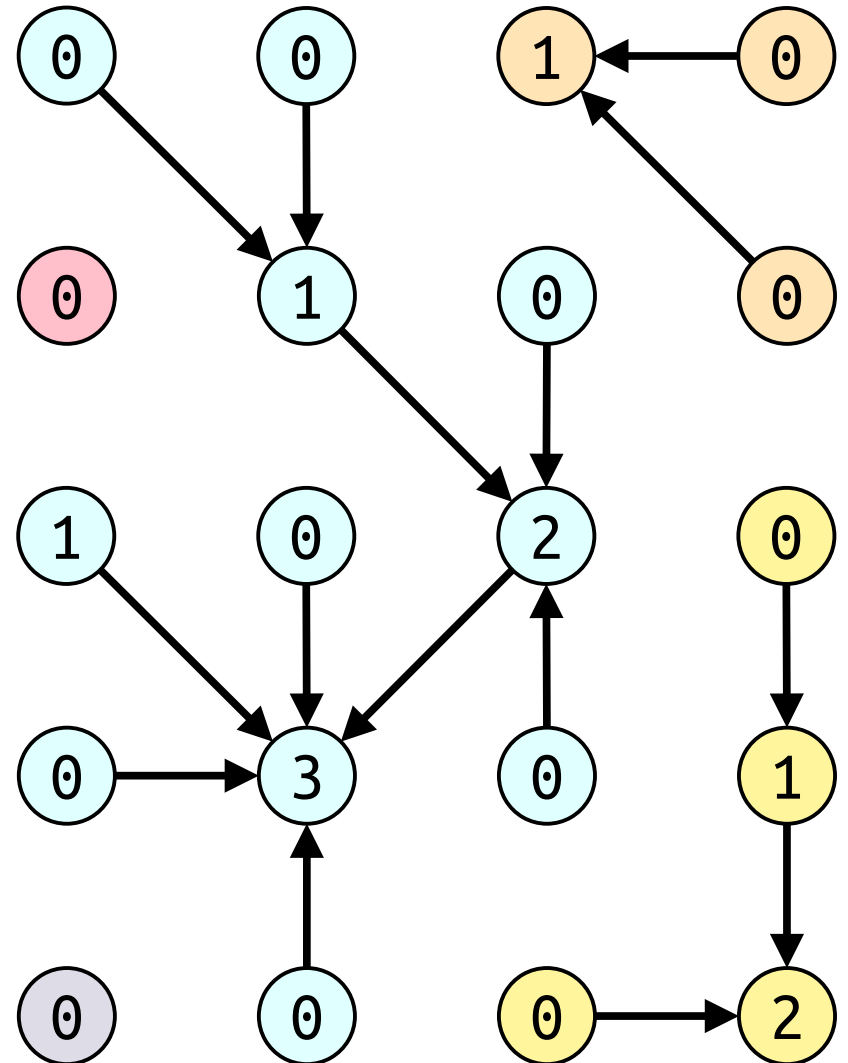
Our Analysis

- We're going to analyze a slightly simplified version of this problem.
- We'll be given a forest \mathcal{F} formed purely from union-by-rank, then do a series of path compressions on it.
 - These don't have to go all the way from a node to its representative.
- Our goal will be to bound the total amount of work done.
- **Great Exercise:** Show that this analysis carries over to the case of interleaved *unions* and *finds*.



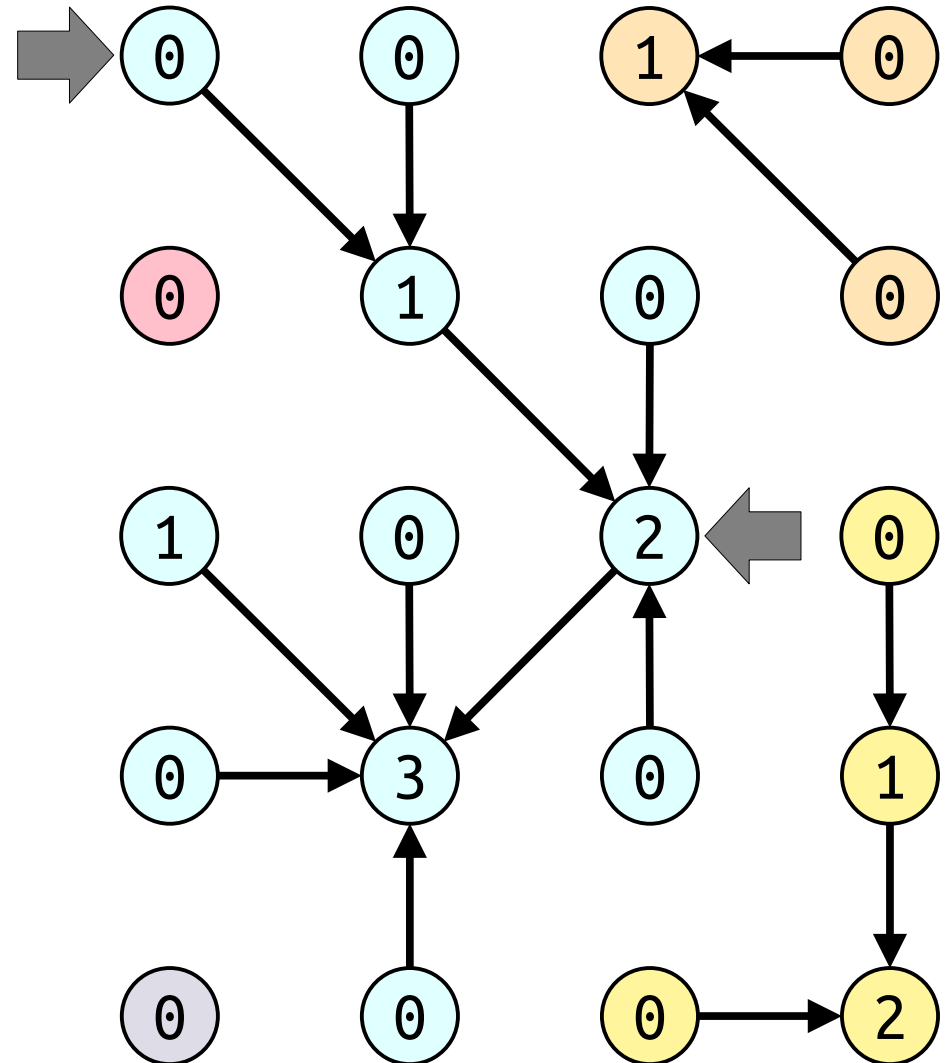
Our Analysis

- We're going to analyze a slightly simplified version of this problem.
- We'll be given a forest \mathcal{F} formed purely from union-by-rank, then do a series of path compressions on it.
 - These don't have to go all the way from a node to its representative.
- Our goal will be to bound the total amount of work done.
- **Great Exercise:** Show that this analysis carries over to the case of interleaved *unions* and *finds*.



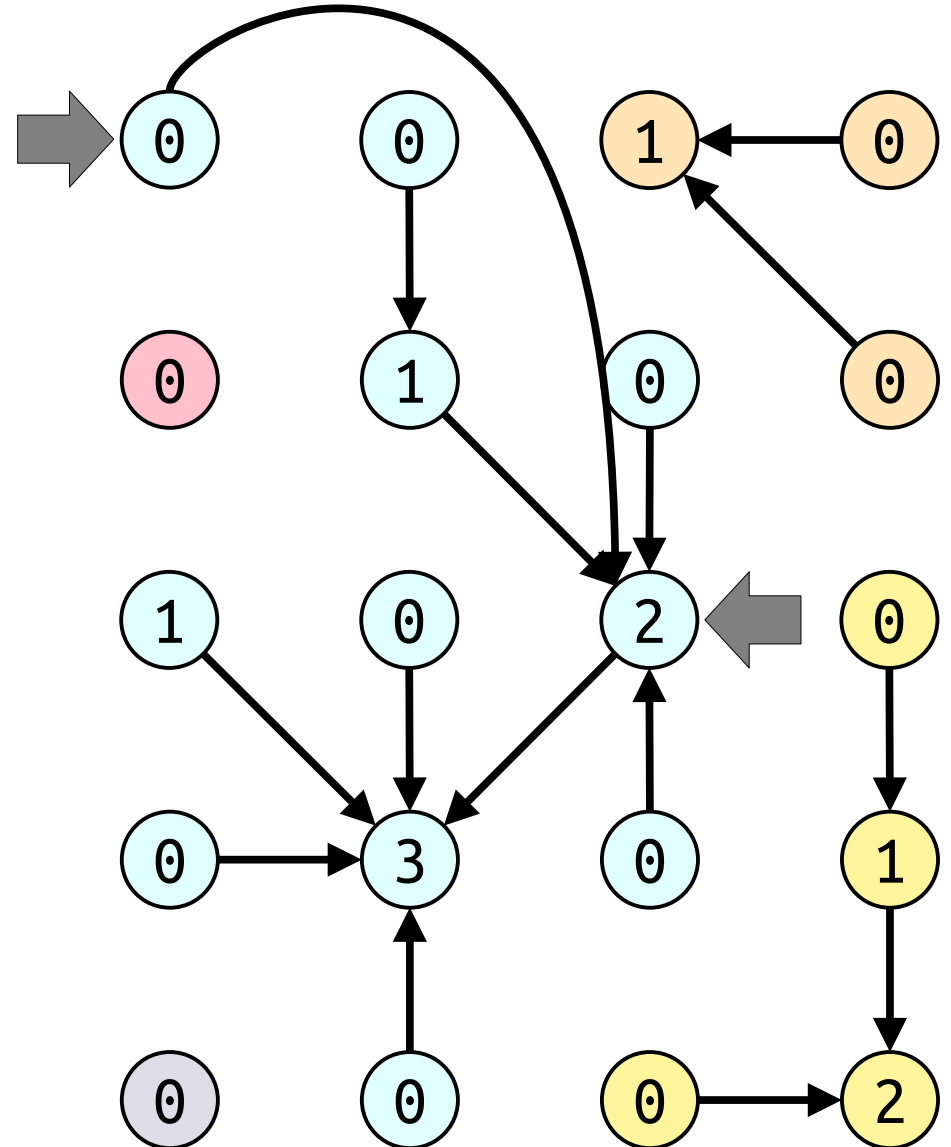
Our Analysis

- We're going to analyze a slightly simplified version of this problem.
- We'll be given a forest \mathcal{F} formed purely from union-by-rank, then do a series of path compressions on it.
 - These don't have to go all the way from a node to its representative.
- Our goal will be to bound the total amount of work done.
- **Great Exercise:** Show that this analysis carries over to the case of interleaved *unions* and *finds*.



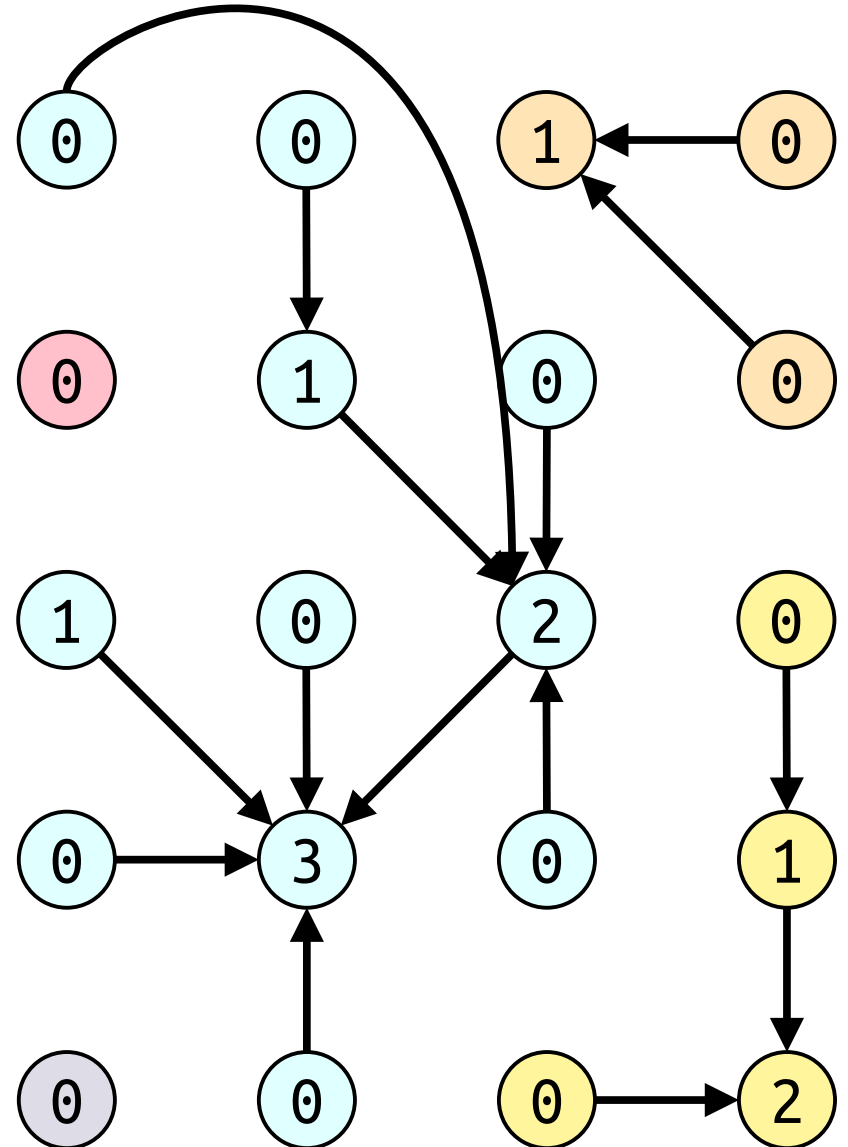
Our Analysis

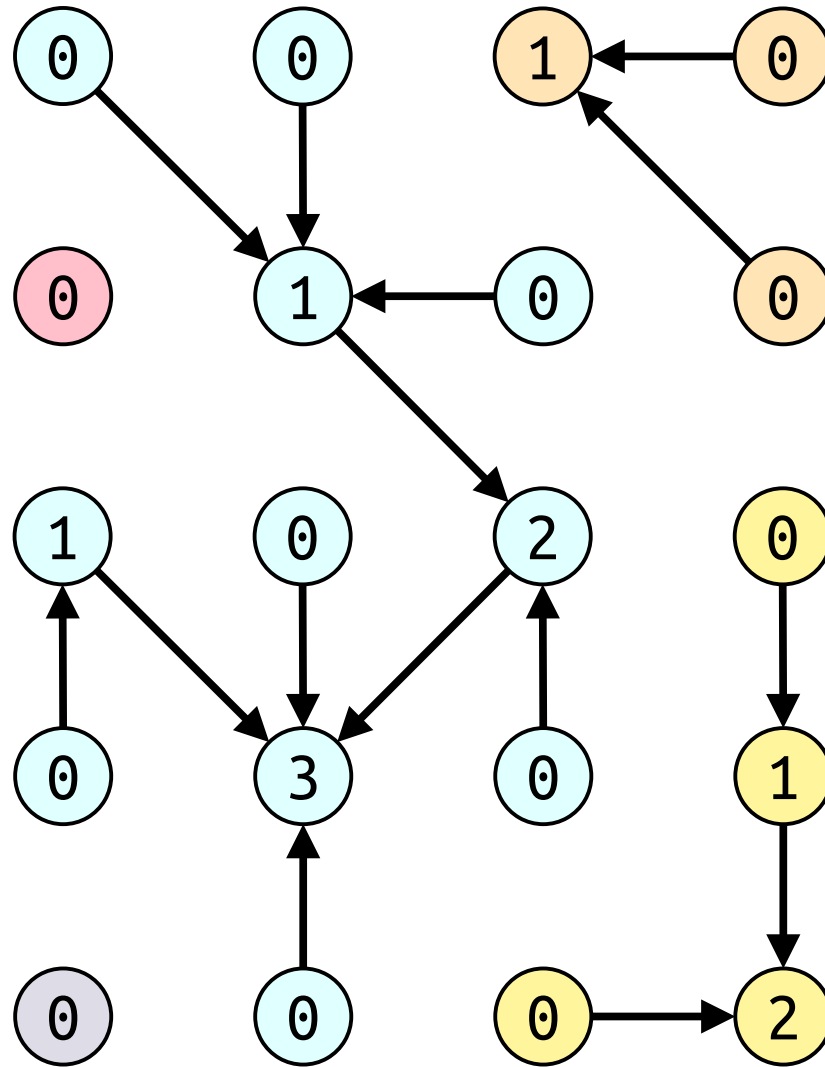
- We're going to analyze a slightly simplified version of this problem.
- We'll be given a forest \mathcal{F} formed purely from union-by-rank, then do a series of path compressions on it.
 - These don't have to go all the way from a node to its representative.
- Our goal will be to bound the total amount of work done.
- **Great Exercise:** Show that this analysis carries over to the case of interleaved *unions* and *finds*.

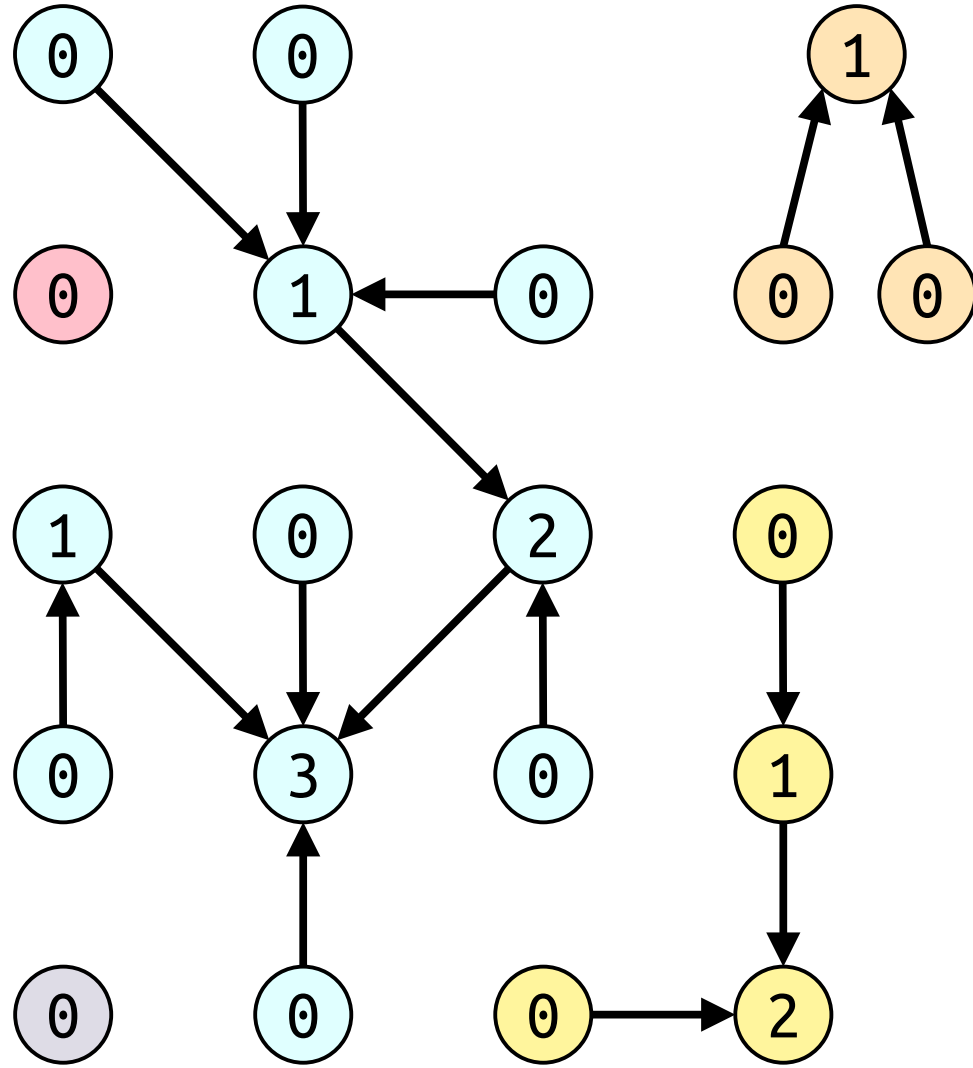


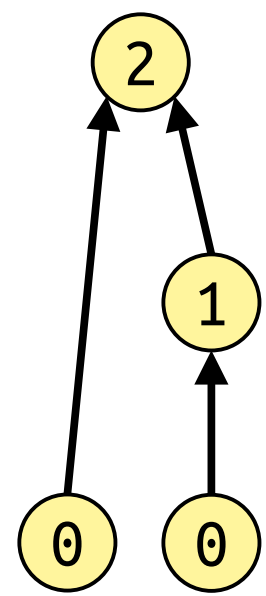
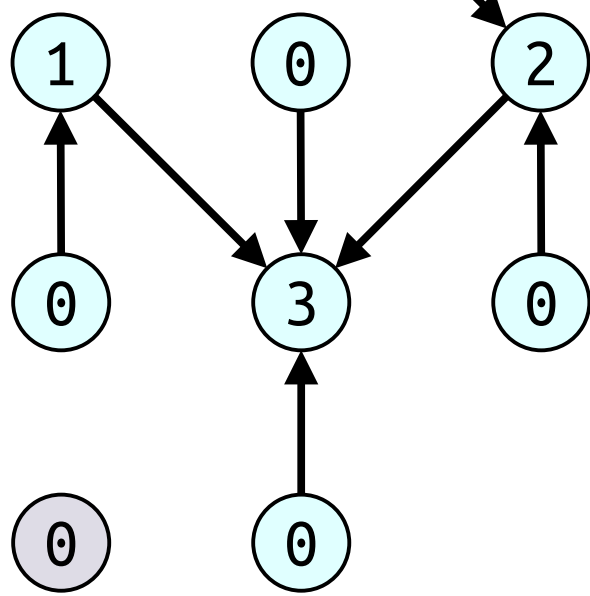
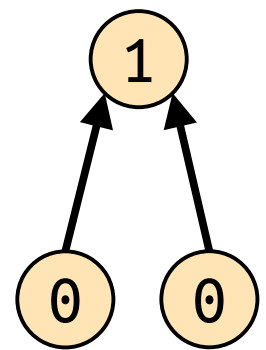
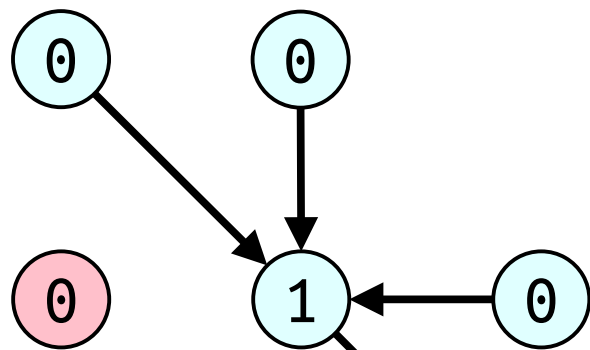
Our Analysis

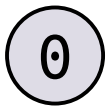
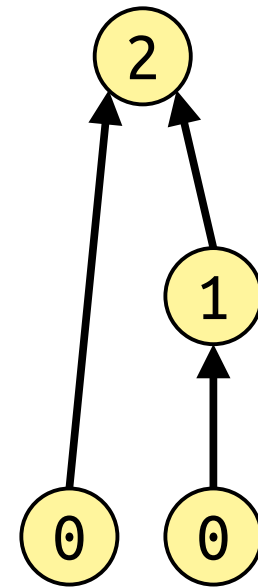
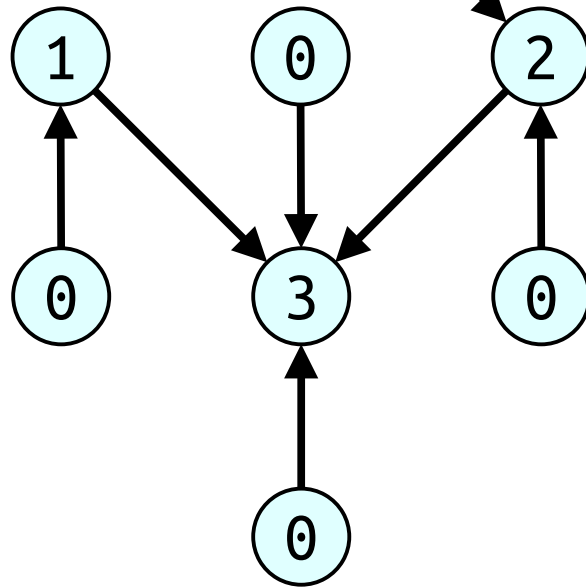
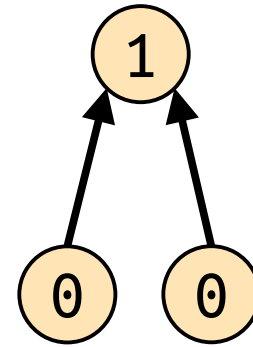
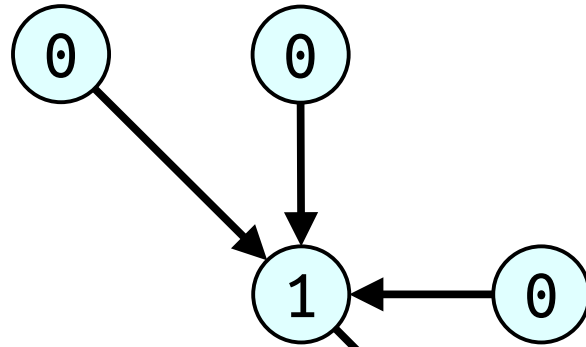
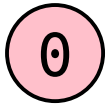
- We're going to analyze a slightly simplified version of this problem.
- We'll be given a forest \mathcal{F} formed purely from union-by-rank, then do a series of path compressions on it.
 - These don't have to go all the way from a node to its representative.
- Our goal will be to bound the total amount of work done.
- **Great Exercise:** Show that this analysis carries over to the case of interleaved *unions* and *finds*.

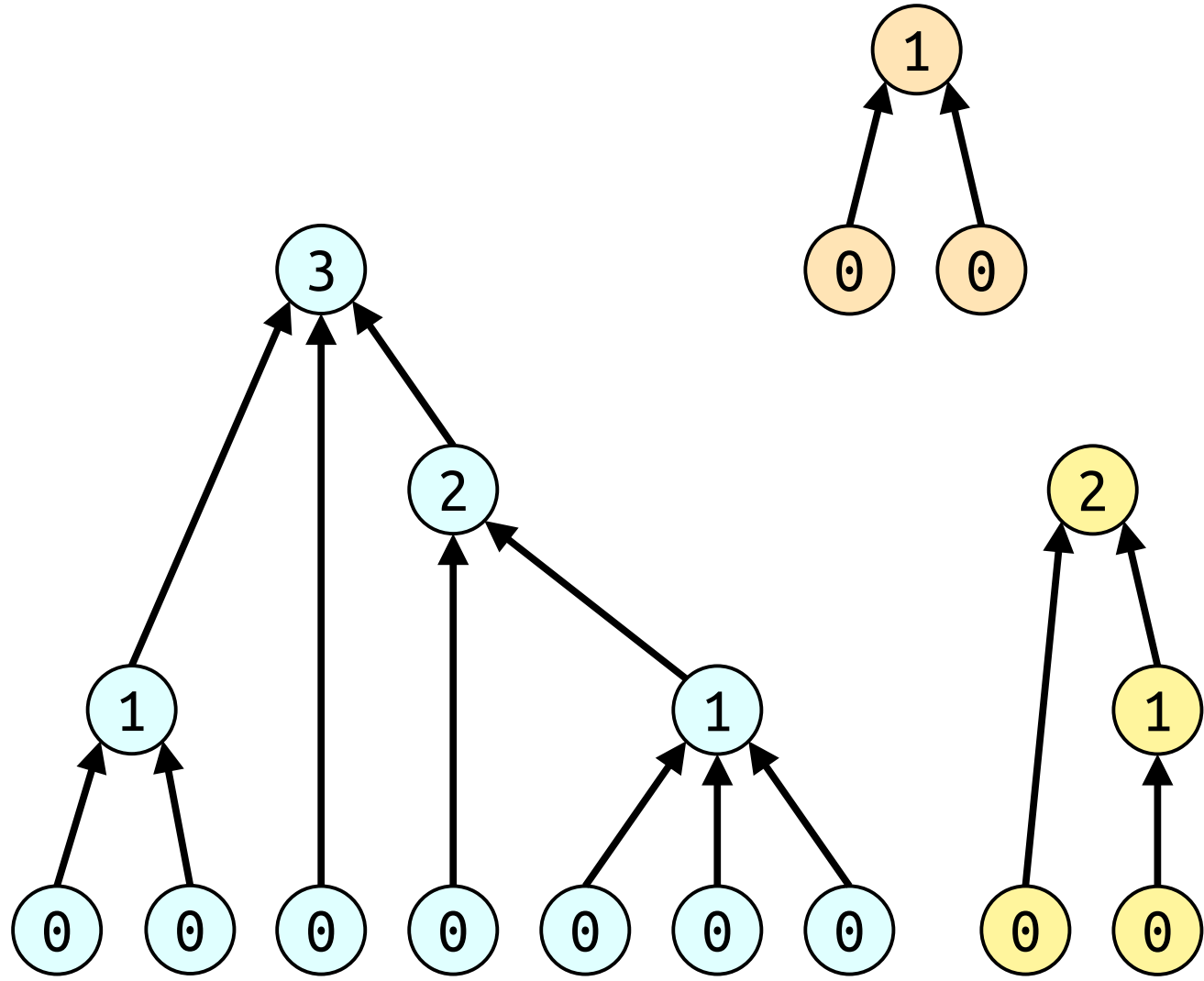
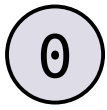


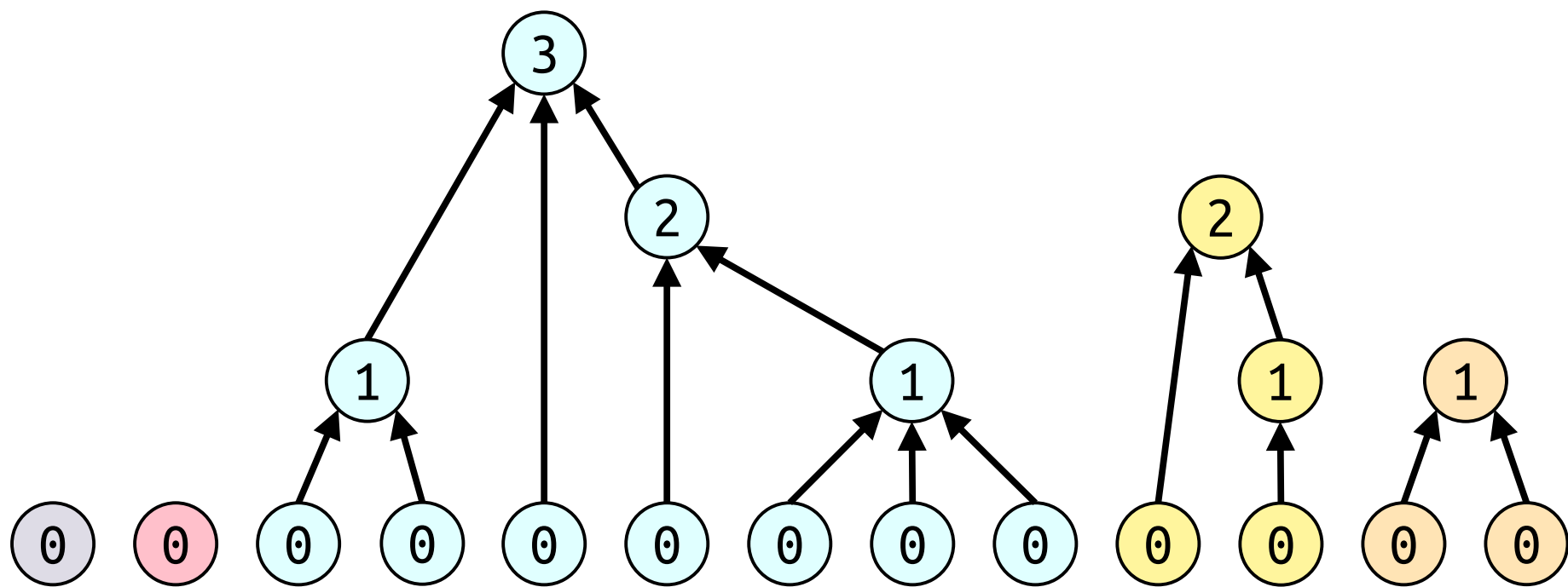






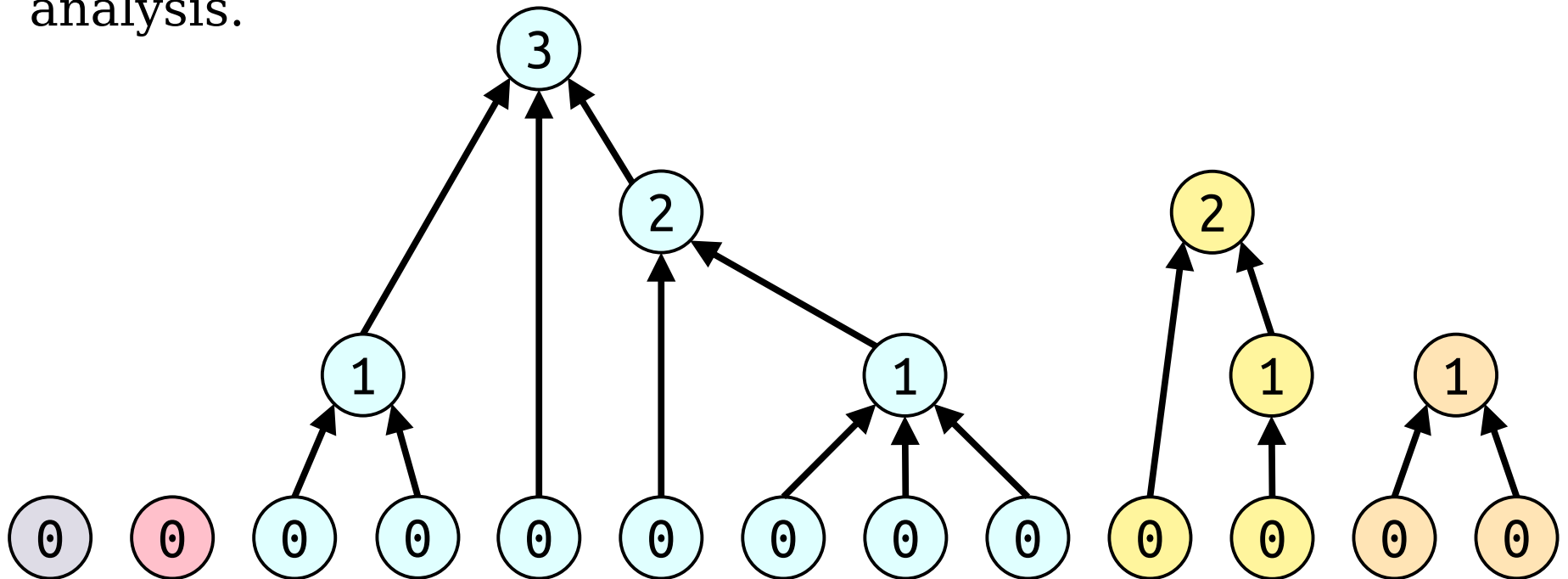






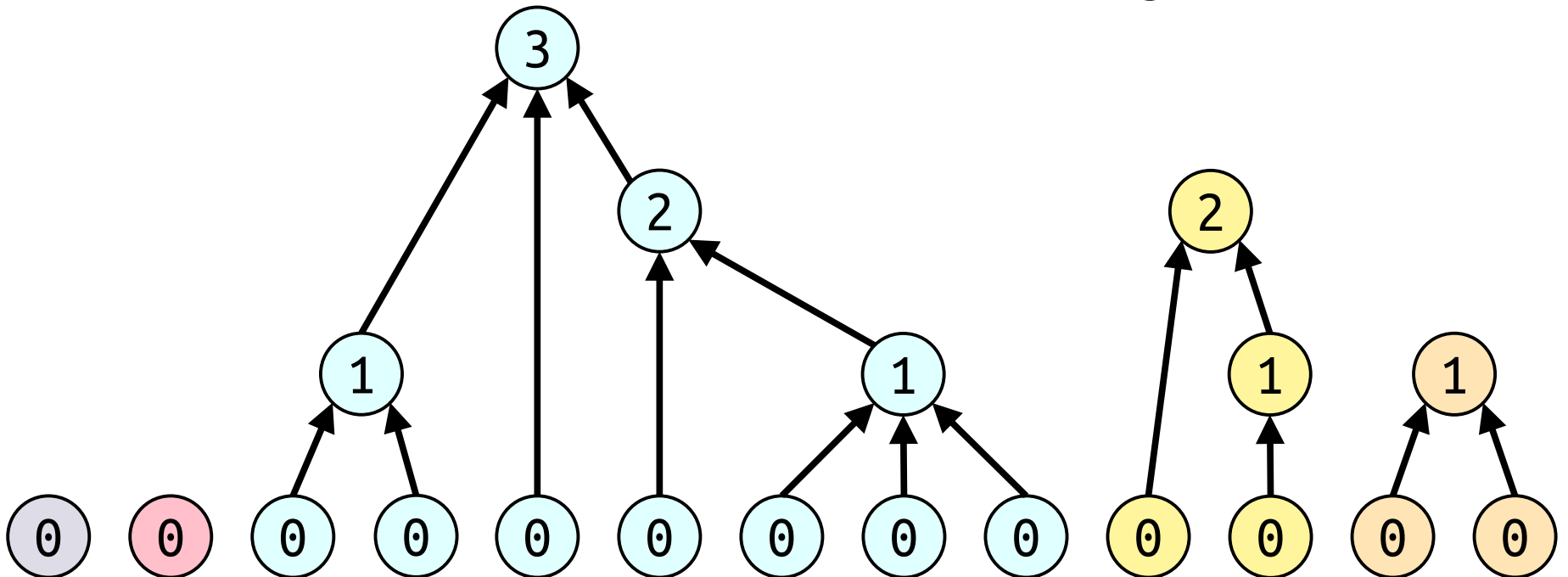
Our Analysis

- 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.



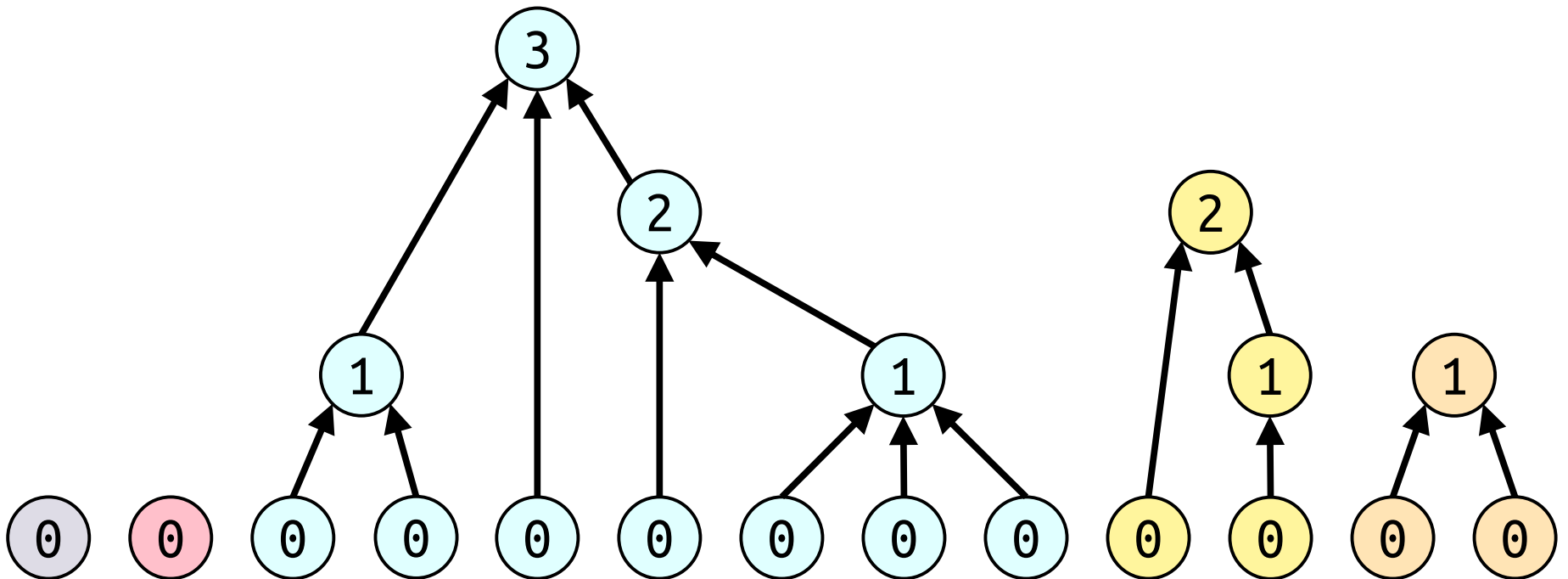
Our Analysis

- We will specifically focus on the number of times a node's parent changes.
- Why?
 - Each operation does $O(1)$ work, plus work proportional to the number of parents changed.
- The total work done is then $\Theta(m + \text{\#changes})$.



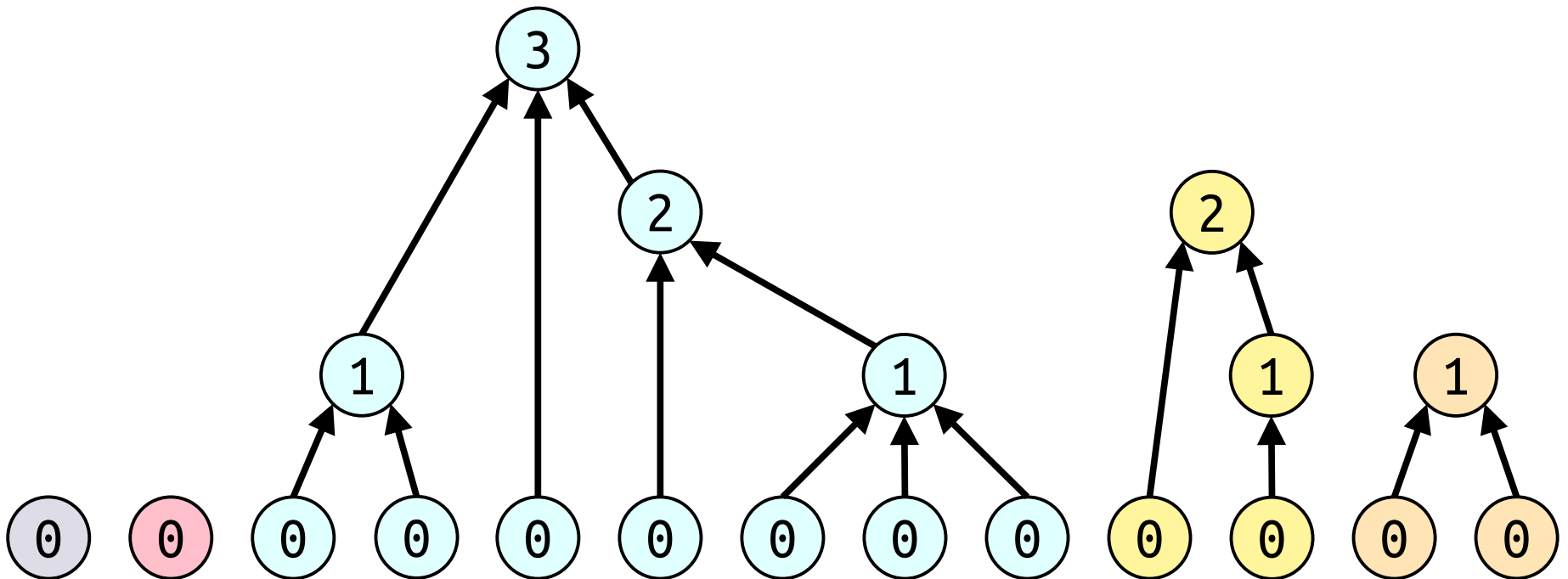
A Starting Analysis

- **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:**
 - **Nodes of nonzero rank:**



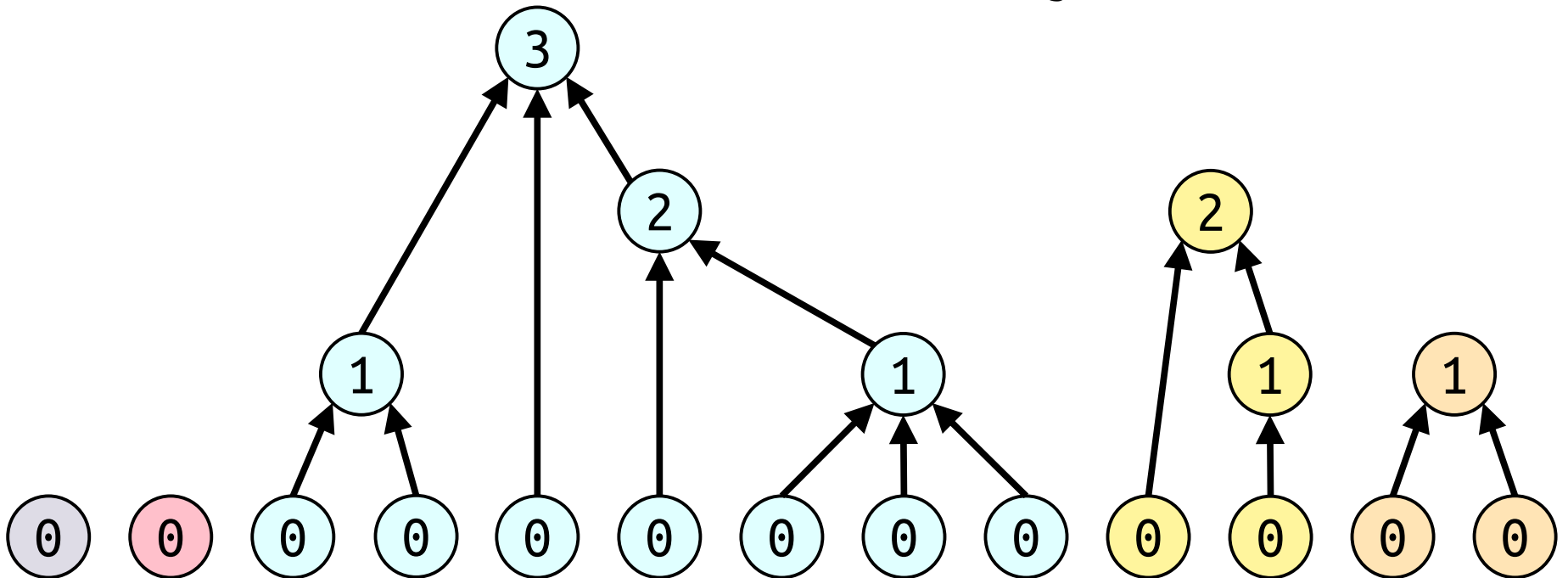
A Starting Analysis

- **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:**



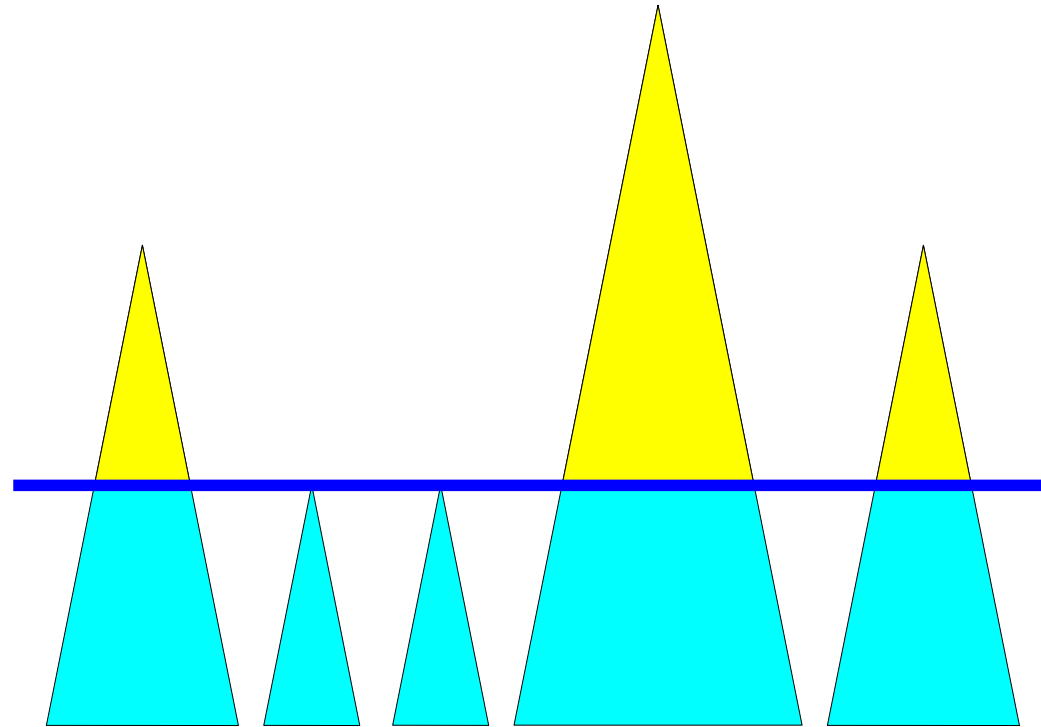
A Starting Analysis

- **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 rank can increase at most r times. There are at most $n / 2$ nodes of nonzero rank. This gives a bound of $n \cdot r / 2$.



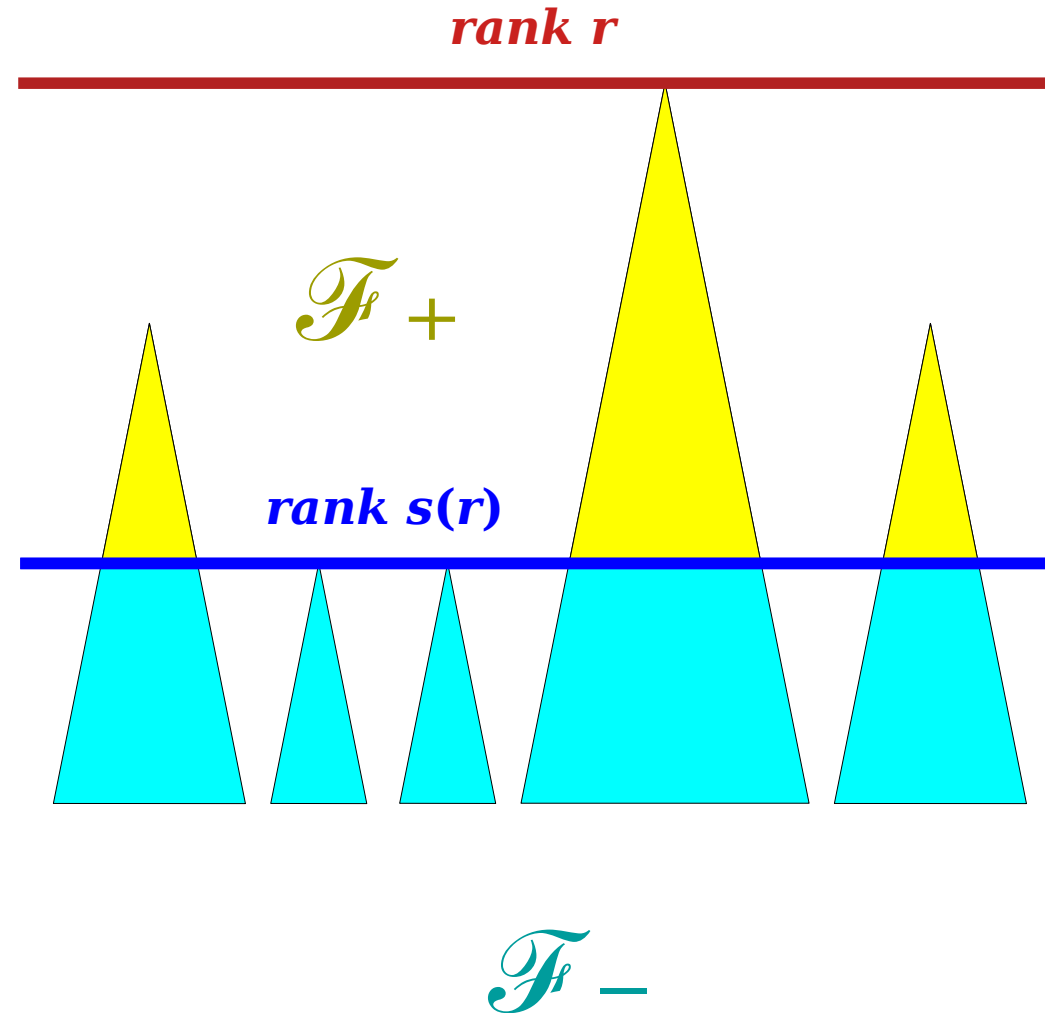
A Starting Analysis

- Our starting analysis is weak.
 - Compressing a path impacts other nodes not on that path.
 - Nodes with high starting rank have 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.



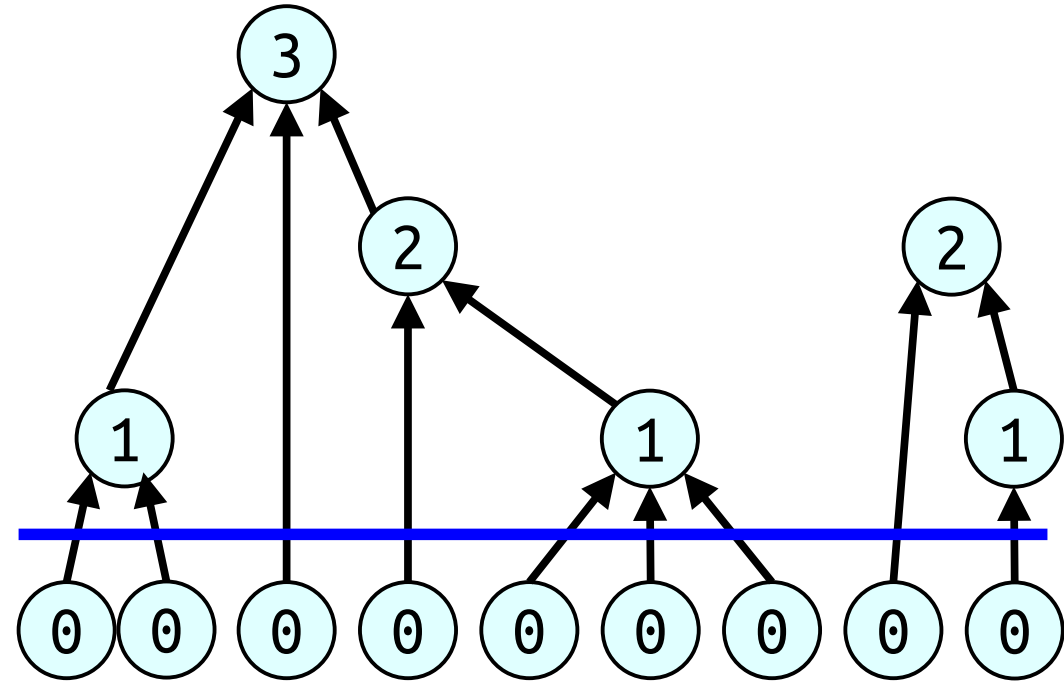
Forest Slicing

- 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}_+ .



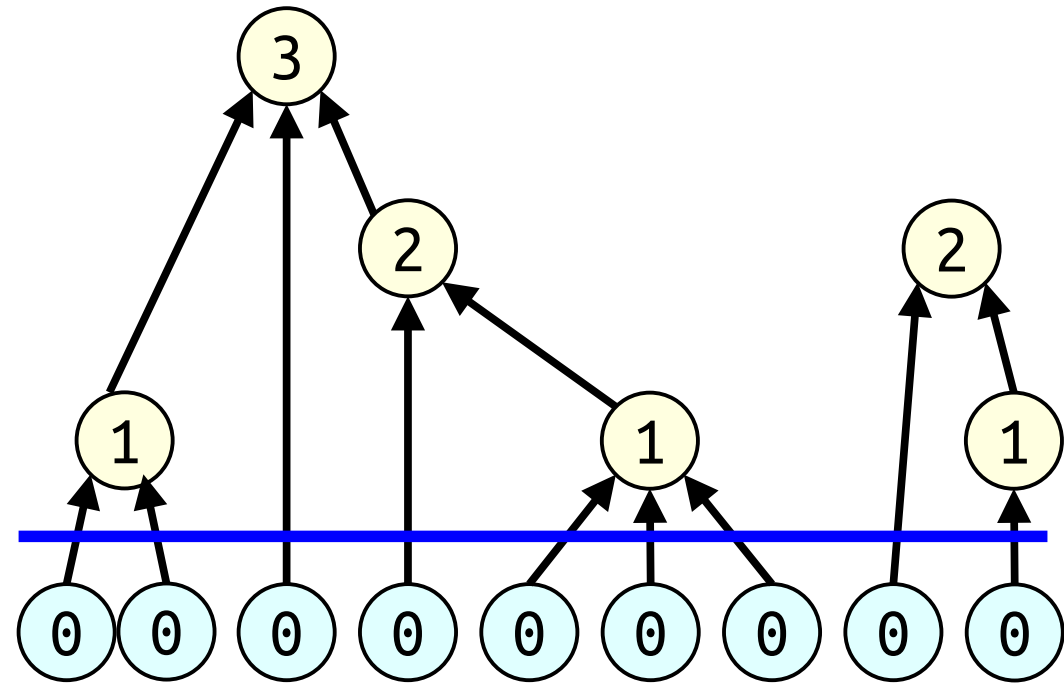
Forest Slicing

- 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}_+ .



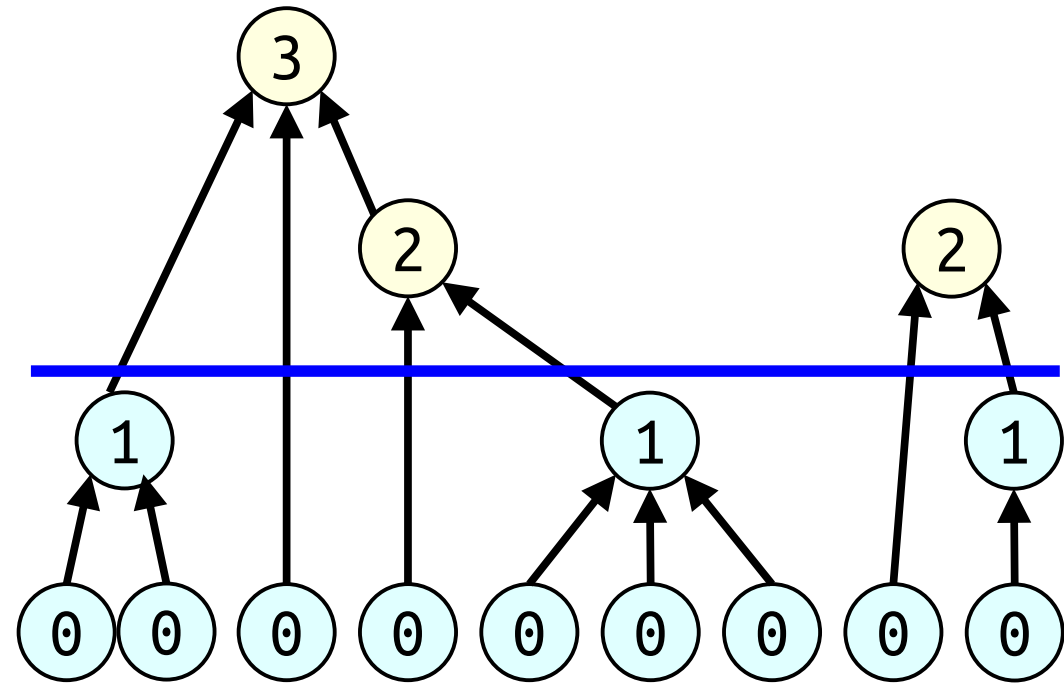
Forest Slicing

- 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}_+ .



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, 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 $C(m, n, 1)$?

Answer at

<https://pollev.com/cs166spr23>

Some Terminology

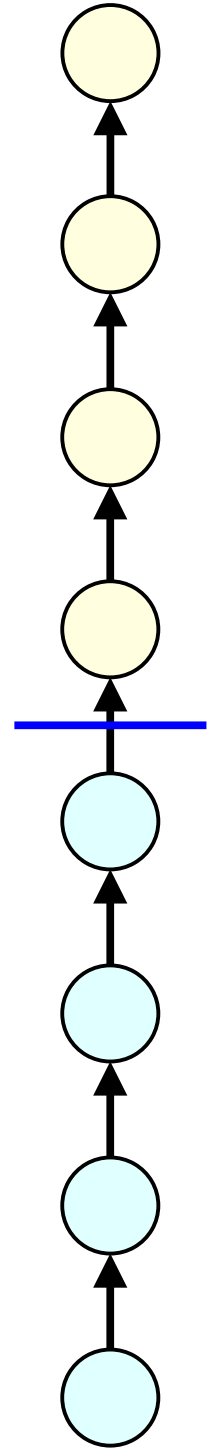
- Let $C(m, n, r)$ be the maximum number of pointer changes that can be made if there are m compresses, n nodes, and the maximum rank is r .
- Using this notation, our earlier result is that

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

- **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**.

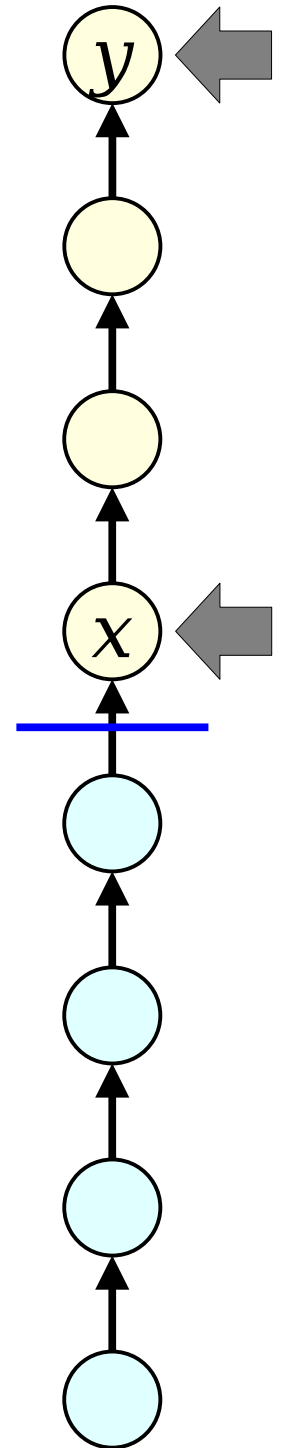
Forest Slicing

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



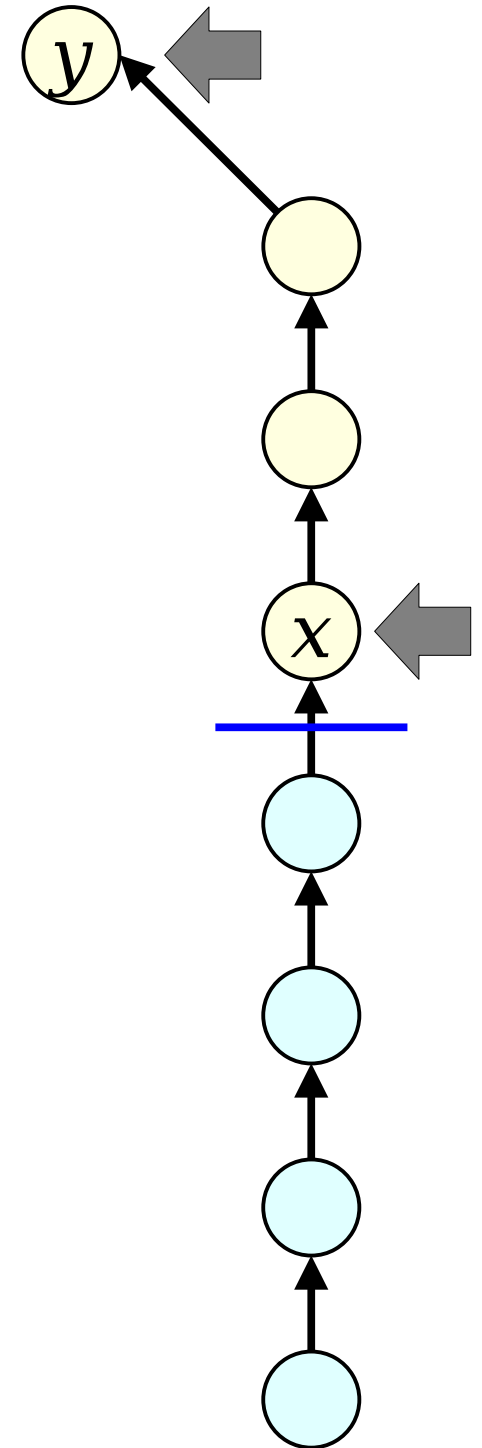
Forest Slicing

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



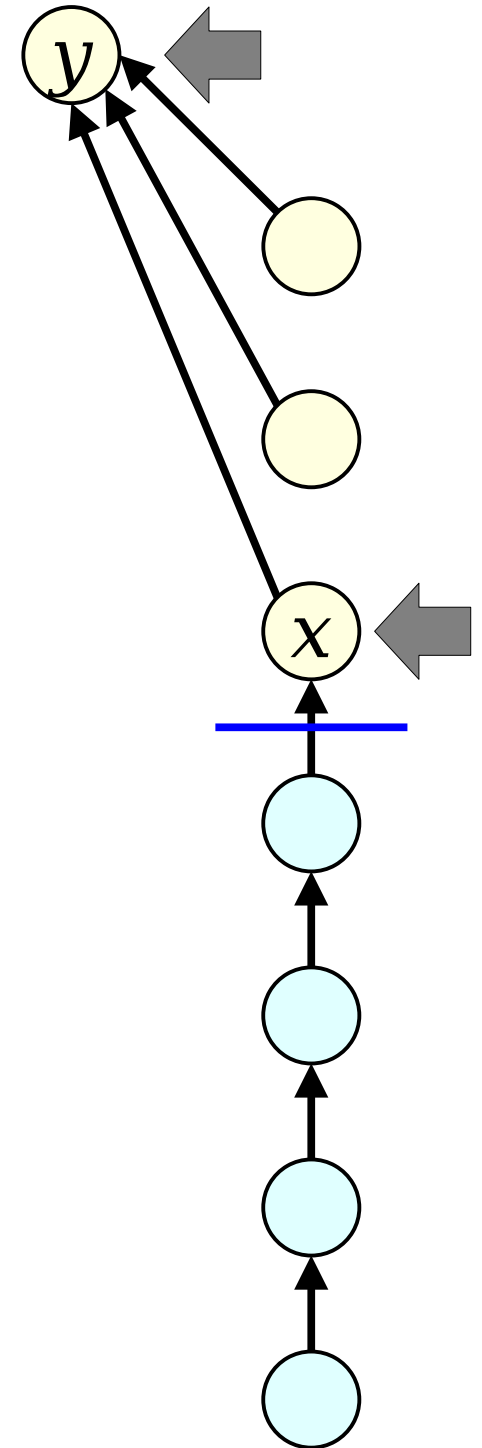
Forest Slicing

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



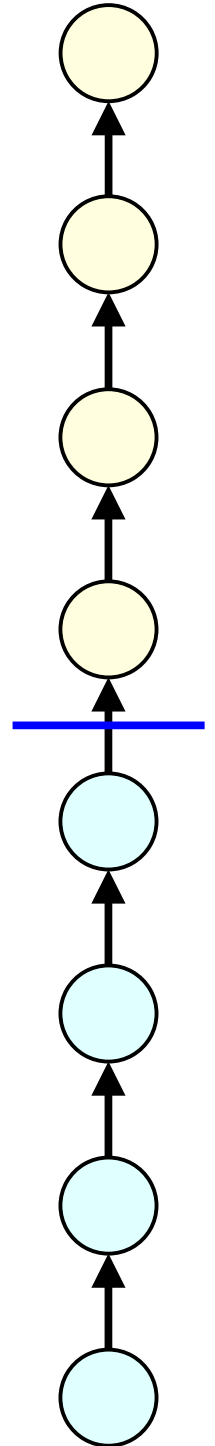
Forest Slicing

- Focus on any one compression from x to y . Let's see how it interacts with \mathcal{F}_- and \mathcal{F}_+ .
- **Case 1:** x and y are both in \mathcal{F}_+ .
- We can recursively handle this compression when bounding the work done purely in \mathcal{F}_+ .



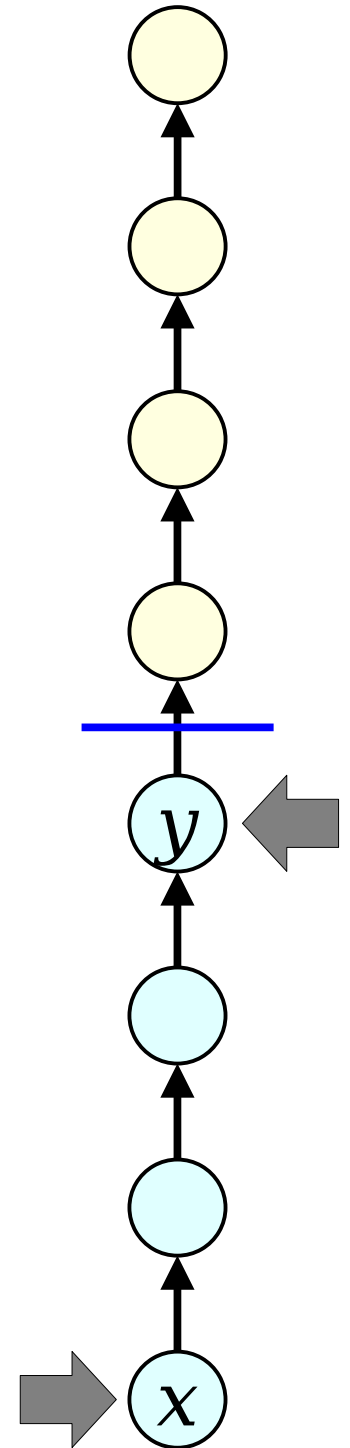
Forest Slicing

- **Case 2:** x and y are both in \mathcal{F}_- .



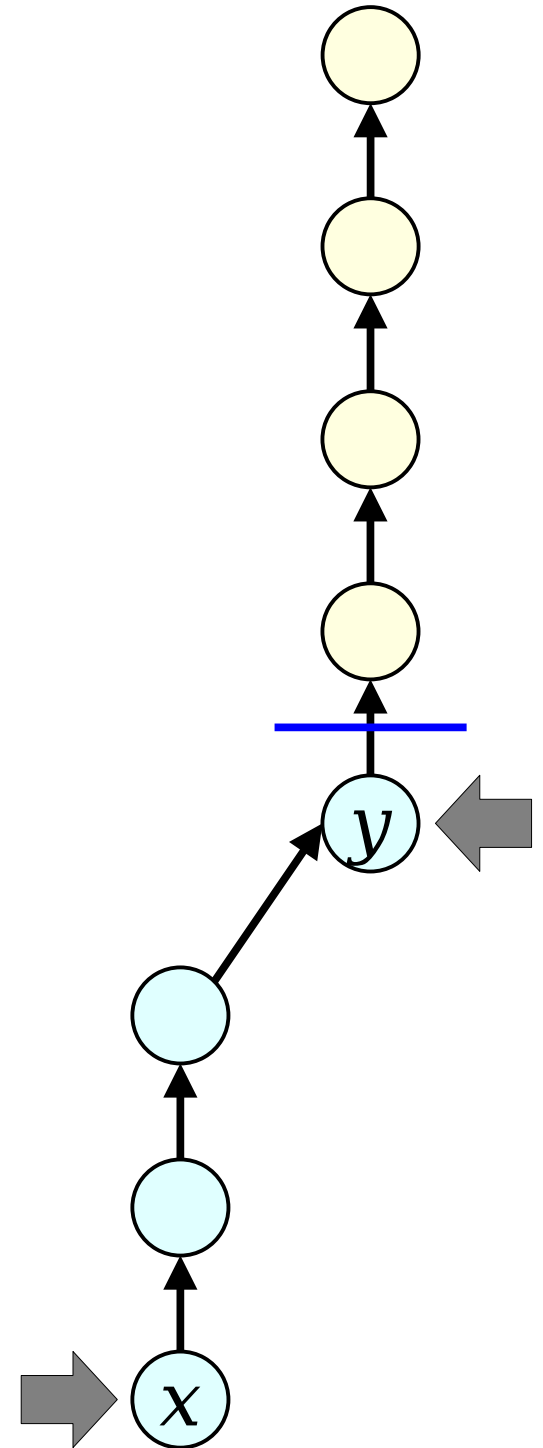
Forest Slicing

- **Case 2:** x and y are both in \mathcal{F}_- .



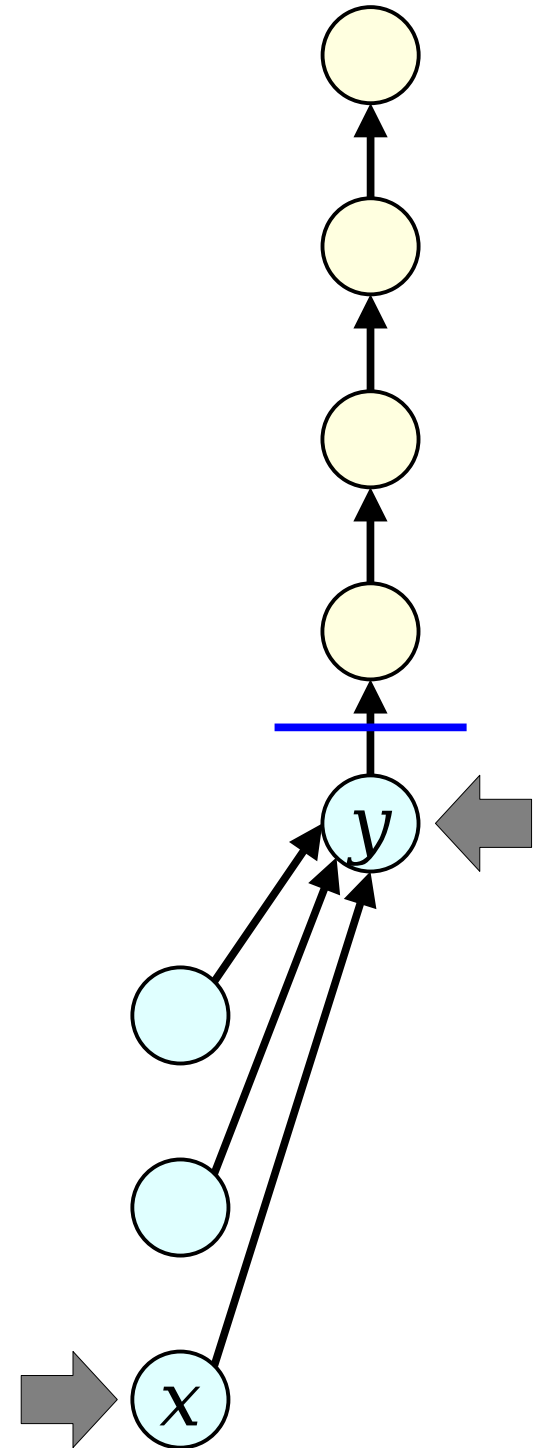
Forest Slicing

- **Case 2:** x and y are both in \mathcal{F}_- .



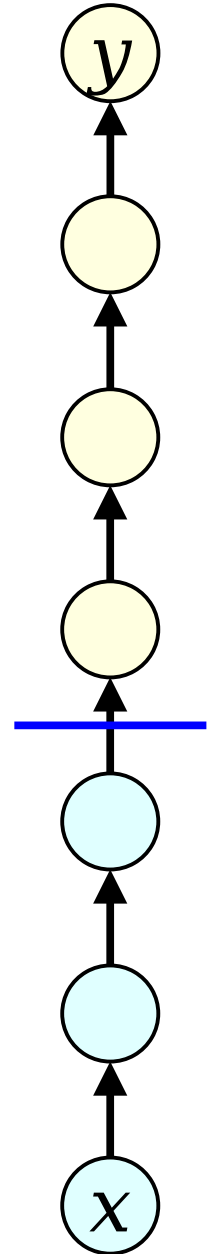
Forest Slicing

- **Case 2:** x and y are both in \mathcal{F}_- .
- We can recursively handle this compression when bounding the work done purely in \mathcal{F}_- .



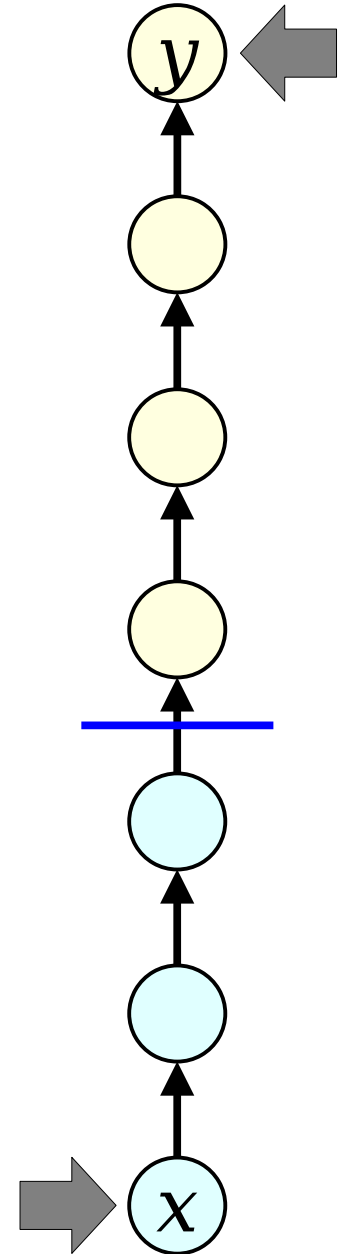
Forest Slicing

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



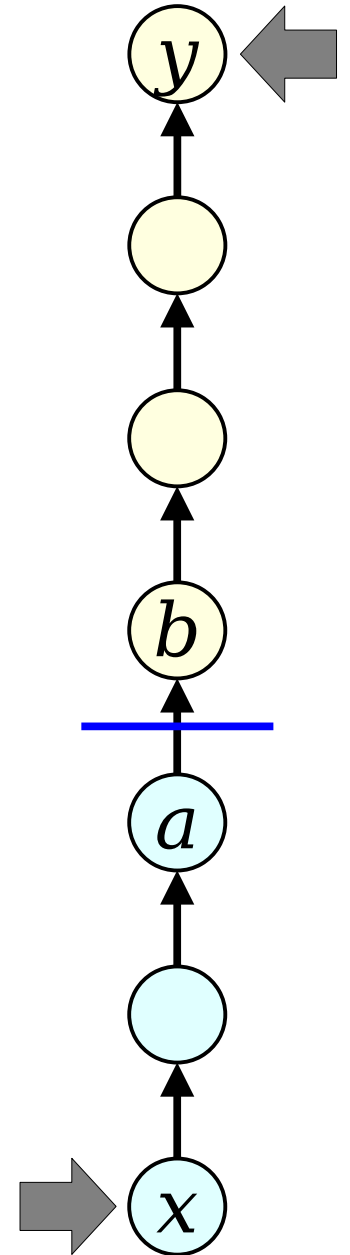
Forest Slicing

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



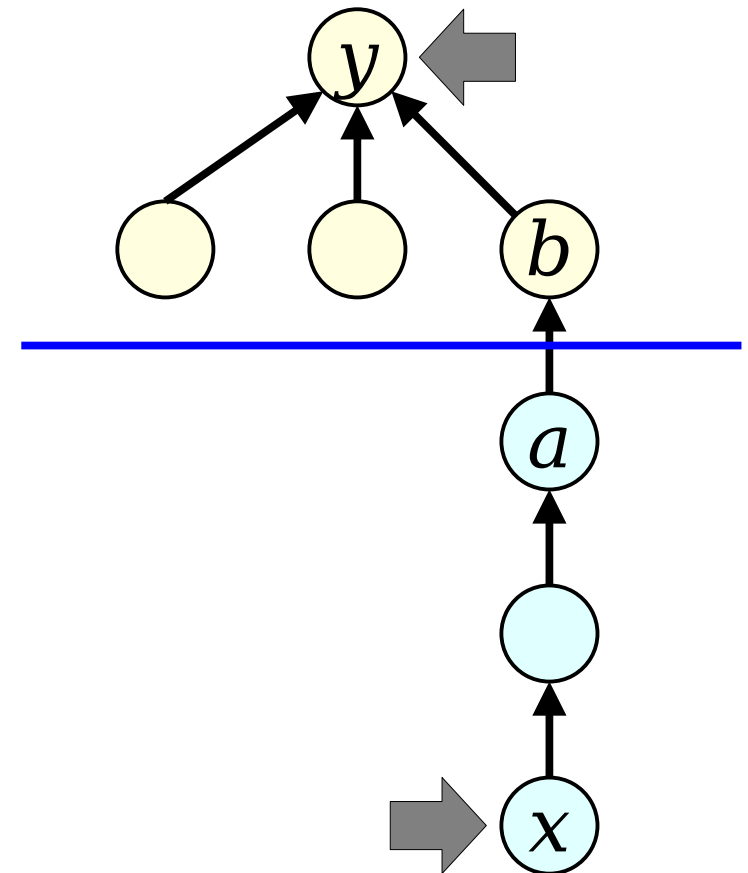
Forest Slicing

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



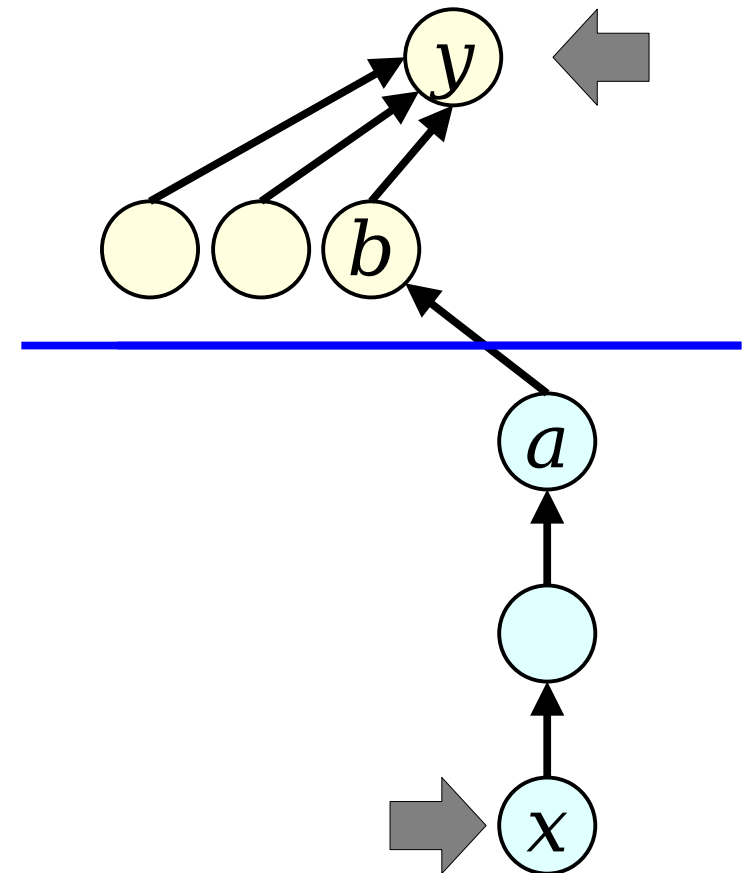
Forest Slicing

- **Case 3:** x is in \mathcal{F}_- and y is in \mathcal{F}_+ .
- We compress from b to y , purely in \mathcal{F}_+ .



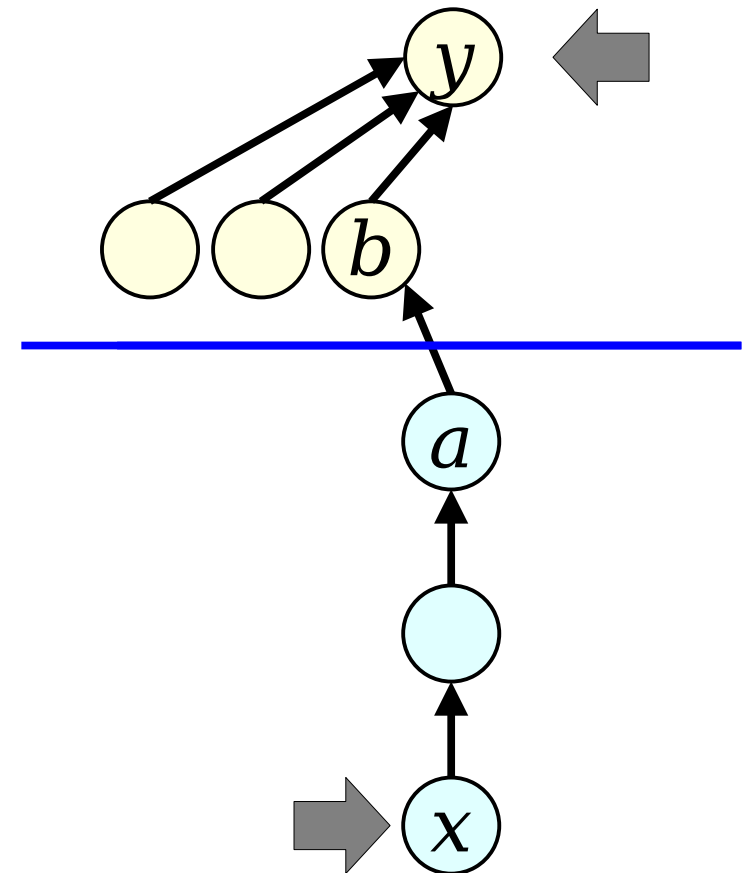
Forest Slicing

- **Case 3:** x is in \mathcal{F}_- and y is in \mathcal{F}_+ .
- We compress from b to y , purely in \mathcal{F}_+ .



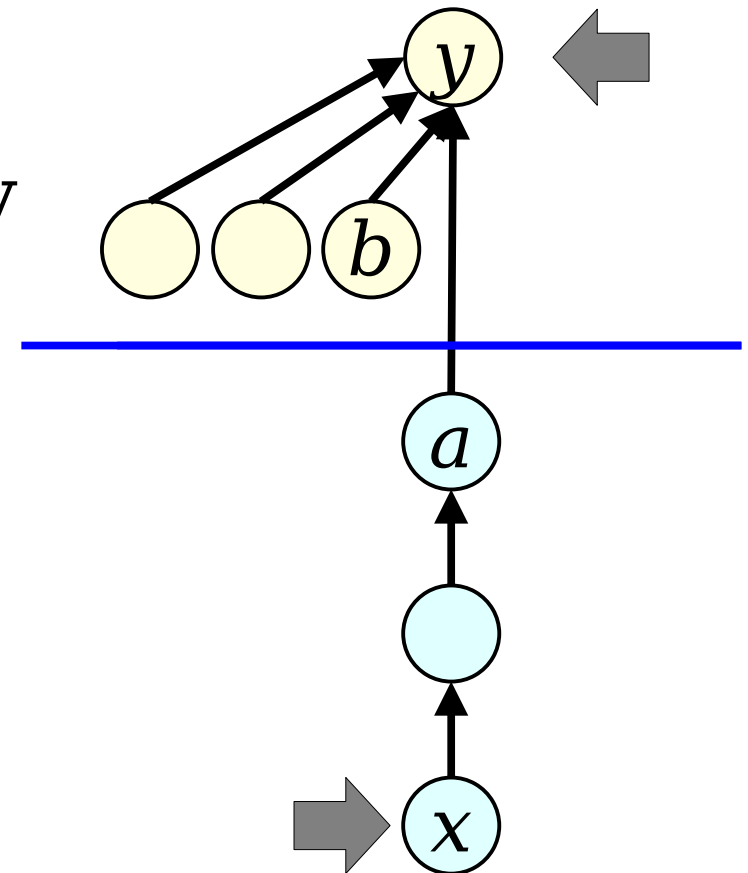
Forest Slicing

- **Case 3:** x is in \mathcal{F}_- and y is in \mathcal{F}_+ .
- We compress from b to y , purely in \mathcal{F}_+ .



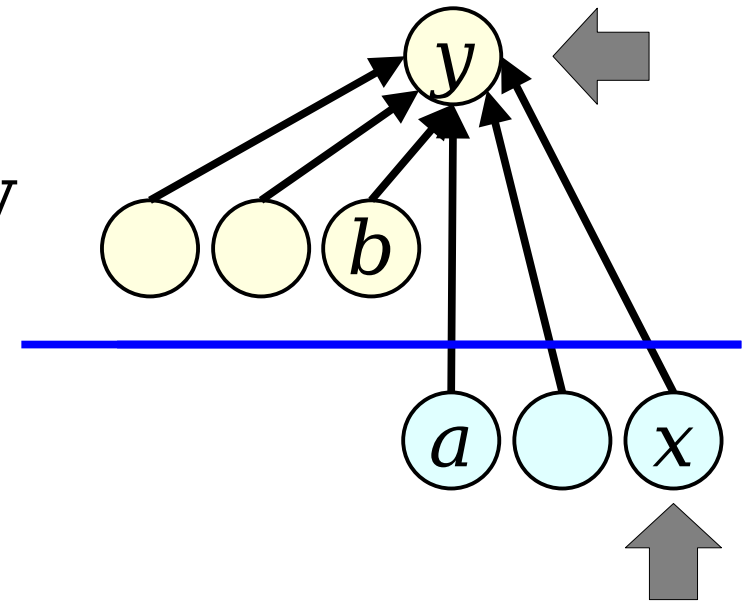
Forest Slicing

- **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}_+ .



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

Forest Slicing

- **Claim:** The cost of compressions crossing from \mathcal{F}_- to \mathcal{F}_+ can be bounded by
 - the cost of some compressions done purely in \mathcal{F}_+ (the top parts of the compressions),
 - the total number of compressions from \mathcal{F}_- to \mathcal{F}_+ (changing the parents of nodes in \mathcal{F}_- whose parents are already in \mathcal{F}_+), and
 - the number of nodes in \mathcal{F}_- whose parents are in \mathcal{F}_- (each of which may get a parent in \mathcal{F}_+ for the first time at most once).

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}_+ .	
Cost of compressions purely in \mathcal{F}_- .	
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}_+ .	
Cost of compressions purely in \mathcal{F}_- .	
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}_+ .	$C(??, ??, ??)$
Cost of compressions purely in \mathcal{F}_- .	
The number of compresses from \mathcal{F}_- to \mathcal{F}_+	
Number of nodes in \mathcal{F}_- with parents in \mathcal{F}_- .	

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

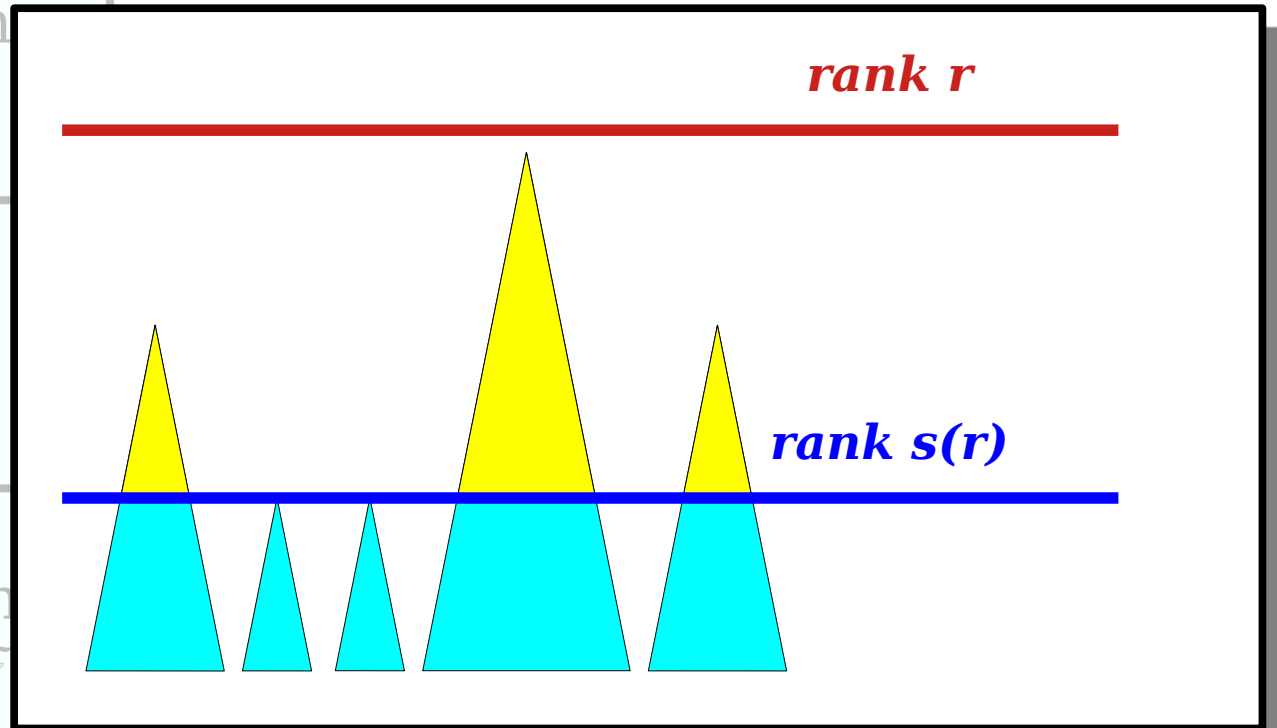
Cost of compressions
purely in \mathcal{F}_+ .

$$C(??, ??, ??)$$

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

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

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



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

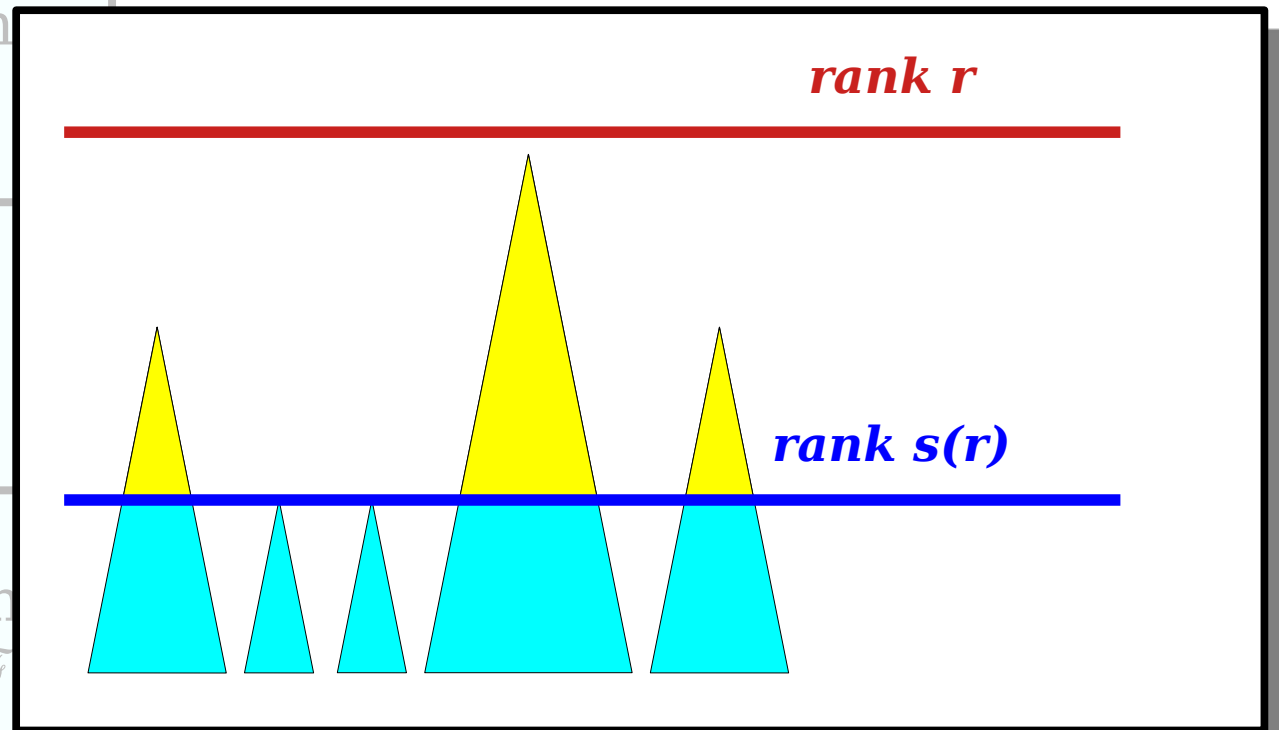
Cost of compressions
purely in \mathcal{F}_+ .

$$C(??, ??, \mathbf{r - s(r) - 1})$$

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

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

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



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

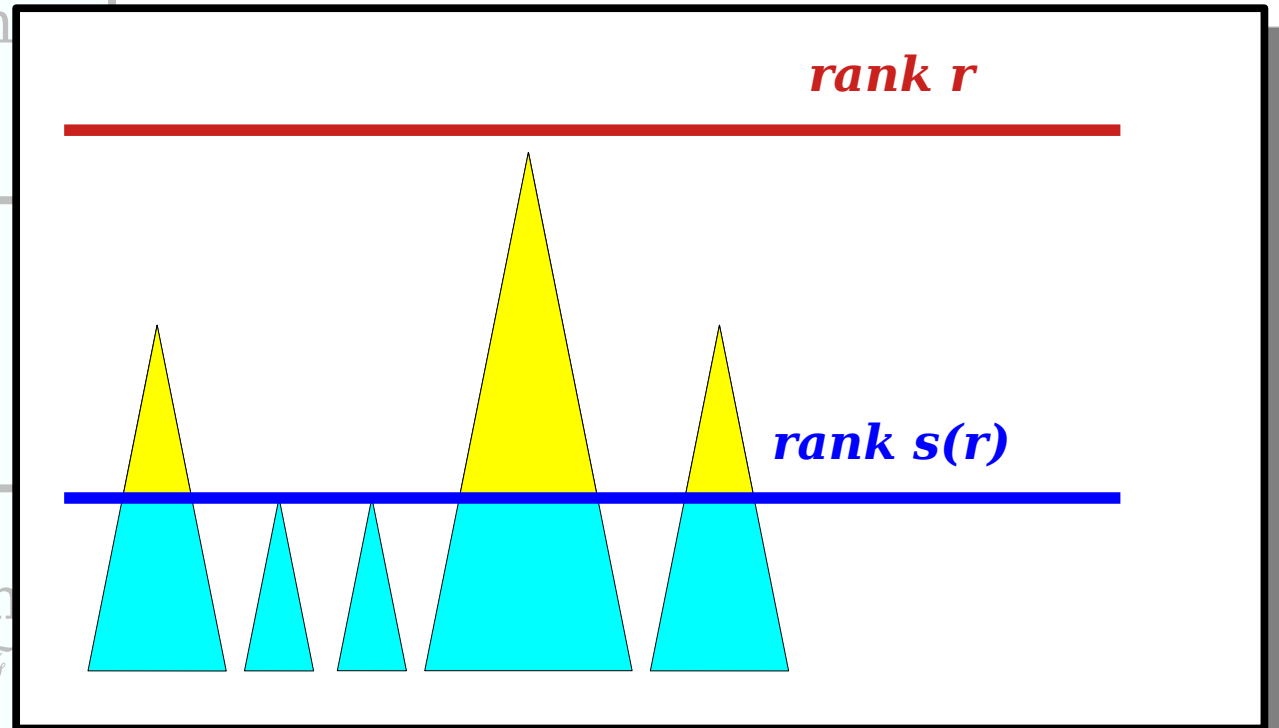
Cost of compressions
purely in \mathcal{F}_+ .

$$C(??, ??, \mathbf{r})$$

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

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

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



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

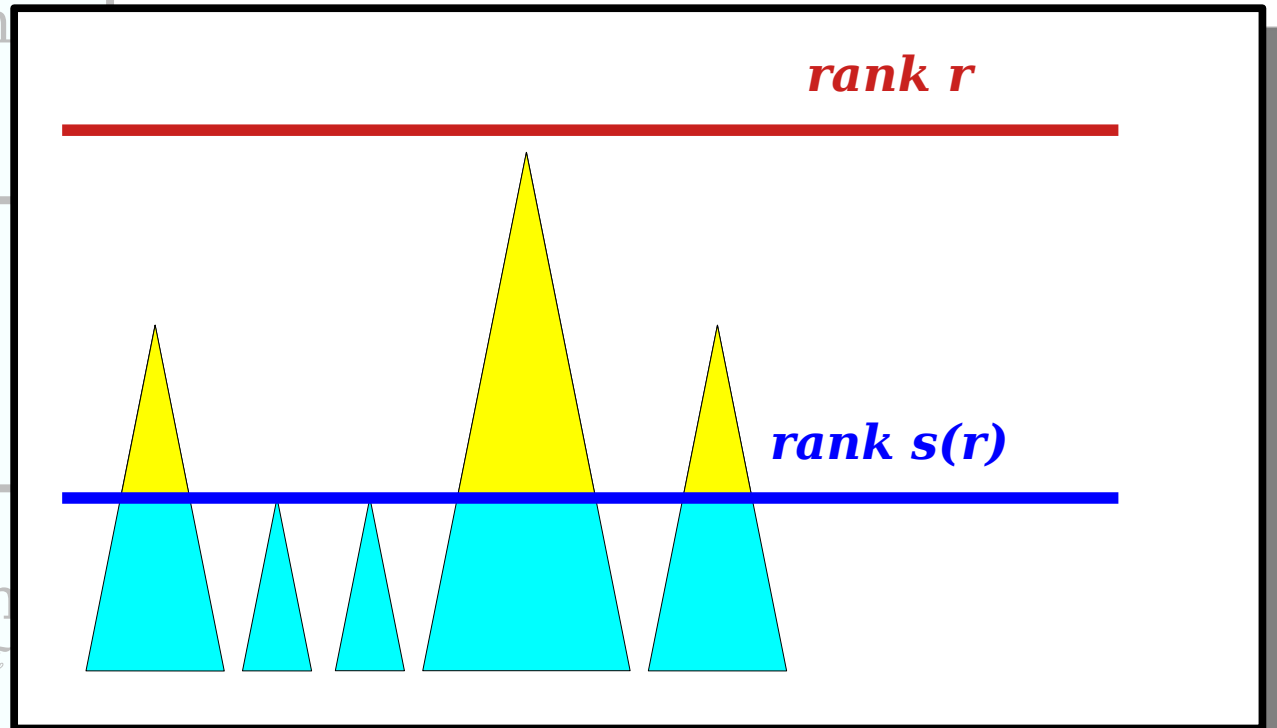
Cost of compressions
purely in \mathcal{F}_+ .

$$C(??, \mathbf{n}_+, \mathbf{r})$$

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

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}_+ .	$C(??, \mathbf{n}_+, \mathbf{r})$
Cost of compressions purely in \mathcal{F}_- .	
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}_+ .

$$C(??, \mathbf{n}_+, \mathbf{r})$$

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 \mathbf{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}_+ .

$$C(\mathbf{m}_+, \mathbf{n}_+, \mathbf{r})$$

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

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

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

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

Let \mathbf{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}_+ .

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

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}_- .	
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}_- .	
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}_- .	$C(??, ??, ??)$
The number of compresses from \mathcal{F}_- to \mathcal{F}_+	
Number of nodes in \mathcal{F}_- with parents in \mathcal{F}_- .	

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

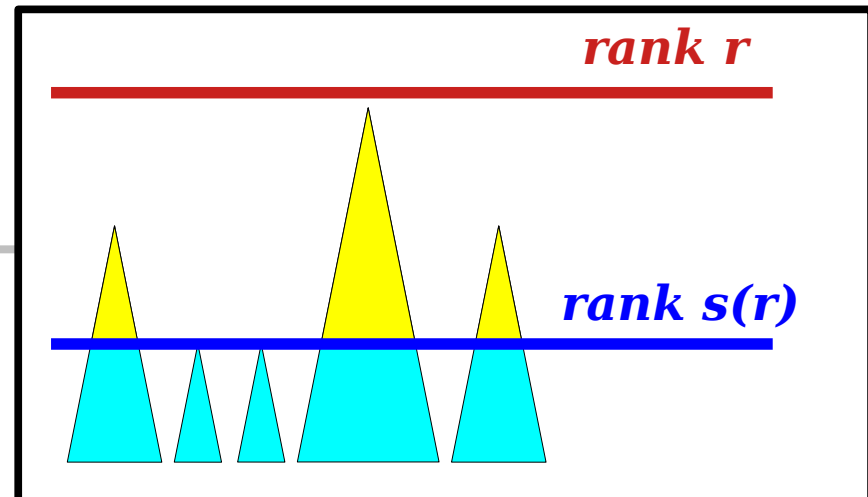
Cost of compressions
purely in \mathcal{F}_+ .

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

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

$$C(??, ??, ??)$$

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



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

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

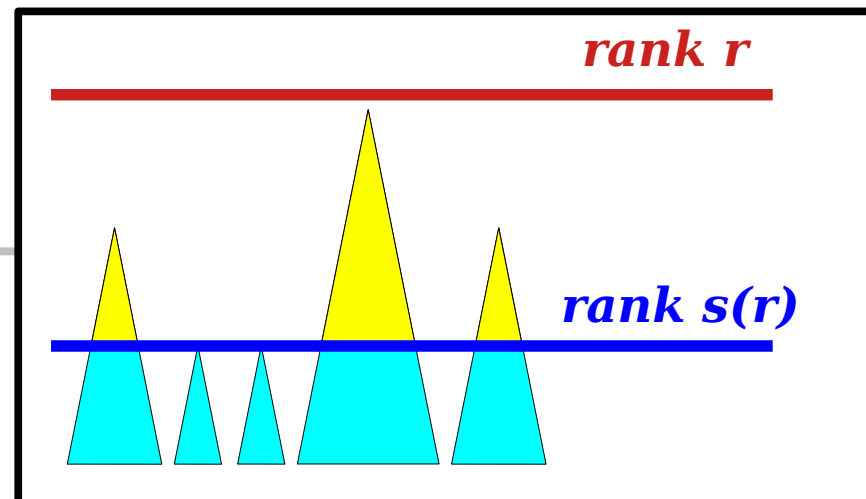
Cost of compressions
purely in \mathcal{F}_+ .

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

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

$$C(??, ??, \mathbf{s(r)})$$

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



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

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

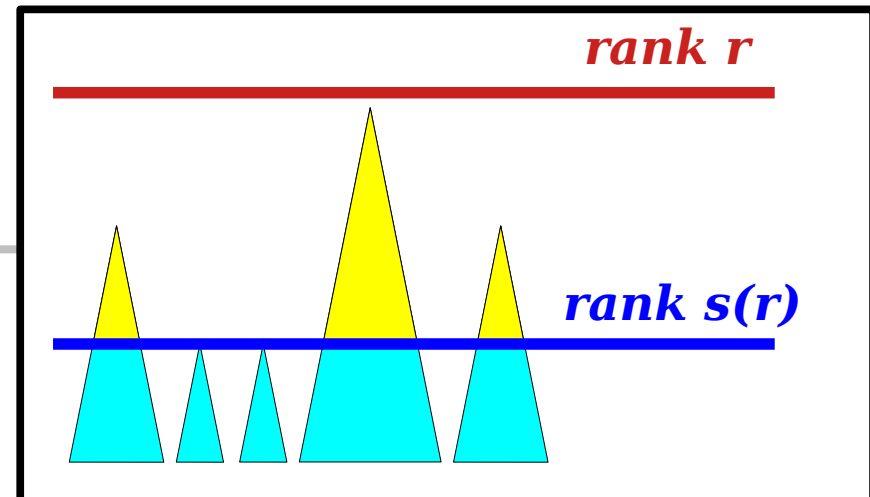
Cost of compressions
purely in \mathcal{F}_+ .

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

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

$$C(??, n - n_+, s(r))$$

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



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

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

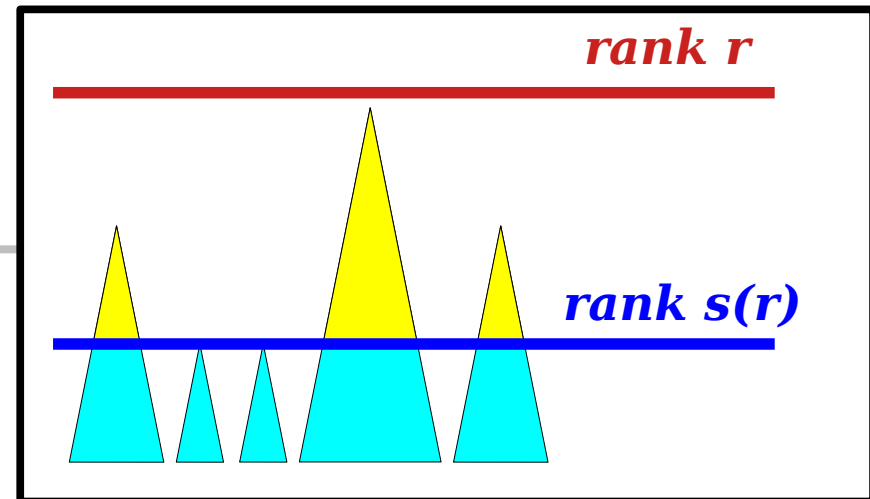
Cost of compressions
purely in \mathcal{F}_+ .

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

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

$$C(??, n, s(r))$$

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}_+ .

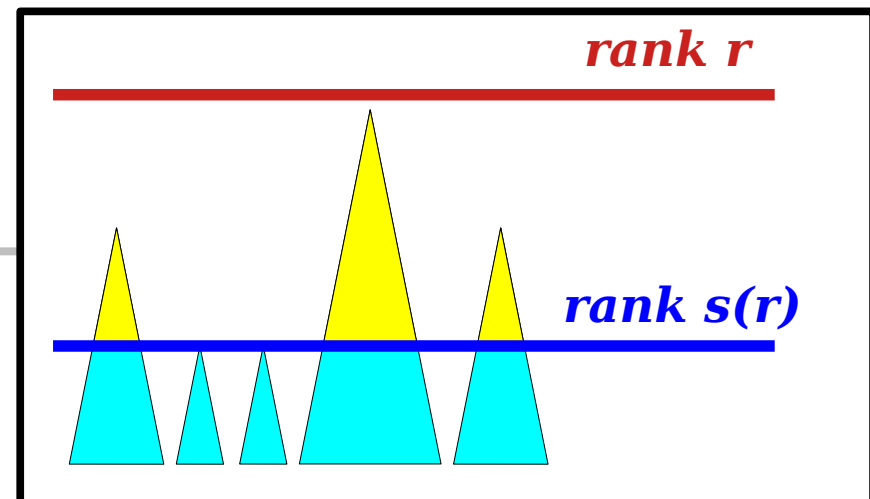
$$C(m_+, n_+, r)$$

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

$$C(\mathbf{m} - \mathbf{m}_+, \mathbf{n}, \mathbf{s}(r))$$

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}_+ .

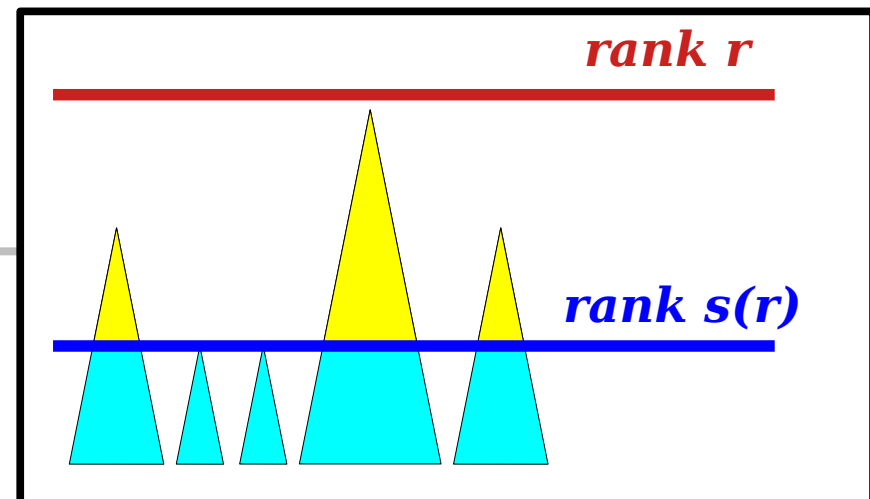
$$C(m_+, n_+, r)$$

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

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

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

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

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

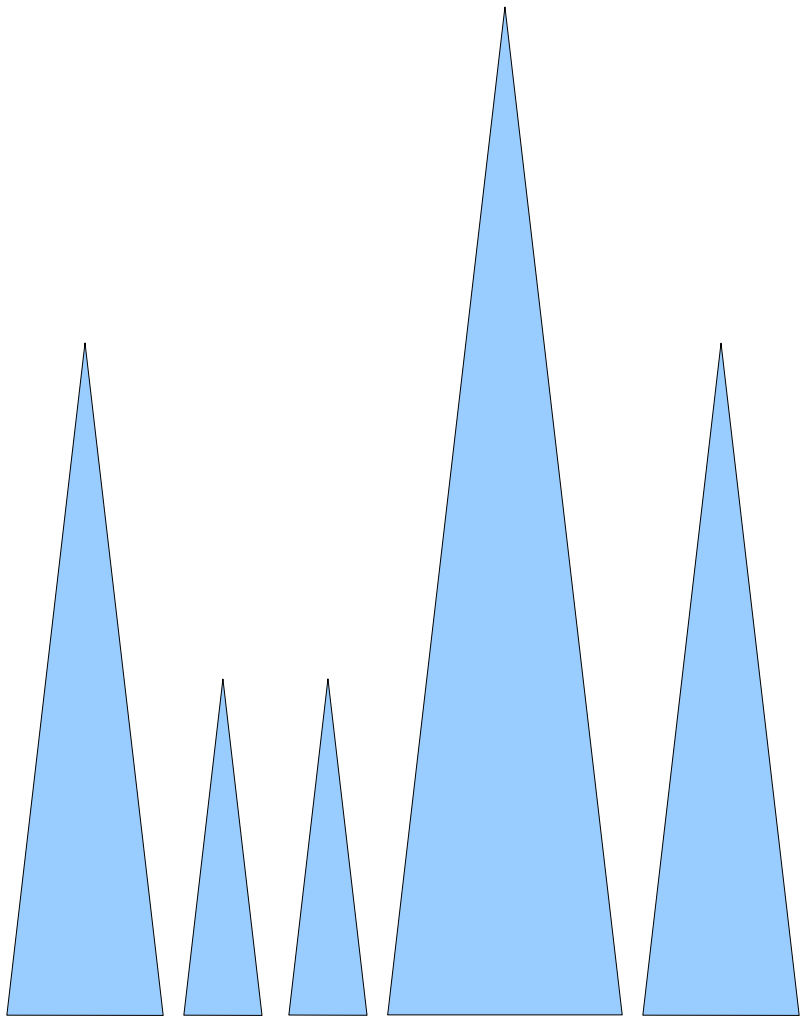
Cost of compressions purely in \mathcal{F}_+ .	$C(m_+, n_+, r)$
Cost of compressions purely in \mathcal{F}_- .	$C(m - m_+, n, s(r))$
The number of compresses from \mathcal{F}_- to \mathcal{F}_+	m_+
Number of nodes in \mathcal{F}_- with parents in \mathcal{F}_- .	(Since this includes all compresses from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

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

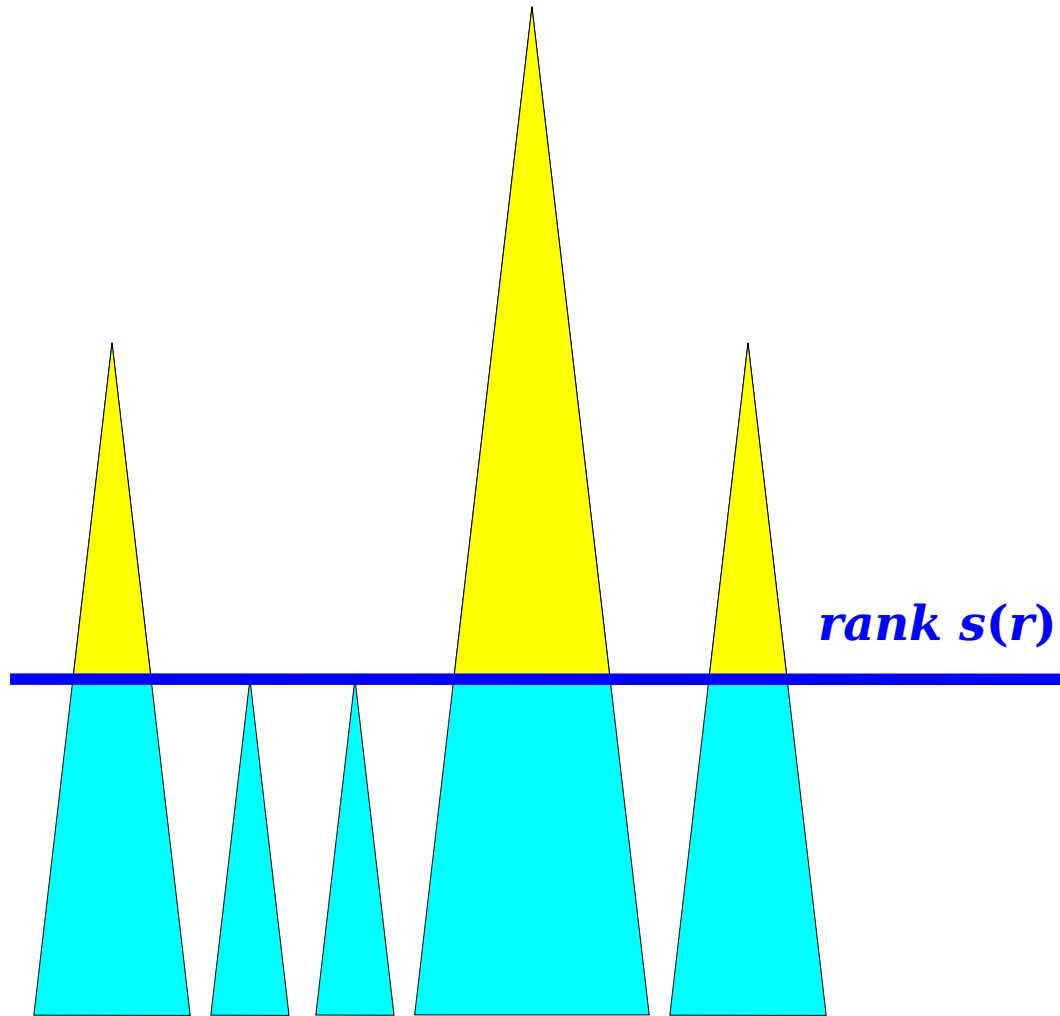
Cost of compressions purely in \mathcal{F}_+ .	$C(m_+, n_+, r)$
Cost of compressions purely in \mathcal{F}_- .	$C(m - m_+, n, s(r))$
The number of compresses from \mathcal{F}_- to \mathcal{F}_+	m_+
Number of nodes in \mathcal{F}_- with parents in \mathcal{F}_- .	

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

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}_- .	

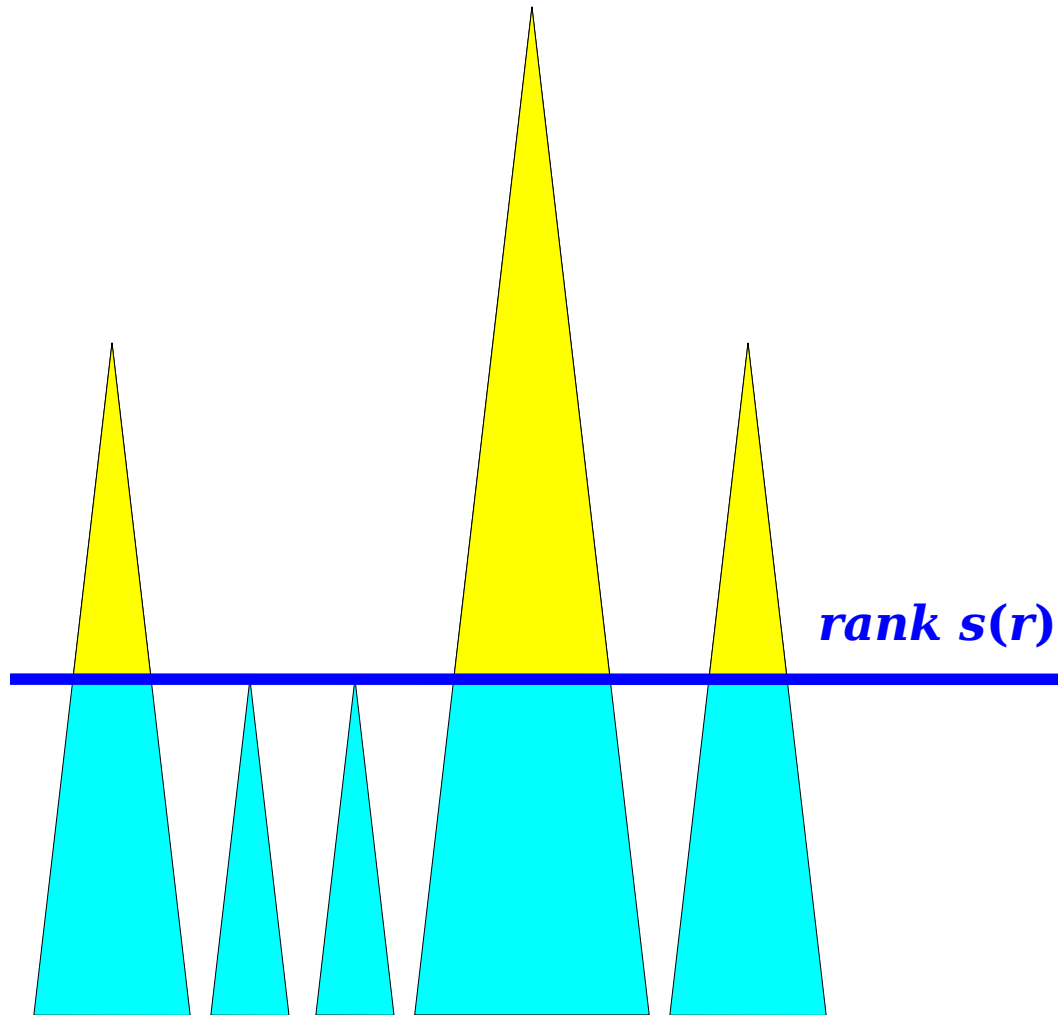


Number of nodes in
 \mathcal{F}^- with parents in \mathcal{F}^- .

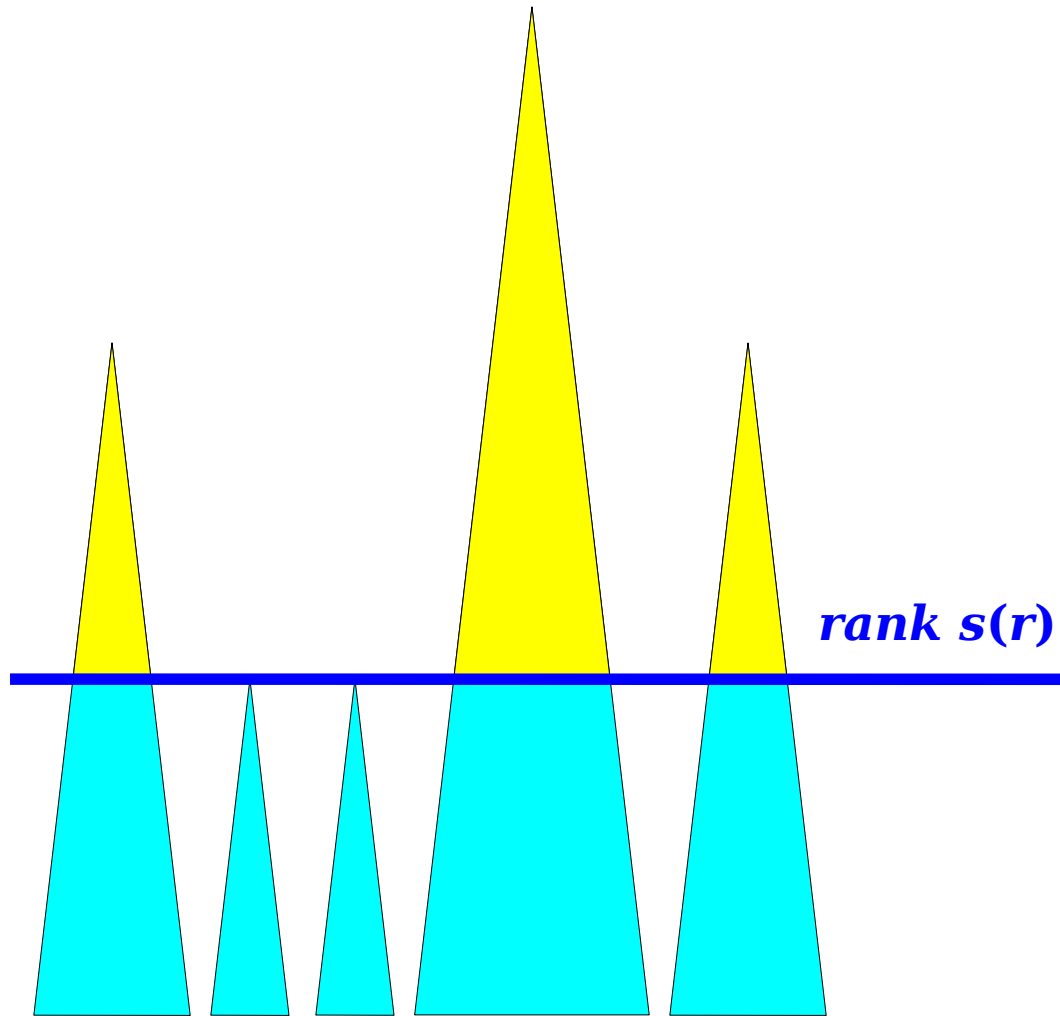


Number of nodes in \mathcal{F}_- with parents in \mathcal{F}_- .

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



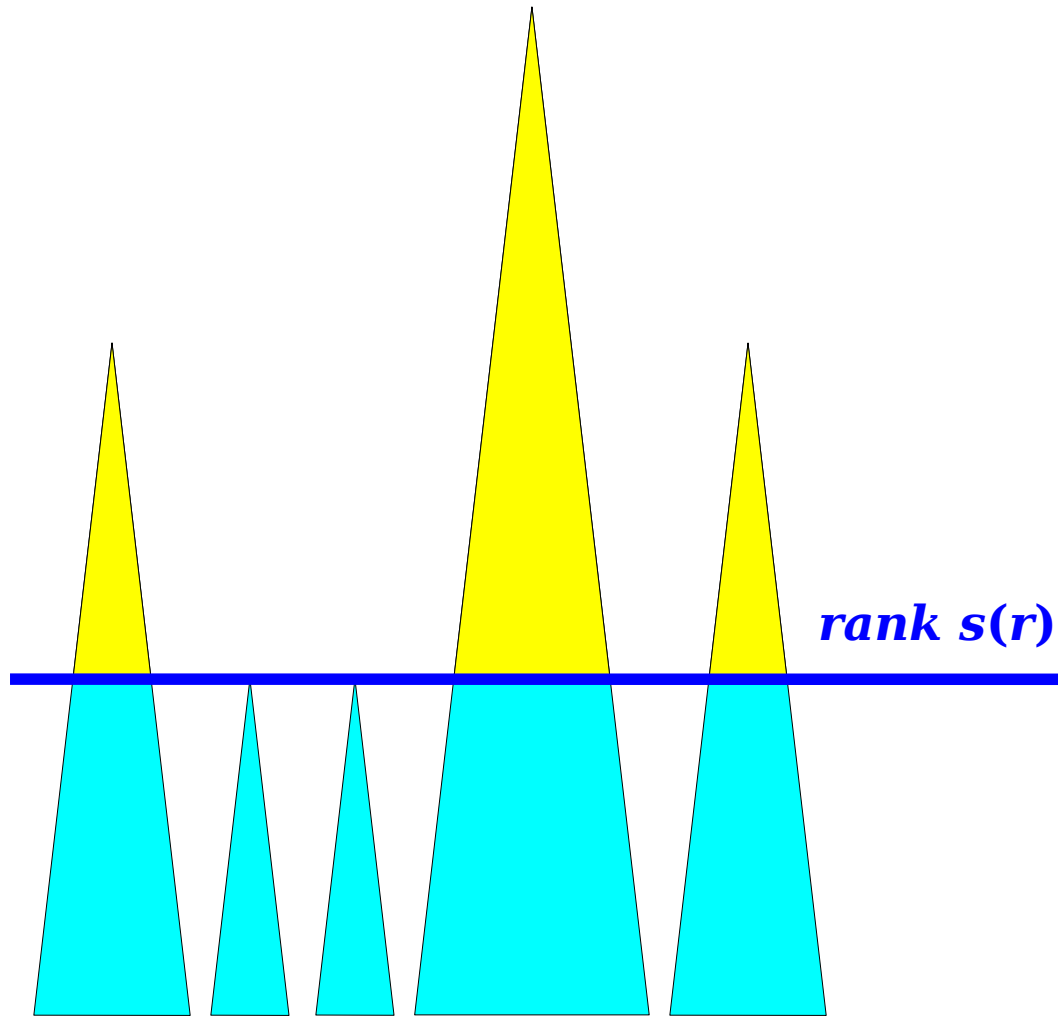
Number of nodes in \mathcal{F}_- with parents in \mathcal{F}_- .



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

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

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

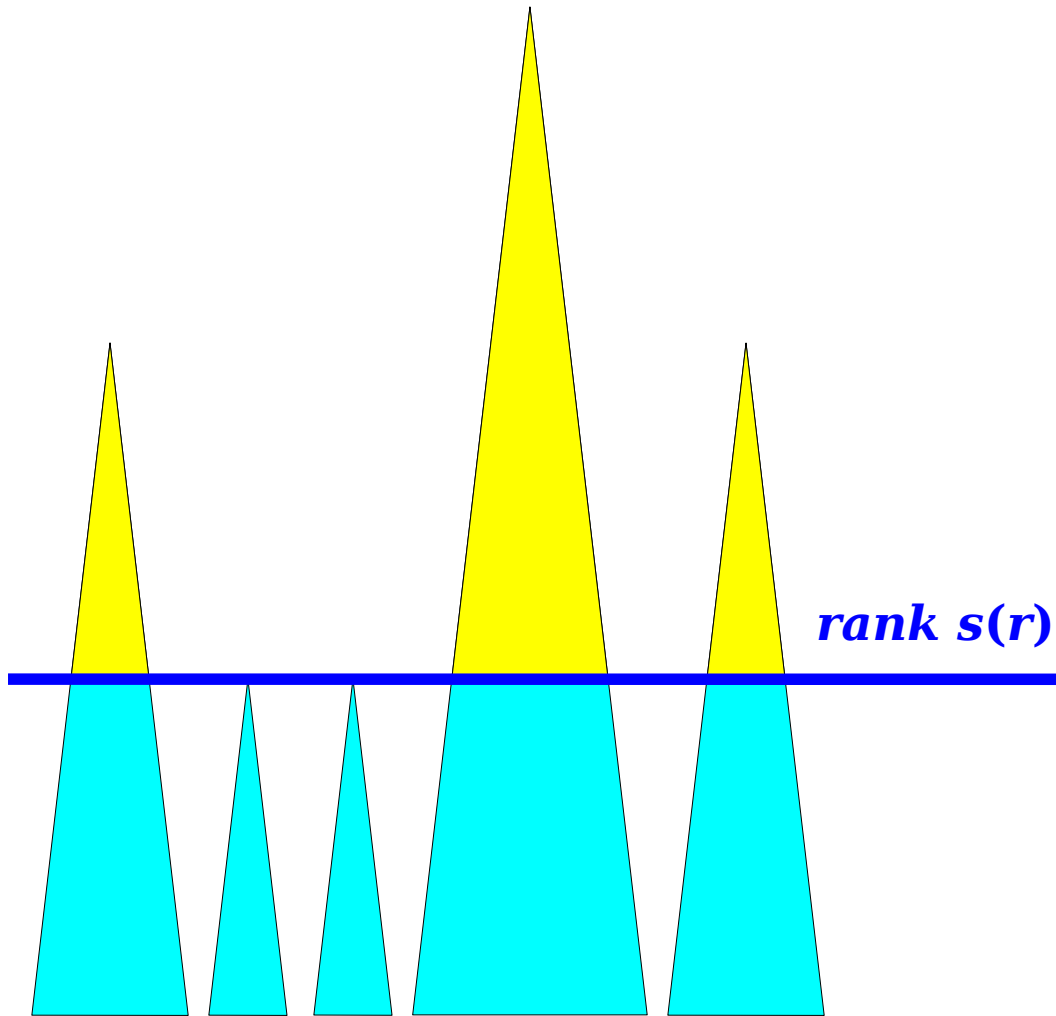


Every node in \mathcal{F}_+ has rank $s(r) + 1$ or greater.

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, and they're all in \mathcal{F}_- .

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



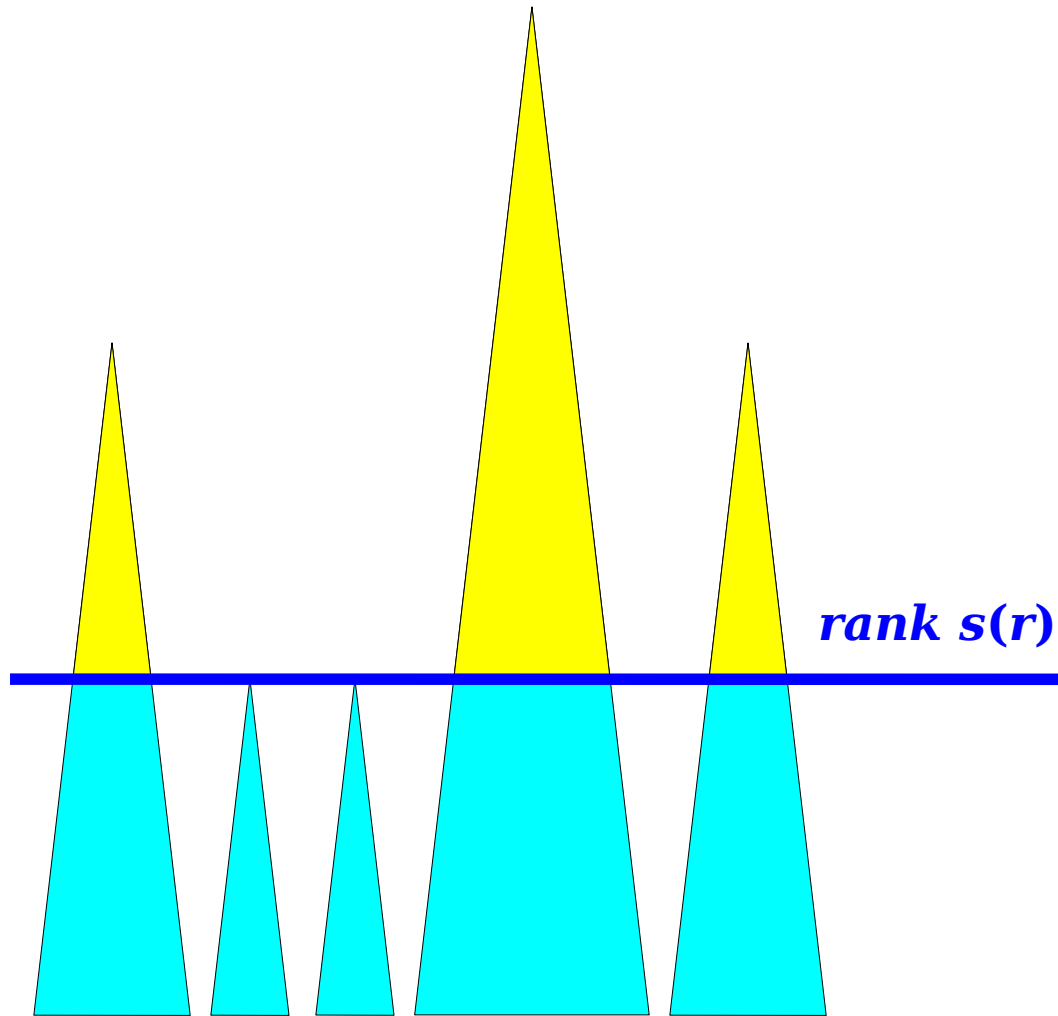
Every node in \mathcal{F}_+ has rank $s(r) + 1$ or greater.

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, and they're all in \mathcal{F}_- .

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

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



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

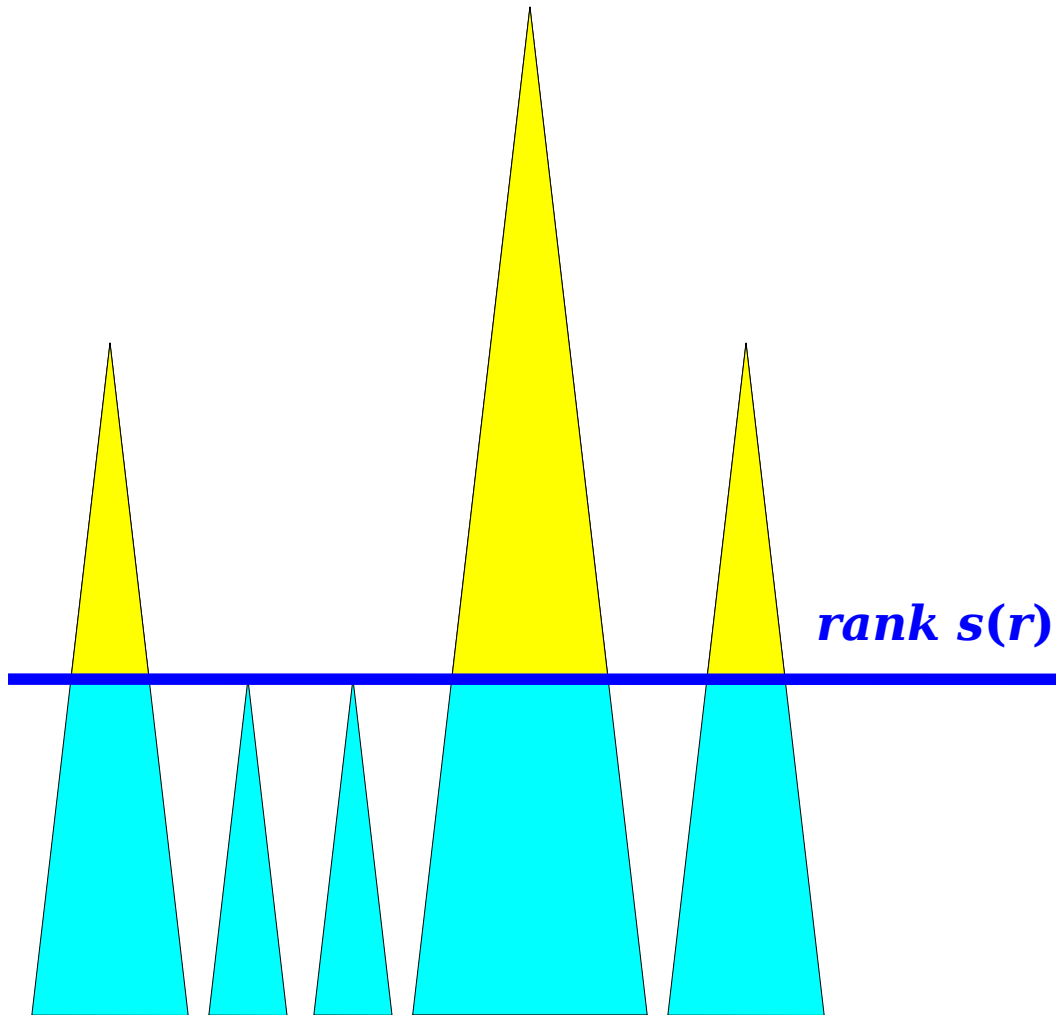
Each rank- k node has children of ranks $0, 1, 2, \dots, k - 1$.

So every node in \mathcal{F}_+ has at least $s(r) + 1$ children, and they're all in \mathcal{F}_- .

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

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

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



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

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

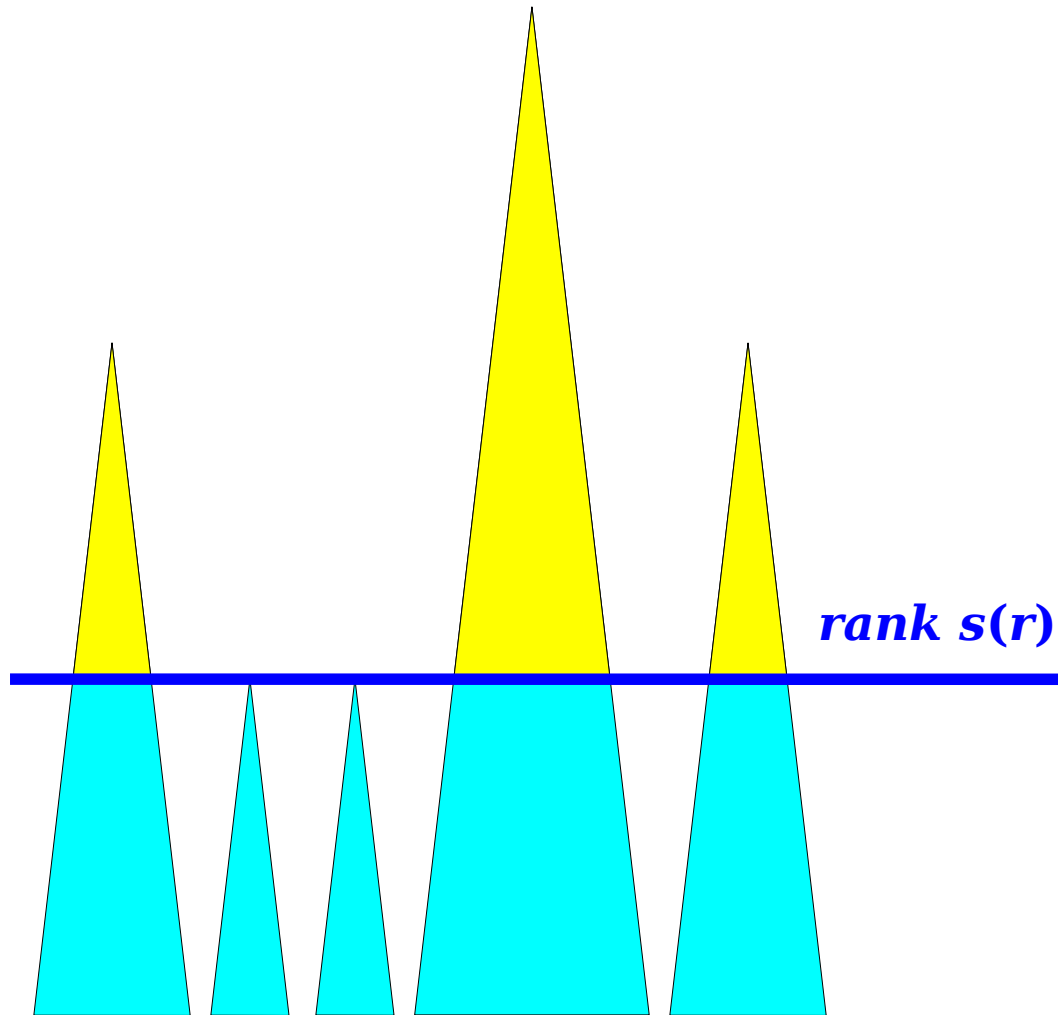
So every node in \mathcal{F}_+ has at least $s(r) + 1$ children, and they're all in \mathcal{F}_- .

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}_+ : $n_+ \cdot (s(r) + 1)$

Number of nodes in \mathcal{F}_- with parents in \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, and they're all in \mathcal{F}_- .

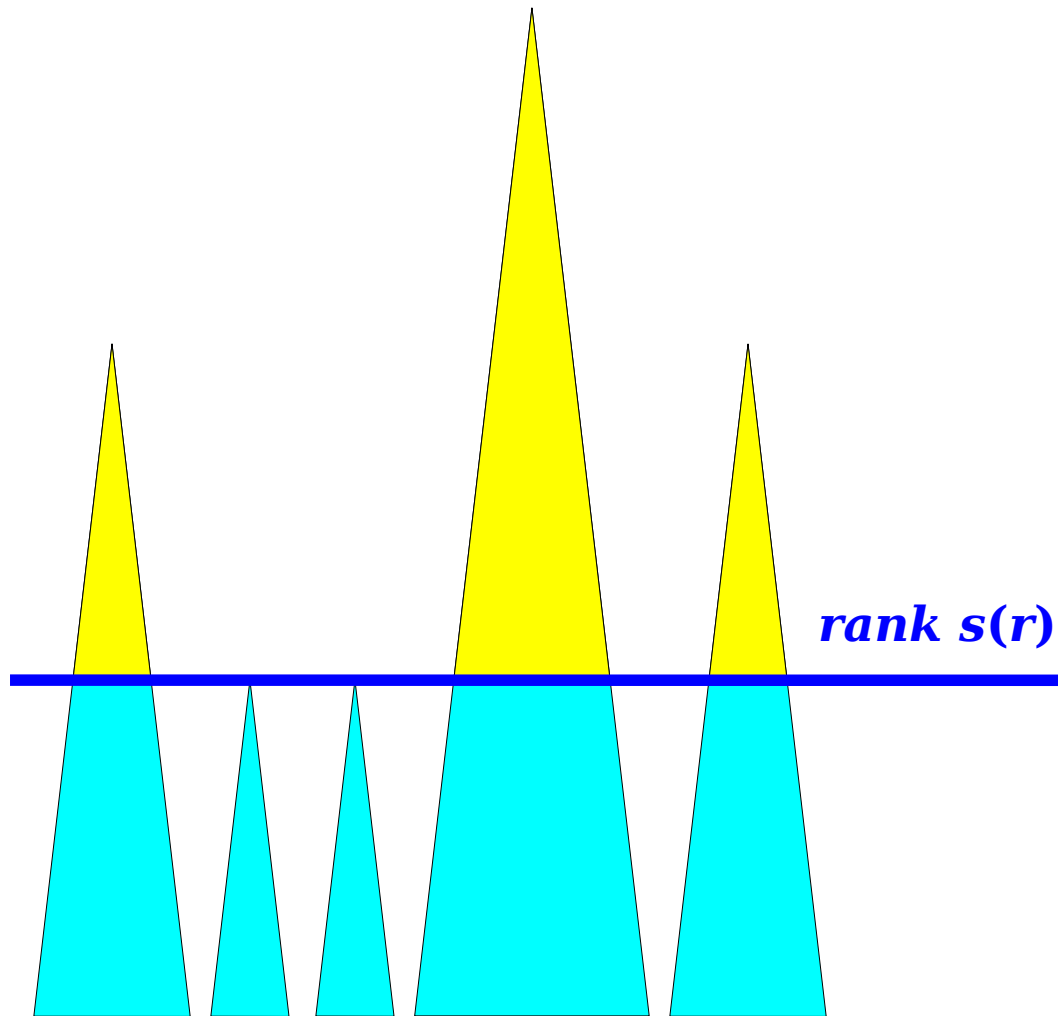
There are n total nodes, and n_+ of them are in \mathcal{F}_+ .

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}_- .



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, and they're all 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}_-.	$n - n_+ \cdot (s(r) + 2)$

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

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}_-.	$n - n_+ \cdot s(r)$

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

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:

$$\begin{aligned} C(m, n, r) \leq & C(m - m_+, n, s(r)) + \\ & C(m_+, n_+, r) + \\ & m_+ + n - n_+ \cdot s(r). \end{aligned}$$

- 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

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

- **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

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

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

The Recurrence

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

- **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

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

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

The Recurrence

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

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

The Recurrence

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

- **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

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

- **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

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

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

The Recurrence

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

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

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, r / 2) + 2m_+ + n + n_+ \cdot (0).$$

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

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, r / 2) + 2m_+ + n + n_+ \cdot (0).$$

- ***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 $s(r) = r / 2$.

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.

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

Starting with m ...

...take some
of m away...

... and add it
into the total.

$$C(\begin{array}{c} \bullet \bullet \bullet \\ \bullet \bullet \bullet \\ \bullet \bullet \bullet \end{array}, n, r) \leq 2(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \end{array}) + n + C(\begin{array}{c} \bullet \bullet \\ \bullet \bullet \\ \bullet \bullet \end{array}, n, r / 2)$$

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

Starting with m ...

...take some
of m away...

... and add it
into the total.

$$C(\begin{array}{c} \bullet \bullet \bullet \\ \bullet \bullet \bullet \\ \bullet \bullet \bullet \end{array}, n, r) \leq 2(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \end{array}) + n$$

$$2(\begin{array}{c} \bullet \bullet \end{array}) + n + C(\begin{array}{c} \bullet \bullet \bullet \\ \bullet \bullet \bullet \end{array}, n, r/4)$$

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

Starting with m ...

...take some
of m away...

... and add it
into the total.

$$C(\text{cluster of 12 colored dots}, n, r) \leq 2(\text{cluster of 4 green dots}) + n$$

$$2(\text{cluster of 2 pink dots}) + n$$

$$2(\text{cluster of 3 light blue dots}) + n + C(\text{cluster of 3 dots: 1 grey, 2 yellow}, n, r/8)$$

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

Starting with m ...

...take some
of m away...

... and add it
into the total.

$$C(\text{cluster of 12 colored dots}, n, r) \leq 2(\text{cluster of 4 green dots}) + n$$

$$2(\text{cluster of 2 pink dots}) + n$$

$$2(\text{cluster of 3 light blue dots}) + n$$

$$2(\text{single grey dot}) + n + C(\text{cluster of 2 yellow dots}, n, r/16)$$

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

Starting with m ...

...take some of m away...

... and add it into the total.

$$C(\text{cluster of 12 colored dots}, n, r) \leq 2(\text{cluster of 4 green dots}) + n$$

$$2(\text{cluster of 2 pink dots}) + n$$

$$2(\text{cluster of 3 light blue dots}) + n$$

$$2(\text{single grey dot}) + n$$

...

$$\leq 2(\text{cluster of 12 colored dots}) + \underline{\hspace{2cm}}$$

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

Starting with m ...

...take some of m away...

... and add it into the total.

$$C(m, n, r) \leq 2(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array}) + n$$

$$2(\bullet \bullet \bullet) + n$$

$$2(\begin{array}{c} \bullet \\ \bullet \end{array}) + n$$

$$2(\bullet) + n$$

...

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

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

How many layers
can this recursion
have?

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

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

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

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

...

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

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

How many layers
can this recursion
have?

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

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

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

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

...

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

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

How many layers
can this recursion
have?

$$C(m, n, r) \leq 2(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array}) + n$$

$$2(\bullet \bullet \bullet) + n$$

$$2(\begin{array}{c} \bullet \\ \bullet \end{array}) + n$$

$$2(\bullet) + n$$

...

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

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

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

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

- ~~**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

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

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

The Recurrence

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

- ~~**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

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

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

The Recurrence

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

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

The Recurrence

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

- ~~**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

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

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

The Recurrence

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

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

The Recurrence

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

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

The Recurrence

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

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

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, \lg r) + \\ 3m_+ + n + \\ n_+ \cdot (0).$$

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

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, \lg r) + 3m_+ + n + n_+ \cdot (0).$$

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

The Recurrence

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

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

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.

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

Starting with m ...

...take some
of m away...

... and add it
into the total.

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

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

Starting with m ...

...take some of m away...

... and add it into the total.

$$C(\text{cluster of 12 colored dots}, n, r) \leq 3(\text{cluster of 4 green dots}) + n$$

$$3(\text{cluster of 3 pink dots}) + n + C(\text{cluster of 6 colored dots}, n, \lg \lg r)$$

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

Starting with m ...

...take some of m away...

... and add it into the total.

$$C(\text{cluster of 12 colored dots}, n, r) \leq 3(\text{cluster of 4 green dots}) + n$$

$$3(\text{cluster of 2 pink dots}) + n$$

$$3(\text{cluster of 3 light blue dots}) + n + C(\text{cluster of 3 dots: 1 grey, 2 yellow}, n, \lg \lg \lg r)$$

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

Starting with m ...

...take some of m away...

... and add it into the total.

$$C(\text{cluster of 13 colored dots}, n, r) \leq 3(\text{cluster of 4 green dots}) + n$$

$$3(\text{cluster of 2 pink dots}) + n$$

$$3(\text{cluster of 3 light blue dots}) + n$$

$$3(\text{single grey dot}) + n$$

...

$$\leq 3(\text{cluster of 13 colored dots}) + \underline{\hspace{2cm}}$$

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

Starting with m ...

...take some
of m away...

... and add it
into the total.

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

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

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

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

...

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

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

How many layers
can this recursion
have?

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

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

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

$$3(\bullet) + n$$

...

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

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

How many layers
can this recursion
have?

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

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

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

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

...

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

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

How many layers
can this recursion
have?

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

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

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

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

...

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

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?***

Notice Something?

If we start with this bound on $C(m, n, r)$ then we get this stronger bound on $C(m, n, r)$:
$m + n \cdot r / 2$	$2m + n \lg r$
$2m + n \lg r$	$3m + n \lg^* r$
$km + n f(r)$??

Answer at

<https://pollev.com/cs166spr23>

Notice Something?

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

Notice Something?

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

Notice Something?

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

The Recurrence

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

- **Assume:** $C(m, n, r) \leq km + n \cdot f(r)$

The Recurrence

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

- **Assume:** $C(m, n, r) \leq km + n \cdot f(r)$

The Recurrence

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

- **Assume:** $C(m, n, r) \leq km + n \cdot f(r)$

The Recurrence

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

- **Assume:** $C(m, n, r) \leq km + n \cdot f(r)$

The Recurrence

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

- **Assume:** $C(m, n, r) \leq km + n \cdot f(r)$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + (k+1)m_+ + n_+ \cdot f(r) + n - n_+ \cdot s(r).$$

- **Assume:** $C(m, n, r) \leq km + n \cdot f(r)$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + (k+1)m_+ + n_+ \cdot f(r) + n - n_+ \cdot s(r).$$

- **Assume:** $C(m, n, r) \leq km + n \cdot f(r)$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + (k+1)m_+ + n_+ \cdot f(r) + n - n_+ \cdot s(r).$$

- **Assume:** $C(m, n, r) \leq km + n \cdot f(r)$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + (k+1)m_+ + n + n_+ \cdot f(r) - n_+ \cdot s(r).$$

- **Assume:** $C(m, n, r) \leq km + n \cdot f(r)$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + \\ (k+1)m_+ + n + \\ n_+ \cdot (f(r) - s(r)).$$

- **Assume:** $C(m, n, r) \leq km + n \cdot f(r)$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + \\ (k+1)m_+ + n + \\ n_+ \cdot (f(r) - s(r)).$$

- **Assume:** $C(m, n, r) \leq km + n \cdot f(r)$

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + \\ (k+1)m_+ + n + \\ n_+ \cdot (f(r) - s(r)).$$

- ***Clever Idea:*** Set $s(r) = f(r)$.

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, s(r)) + \\ (k+1)m_+ + n + \\ n_+ \cdot (f(r) - s(r)).$$

- ***Clever Idea:*** Set $s(r) = f(r)$.

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n + n_+ \cdot (f(r) - f(r)).$$

- ***Clever Idea:*** Set $s(r) = f(r)$.

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n + n_+ \cdot (0).$$

- ***Clever Idea:*** Set $s(r) = f(r)$.

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n + n_+ \cdot (0).$$

- ***Clever Idea:*** Set $s(r) = f(r)$.

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n.$$

- ***Clever Idea:*** Set $s(r) = f(r)$.

The Recurrence

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n.$$

- ***Clever Idea:*** Set $s(r) = f(r)$.

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n$$

Starting with m ...

...take some
of m away...

... and add it
into the total.

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n$$

Starting with m ...

...take some
of m away...

... and add it
into the total.

$$C(\begin{array}{c} \bullet \bullet \bullet \\ \bullet \bullet \bullet \\ \bullet \bullet \bullet \end{array}, n, r) \leq (k+1)(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \end{array}) + n + C(\begin{array}{c} \bullet \bullet \bullet \\ \bullet \bullet \bullet \\ \bullet \bullet \bullet \end{array}, n, f(r))$$

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n$$

Starting with m ...

...take some
of m away...

... and add it
into the total.

$$C(\text{cluster of 12 colored dots}, n, r) \leq (k+1)(\text{cluster of 4 green dots}) + n$$

$$(k+1)(\text{cluster of 3 purple dots}) + n + C(\text{cluster of 5 colored dots}, n, f(f(r)))$$

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n$$

Starting with m ...

...take some
of m away...

... and add it
into the total.

$$C(\text{cluster of 12 colored dots}, n, r) \leq (k+1)(\text{cluster of 4 green dots}) + n$$

$$(k+1)(\text{cluster of 3 purple dots}) + n$$

$$(k+1)(\text{cluster of 2 blue dots}) + n + C(\text{cluster of 2 yellow dots}, n, f(f(f(r))))$$

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n$$

Starting with m ...

...take some of m away...

... and add it into the total.

$$C(\text{cluster of 12 colored dots}, n, r) \leq (k+1)(\text{cluster of 4 green dots}) + n$$

$$(k+1)(\text{3 purple dots}) + n$$

$$(k+1)(\text{2 light blue dots}) + n$$

$$(k+1)(\text{1 grey dot}) + n$$

...

$$\leq (k+1)(\text{cluster of 12 colored dots}) + \underline{\hspace{2cm}}$$

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n$$

Starting with m ...

...take some
of m away...

... and add it
into the total.

$$C(m, n, r) \leq (k+1) \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) + n$$

$$(k+1) \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) + n$$

$$(k+1) \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) + n$$

$$(k+1) \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) + n$$

...

$$\leq (k+1)m + \underline{\hspace{2cm}}$$

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n$$

How many layers
can this recursion
have?

$$C(m, n, r) \leq (k+1)(\text{⊕}) + n$$

$$(k+1)(\text{⊙}) + n$$

$$(k+1)(\text{⊗}) + n$$

$$(k+1)(\text{⊙}) + n$$

...

$$\leq (k+1)m + \underline{\hspace{2cm}}$$

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n$$

How many layers
can this recursion
have?

$$C(m, n, r) \leq (k+1)(\text{⊕}) + n$$

$$(k+1)(\text{⊙⊙}) + n$$

$$(k+1)(\text{⊙⊙⊙}) + n$$

$$(k+1)(\text{⊙}) + n$$

...

$$\leq (k+1)m + nf^*(r)$$

$$C(m, n, r) \leq C(m - m_+, n, f(r)) + (k+1)m_+ + n$$

How many layers
can this recursion
have?

$$C(m, n, r) \leq (k+1) \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) + n$$

$$(k+1) \left(\begin{array}{ccc} \bullet & \bullet & \bullet \end{array} \right) + n$$

$$(k+1) \left(\begin{array}{cc} \bullet & \bullet \\ \bullet & \end{array} \right) + n$$

$$(k+1) \left(\begin{array}{c} \bullet \end{array} \right) + n$$

...

$$\leq (k+1)m + nf^*(r)$$

Interpreting This Result

- We now have a family of bounds on the cost of operations on a disjoint-set forest:

$$m + n \cdot (r/2)$$

$$2m + n \lg r$$

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

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

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

...

- Which of these is the “best” bound?

Interpreting This Result

- For now, focus on these bounds:

$$2m + n \lg r$$

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

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

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

$$6m + n \lg^{****} r$$

- More generally, we have bounds of 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:

$$\alpha(z) = \min\{ k \in \mathbb{N} \mid \log^{*(k)} z \leq 1 \}$$

- Intuitively, this counts how many times you have to put stars on $\log^{***...***} z$ before it drops to 1.
- 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 have a bound of

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

- Picking $k = \alpha(r) = \alpha(\log n)$, and the bound on the cost of any series of m operations is.

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

- 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.

Next Time

- ***Euler Tour Trees***
 - Fully dynamic connectivity in forests.
- ***Augmented Dynamic Trees***
 - Figuring out information about connected components in sublinear time.