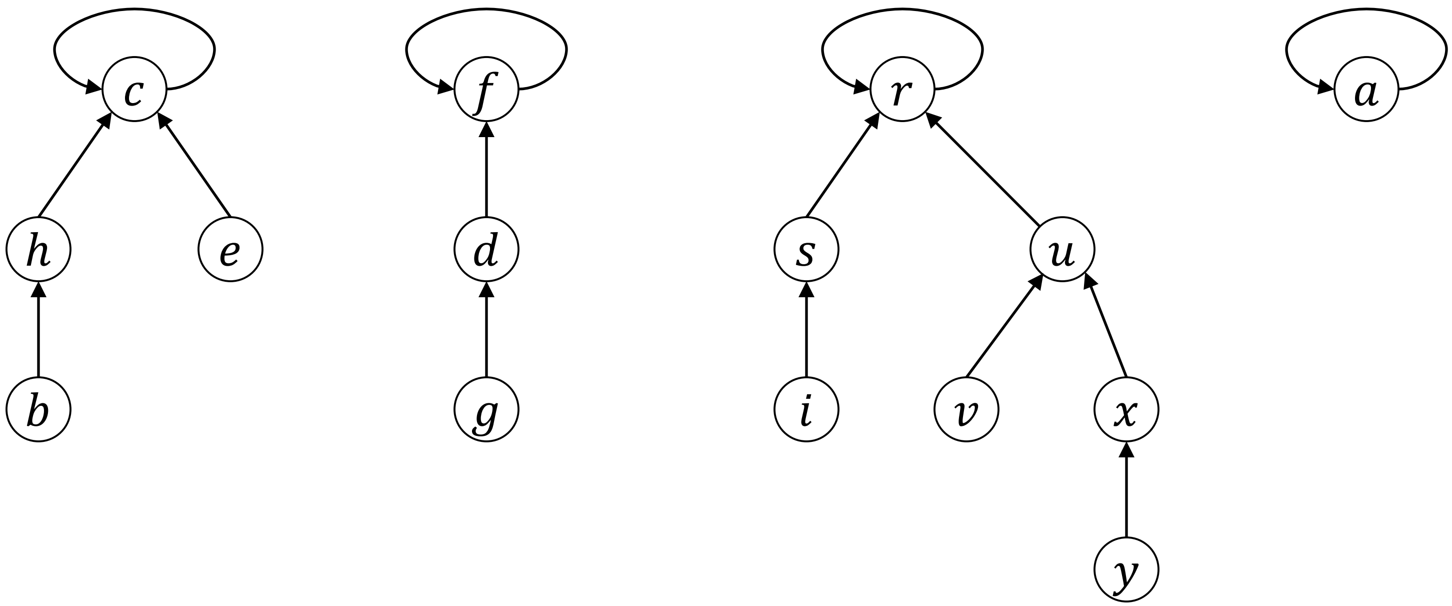
Algorithm Theory

Chapter 5
Data Structures

Part II:
Union Find: Disjoint-Set Forests

Fabian Kuhn

Disjoint-Set Forests

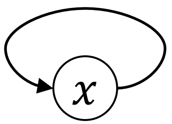


- Represent each set by a tree

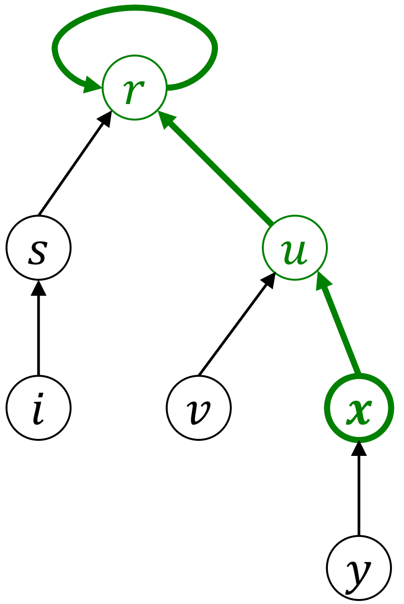
- Representative of a set is the root of the tree

Disjoint-Set Forests

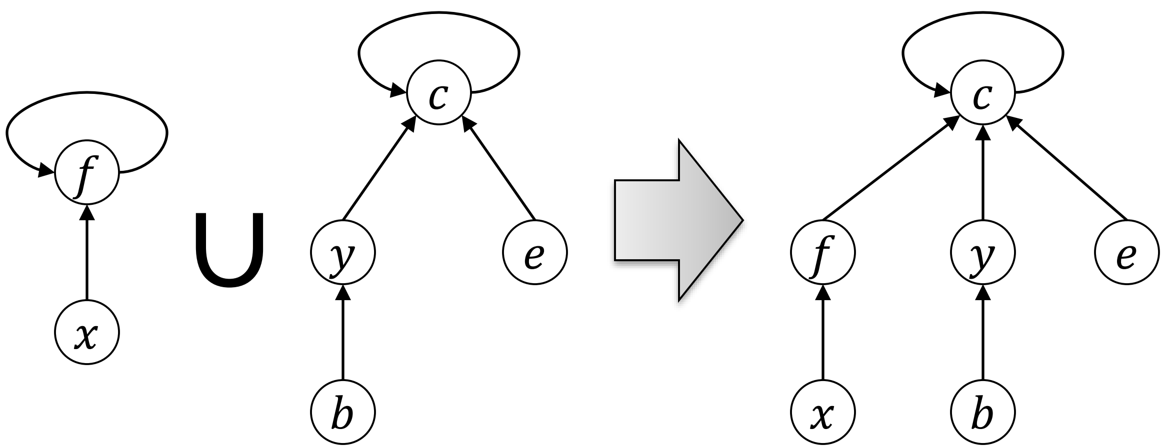
make_set(x): create new one-node tree



find(x): follow parent point to root
(parent pointer to itself)



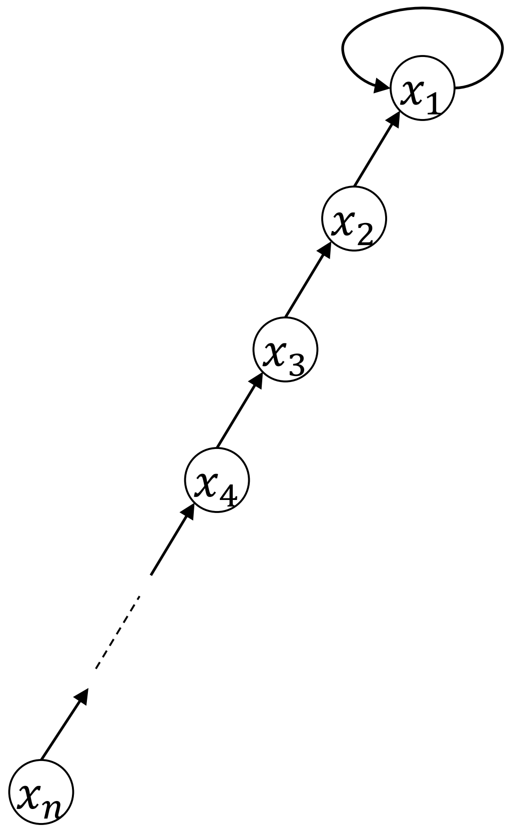
union(x, y): attach tree of x to tree of y



Bad Sequence

Bad sequence leads to tree(s) of depth $\Theta(n)$

- $\text{make\_set}(x_1), \text{make\_set}(x_2), \dots, \text{make\_set}(x_n),$
  $\text{union}(x_1, x_2), \text{union}(x_1, x_3), \dots, \text{union}(x_1, x_n)$



Union-By-Rank Heuristic

- We could use the same union-by-size idea as before and always attach the smaller tree to the larger tree.
- Instead, we use an alternative, slightly different idea

Union of sets S_1 and S_2:

- Each tree node v has a rank $r(v)$, initially, after make_set: $r(v) = 0$
- Union of two trees (for sets S_1 and S_2) with roots v_1 and v_2
- If $r(v_1) \neq r(v_2)$, attach root of smaller rank to root of larger rank
- Otherwise, attach v_2 as to v_1, increment rank $r(v_1)$ of root v_1

Remark: The rank $r(v)$ is the height of the subtree rooted at v

- Initially, v is a one-node tree of height 0 and $r(v) = 0$
- Union operation:
  - If $r(v_1) \neq r(v_2)$, height of all subtrees stays the same, ranks do not change.
  - If $r(v_1) = r(v_2)$, height of subtree of v_1 and the rank $r(v_1)$ grow by 1

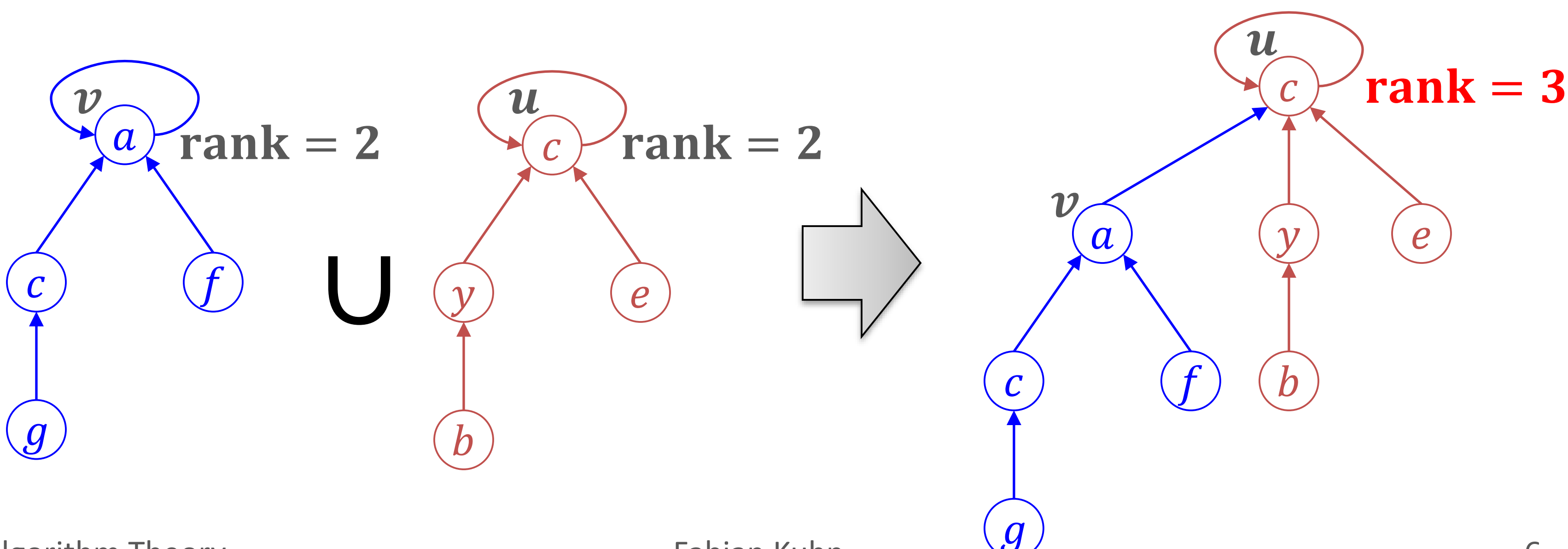
Union-By-Rank Heuristic

Lemma: The subtree rooted at a node v of rank $r(v)$ has at least $2^{r(v)}$ nodes.

Proof:

- A new node is in a tree of size 1 and has rank 0.

- Afterwards, the rank can only change in a union operation
  - If a node u gets v as a new child and $r(u) = r(v) = r$
  - Rank of root increases by 1, new tree size $\geq 2 \cdot 2^r = 2^{r+1}$



Union-By-Rank Heuristic

Lemma: Let $u = u_1, u_2, \dots, u_k = v$ be the path from a node u to the root v in the tree containing node u. We then have

$$r(u_1) < r(u_2) < \cdots < r(u_k).$$

Proof:

- We show that for any two nodes x and y such that y is the parent of x, we always have $r(x) < r(y)$.
- Node y becomes the parent of x when the tree of root x is attached to the tree of root y in a union operation.
  - Either, we then have $r(y) > r(x)$
  - Or, we have $r(y) = r(x)$. Then, the rank of y is increased by 1
  - Afterwards, only the rank of y can change (increase)

Corollary 1: The subtree of a node v has height at most $r(v)$.

Corollary 2: All trees have height $O(\log n)$.

Union-Find Algorithms

Recall: m operations, n of the operations are make_set-operations

Linked List with Union-By-Size:

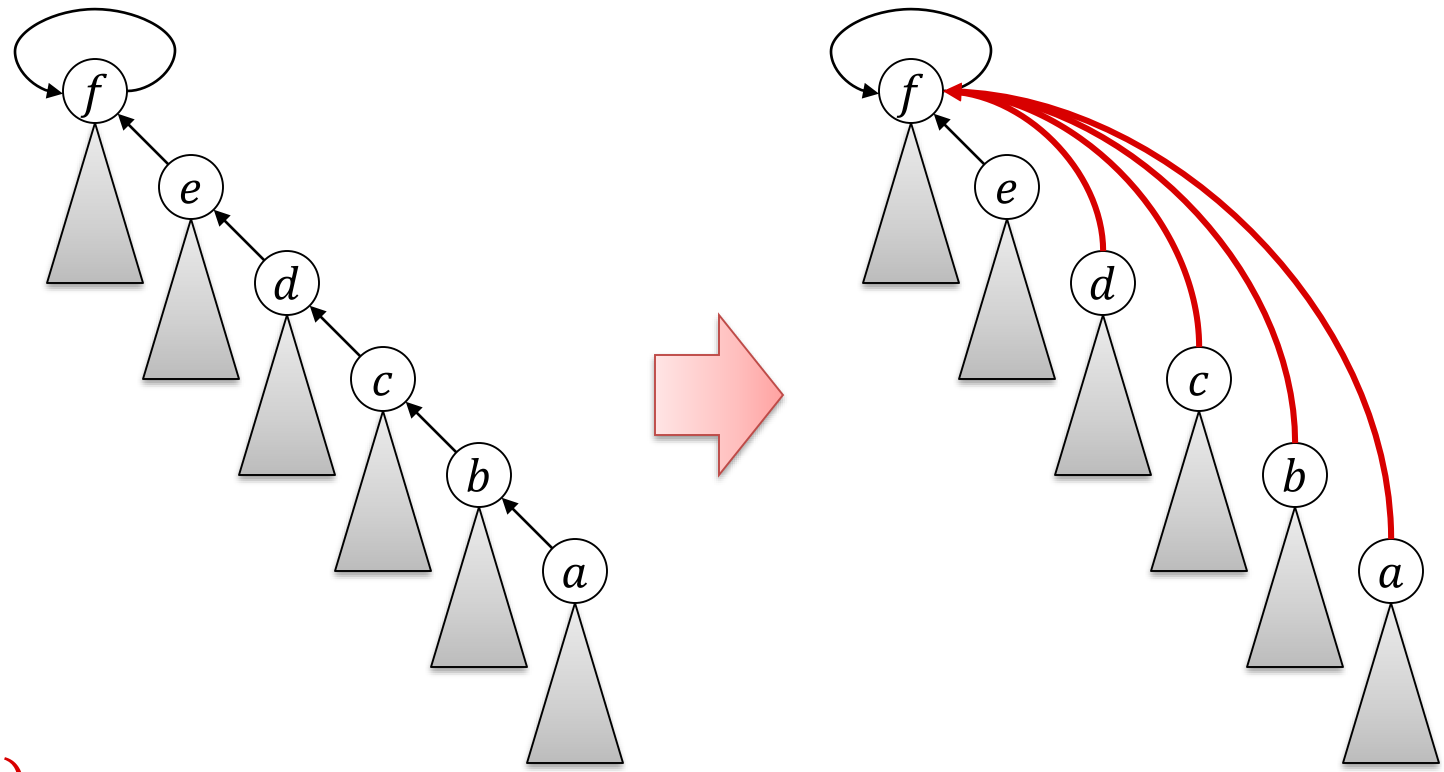
- make_set: worst-case cost $O(1)$
- find : worst-case cost $O(1)$
- union : amortized worst-case cost $O(\log n)$

Disjoint-Set Forest with Union-By-Rank (or Union-By-Size):

- make_set: worst-case cost $O(1)$
- find : worst-case cost $O(\log n)$
- union : worst-case cost $O(\log n)$

Can we make this faster?

Path Compression During Find Operation



find(a):

1. **if** $a \neq a.parent$ **then**
2. $\quad a.parent := \text{find}(a.parent)$
3. **return** $a.parent$

Union-By-Rank and Path Compression

Theorem: Using the combined union-by-rank and path compression heuristic, the running time of m union-find operations on n elements (at most n make_set-operations) is

$$\boldsymbol{\Theta}\big(\boldsymbol{m} \cdot \boldsymbol{\alpha}(\boldsymbol{n})\big),$$

where $\alpha(n)$ is the inverse of the Ackermann function.

- $\alpha(n)$: extremely slowly growing function.
- For all practical purposes $\alpha(n) \leq 4$
  - E.g., as long as n is less than the number of atoms in the universe…

We will show the following slightly weaker statement:

- The running time of m operations is at most $O(m \cdot \log^* n)$
- $\log^* n$ is a function that grows almost as slowly as $\alpha(n)$.

Union-By-Rank and Path Compression

The rank of a node v is still defined in the same way:

- $r(v)$ is initialized to 0
- every time a node u is attached as the child of v in a union operation, if $r(u) = r(v)$, $r(v)$ is increased by 1.
  - The rank is now just an upper bound on the height of a tree.

The two lemmas from before are still true.

Lemma: The subtree rooted at node v has $\geq 2^{r(v)}$ nodes.

- The same argument as before works.

Lemma: Let $u = u_1, u_2, \dots, u_k = v$ be the path from a node u to the root v in the tree containing node u. We then have

$$r(u_1) < r(u_2) < \cdots < r(u_k).$$

- Node y can become parent of x during a path compression.
- But then, y was an ancestor of x and we also have $r(y) > r(x)$.

Simple Observations

Path compression in union operations:

- We will assume that a union operation consists of two find operations and the constant-time operation for merging the trees.
  - We can then concentrate on the cost and the effect of find operations.

Rank:

- The rank of a node can only change as long as the node is the root of some tree. As soon as the node has a parent, the rank is fixed.
- The number of nodes of rank r is at most $n/2^r$
  - Each such node has a subtree of size $\geq 2^r$ with nodes of smaller rank.

Find operation:

- Consider a node u with parent v and assume that the edge $\{u, v\}$ is traversed in a find operation.
  - If v is not the root of the tree, u gets a new parent w with $r(w) > r(v)$
  - Node v will not be the parent of u in the future

Node Buckets

- We place all the non-root nodes in some bucket
  - As soon as a node is in a bucket, the rank of the node is fixed

Buckets:

B buckets

- rank 0
- rank 1
- ranks $2, 3$
- ranks $4, \dots, 15$
- $\vdots$
- ranks $R, \dots, 2^R - 1$
- ranks $2^R, \dots, 2^{2^R} - 1$

Each bucket contains all non-root nodes with ranks between r and $2^r - 1$ for some integer $r \geq 0$.

Maximum number of nodes in bucket:

- Nodes with rank $i \leq n/2^i$
- Nodes with rank in $[r, 2^r - 1]$:

$$\sum_{i=r}^{\infty} \frac{n}{2^i} = \frac{2n}{2^r}$$

Cost of Find Operations

Total cost of all find operations $= \boldsymbol{O}($#traversed edges$) =$

1. Number of edges to a root node
   - $\leq O(m)$ (1 per find operation)
2. Number of edges to a non-root node in a different bucket
   - $\leq O(m \cdot B)$ ($\leq B$ per find operation)
3. Number of edges to a non-root node in same bucket
   - Let's count these kind of edges for each node v in some bucket with ranks between r and $2^r - 1$
   - Node v sees every non-root parent w at most once, and when changing the parent, the new parent has a higher rank.
   - The number of such traversed edges for each node is therefore at most the number of ranks in the bucket $\Longrightarrow < 2^r$
   - Number of nodes in bucket is $\leq {}^{2n}/_{2^r}$
   - At most $2n$ such edges per bucket $\Longrightarrow \leq O(n \cdot B)$ cost overall

Overall find cost: $\boldsymbol{O(m \cdot B)}$

What is B?

The Number of Buckets

Definition:

$$\log^*(n) := \begin{cases} 0, & \text{if } n \leq 1 \\ 1 + \log^*(\log_2 n)\,, & \text{otherwise} \end{cases}$$

- $\log^* n$ is the number of logarithms we have to take to get a value at most 1.
- $\log^* n$ grows extremely slowly:
  - $\log^*(4) = 2, \log^*(16) = 3, \log^*(65536) = 4, \log^*(2^{65536}) = 5$

Claim: The number of buckets is $B = O(\log^* n)$.

- Buckets contain ranks
  $[r_0, 2^{r_0} - 1], [r_1, 2^{r_1} - 1], [r_2, 2^{r_2} - 1], \dots$, with $r_{i+1} = 2^{r_i}$
- Therefore:
  $r_0 = \log(r_1) = \log\log(r_2) = \log\log\log(r_3) = \log \dots \log(r_B)$

Union-By-Rank and Path Compression

Theorem: Using the combined union-by-rank and path compression heuristic, the running time of m union-find operations on n elements (at most n $\mathrm{make\_set}$-operations) is

$$\boldsymbol{\Theta(m \cdot \log^* n)}.$$

- Cost of make-set and union is $O(n)$
  - When ignoring the find operations that are contained in union operations.
- Cost of find operations is $O(m \cdot B) = O(m \cdot \log^* n)$
  - Including the find operations that are contained in union operations