Data Structures

Disjoint Sets 3

CS 225

October 20, 2023

Brad Solomon & G Carl Evans

UNIVERSITY OF
ILLINOIS
URBANA-CHAMPAIGN

Department of Computer Science

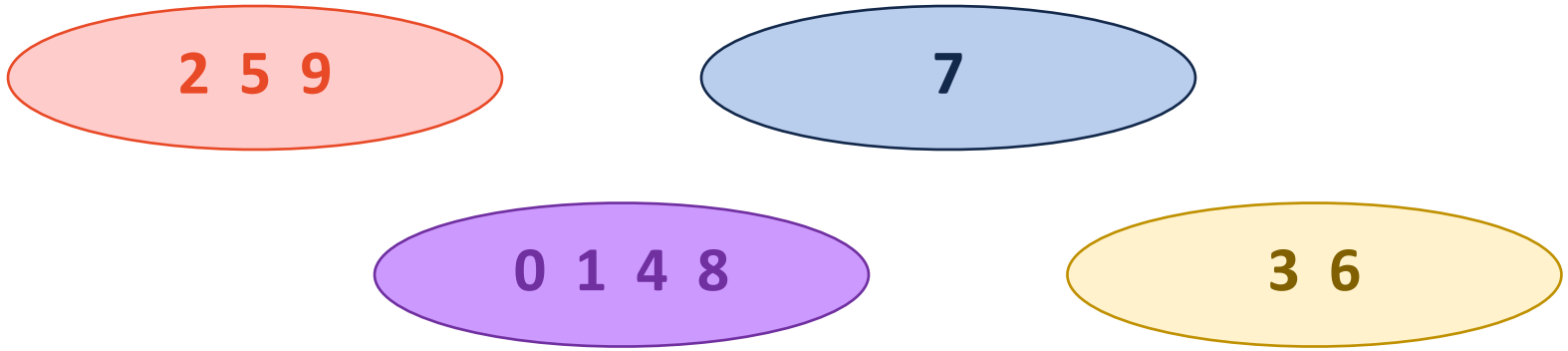
Learning Objectives

Discuss efficiency of disjoint sets

Introduce path compression and rank

Prove efficiency of disjoint sets (again)

Disjoint Sets



Key Ideas:

- Each element exists in exactly one set.
- Every item in each set has the same representation
- Each set has a different representation

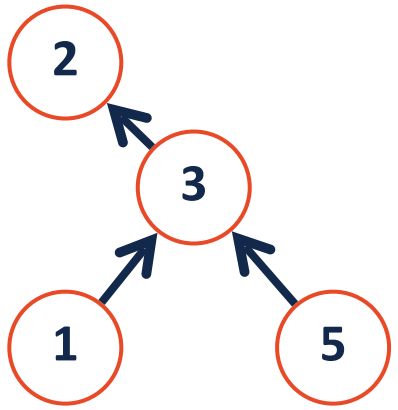
Disjoint Sets Representation

We can represent a disjoint set as an array where the key is the index

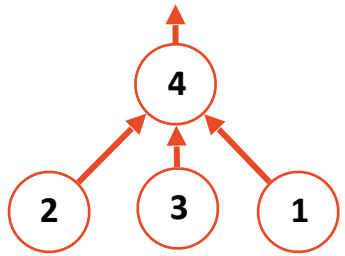
The values inside the array stores our sets as a pseudo-tree (UpTree)

Negative values denote representative elements (the root)

All other set members store the index to a parent of the UpTree



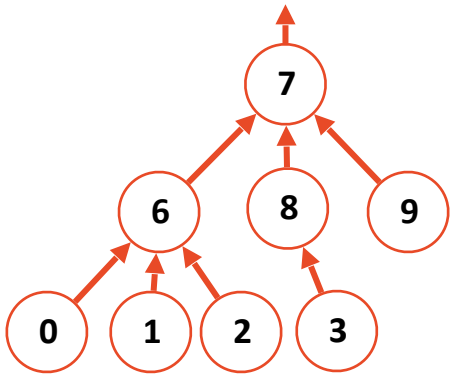
Disjoint Sets – Best and Worst UpTree



0	1	2	3	4
	3	4	2	-1

0	1	2	3	4
	4	4	4	-1

Disjoint Sets – Smart Union



Union by height

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8	−4	10	7	−3	7	7	4	5

***Idea**: Keep the height of the tree as small as possible.*

Union by size

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8	−4	10	7	−8	7	7	4	5

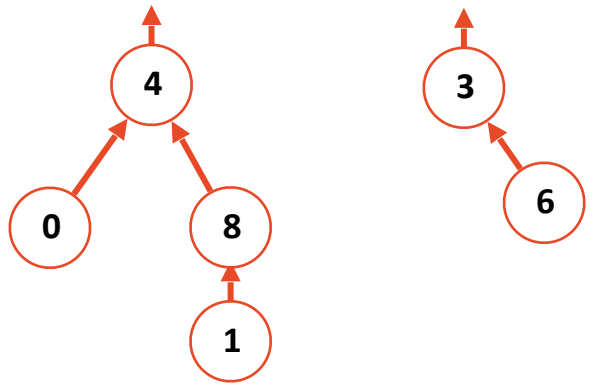
***Idea**: Minimize the number of nodes that increase in height*

Claim that both guarantee the height of the tree is: O(log n).

Disjoint Sets Union by Size

unionBySize(4, 3)

```
 1  void DisjointSets::unionBySize(int root1, int root2) {
 2    int newSize = arr_[root1] + arr_[root2];
 3
 4    if ( arr_[root1] < arr_[root2] ) {
 5
 6      arr_[root2] = root1;
 7
 8      arr_[root1] = newSize;
 9
10    } else {
11
12      arr_[root1] = root2;
13
14      arr_[root2] = newSize;
15
16    }
  }
```



0	1	2	3	4	5	6	7	8	9
4	8		-2	-4		3		4	

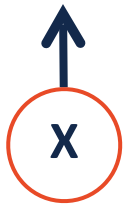
Disjoint Sets Union by Size

Claim: Sets unioned by size have a height of at most $O(\log_2 n)$

Claim: An UpTree of height **h** has nodes $\geq 2^h$

Base Case:

Base case height is 0, has one node.



vs.

$$2^0 = 1$$

Base case holds!

Disjoint Sets Union by Size

Claim: An UpTree of height **h** has nodes $\geq 2^h$

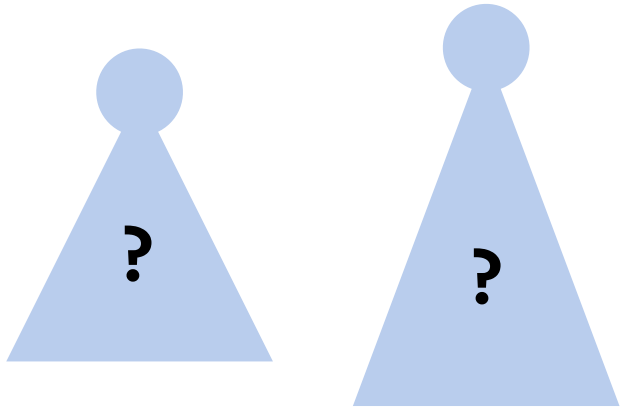
IH: Claim is true for $< i$ unions, prove for ith union.

(We have done $i - 1$ total unions and plan to do ***one*** more)

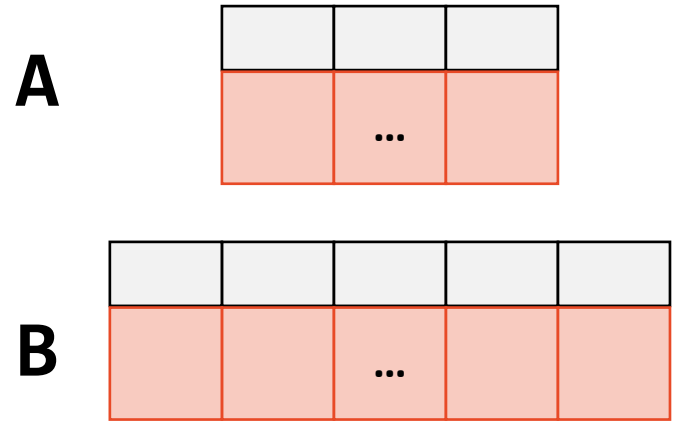
Case 1: h(A) < h(B)

Case 2: h(A) == h(B)

Case 3: h(A) > h(B)



n(B) ≥ n(A)



Disjoint Sets Union by Size

$n(B) \geq n(A)$

Claim: An UpTree of height **h** has nodes $\geq 2^h$

IH: Claim is true for $< i$ unions, prove for ith union.

Case 1: height(A) < height(B)

Disjoint Sets Union by Size

$n(B) \geq n(A)$

Claim: An UpTree of height **h** has nodes $\geq 2^h$

IH: Claim is true for $< i$ unions, prove for ith union.

Case 2: height(A) == height(B)

Disjoint Sets Union by Size

$n(B) \geq n(A)$

Claim: An UpTree of height **h** has nodes $\geq 2^h$

IH: Claim is true for $< i$ unions, prove for ith union.

Case 3: height(A) > height(B)

Disjoint Sets Union by Size

$n(B) \geq n(A)$

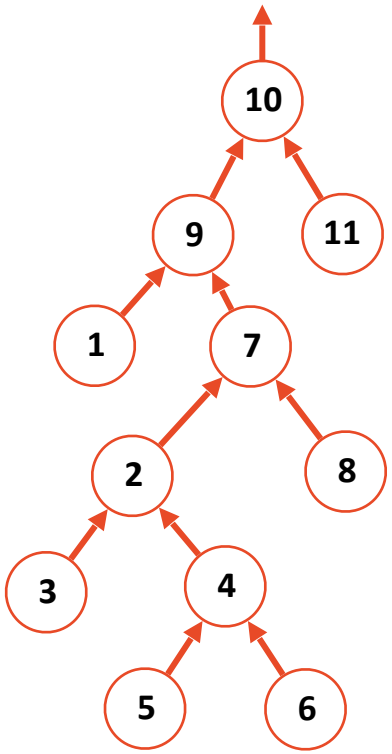
Proven: An UpTree of height **h** has nodes $\geq 2^h$

IH: Claim is true for $< i$ unions, prove for ith union.

Each case we saw we have $n \geq 2^h$.

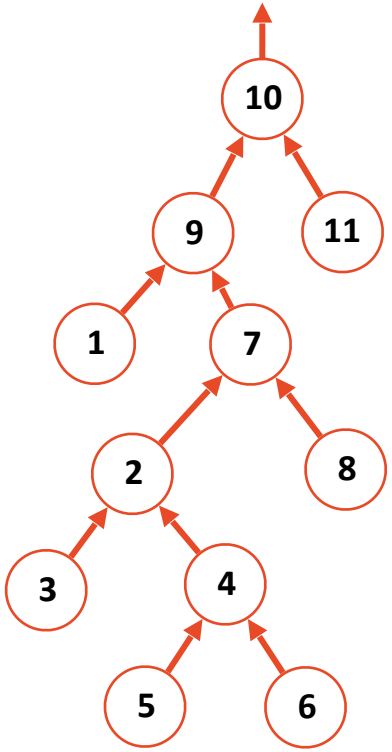
Path Compression

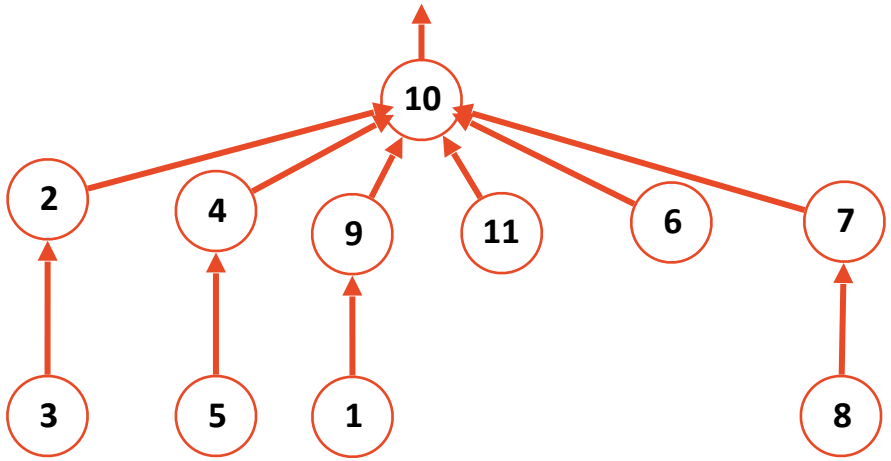
`Find(6)`



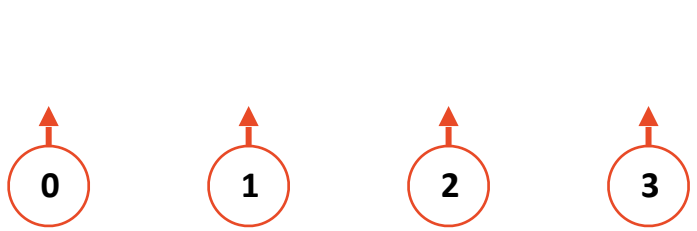
Path Compression

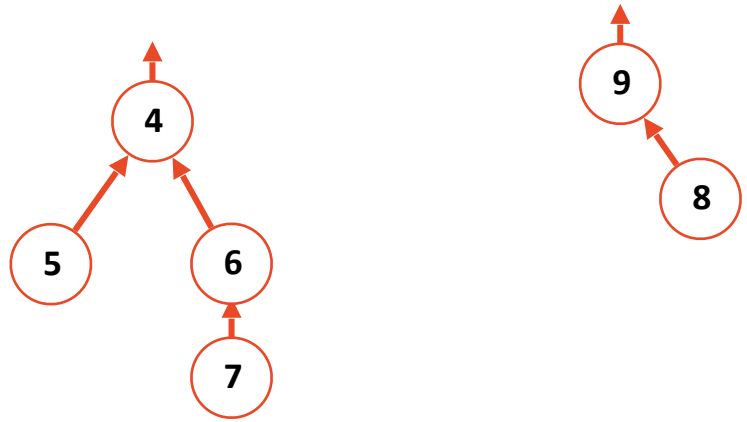
Find(6)





Disjoint Sets – Union by Rank (not height!)





0	1	2	3	4	5	6	7	8	9

Union by Rank (Not Height)

The change: New UpTrees have rank = 0

Let A, B be two sets being unioned. If:

rank(A) == rank(B): The merged UpTree has rank + 1

rank(A) > rank(B): The merged UpTree has rank(A)

rank(B) > rank(A): The merged UpTree has rank(B)

This is identical to height (with a different starting base)!

Union by Rank

Claim: An UpTree of rank **r** has nodes $\geq 2^r$.

Base Case:

Inductive Step: IH holds for all UpTrees up to $k < r$

Try solving yourself before seeing answer (next slide)!

Union by Rank - Proof

Much like before we will show that in a tree with a root of rank r there are $nodes(r) \geq 2^r$

Base Case: UpTree of rank = 0 has 1 node $2^0 = 1$

Inductive Hypothesis: for all trees of ranks $k,\ k < r,\ \ nodes(k) \geq 2^k$

A root of rank r is created by merging two trees of rank $r - 1$

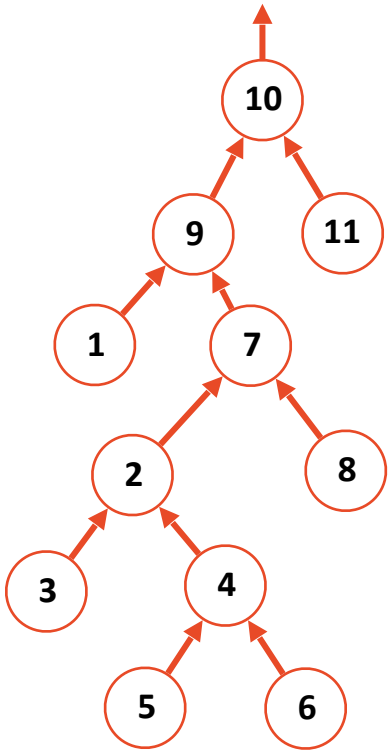
by IH each of those trees have $nodes(r-1) \geq 2^{r-1}$

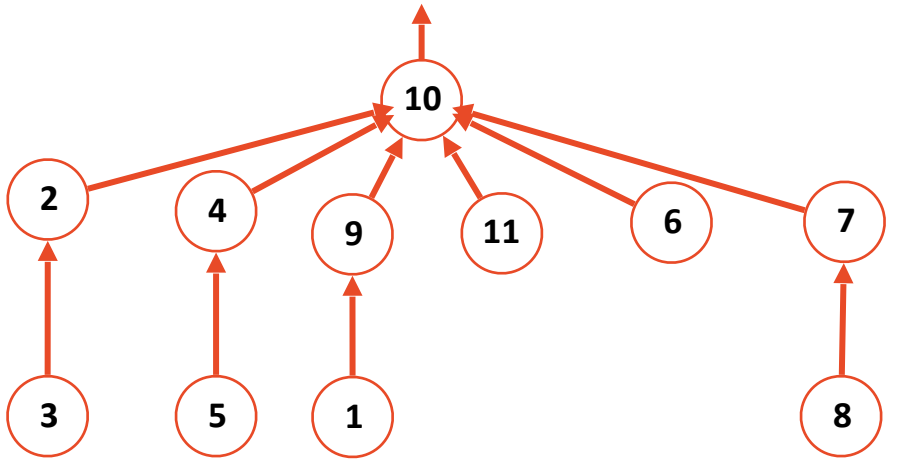
so, tree a of rank r has $nodes(r) \geq 2 \times 2^{r-1} \geq 2^r$

Taking the inverse, we get a height of $O(\log(n))$

Union by Rank w/ Path Compression

How does rank w/ path compression affect our runtime?





Union by Rank w/ Path Compression

1. Rank only changes for roots and can only increase (unlike height!)

2. For all non-root nodes x, **rank(x) < rank(parent(x))**

3. If parent(x) changes, then our new parent has larger rank.

Union by Rank w/ Path Compression

4. min(nodes) in a set with a root of rank r has $\geq 2^r$ nodes.

5. Since there are only n nodes the highest possible rank is $\lfloor log\ n \rfloor$.

Union by Rank w/ Path Compression

6. For any integer r, there are at most $\frac{n}{2^r}$ nodes of rank r.

Amortized Time (Rank w/ Path Compression)

For **n** calls to makeSets() [**n items**] and **m** find() calls the max work is…

This gives us a more accurate picture since each find can make our search a faster!

Two cases of find():

1. We search for root [or a node whose parent is root]

2. We search for a node where neither above apply.

Amortized Time (Rank w/ Path Compression)

Put every non-root node in a bucket by rank!

Structure buckets to store ranks $[r, 2^r - 1]$

Ranks	Bucket
0	0
1	1
2 - 3	2
4 - 15	3
16 – 65535	4
65536 – 2^{65536}-1	5

Iterated Logarithm Function (log^*n)

log^*n is piecewise defined as

$$0 \text{ if } n \leq 1$$

otherwise

$$1 + log^*(\log n)$$

Amortized Time (Rank w/ Path Compression)

Let $|B_r|$ be the size of the bucket with min rank r.

What is $max(|B_r|)$?

Ranks	Bucket
0	0
1	1
2 - 3	2
4 - 15	3
16 – 65535	4
65536 – 2^{65536}-1	5

Amortized Time (Rank w/ Path Compression)

The work of **find(x)** is the steps taken on the path from a node x to the root (or immediate child of the root) of the UpTree containing x

We can split this into two cases:

Case 1: We take a step from one bucket to another bucket.

Case 2: We take a step from one item to another inside the same bucket.

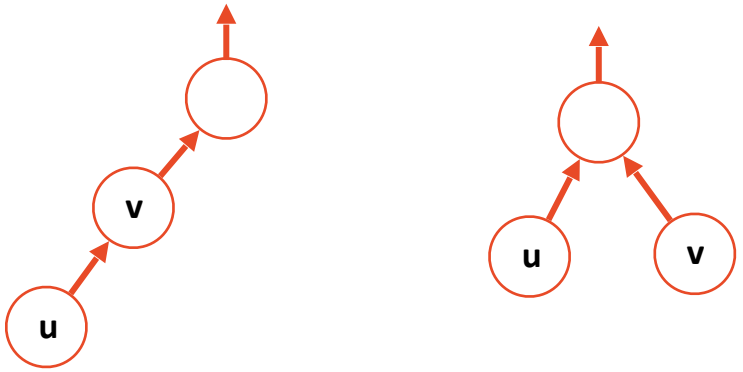
Amortized Time (Rank w/ Path Compression)

Case 2: We take a step from one item to another *inside* the same bucket.

Let's call this the step from **u** to **v**.

Every time we do this, we do path compression:

We set parent(u) a little closer to root



How many total times can I do this for each **u** in $|B_r|$?

How many nodes are in $|B_r|$?

Final Result

For **n** calls to makeSets() [**n items**] and **m** find() calls the max work is:

Even Better

In case that still seems too slow tightest bound is actually

$$\Theta(m\ \alpha(m, n))$$

Where $\alpha(m, n)$ is the inverse Ackermann function which grows much slower than log*n.

Proof well outside this class.

Randomized Algorithms

A **randomized algorithm** is one which uses a source of randomness somewhere in its implementation.

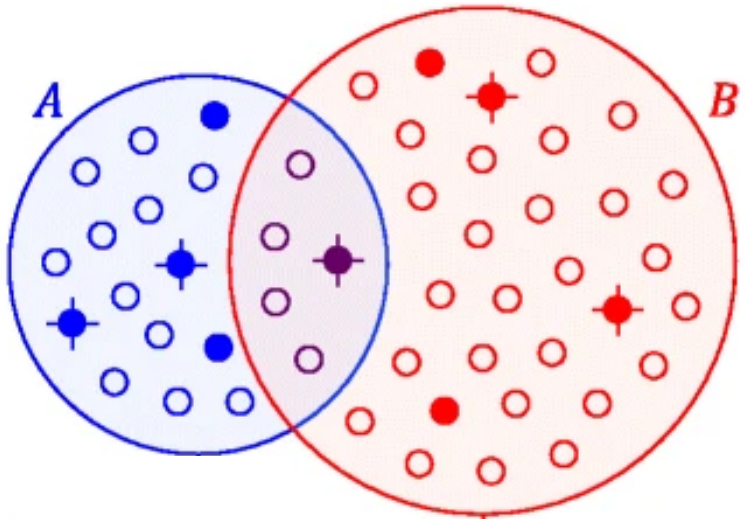
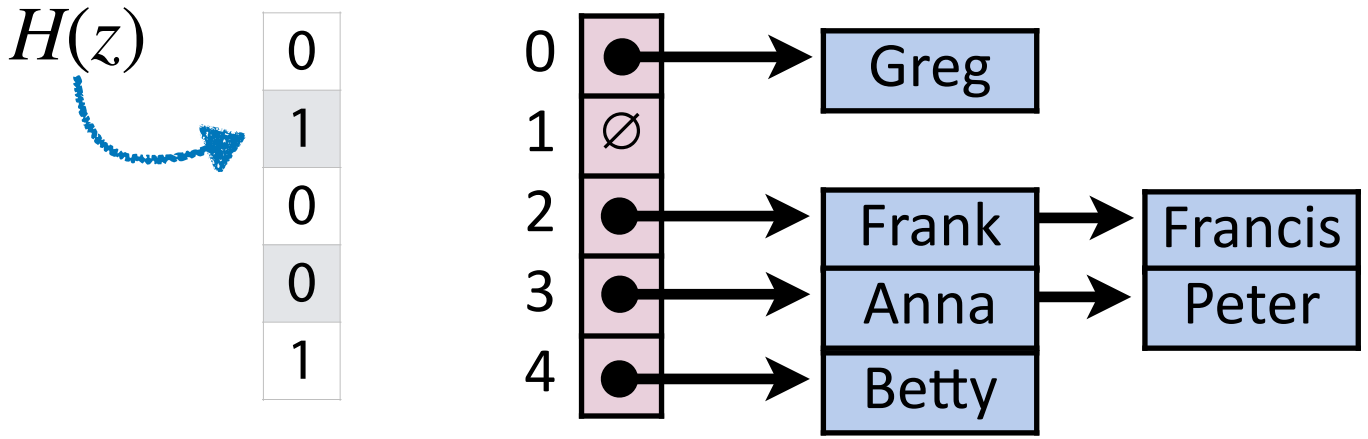


Figure from Ondov et al 2016



$H(x)$	0	2	1	0	0	4	0	2	0	6
$H(y)$	1	0	2	3	1	0	3	4	0	1
$H(z)$	2	1	0	2	0	1	0	0	7	2