CS 225

Data Structures

March 12 – Disjoint Sets

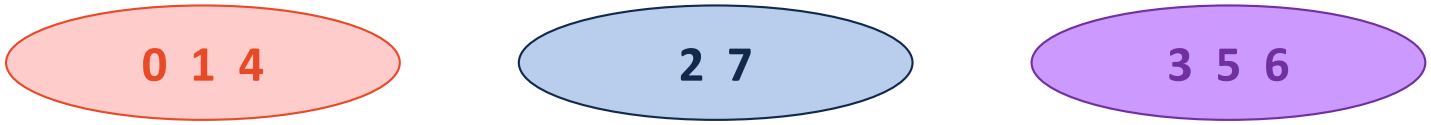
G Carl Evans

Disjoint Sets ADT

- Maintain a collection S = $\{s_0, s_1, \ldots s_k\}$
- Each set has a representative member.
- API:

```
void makeSets(int number);
void union(int k1, const int k2);
int  find(int k);
```

Implementation #1



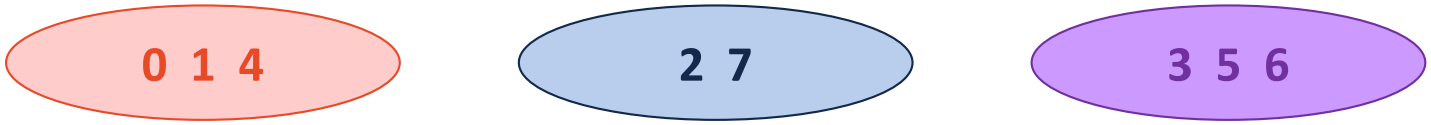
0	1	2	3	4	5	6	7

Find(k):

Union(k1, k2):


YOU EXPECTED A NEW DATA STRUCTURE
BUT IT WAS ME, TREE ALL ALONG
imgflip.com

Implementation #2



0	1	2	3	4	5	6	7

Find(k):

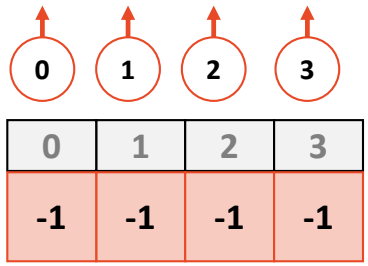
Union(k1, k2):

Implementation #2

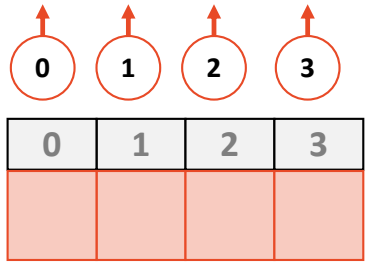
- We will continue to use an array where the index is the key

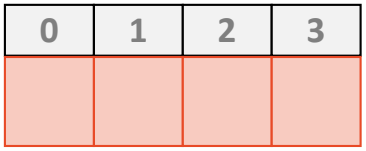
- The value of the array is:
  - **-1,** if we have found the representative element
  - **The index of the parent**, if we haven't found the rep. element

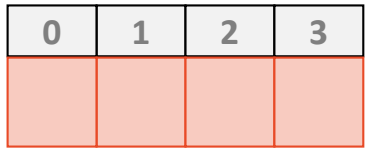
- We will call theses **UpTrees**:

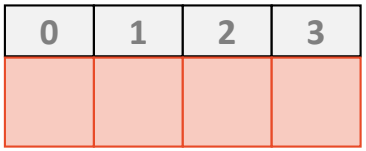


UpTrees

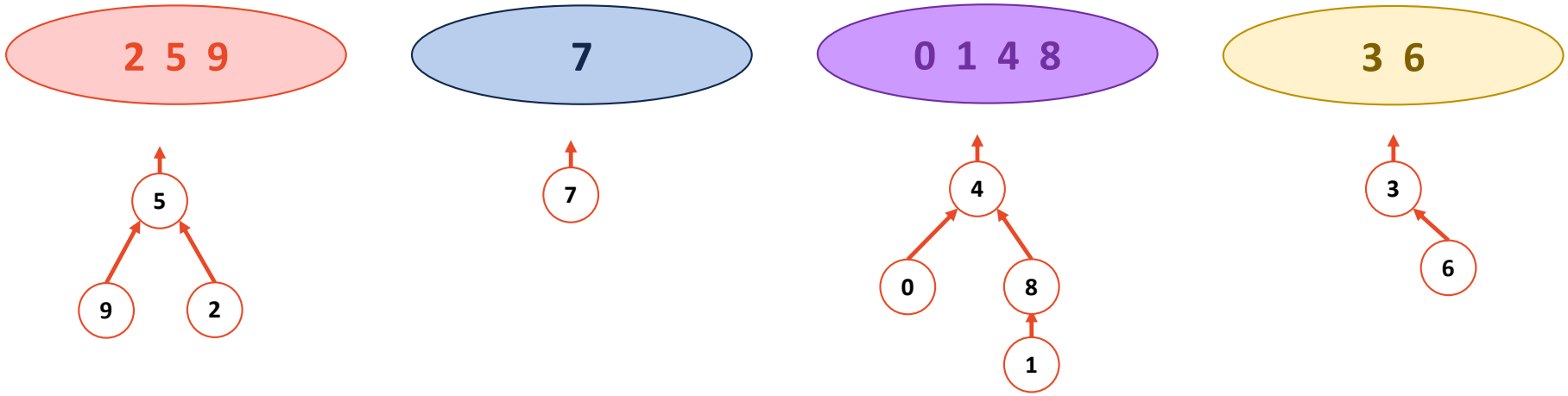








Disjoint Sets



0	1	2	3	4	5	6	7	8	9

Find(k):

Union(k1, k2):

Disjoint Sets Find

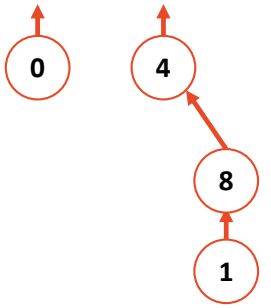
```
1  int DisjointSets::find(int i) {
2    if ( s[i] < 0 ) { return i; }
3    else { return find( s[i] ); }
4  }
```

Running time?

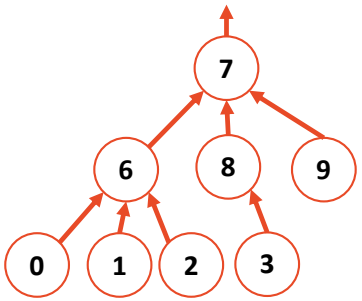
What is the ideal UpTree?

Disjoint Sets Union

```
1  void DisjointSets::union(int r1, int r2) {
2
3
4  }
```

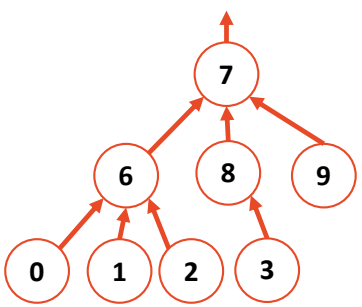


Disjoint Sets – Union



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

Disjoint Sets – Smart Union

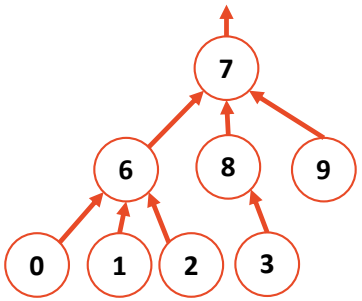


Union by height

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

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

Disjoint Sets – Smart Union

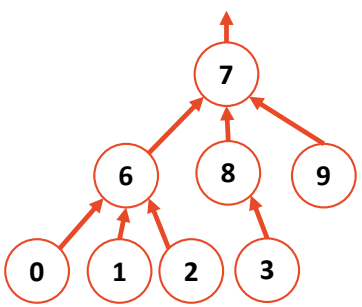


Union by size

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

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

Disjoint Sets – Smart Union



Union by height

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

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

Both guarantee the height of the tree is: _____.

Disjoint Sets Find

```
1  int DisjointSets::find(int i) {
2    if ( s[i] < 0 ) { return i; }
3    else { return find( s[i] ); }
4  }
```

```
1  void DisjointSets::unionBySize(int root1, int root2) {
2    int newSize = arr_[root1] + arr_[root2];
3
4    // If arr_[root1] is less than (more negative), it is the larger set;
5    // we union the smaller set, root2, with root1.
6    if ( arr_[root1] < arr_[root2] ) {
7      arr_[root2] = root1;
8      arr_[root1] = newSize;
9    }
10
11   // Otherwise, do the opposite:
12   else {
13     arr_[root1] = root2;
14     arr_[root2] = newSize;
15   }
16 }
```

Union by Size (limit on height)

To show that every tree in a disjoint set data structure using union by size has a height of at most O(log n) we will show that the inverse.

Base Case

Inductive Hypothesis

Union by Size

Case 1

Union by Size

Case 2

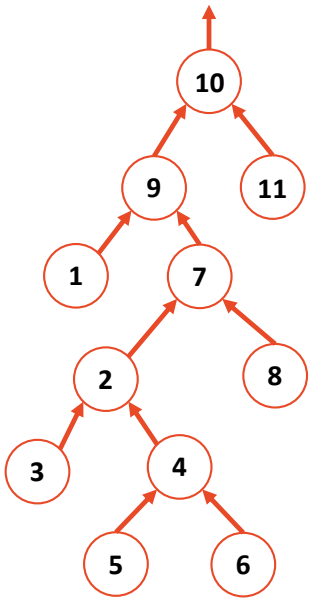
Union by Height (limit on height)

Much like before we will show the min(nodes) in a tree with a root of height $k \geq 2^k$

Base Case

IH

Path Compression



Disjoint Sets Analysis

The **iterated log** function:

The number of times you can take a log of a number.

$$\log^*(n) = \begin{cases} 0 & , n \leq 1 \\ 1 + \log^*(\log(n)) & , n > 1 \end{cases}$$

What is $\mathbf{lg^*(2^{65536})}$?

Disjoint Sets Analysis

In a Disjoint Sets implemented with smart **unions** and path compression on **find**:

Any sequence of **m union** and **find** operations result in the worse case running time of O(_____),

where **n** is the number of items in the Disjoint Sets.