CS 225

# #32: Disjoint Sets Finale + Graphs Intro

November 6, 2019 · *G Carl Evans*

## Implementation – DisjointSets::union

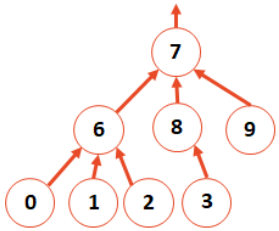
DisjointSets.cpp (partial)

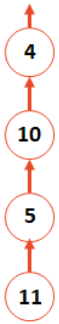
```
1  void DisjointSets::union(int r1, int r2) {
2
3
4  }
```

How do we want to union the two UpTrees?

---

## Building a Smart Union Function




The implementation of this visual model is the following:

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

What are possible strategies to employ when building a "smart union"?

**Smart Union Strategy #1:** _____

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

**Metadata at Root:**

After `union( 4, 7 )`:

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

---

**Smart Union Strategy #2:** _____

**Idea:** Minimize the number of nodes that increase in height.
*(Observe that the tree we union have all their nodes gain in height.)*

**Metadata at Root:**

After `union( 4, 7 )`:

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

---

## Smart Union Implementation:

DisjointSets.cpp (partial)

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

## How do we improve this?

**DisjointSets.cpp (partial)**

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

**DisjointSets.cpp (partial)**

```
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
5    // larger set; we union the smaller set, root2, with root1.
6    if ( arr_[root1] < arr_[root2] ) {
7      arr_[root2] = root1;
8      arr_[root1] = newSize;
9    }
10   // Otherwise, do the opposite:
11   else {
12     arr_[root1] = root2;
13     arr_[root2] = newSize;
14   }
15 }
```

**Running Time:**

- Worst case running time of find(k)**:**
- Worst case running time of union(r1, r2), given roots:
- New function: "Iterated Log":

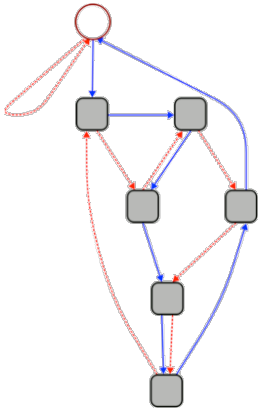
  **log*(n)** :=

- Overall running time:
  - A total of **m** union/find operation runs in:

## A Review of Major Data Structures so Far

| Array-based | List/Pointer-based |
|---|---|
| - Sorted Array | - Singly Linked List |
| - Unsorted Array | - Doubly Linked List |
| - Stacks | - Skip Lists |
| - Queues | - Trees |
| - Hashing | - BTree |
| - Heaps | - Binary Tree |
| - Priority Queues | - Huffman Encoding |
| - UpTrees | - kd-Tree |
| - Disjoint Sets | - AVL Tree |

## An Introduction to Graphs



**Motivation:**
Graphs are awesome data structures that allow us to represent an enormous range of problems. To study these problems, we need:

1. A common vocabulary to talk about graphs
2. Implementation(s) of a graph
3. Traversals on graphs
4. Algorithms on graphs

**CS 225 – Things To Be Doing:**

1. Theory Exam 3 starts Thursday; **Practice Exam Available!**
2. MP5 due tonight at 11:59pm
3. Lab Section: lab_puzzles coming up this week in lab!
4. Daily POTDs are ongoing!