Announcements

Programming assignment #3 ⇒ Review (passed back today)

- Test cases ⇒ Empty, 3 million random, 3 million decending, 3 million assending
- Test cases (extra credit) ⇒ Empty, 3 million duplicate random, 3 million duplicate decending, 3 million duplicate decending
- Compare r-p > threshold not A.size() > threshold
- Pass by reference, default in C++ is to pass by value
- In Java be sure to use stringbuilder
- # includes should be ordered as the standard libary includes first, then your own includes
- Do not use #define, malloc() free(), printf(), c-headers, and other c stuff in cpp
- Make sure to include the required comments
- Do not use C++ 11 specific features if not needed
- Overhead for med of 3 should have been a low value. The benefit for sorted cases outweighs any overhead for unsorted cases

Midterm ⇒ Next class

- 2 hour exam (3 hour class)
- ✗ No disjoint sets, counting sort, radix sort (stand alone), bucket sort (stand alone), Φ value
- ✓ Hash tables ⇒ Only about the version we are coding (chaining, multiplication)
- ✓ Quicksort ⇒ Lumuto or hoare
- ✓ Radix bucket hybrid ⇒ The one from the assignment
- ✓ PQ/heapsort/heap ⇒ More likely to be about a min heap
- ✓ Linkedlist /stack /queue
- ✓ Big oh
- 7 of the questions are "write this function"

Disjoint Sets

Overview ⇒ Track equivalence classes. Aka union-find

- Group n items into between n and 1 set(s) based on what you decide what equivallency means
- Self healing towards best time complexity
- Operation #1 ⇒ Given an item, which set is it a member of (find)
- Operation #2 ⇒ Union two sets (yields one set) (union)

Implementation

- ✗ Linkedlist of elements in same set (slow, O(n) find)
- ✓ Disjoint set forest (Expected case O(1), worst case O(n) but self healing towards O(1))

Time budget => O(1)

- Our operations will need to make decisions quicky with little information to stay within the O(1) budget
- Heuristic #1 => Union by rank
- Heuristic #2 => Path compression

Root element => Representative element of a set

- An element of the set is its name

Make set

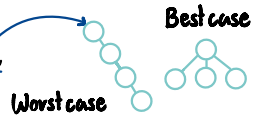
- Consumed by the constructor
- Pseudo code

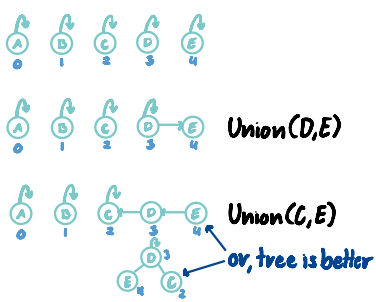
```
Constructor(int size){
    p=new int[size]; //parent
    r=new int[size];//rank
    for(int i=0;i<size;i++){
        p[i]=i;   //parent is itself
        r[i]=0;   //all initally rank zero
    }
}
```

Union => Union by rank

1. Determine representative element
2. Combine by rank to try to avoid worst case

Rank is an estimated upper bound





- Pseudo code

```
union(x,y){
    //bounds check x and y, if x==y return;
    link(find(x), find(y));
}

link(x,y){//O(1)
    //Tuck smaller rank under larger rank, no rank change
    //If same rank, then one rank increases by one (one on top)
    if(r[x]>r[y]){
    p[y]=x;
    }else{  //also includes where r[x]==r[y]
        p[x]=y;
        if(r[x]==r[y]){
```

```
            r[y]++;
        }
    }
}
```

Find ⇒ Path compression

- Within a find call, flatten the structure
- Find what we are looking for and fix the parents at the same time
- Does not change the rank
- Pseudo code

```
find(x){
    if(p[x]==x){
        return x;
    }
    p[x]=find(p[x]);
    return p[x];
}
```

```
//Shorter version
find(){
    if(x!=p[x]){
        p[x]=find(p[x]);
    return p[x]
}
```

Extra functions

- same_component(x,y){return find(x)==find(y)
- get_num_sets(), get_num_elements...

Assignment#6

- Generate mazes of nxn size (min 3x3)
- Pick walls (adjacent cells) randomly, only remove walls if cells not connected
- Equivalency class ⇒ Mutually reachable cell
- Each square is represented by a hex number of how the walls are arranged. Try to do with bitwise
- To join two cells (remove a wall) use union()
- To randomly select cells, make a list and shuffle it