CS 61B Fall 2020 — Disjoint Sets and Asymptotics

Discussion 6: September 28, 2020

1 Disjoint Sets, a.k.a. Union Find

In lecture, we discussed the Disjoint Sets ADT. Some authors call this the Union Find ADT. Today, we will use union find terminology so that you have seen both.

(a) What are the last two improvements (out of four) that we made to our naive implementation of the Union Find ADT during lecture 14 (Monday's lecture)?

1. Improvement 1: ______________________________

2. Improvement 2: ______________________________

The naive implementation was maintaining a record of every single connection. Improvements made were:

```
- Keeping track of sets rather than connections (QuickFind)
- Tracking set membership by recording parent not set # (QuickUnion)
- Union by Size (WeightedQuickUnion)
- Path Compression (WeightedQuickUnionWithPathCompression)
```

We will focus on attention on the last two, union by size and path compression.

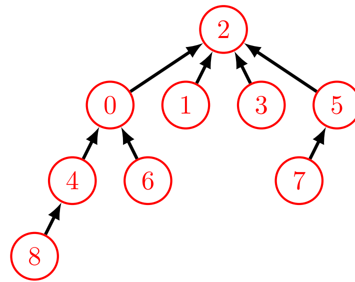
(b) Assume we have nine items, represented by integers 0 through 8. All items are initially unconnected to each other. Draw the union find tree, draw its array representation after the series of `connect()` and `find()` operations, and write down the result of `find()` operations using **WeightedQuickUnion**. Break ties by choosing the smaller integer to be the root.

Note: `find(x)` returns the root of the tree for item `x`.

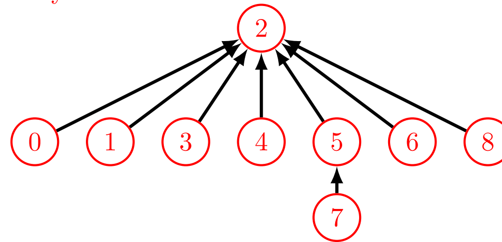
```
connect(2, 3);
connect(1, 2);
connect(5, 7);
connect(8, 4);
connect(7, 2);
find(3);
connect(0, 6);
connect(6, 4);
connect(6, 3);
find(8);
find(6);
```

`find()` returns 2, 2, 2 respectively.

The array is [2, 2, -9, 2, 0, 2, 0, 5, 4].



(c) Repeat the above part, using **WeightedQuickUnion with Path Compression**. `find()` returns 2, 2, 2 respectively.
The array is [2, 2, -9, 2, 2, 2, 2, 5, 2].



2 Asymptotics

(a) Order the following big-O runtimes from smallest to largest.

$$O(\log n), O(1), O(n^n), O(n^3), O(n \log n), O(n), O(n!), O(2^n), O(n^2 \log n)$$

$O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2 \log n) \subset O(n^3) \subset O(2^n) \subset O(n!) \subset O(n^n)$

(b) Are the statements in the right column true or false? If false, correct the asymptotic notation ($\Omega(\cdot)$, $\Theta(\cdot)$, $O(\cdot)$). Be sure to give the tightest bound. $\Omega(\cdot)$ is the opposite of $O(\cdot)$, i.e. $f(n) \in \Omega(g(n)) \iff g(n) \in O(f(n))$.

$f(n) = 20501$	$g(n) = 1$	$f(n) \in O(g(n))$
$f(n) = n^2 + n$	$g(n) = 0.000001n^3$	$f(n) \in \Omega(g(n))$
$f(n) = 2^{2n} + 1000$	$g(n) = 4^n + n^{100}$	$f(n) \in O(g(n))$
$f(n) = \log(n^{100})$	$g(n) = n \log n$	$f(n) \in \Theta(g(n))$
$f(n) = n \log n + 3^n + n$	$g(n) = n^2 + n + \log n$	$f(n) \in \Omega(g(n))$
$f(n) = n \log n + n^2$	$g(n) = \log n + n^2$	$f(n) \in \Theta(g(n))$
$f(n) = n \log n$	$g(n) = (\log n)^2$	$f(n) \in O(g(n))$

- True, although $\Theta(\cdot)$ is a better bound.
- False, $O(\cdot)$. Even though n^3 is strictly worse than n^2, n^2 is still in $O(n^3)$ because n^2 is always as good as or better than n^3 and can never be worse.
- True, although $\Theta(\cdot)$ is a better bound.
- False, $O(\cdot)$.
- True.
- True.
- False, $\Omega(\cdot)$.

(c) Give the worst case and best case runtime in terms of M and N. Assume `ping` is in $\Theta(1)$ and returns an **`int`**.

```
1  int j = 0;
2  for (int i = N; i > 0; i--) {
3      for (; j <= M; j++) {
4          if (ping(i, j) > 64) break;
5      }
6  }
```

Worst: $\Theta(M + N)$, Best: $\Theta(N)$ The trick is that `j` is initialized outside the loops!

(d) Assume `mrpoolsort(array)` is in $\Theta(N \log N)$ and returns `array` sorted.

```
1  public static boolean mystery(int[] array) {
2      array = mrpoolsort(array);
3      int N = array.length;
```

```
4       for (int i = 0; i < N; i += 1) {
5           boolean x = false;
6           for (int j = 0; j < N; j += 1) {
7               if (i != j && array[i] == array[j]) x = true;
8           }
9           if (!x) return false;
10      }
11      return true;
12  }
```

1. Give the worst case and best case runtime where N = `array.length`. What is `mystery()` doing?

   `mystery()` returns true if every `int` has a duplicate in the array (ex. `{1, 2, 1, 2}`) and false if there is any unique `int` in the array (ex. `{1, 2, 2}`). Its runtime is $\Theta(NlogN + N^2) = \Theta(N^2)$ for the worst case the if statement always sets x to true. The best case is if we we don't set x to be true in the very first loop, which allows us to only go through the entire array once giving us $\Theta(NlogN + N) = \Theta(NlogN)$.

2. Now that you know what `mystery()` is doing, try to come up with a way to implement `mystery()` that runs in $\Theta(NlogN)$ time. Can we get any faster?

   ```
   public static boolean mystery(int[] array) {
       array = mrpoolsort(array);
       int N = array.length;
       int curr = array[0];
       boolean unique = true;

       for (int i = 1; i < N; i += 1) {
           if (curr == array[i]) {
               unique = false;
           } else if (unique) {
               return false;
           } else {
               unique = true;
               curr = array[i];
           }
       }
       return true;
   }
   ```

   There is a possible $\Theta(N)$ solution, but that involves data structures we haven't covered yet!