CS 61B Spring 2021

Disjoint Sets and Asymptotics

Discussion 6: February 22, 2021

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)?

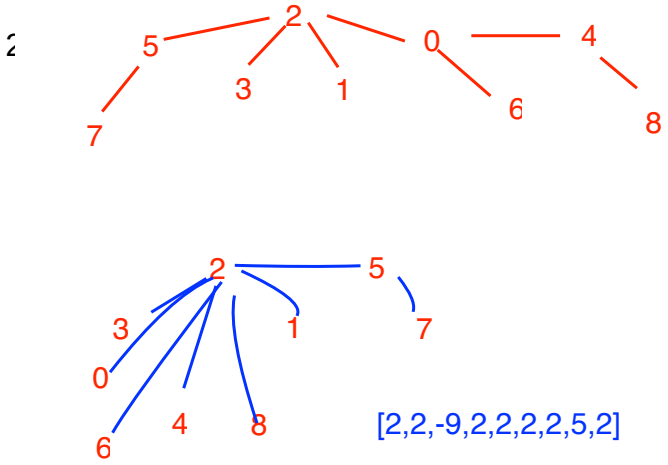
1. Improvement 1: Weighted Quick Union - merge on size, merge with smaller size as easier
2. Improvement 2: Path Compression - find root on first call, then set all connects to the root

(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** without path compression. 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);  2
connect(0, 6);
connect(6, 4);
connect(6, 3);
find(8);  2
find(6);  2
```

[2, 2,-9,2, 0, 2, 0, 5, 4]



[2,2,-9,2,2,2,2,5,2]

(c) *Extra:* Repeat the above part, using **WeightedQuickUnion with Path Compression**.

Part B in Red, Part C in Blue

(d) What is the runtime for "connect" and "isConnected" operations using our Quick Find, Quick Union, and Weighted Quick Union ADTs? Can you explain why the Weighted Quick union has better runtimes for these operations than the regular Quick Union?

	Quick Find	Quick Union	Weighted Quick Union
Connect:	Theta (n)	Theta(n)	Theta(log n)
isConnected:	Theta(1)	Theta (n)	Theta(log n)

Weighted Quick Union has better runtimes because height of tree is at most log n, while Quick Union has height at most of n

2 Asymptotics

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

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

0(1), O(log n), O(n), O(n log n), O(n^2), O(n^2 log n), O(n^3), O(2^n), O(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. $f(n) \in \Omega(g(n)) \iff g(n) \in O(f(n))$.
Hint: Make sure to simplify the runtimes first.

i) $f(n) = 20501$	$g(n) = 1$	$f(n) \in O(g(n))$
ii) $f(n) = n^2 + n$	$g(n) = 0.000001n^3$	$f(n) \in \Omega(g(n))$
iii) $f(n) = 2^{2n} + 1000$	$g(n) = 4^n + n^{100}$	$f(n) \in O(g(n))$
iv) $f(n) = \log(n^{100})$ 100 log n	$g(n) = n \log n$	$f(n) \in \Theta(g(n))$
v) $f(n) = n \log n + 3^n + n$	$g(n) = n^2 + n + \log n$	$f(n) \in \Omega(g(n))$
vi) $f(n) = n \log n + n^2$	$g(n) = \log n + n^2$	$f(n) \in \Theta(g(n))$
vii) $f(n) = n \log n$	$g(n) = (\log n)^2$ log n log n	$f(n) \in O(g(n))$

i) True, this an accurate tightest bound. T False, an accurate tightest bound would be _____
ii) True, this an accurate tightest bound. False, an accurate tightest bound would be n^2
iii) True, this an accurate tightest bound. T False, an accurate tightest bound would be _____
iv) True, this an accurate tightest bound. False, an accurate tightest bound would be log n
v) True, this an accurate tightest bound. T False, an accurate tightest bound would be _____
vi) True, this an accurate tightest bound. T False, an accurate tightest bound would be _____
vii) True, this an accurate tightest bound. False, an accurate tightest bound would be n log n

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

Worst Case: This is a rectangle of N * M. So runtime is N * M
Best Case: every call to j breaks the inner for loop ie Theta(1). So runtime is N

Upper Bound: O(N*M)
Lower Bound: Omega(N)

(d) Below we have a function that returns true if every int has a duplicate in the array, and false if there is any unique int in the array. Assume `sort(array)` is in $\Theta(N \log N)$ and returns `array` sorted.

```
1  public static boolean noUniques(int[] array) {
2      array = sort(array);
3      int N = array.length;
4      for (int i = 0; i < N; i += 1) {
5          boolean hasDuplicate = false;
6          for (int j = 0; j < N; j += 1) {
7              if (i != j && array[i] == array[j]) {
8                  hasDuplicate = true;
9              }
10         }
11         if (!hasDuplicate) return false;
12     }
13     return true;
14 }
```

1. Give the worst case and best case runtime where $N =$ `array.length`.

   Worst Case:
   Say we have an array with length 5. sort(array) is N Log N
   if we have no dups, loops run 25 times for each i and j This is Theta(N^2)

   Best Case:
   say item 0 and item 1 are duplicates. theres 6 operations (inner loop goes 5 times, then 1 extra operation.
   This is N + 1, or runtime N. But we have sort(array) = N * Log N, so best case runtime is Theta(N Log N)

2. Try to come up with a way to implement `noUniques()` that runs in $\Theta(NlogN)$ time. Can we get any faster?

```
public static boolean noUniques(int[] array) {
    array = sort(array);
    int N = array.length
    int curr = array[0]
    boolean unique = true;
    for (int i = 1; i < N; i += 1) {
        if (curr[0] = array[i]) {
              unique = false;
        }
        else if (unique) {
            return false;
         }
        else {
            unique = true;
            curr = array[i]
    }
}
```