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: _____
2. Improvement 2: _____

(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);
connect(0, 6);
connect(6, 4);
connect(6, 3);
find(8);
find(6);
```

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

(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?

2 Asymptotics

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

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

T(N)∈O(f(N) 当输入规模N足够大时，T(N)的增长速度不会超过f(N)的一个常数倍
T(N)∈Θ(f(N) 存在k1, k2使 k1*f(N) <= T(N) <= k2*f(N) 对任意N>=N0成立
T(N)∈Ω(f(N) 算法运行时间至少为f(N)的增长速度

(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) $f(n) = n^2 + n$	$g(n) = 0.000001n^3$	$f(n) \in \Omega(g(n))$ ⟺ $g(n) \in O(f(n))$
iii) $f(n) = 2^{2n} + 1000$ $2^n \cdot 2^n$	$g(n) = 4^n + n^{100}$ $(2^2)^n = 2^{2n}$	$f(n) \in O(g(n))$
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) $f(n) = n \log n$	$g(n) = (\log n)^2$	$f(n) \in O(g(n))$

i) True, this an accurate tightest bound.	False, an accurate tightest bound would be ____
ii) True, this an accurate tightest bound.	False, an accurate tightest bound would be ____
iii) True, this an accurate tightest bound.	False, an accurate tightest bound would be ____
iv) True, this an accurate tightest bound.	False, an accurate tightest bound would be ____
v) True, this an accurate tightest bound.	False, an accurate tightest bound would be ____
vi) True, this an accurate tightest bound.	False, an accurate tightest bound would be ____
vii) True, this an accurate tightest bound.	False, an accurate tightest bound would be ____

(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 : $\Theta(M \cdot N)$

best : $\Theta(N)$

(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 }
```

O(NlogN)

N^2

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

   worst: $\Theta(N^2)$ best: $\Theta(N \log N)$

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

```
for (int i = 0, j = 0; i < n; i++)
    if ( j >= n) { break; }
    if (a[i] != a[j]) {
        j ++;
    } else {
        return false;
    }
}
```