CS 61B Fall 2024

Asymptotics, Disjoint Sets

Discussion 05: September 30, 2024

1 Asymptotics

(a) Say we have a function `findMax` that iterates through an unsorted `int` array one time and returns the maximum element found in that array. Give the tightest lower and upper bounds ($\Omega(\cdot)$ and $O(\cdot)$) of `findMax` in terms of `N`, the length of the array. Is it possible to define a $\Theta(\cdot)$ bound for `findMax`?

(b) 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  }
```

(c) 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 }
```

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

2 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) 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);
```

Below is an implementation of the `find` function for a Disjoint Set. Given an integer `val`, `find(val)` returns the root value of the set `val` is in. The helper method `parent(int val)` returns the direct parent of `val` in the Disjoint Set representation. Assume that this implementation only uses **QuickUnion**.

```
1  public int find(int val) {
2      int p = parent(val);
3      if (p == -1) {
4          return val;
5      } else {
6          int root = find(p);
7          return root;
8      }
9  }
```

(b) If `N` is the number of nodes in the set, what is the runtime of `find` in the worst case? Draw out the structure of the Disjoint Set representation for this worst case.

(c) Using a function `setParent(int val, int newParent)`, which updates the value of `val`'s parent to `newParent`, modify `find` to achieve a faster runtime using path compression. You may add at most one line to the provided implementation.

(d) *Extra Practice:* Draw out the tree and array representation for the following WeightedQuickUnion with path compression that has 9 elements from 0 to 8. Break ties by choosing the smaller integer to be root.

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