CS 381 HW 2

Ankur Dhoot

September 11, 2016

1 Q1

(a) Claim: A set can have exactly 0 or 2 MegaGuys.
Proof: It's trivial that a set can have 0 MegaGuys (suppose everyone knows each other). It's also clear that a set can have exactly two MegaGuys. By the definition of a MegaGuy, it's impossible for a set to contain only one MegaGuy (a MegaGuy knows exactly one other person, who must be a MegaGuy as well). Thus, what's left to prove is that a set cannot have more than two MegaGuys. We'll proceed by contradiction. Suppose a set S contains $n > 2$ MegaGuys. Let the MegaGuys be denoted by $G = \{s_1, s_2, ..., s_n\}$. By definition, all $s \in S$ know s_1 and s_1 knows exactly one other person who is a MegaGuy, say $s_m \in G$. But then any s_i $i \neq m$ cannot be a MegaGuy, since s_i is not known by s_1, a contradiction. Thus, there can be no more than two MegaGuys in S.

(b) In the celebrity problem, we're able to eliminate one person if given two. However, in the MegaGuy problem, it's possible that the two people we select could both be MegaGuys. Thus, we'll need to select three people in order to eliminate one (since at most two can be MegaGuys).

Let x,y,z denote the three people selected from S. We can then eliminate x based on either of two conditions:
1. If x knows y and x knows z (If x was a MegaGuy, x knows only one other person)
2. If y doesn't know x or z doesn't know x (A MegaGuy is known by all)
The same respective checks can also be used to eliminate y or z.

It's guaranteed that we'll eliminate one of x,y,z. Why? It's clear that x can know neither y nor z, exactly one of y or z, or both y and z. In the first case, x knows neither, so we can eliminate either of y or z since they aren't known by all. In the second case, x knows exactly one of y or z, so we can eliminate the guy x doesn't know. In the last case, x knows both y and z, so we can eliminate x since a MegaGuy only knows one other person. Thus,

we eliminate a guy who can't be a MegaGuy.
The guy eliminated is also not a celebrity. Why? If eliminated, that guy either knew two other guys, in which case he can't be a celebrity. Otherwise, he wasn't known by both of the other guys, in which case he can't be a celebrity.
Thus, the guy eliminated is neither a MegaGuy nor a celebrity.

The code which returns the element to be eliminated:

```
eliminate(x,y,z,Know) {

    //checks to eliminate x
    if Know[x,y] && Know[x,z]
        return x
    if !Know[y,x] || !Know[z,x]
        return x

    //checks to eliminate y
    if Know[y,x] && Know[y,z]
        return y
    if !Know[x,y] || !Know[x,z]
        return y

    //checks to eliminate z
    if Know[z,x] && Know[z,y]
        return z
    if !Know[x,z] || !Know[y,z]
        return z
}
```

The code above does at most 12 array accesses and runs in O(1) time.

(c) The code for finding MegaGuys or celebrities:

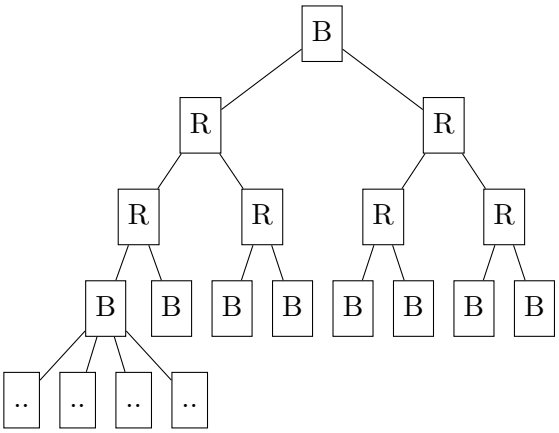
```
MegaGuysOrCelebrity(S, Know) {
    Let L be a list
    Add all elements of S to L

    //remove elements that are neither a celebrity nor a MegaGuy
    while (L.size >= 3) {
        x = L.remove()
        y = L.remove()
        Z = L.remove()
        toBeEliminated = eliminate(x,y,z,Know)
```

```
        //add back the two elements that weren't eliminated
        if (toBeEliminated == x) {
            L.insert(y)
            L.insert(z)
        } elif (toBeEliminated == y) {
            L.insert(x)
            L.insert(z)
        } else {
            L.insert(x)
            L.insert(y)
        }
    }

    //2 elements left in L
    s = L.remove()
    t = L.remove()

    if isMegaGuyPair(s,t)
        return (s,t)

    if isCelebrity(s)
        return s

    if isCelebrity(t)
        return t
}
```

Basic Idea: Use eliminate() to whittle down S until we're left with two possible elements. All elements eliminated are neither MegaGuys nor Celebrities. Then, we need to check whether the remaining two elements are a MegaGuy pair, or if either element is a celebrity.
To check if the two elements, (s,t) are a MegaGuy pair:
1. Check that s and t know each other, [2 array accesses]
2. Check that s doesn't know any other elements, but all other elements know s, [(N-2) + (N-2) = 2(N-2) array accesses]
3. Check that t doesn't know any other elements, but all other elements know t, [(N-2) + (N-2) = 2(N-2) array accesses]
Thus, we can check if a pair (s,t) is a MegaGuy Pair in O(N). We also know (given in class) that we can check whether a single element is a celebrity in O(N) time. Thus, the checks for MegaGuys and celebrities take O(N) time total. The list, L, takes O(N) time to be created and initialized. The while loop runs in O(N) since each time through, we eliminate one element from L and the call to eliminate takes O(1) time. Thus, the total algorithm runs in O(N) time.

Note that an element can't be a MegaGuy and a celebrity, nor can there be two celebrities. Thus, if a MegaGuy Pair or celebrity exists, we're guaranteed to return it with the last three checks in the algorithm.

Q2



(a) It's clear that M(0) = 0 since this corresponds to the root being a leaf. For $k > 0$, we can see from the tree above that the most number of nodes a tree can have is bounded by 1 + 2 + 4 + 8M(k-1). (1 for the root + 2 if both children are red + 4 if all 4 grandchildren are red + 8 black great children with black height k-1)
Thus M(k) = 7 + 8M(k-1) for $k > 0$
M(0) = 0

(b) Proof: By induction on k
Base Case: k = 0, We defined the base case as $M(0) = 0 = 2^{3*0} - 1$. Thus, the base case holds.

Induction Step:

Induction Hypothesis: Suppose that the statement holds for all positive $j < k$. That is $M(j) = 2^{3k} - 1$

Now we must show the statement holds for k. That is, we must show $M(k) = 2^{3k} - 1$.

M(k) = 7 + 8M(k-1)
$= 7 + 8 * (2^{3(k-1)} - 1)$ (by induction hypothesis)
$= -1 + 2^3 2^{3(k-1)}$
$= -1 + 2^{3k}$
$= 2^{3k} - 1$

Thus, having established the basis and induction step, the claim follows by the principle of mathematical induction.