CPSC 320: Intermediate Algorithm Design and Analysis
Assignment #2, due Thursday, January 26th, 2012 at 11:00

[10] 1. Suppose that you have $n \geq 3$ coins that are numbered from 1 to n, but are identical in appearance otherwise; either all are genuine or exactly one of them is fake. The fake coin does not have the same weight as the genuine ones, but it is unknown whether it is lighter or heavier. You also have a balance scale with which you can compare any two sets of coins. That is, by tipping to the left, to the right, or staying even, the balance scale will tell you whether the sets weigh the same or which of the sets is heavier than the other, but not by how much. You wish to find whether all the coins are genuine and, if not, to find the fake coin and establish whether it is lighter or heavier than the genuine ones.

Prove that no matter how clever you are, you will need to make at least $\lceil \log_3(2n+1) \rceil$ weighings in the worst case. Hints: use decision trees, as your proof needs to consider all possible algorithms for this problem; how many possible "answers" are there?

Solution: First, let us consider the number of possible answers an algorithm might return. There are n coins that might be fake, and each of them could be either lighter or heavier than the genuine coins. This gives us $2n$ possible answers. It is also possible that all coins are genuine, which is a $2n + 1^{\text{st}}$ answer.

The remainder of the proof is almost identical to the $\Omega(n \log n)$ lower bound on the number of comparisons between array elements for a comparison sort. Consider any algorithm $\mathcal{A}lg$ that solves this problem. For each n, there is a decision tree T_n that corresponds to algorithm $\mathcal{A}lg$ running on an array of size n. Unlike the proof we did in class, T_n is a ternary tree (not binary) since each weighing has three possible outcomes: tipping to the left, tipping to the right, or staying even. Because $\mathcal{A}lg$ needs to be able to produce $2n + 1$ distinct solutions, T_n has at least $2n + 1$ leaves.

Let h_n be the height of T_n. Since every ternary tree with height h_n has at most 3^{h_n} leaves, we have $3^{h_n} \geq 2n + 1$, and hence $h_n \geq \log_3(2n + 1)$. Since h_n is an integer, this means that $h_n \geq \lceil \log_3(2n + 1) \rceil$. Therefore $\mathcal{A}lg$ needs to perform at least $\lceil \log_3(2n + 1) \rceil$ weighings in the worst case.

[10] 2. Prove (using the definitions of O, Ω, etc.) or disprove (by giving a counterexample) each of the following statements about two functions $f, g : \mathbf{N} \to \mathbf{R}^+$:

a. If $f \in \Omega(g)$, then $f^2 \in \Omega(g^2)$, where f^2 is defined by $f^2(n) = (f(n))^2$ and g^2 is defined similarly.

Solution: This is true. If $f \in \Omega(g)$, then there are constants $c \in \mathbf{R}^+$ and $n_0 \in \mathbf{N}$ such that for every $n \geq n_0$, $f(n) \geq cg(n)$. Hence, for $n \geq n_0$,

$$f^2(n) \geq (cg(n))^2 \geq c^2 g(n)$$

and so f^2 and g^2 satisfy the definition of Ω using c^2 and n_0 as the two constants.

b. If $f \in O(g)$, then $2^f \in O(2^g)$, where 2^f is defined by $2^f(n) = 2^{f(n)}$ and 2^g is defined similarly.

Solution: This is false. Consider $f(n) = 2n$ and $g(n) = n$. Clearly $f \in O(g)$, using $c = 1/2$ and $n_0 = 1$. However

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^{2n}}{2^n} = \lim_{n\to\infty} \frac{4^n}{2^n} = \lim_{n\to\infty} 2^n = +\infty$$

and therefore $2^f \notin O(2^g)$.

[10] 3. You are facing a wall that stretches infinitely in both directions. There is a door in the wall, but you do not know either how far away nor in which direction it is. You can see the door only when you are right next to it.

a. Design an algorithm that enables you to reach the door by walking at most $O(n)$ steps where n is the (unknown to you) number of steps between your initial position and the door.

Solution: The trick is to increase the lengths of your walks according to a geometric series. For instance, the following will work:

- On the first "iteration", you walk 1m towards the left, then 2m towards the right, then 1m towards the left. During this iteration, you walked 4m and covered every position within 1m of your starting point.
- Next walk 2m towards the left, then 4m towards the right, then 2m towards the left. During this iteration, you walked 8m and covered every position within 2m of your starting point.
- Then walk 4m towards the left, then 8m towards the right, then 4m towards the left. During this iteration, you walked 16m and covered every position within 4m of your starting point.

In general, in the i^{th} iteration, you walk 2^{i-1}m towards the left, then 2^im towards the right, then 2^{i-1}m towards the left. You thus walk 2^{i+1}m and cover every position within 2^{i-1}m of your starting point.

b. Prove that by following your algorithm, you will walk $O(n)$ steps.

Solution: Suppose we find the door during the i^{th} iteration of the algorithm. This means that $n \leq 2^{i-1}$. Because the door was not found in the $(i-1)^{\text{st}}$ iteration, we also know that $n > 2^{i-2}$. The total distance walked is

$$\sum_{j=1}^{i} 2^{j+1} = 4\sum_{j=0}^{i-i} 2^j = 4(2^i - 1)$$

meters. Now,

$$4(2^i - 1) \leq 4 \cdot 2^i = 16 \cdot 2^{i-2} < 16n.$$

Therefore the distance walked is in $O(n)$.