MASSACHUSETTS INSTITUTE OF TECHNOLOGY

15.082J Fall 2010
HW 4 Due Session 15

Hand in the exercises labeled 4,5,6,7.
1. 7.9

2. 7.13

3. 7.15

4. $O()$ notation

Consider a strictly positive function $f(n)$ defined on the positive integers. Then formally, $O\big(f(n)\big)$ refers to a particular set of functions, namely those that (in a sense) are 'smaller' than $f(n)$. More precisely, a function $g(n)$ belongs to the set of functions $O\big(f(n)\big)$ if and only if there exists some fixed constant C, independent of n, s.t. $g(n) \leq Cf(n)$ for all integers $n \geq 1$. We note that this constant C can be different for different g.

a. Suppose $g(n) \in O\big(f(n)\big)$. Is it necessarily true that $\big(g(n)\big)^2 \in O\bigg(\big(f(n)\big)^2\bigg)$? Formally prove your answer.

b. Suppose $g(n) \in O\big(f(n)\big)$. Is it necessarily true that $2^{g(n)} \in O\big(2^{f(n)}\big)$? Formally prove your answer.

c. Is $n! \in O(2^n)$? Formally prove your answer.

5. Naive-Ford-Fulkerson

Consider a more naive version of the Generic Augmenting Path Algorithm(a.k.a. Ford-Fulkerson algorithm), where one operates on the true network instead of the residual network. In particular, the algorithm operates as follows. It finds *any* directed s-t path in the network, pushes the maximum amount of flow possible along that path, and then reduces the capacities of all edges along that path by the corresponding amount. The algorithm then repeats, iterating on the resulting reduced-capacity network, without ever considering any kind of residual network.

a. Prove that the Naive-Ford-Fulkerson algorithm may find a strictly sub-optimal flow (i.e. give an example).

b. Explain in words the fundamental importance of working on the residual network, as opposed to the 'true' network.

6. Amortized analysis of algorithms

We analyzed the runtime of several flow algorithms in class. Often the arguments proceeded by showing that certain operations only happen so many times, and thus the total number of operations can be sufficiently bounded. Consider the following simple algorithm INCREMENT for incrementing a binary number n by 1. The algorithm proceeds as follows. It starts at the left-most bit of n. If the first bit of n is a 1, it flips that bit to a 0 and moves one bit to the right. As long as the algorithm continues to see 1's, the algorithm flips the current bit to a 0 and again moves one to the right. The first time the algorithm sees a 0, it flips that bit to a 1 and terminates. If the algorithm reaches the end of the bit-string, it appends a new bit equal to 1 to the end of the string. Note that this is equivalent to implementing the 'carry-over' in simple addition. Suppose we iteratively use the algorithm INCREMENT to increment the n-bit 0's string to the n-bit 1's string. Furthermore, suppose the only cost incurred by INCREMENT is for flipping a bit, which always has a cost of 1.

a. How many times do we iteratively use INCREMENT? What is the maximum cost incurred by INCREMENT in any given iteration?

b. During which iterations does INCREMENT flip the first bit? What about the second bit? What about the kth bit? Prove this formally.

c. What is the cost of using INCREMENT to increment the n-bit 0's string to the n-bit 1's string? Why is this surprising in light of your answer to part a.?

d. Is INCREMENT a polynomial time algorithm for generating the n-bit 1's string from the n-bit 0's string?

7. 7.14

15.082J / 6.855J / ESD.78J Network Optimization
Fall 2010