Design and Analysis of Algorithms March 15, 2012
Massachusetts Institute of Technology 6.046J/18.410J
Profs. Dana Moshkovitz and Bruce Tidor Handout 12

Problem Set 5

This problem set is due **at 9:00pm** on **Wednesday, March 21, 2012**.

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of the first page of your solution with your name, the course number, the problem number, your recitation section, the date, and the names of any students with whom you collaborated. The homework template (LaTeX) is available on the course website.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions *that are described clearly*. Convoluted and opaque descriptions will receive lower marks.

Exercise 5-1. Do Exercise C.5-1 on page 1214 of CLRS.

Exercise 5-2. Do Exercise C.5-5 on page 1214 of CLRS.

Exercise 5-3. Do Exercise C.5-6 on page 1215 of CLRS.

Exercise 5-4. Do Exercise 11.3-3 on page 269 of CLRS.

Exercise 5-5. Do Exercise 11.3-5 on page 269 of CLRS.

Exercise 5-6. Do Exercise 11.5-1 on page 282 of CLRS.

Exercise 5-7. Do Problem 11-2 on page 283 of CLRS.

Problem 5-1. High Probability Bounds on Randomized Select

Recall that in lecture, we discussed an algorithm for randomized select, for which you can find pseudocode in CLRS on page 216 in section 9.2. In this problem, you'll explore the problem of high probability bounds for randomized select.

We'll go through a few steps to disprove the following statement, which we will refer to as $(*)$:

Let $T(n)$ be the running time of RANDOMIZED-SELECT on an input of size n. Then there exist integers n_0, $c \geq 1$ such that for all $n \geq n_0$, $P(T(n) > cn) \leq \frac{1}{n}$.

(a) Let b be a real number such that $1/2 < b < 1$, and let A_i be the array in the i^{th} recursion. Define a *bad pivot choice* in the i^{th} recursion as one that results in $|A_{i+1}| > b|A_i|$. (Note that in class, we have been using $b = 9/10$.)

Give a lower bound on the probability of having k bad pivot choices in a row. (The lower bound should hold for any input to RANDOMIZED-SELECT. Make it as tight as you can; you'll need it in part (c).)

(b) If our initial array size is n, then after one bad pivot choice, the next array is of size at least bn. Recall that the running time at each recursive call requires time equal to at least the size of the array.

Give a precise lower bound (do not use big-O notation) on the total running time of k recursive calls to RANDOMIZED-SELECT, if in every recursive call we chose a bad pivot.

(c) Use a proof by contradiction to disprove the statement $(*)$ above.

(Hint: Suppose there exists a c, n_0 satisfying the equation above. Then pick $b = 1 - \frac{1}{2c}$.)

Problem 5-2. Random Vectors and Matrices

Random vectors are good choices for hashing and testing, because they are unlikely to be orthogonal to a given (non-zero) input vector. In this problem, we will perform all operations in $\mathbb{Z}_p$ for some prime number p (which is to say we take the result of any arithmetic operation $\mod p$).

(a) You are given a non-zero vector $\vec{u} \in \mathbb{Z}_p^n$, and some number $c \in \mathbb{Z}_p$. Prove that if another vector $\vec{v} \in \mathbb{Z}_p^n$ has each element chosen independently and uniformly at random from $\mathbb{Z}_p$, then the probability that $\vec{v} \cdot \vec{u} = c$ is $1/p$.

(Hint: Observe that any non-zero vector $\vec{u}$ of size n has $n - 1$ elements which can be arranged into a non-zero vector of size $n - 1$. Then use induction to prove the claim.)

In class, we have seen a couple examples of universal hash families. We will now devise another universal hash family based on random matrices.

(b) You are given two vectors $\vec{x}, \vec{y} \in \mathbb{Z}_p^n$ such that $\vec{x} \neq \vec{y}$. Using the result from part (a), show that if $\vec{v}$ is a random vector as before, then $\Pr(\vec{v} \cdot \vec{x} = \vec{v} \cdot \vec{y}) = 1/p$.

(c) Using the result from part (b), show that if A is an $m \times n$ matrix with each element chosen independently and at random from $\mathbb{Z}_p$, then $\Pr(A\vec{x} = A\vec{y}) = 1/p^m$.

(d) Conclude that the family $\mathcal{H}$ of all such functions $h_A(\vec{x}) = A\vec{x}$ where A is an $m \times n$ matrix with elements in $\mathbb{Z}_p$, is universal.

In the implementation of the encryption algorithm BitWhipper™, the key step is to perform the composition of two hashes to compute the vector $\vec{y} \in \mathbb{Z}_2^k$ such that $\vec{y} = h_B(h_A(\vec{x}))$, where $\vec{x} \in \mathbb{Z}_2^n$, A is an $m \times n$ matrix, B is a $k \times m$ matrix, and h_A and h_B are defined as in part (d) with $p = 2$.

Note that all operations here are performed in $\mathbb{Z}_2$, which means all additions are bit-wise XOR.

Ben Bitdiddle gives you a $k \times n$ matrix C over $\mathbb{Z}_2$, which he says is equal to $B \cdot A$. If he's right, you could just use $h_C(\vec{x})$ in the implementation of BitWhipper™, in lieu of $h_B(h_A(\vec{x}))$. You want to make sure that $C = B \cdot A$, but you don't want to multiply the matrices because it will take you $O(kmn)$ time.

In part (e), you will come up with a randomized algorithm that runs much faster than $O(kmn)$ time, and also runs faster than any known algorithm for rectangular matrix multiplication.

(e) Using the result from part (a), devise a randomized algorithm to determine if $C = B \cdot A$. Show that your algorithm is correct with probability at least 90%.

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2012