Introduction to Algorithms October 21, 2002
Massachusetts Institute of Technology 6.046J/18.410J
Professors Erik Demaine and Shafi Goldwasser Handout 17

Problem Set 5

This problem set is due **in lecture** on **Wednesday, October 30.**

Reading: Chapter 14; §33.1-33.2

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 each sheet with your name, the course number, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.

Each problem should be done on a separate sheet (or sheets) of three-hole punched paper.

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 your essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudocode.
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. Graders will be instructed to take off points for convoluted and obtuse descriptions.

Exercise 5-1. Do Exercise 14.1-5 on page 307 of CLRS.

Exercise 5-2. Do Exercise 14.2-2 on page 310 of CLRS.

Exercise 5-3. Do Exercise 14.3-5 on page 317 of CLRS.

Exercise 5-4. Do Exercise 33.1-4 on page 939 of CLRS.

Exercise 5-5. Do Exercise 33.2-6 on page 947 of CLRS.

Problem 5-1. We have seen in class how SEARCH and INSERT can be performed on a randomized skip list in $O(\log n)$ time, with high probability (where n is the total number of elements in the structure). However, this assumes that the structure is used as a "black box": the actual layout of the structure is opaque to its user, and the operations performed are independent of the random coin flips used in constructing the skip list. In many contexts, these assumptions are reasonable; in this case we say that the user of the structure is ***oblivious***. Our analysis from class assumes an oblivious user.

However, in some situations this assumption is unfounded. For example, an adversarial user might be able to infer something about the layout of the data structure by performing "timing attacks" (observing the exact running time of a SEARCH or INSERT and adapting its queries based on that data). We say that an adversary is ***omniscient*** if it learns the results of every coin flip (and thereby, the exact layout of the skip list at all times). Note that an omniscient adversary cannot *control* the outcomes of the coin flips, it merely learns what they are. The point of this problem is to investigate how well (actually, how badly) skip lists tolerate omniscient adversaries.

(a) A skip list begins with just two boundary elements, $-\infty$ and ∞, at the lowest (and only) level. Define an ***uninterrupted chain*** to be a chain of consecutive elements in the lowest level, none of whose elements appear in any higher level. Let ℓ_i be the length of the longest uninterrupted chain after inserting the ith element. Give an omniscient adversary strategy for adaptively inserting elements that has the following properties: for all i, $\ell_{i+1} \geq \ell_i + 1$ with probability at least $1/2$, and $\ell_{i+1} \geq \ell_i$ with probability 1. In other words, with each insertion, the length of the longest uninterrupted chain must never become any shorter, and it must increase with probability at least $1/2$. You may assume that, for any distinct keys x and z, the adversary can find a key y such that $x < y < z$.

(b) Prove that, by inserting n elements according to your strategy from above, the length of the longest uninterrupted chain is $\Omega(n)$ with high probability.

(c) What are the implications of this strategy on the running time of SEARCH and INSERT? Contrast this with the performance of a red-black tree in the presence of an omniscient adversary. Be brief.

Problem 5-2. You are soon to be a contestant on *Survivor: MIT*. The format of the show is as follows: the n contestants choose among seats, at a round table, consecutively labelled 1 through n. Then, the host goes around the table (starting at chair 1) and kicks off every mth remaining contestant, until none remain. The longer a contestant remains, the more fabulous the cash and prizes he receives. The (n, m)-***Survivor permutation*** is the order in which the contestants are kicked off the show. For example, the $(7, 3)$-Survivor permutation is $(3, 6, 2, 7, 5, 1, 4)$.

You don't know n (the number of contestants) or m ahead of time, but you definitely want fabulous prizes. Therefore you need a quick way to determine the order of elimination so you can grab the best seat (or the second-best if it has already been taken, etc.).

(a) Give an $O(nm)$-time algorithm that, given integers n and m, ouputs the (n, m)-Survivor permutation. *Hint:* Think about a circular, doubly-linked list.

(b) The show's producers tell you that m could in fact be very large (up to n), so you decide your algorithm from above isn't efficient enough. Give an $O(n \log n)$-time algorithm that, on integers m and n, outputs the (n, m)-Survivor permutation. *Hint:* How can you efficiently find the mth successor in a dynamic set?

Problem 5-3. As the new accountant for AndersenArthur (the previous accountant is on an extended "leave of absence"), you must design a data structure that stores the history of all the transactions on its bank account. In keeping with corporate policy, this data structure should support insertion of past and future transactions, as well as deletion of existing transactions.

For this problem, you may assume that no two transactions occur on the same date. The data structure should be able to support the following operations:

INITIALIZE: Initialize the account. The initial balance in the account is \$0. This operation should take $O(1)$ time.

INSTRANS(*sum*, *date*): Insert a given transaction at a given date. The sum can be either positive or negative, and should be added to the balance in the account starting from the following day. Notice that the date can be arbitrary (not necessarily today's date). This operation should take $O(\log n)$ time, where n is the number of transactions in the database.

DELTRANS(*date*): Delete the transaction that occurs at the given date, if there is any. When a transaction is deleted, the corresponding sum should be subtracted from the balance in the account starting from the following day. This operation should take $O(\log n)$ steps.

BALANCE(*date*): Returns the balance in the account at the beginning of the given date. This operation should take $O(\log n)$ time.

Describe the data structure you would use, give clear and concise descriptions of the operations, and argue why they meet the desired running times. **Do not write code**.

Problem 5-4. Consider n chords on a circle. Suppose the circle is centered at $(0, 0)$ in the xy-plane, and the chords are given as an unordered list of pairs of their (x, y) endpoints. In this problem, we will develop an $O(n \log n)$ algorithm for counting the number of pairs of chords that intersect inside the circle. (For example, if the n chords are all diameters that meet at the center, then the correct answer is $\binom{n}{2}$.) We assume that no two chords share an endpoint.

(a) Suppose we want to sort the endpoints by their counter-clockwise angles, relative to the positive x-axis (i.e., their polar angles). To do this, we need an algorithm to compare two points: $p_1 > p_2$ if the angle of p_1 is greater than the angle of p_2. Give an $O(1)$-time algorithm to perform such a comparison, which does not rely upon trigonometric functions or division. You may assume that the points lie on the circle.

(b) Now we want to label every endpoint as either A_i or B_i, such that the following properties hold:

1. Each chord has a unique $j \in \{1, \ldots, n\}$ for which one endpoint is labelled A_j and the other endpoint is labelled B_j.
2. In a counter-clockwise sweep around the circle, starting at the positive x-axis, A_j appears before B_j and before A_{j+1}, for all j.

Give an $O(n \log n)$-time algorithm to perform this labelling.

(c) Give an $O(n \log n)$-time algorithm to compute the number of pairs of chords that intersect inside the circle. Note that two chords CD and EF intersect if and only if exactly one of the points E, F appears between C and D on the circle. *Hint:* You may want to use dynamic order statistics.