Problem Set 13

Due: Friday, December 5, 2003.

NONCOLLABORATIVE Problem 1. Consider a collection of faculty available to give you advice about your future path in life. Since they are busy, each only answers yes or no questions. Some are wiser than others, and are more often correct in their advice. You would like, looking back, to have taken the advice of the wisest faculty member, but ahead of time you don't know who that is. Consider the following online algorithm: each time you have a question, you ask all the faculty, and go with a "weighted majority" opinion as follows:

1. Initially, each faculty member has a weight of 1.
2. After asking a question, take the (yes/no) answer of larger total weight.
3. Upon discovering which answer is correct, halve the weight of each faculty member who answered incorrectly.

You will show that, using this algorithm, you make at most $2.41(m + \lg n)$ mistakes, where m is the number of mistakes made by the wisest faculty, and n is the number of faculty. (Thus, in the language of asymptotic competitive ratios, you are 2.41-competitive against asking the best faculty member.)

(a) Prove that, when the online algorithm makes a mistake, the total weight of the faculty decreases by a factor of $4/3$. Use this to upper bound the total weight assigned to faculty.

(b) Lower-bound the weight assigned to faculty by considering ths weight of the wisest faculty member in term of m.

(c) Combine the above two parts to prove the claim.

With randomization, you can do better. We modify the online algorithm as follows:

1. Instead of going with the majority, choose one faculty member with probability proportional to his weight, and follow his opinion.
2. Instead of multiplying incorrect faculty weights by $1/2$, multiply them by some $\beta < 1$ to be determined later.

(d) For a given question, let F denote the fraction of the weight of faculty with the wrong answer. Give an expression for the expected (multiplicative) change in the total weight as a result of reweighting for this question.

(e) Arguing much as in the deterministic case, prove that the (expected) number of wrong answers will be at most

$$\frac{m \ln(1/\beta) + \ln n}{1 - \beta}$$

so for example, setting $\beta = 1/2$ gives $1.39m + 2\ln n$, for a competitive ratio of 1.39.

(f) Show that if one uses the "right" β, one can limit the number of errors to $m + \ln n + O(\sqrt{m \ln n})$, achieving a 1-competitive algorithm (in the asymptotic sense). (This β can actually be found online be "repeated doubling" as the mistakes build up.)

Problem 2. Suppose you are given N line segments in the plane, each of which is horizontal or vertical. Show how to compute all K intersections among the line segments using $O(\frac{N}{B} \log_{M/B} \frac{N}{B} + K/B)$ memory transfers in the external-memory model. (**Hint:** Use a sweep-line algorithm and buffer trees.) You may assume that no two horizontal segments intersect and that no two vertical segments intersect.

Problem 3. Suppose you are given a linked list: there is an array of N nodes in arbitrary order, and each node stores a pointer to the next element (successor) in the list. In this problem, you will solve the *list ranking* problem: compute the *rank* or *index* of each node in the list. You may assume a black box that sorts using $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers in the cache-oblivious model.

(a) Show how to add to each node a pointer to its successor's successor using $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers in the cache-oblivious model. (**Hint**: Use sorting.)

(b) Suppose we scan the nodes and, for each node, label the node with the outcome of an independent fair coin flip. What local rule can we use to select αN nodes in expectation, where $0 < \alpha < 1$ is a constant of your choice, with the property that no two selected nodes are adjacent in the list? Show how to label each node accordingly as selected or unselected using $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers in the cache-oblivious model.

(c) Show how to unlink the selected nodes from the list, leaving just the unselected nodes in the list, using $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers in the cache-oblivious model.

(d) Suppose that we recursively solve the list-ranking problem on the list of unselected nodes. Show how to then solve the original list-ranking instance using $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers in the cache-oblivious model.

(e) What is the total number of memory transfers used by the recursive algorithm outlined for list ranking?

Problem 4. Suppose you have a large binary suffix tree given by a pointer representation: there is an array of N nodes in arbitrary order, and each node stores a pointer to its left child (if there is one) and its right child (if there is one). You want to be able to support queries of the form "how many occurrences are there of substring x?" which requires knowing the size of every subtree. You will show how to quickly preprocess the tree to store subtree sizes in the cache-oblivious model. You may assume a black box that sorts using $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers in the cache-oblivious model.

(a) Show how to compute parent pointers and whether each node is a left child or right child of its parent using $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers in the cache-oblivious model. (**Hint:** Use sorting.)

(b) Given parent pointers and whether each node is a left child or right child, show how to construct a linked list representing an Euler tour of the tree (the order in which nodes are visited and revisited in a depth-first traversal of the tree) using $O(1 + N/B)$ memory transfers in the cache-oblivious model. Your linked list must fit in an array of size $O(N)$.

(c) Suppose that you have solved the list-ranking problem from Problem 3 on the linked list representing the Euler tour using $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers. Show how to compute the size of the subtree rooted at every node using $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers in the cache-oblivious model.

Problem 5. Suppose you are an Internet router and you see a constant stream of IP addresses passing by. You'd like to keep track of which IP addresses occur frequently during a day.

(a) Suppose first that you are have space for just a single counter and a single register. Initially you set the register to the first address you see on the stream and you set the counter to 1. Whenever you see the address in the register, you increment the counter. Whenever you see any address other than the one in the register, you decrement the counter. Whenever the counter reaches zero, you set the register to the next address that appears on the stream and you set the counter to 1. Prove that, if any element appears with frequency $> 1/2$ during the day, then at the end of the day, that element is stored in the register.

(b) Generalize to the case of k counters with k corresponding registers: find a scheme with the property that, if any element appears with frequency $> 1/(k+1)$ during the day, then at the end of the day, that element is stored in a register.