Massachusetts Institute of Technology
6.854J/18.415J: Advanced Algorithms
David Karger

Handout 19
Wednesday, November 7, 2012

Problem Set 10

Due: Wednesday, November 14, 2012.

Collaboration policy: collaboration is *strongly encouraged*. However, remember that

1. You must write up your own solutions, independently.
2. You must record the name of every collaborator.
3. You must actually participate in solving all the problems. This is difficult in very large groups, so you should keep your collaboration groups limited to 3 people in a given week.
4. **No bibles. This includes solutions posted to problems in previous years.**

NONCOLLABORATIVE Problem 1. A *conservative algorithm* is one that makes at most k page faults on any consecutive subsequence of the input that contains at most k pages. Here k is the size of the cache.

(a) Prove that LRU and FIFO are conservative

(b) Prove that any conservative algorithm is k-competitive.

Problem 2. Here we will generalize from the Double Coverage algorithm for k-server on the line to k-server on trees. We say that a server s_i is a "neighbor" of a request if there is no other server on the path from s_i to the request.

Algorithm DC-TREE: At each time, all the servers neighboring the request are moving at a constant speed toward the request.

For analysis, we will use the same potential function used for the line: $\Phi = kM_{\min} + \sum_{\text{DC}}$, where $M_{\min}$ is the minimum cost matching between OPT's and DC's servers, and $\sum_{\text{DC}} = \sum_{i<j} d(s_i, s_j)$. Note that the set of neighbors may decrease during the service process since a moving server may become "blocked" by other servers.

(a) Show that **DC-TREE** is k-competitive.

Hint: Break the algorithm into phases, where in each phase the number of neighbors is fixed. Now consider the changes to each of the parts of the potential function within a phase.

(b) Show that any algorithm for k-server on a tree can be used to solve the paging problem by modeling paging as k-server on a particularly simple tree. Note that k-server algorithm can place servers midway along edges, which doesn't make sense for paging, so you have to "reinterpret" it a bit to get a paging algorithm.

(c) What standard paging algorithm do you get when you apply the above reduction using DC-TREE?

Problem 3. Consider the problem of finding a lifetime companion. Among the pool of k potential partners at the university (MIT students being snobby about this) you can choose and date any one for a while and by doing so measure their suitability compared to all your previous dates'. You then decide whether to stay together forever or break up forever. Your goal is to find the most suitable companion, as measured by a rank ordering of all the possibilities.

(a) Show that any deterministic strategy for choosing dates and deciding whether to break up is terrible from a competitive perspective: you can be forced by fate to end up with the absolutely worst possible choice.

(b) Devise a randomized strategy that does better, giving you a constant probability of ending up with the absolute best companion.

(c) Suppose that you want to play for slightly less high stakes. Give an algorithm minimizing the *expected rank* of your final choice. You will receive full credit for achieving rank $O(\log k)$ and partial credit for worse bounds.

(d) Comment on potential ramifications for dating at MIT.

OPTIONAL (e) Show how to achieve expected rank $O(1)$.

Problem 4. 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 the weight of the wisest faculty member in terms 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 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 by "repeated doubling" as the mistakes build up.)

Problem 5. In many applications, one wants to do range searching among objects other than points. In this problem, we will see that we can reduce several problems of this flavor to normal orthogonal range searching.

(a) Let S be a set of n axis-parallel rectangles in the plane (i.e., the sides of the rectangles are vertical and horizontal). We want to be able to report all rectangles in S that are completely contained in a query rectangle $[x : x'] \times [y : y']$. Describe a data structure for this problem that uses $O(n \log^3 n)$ storage and has $O(\log^4 n + k)$

query time, where k is the number of reported answers. (**Hint:** Transform the problem to an orthogonal range searching problem in some higher-dimensional space.)

(b) Let P consist of a set of n polygons in the plane. Again, describe a data structure that uses $O(n \log^3 n)$ storage and has $O(\log^4 n + k)$ query time to report all polygons completely contained in the query rectangle, where k is the number of reported answers (note that as a special case, you can report containment of line segments).

Problem 6. How long did you spend on this problem set? Please answer this question using the Google form that is sent to you via a separate email. This problem is mandatory, and thus counts towards your final grade. It is due by the Monday 2:30pm after the pset due date. You can find the link to the form on the course website.