15-451/651 Algorithm Design & Analysis, Spring 2024

Recitation #11

Objectives

- Practice designing and reasoning about algorithms in the streaming model
- Understand the objective of an online algorithm, how to create one, and how to bound its competitive ratio.

Recitation Problems

1. **(Plausible Majority Elements)** The ε-heavy-hitters algorithm guarantees that it will output all elements $e \in \Sigma$ such that $\text{count}_t(e) > \varepsilon t$ if they exist, but makes no guarantees whatsoever about the sorts of false positives it may emit. Suppose for example that we want to find the majority element (aka $\varepsilon = \frac{1}{2}$).

   (a) Show that it is possible for the ε-heavy-hitters algorithm to output an element which appears only once in an arbitrarily large stream.

   (b) Come up with a one-pass streaming algorithm that guarantees it will output the majority element if there is one, but will never output an element that appears less than a third of the time (an implausible majority element).

   Hint: Your algorithm might need to output more than one element.

   (c) What is your algorithm's asymptotic space complexity as a function of t and $|\Sigma|$?

   (d) Generalize your algorithm to guarantee it will output all elements e such that $\text{count}_t(e) > \varepsilon_1 t$ but never output an element e' such that $\text{count}_t(e') < \varepsilon_2 t$ for $\varepsilon_1 > \varepsilon_2$. What is its asymptotic space complexity?

2. **(Online Paging)** In the paging problem, we have a disk with N **pages**, and a *cache* with space for k **pages** for some $k < N$. A sequence of *page requests* are made, where each request is for a particular page i, with $1 \leq i \leq N$, which requires us to load that page into the cache. If the page is already in the cache, the request is free (it costs 0). If the page is not already in the cache (this is called a *miss*), we must pay a cost of 1 to load it in, but we must also *evict* one of the current k pages in the cache to make space for it. We assume that the cache can start with any pages already in it.

   The goal is to minimize the number of misses in a sequence of requests, so for any algorithm trying to do this, the important question is **which page to evict**.

   Here are two algorithms that solve this question:

   - *Least-Recently-Used* (LRU): Evict the least recently used page. Start out the cache with pages $1, 2, \ldots, k$.
   - *Longest-Forward-Distance* (LFD): Evict the page whose next request is farthest in the future. Start out the cache with the first k unique pages in the request sequence.

   LRU is an online algorithm because it only uses information from the past, but LFD is not, since it requires knowing the future (it is an *offline algorithm*). In fact, LFD can be shown to be an optimal offline algorithm for this problem. Now we want to show that LRU is k-competitive.

   (a) Say we have $k = 3$, $N = 4$, and the request sequence is $1, 2, 3, 2, 4, 3, 4, 1, 2, 3, 4$. How does LRU and LFD perform?

(b) Now, we will prove why we cannot do better than a competitive ratio of k with LRU by proving something stronger. Show that you cannot do better than a competitive ratio of k with **any** deterministic online algorithm.

Hint: If we set $N = k + 1$, what request sequence would be the worst case for any online algorithm? How does this compare to LFD, which can see future requests at any time?

(c) Now, we will show that LRU can indeed achieve a competitive ratio of k. Define the notion of a **phase** as a contiguous subsequence of requests that contain k distinct requests, such that the first request of a phase is distinct from all requests in the previous phase. Consider a series of requests that has m phases, with the first phase starting on the first page fault.

Notice that LRU incurs at most k page faults in each phase, and thus the total number of page faults is at most $k \cdot m$, since the first phase starts on the first page fault. Now, prove that *any* algorithm must pay a cost of at least m in total, which shows that LRU is k-competitive.

Hint: For any pair of phases i and $i+1$, look at the sequence from the second request of phase i to the first request of phase $i+1$ inclusive.

3. **(Randomized Paging (Optional advanced problem))** We just proved that LRU has a competitive ratio of k for the online paging problem, and also that k is a lower bound for any deterministic paging algorithm, so LRU is optimal. So, are we done with paging? No! We can still design an algorithm with a **much** better competitive ratio by taking advantage of randomization! This is an example of a problem where randomization is not just a convenience, but an actual necessity to break through lower bounds!

   (a) Here, we present a randomized algorithm with a competitive ratio of just $O(\log k)$.

   Marking Algorithm

   i. The initial state is pages $1, ..., k$ in fast memory. Start with these pages all marked.

   ii. When a page is requested:

   - If it's in fast memory already, mark it.
   - If not, throw out a random unmarked page. (If all pages in fast memory are marked, unmark everything first. For analysis purposes, we will call this the start of a "phase." Then, bring in the page and mark it.

   We can think of this as a 1-bit randomized LRU, where marks represent "recently used" vs "not recently used". When it needs to evict an element, instead of evicting the least-recently used, it evicts a random element that was not among the most recently used ones. We will analyze the special case where $N = k+1$. Prove that when the marking algorithm is run on a sequence σ of accesses,

$$\frac{\mathbb{E}[MARKING(\sigma)]}{OPT(\sigma)} \leq H_k$$

   where $H_k := 1 + 1/2 + ... + 1/k$ is the k^{th} harmonic number.

(b) **(Extra advanced, extra optional problem)** Prove that $\Omega(\log k)$ is a lower bound for the competitive ratio of any randomized paging algorithm.