

15-451/651 Algorithm Design & Analysis, Fall 2025

Recitation #11

Objectives

- Practicing using **potential functions** to analyze the competitive ratio of an online algorithm
- Review the online paging / caching problem from lecture
- Understand the data stream model and streaming algorithms, and their analysis

Recitation Problems

1. (**Favoritism**) You're given a rooted, unweighted tree T of n nodes (not necessarily a binary tree, or a balanced tree). For each node in the tree, one of the children of that node is designated as the "favorite child". Initially the left-most children are all favorites, but they may change over time.

We want to perform a sequence of $\text{Traverse}(x)$ operations. The $\text{Traverse}(x)$ operation traverses the path from the root of the tree to the node x (assume that the algorithm knows the path it needs to take in order to get from the root to x). The cost of a traversal is the number of nodes visited (not counting the root), except that visiting a favourite child is free!

To help improve the cost, the algorithm is allowed to change the favorite child of any node at any time. Changing the favorite child of a node costs one. Consider the following greedy online algorithm: before performing a traversal, make all of the nodes on the path the favourite child of their parent if they are not already.

Using a potential function Φ , prove that this algorithm is 2-competitive, that is, it incurs a cost at most double that of an optimal omniscient algorithm that can see the future and plan ahead.

2. (**Unbiased approximate counting**) In lecture we learned about the CountMin algorithm, which is called a *biased* estimator because it always overestimates its true value. In this problem we analyze a different estimator that does not have this feature. The idea is relatively simple: the CountMin algorithm is an overestimate because all of the counts are added and some may collide. To fix this, instead of always adding counts, for each element, we randomly choose whether to add or subtract its count.

Hash functions We start by selecting not one, but *two* hash functions this time. We let $h : \Sigma \rightarrow [k]$ be a hash function chosen from a *universal* family, and $g : \Sigma \rightarrow \{-1, 1\}$ be a hash function chosen from a *pairwise independent* family.

The algorithm The algorithm proceeds in a similar manner to CountMin, using the hash function g to decide whether to add or subtract the count of a particular element.

```

when update  $a_t$  arrives
if ( $a_t ==$  (add,  $e$ )) then
    counts[ $h(e)$ ] +=  $g(e)$ 
else
    counts[ $h(e)$ ] -=  $g(e)$ 

```

We define the estimate of element e to be

$$\text{est}(e) := g(e) \cdot \text{counts}[h(e)].$$

- (a) We want to show that the expected value of $\text{est}(e)$ is exactly $\text{count}(e)$, i.e., the true count. This is called an *unbiased estimator*. First, similar to lecture, give an expression for $\text{est}(e)$ in terms of the true counts and the hash functions.
 - (b) Using Part (a), show that $\mathbb{E}[\text{est}(e)] = \text{count}(e)$. (Hint: remember to use the fact that g and h are *independent* and that g is *pairwise independent*!)
3. **(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 $|\Sigma|$?