CS420+500: Advanced Algorithm Design and Analysis

Lectures: March 29 + 31, 2017

Prof. Will Evans *Scribe: Matthew Hounslow*

In this lecture we:

- Discussed Randomized Online Algorithms;
- Random Marking Mouse (RMM);
- Hash Functions

Note: Friday next week there will be a final review session from 1.30 - 3.00.

1 Randomized Online Algorithms

We talked a little about this in the previous lecture, but continued on with our Random Marking Mouse. This is described below.

1.1 Random Marking Mouse (RMM)

Locations the mouse can hide are equivalent to different pages ($1, 2, ..., m$ different pages), locations where mouse isn't are pages in the cache. A cat probe sequence is the equivalent of a page request sequence. Deterministic mouse can achieve no better than $(m - 1)$-competitive.

The Random Marking Mouse (RMM) performs the following algorithm:

1. Start at a random location
2. When cat probes spot/location the mouse then marks it
3. When cat probes the mouse's location, mouse moves to a <u>random</u> unmarked spot.
4. If mouse as at the last unmarked spot it clears the marks [a new phase begins]

Claim 1. $E[RMM_{cost}(p_1p_2...p_n)] \leq O(\log m) \cdot\ OPT(p_1p_2...p_n)$

Proof. Initially the probability that mouse is at any location is $\frac{1}{m}$. The first cat probe therefore finds the mouse with probability $\frac{1}{m}$. Whether the first probe finds the mouse or not, the mouse is now at any of the $m - 1$ unprobed locations with equal probability. This means that our second cat probe has a $\frac{1}{m-1}$ probability of finding the mouse. This continues to the third probe ($\frac{1}{m-2}$) and so forth.

We can define the **expected number of times RMM is found during a phase** using indicator variables. An indicator variable is defined as follows:

$$X_i = \begin{cases} 0 \text{ if mouse is not found on probe } i \\ 1 \text{ if mouse is found on probe } i \end{cases}$$

E[number of times RMM is found during phase] $= E[\sum_{i=1}^{m} X_i]$ This ends up giving us a harmonic series:

$$E[\sum_{i=1}^{m} X_i] = \frac{1}{m} + \frac{1}{m-1} + \frac{1}{m-2} + ... + \frac{1}{1} = H_m \approx \ln m$$

Since every spot is probed by the end of this sequence of probes (called a phase), the optimal mouse OPT must move at least once. Hence the **number of times OPT is found during phase** ≥ 1.

□

Claim 2. *For all mice A (deterministic or randomized) there exists a sequence $p_1 p_2 ... p_n$ such that*

$$E[A_{cost}(p_1 p_2 ... p_n)] > (\log m) \cdot OPT(p_1 p_2 ... p_n)$$

Proof idea: Show that a cat exists that will cause A to move $> \log m$ times more than OPT.

Proof. If cat probes at random then no matter what A does, cat finds it with probability $\frac{1}{m}$. The expected number of times A must move over sequence of t probes is $\frac{t}{m}$.

So how many random cat probes until cat examines all m locations? This is related to the coupon collector problem $\Rightarrow \boxed{m \ln m}$.

So OPT moves once every $m \ln m$ probes, while A moves $\frac{m \ln m}{m}$ times. Therefore A faults $\geq (\ln m) \cdot OPT$ □

2 Hash Functions

2.1 Universal Hash Functions

Universal Hash Functions : A set of hash functions H, such that each maps keys to indices. It is universal if for all distinct keys $k, l \in U$ (where U is the set of keys), the number of hash functions $h \in H$ such that $h(k) = h(l)$ is at most $\frac{|H|}{m}$ where m, is the size of the hash table.

Fixed hash functions are usually a very bad idea, why? Because smart people will find a sequence of keys that will cause your hash function to break!

It should be noted however that this is possibly not true for hash functions that use a cryptographically secure hash function, meaning a function h for which it is believed to be computationally expensive, given $h(k)$ to find k (or any value l such that $h(l) = h(k)$). Unfortunately, these functions can be slow to compute, meaning calculating $h(k)$ given k is slower than for typical hash functions. A better idea is to choose a hash function at random from a set of good hash functions (such as a **universal set of hash functions**).

2.2 Chaining using universal hash functions

Hash n keys into a table T of size m, where we consider a load factor $\alpha = \frac{n}{m}$. The **load factor** is the average number of items that hash to the same location.

Using a randomly chosen $h \in_R H$. In this case $\in_R$ means chosen at random uniformly. Let n_i be the number of items in bucket i.

Theorem 3. *For any key* $k, E[n_{h(k)}] \leq \begin{cases} \alpha + 1 \text{ if key } k \in T \\ \alpha \text{ if } k \notin T \end{cases}$