CS420+500: Advanced Algorithm Design and Analysis

Lectures: March 27th + March 29th, 2017

Prof. Will Evans *Scribe: Anya McGee*

In this lecture we:

- Discussed how to compare online algorithms
- Proved that deterministic algorithms are bounded on how competitive they can be
- Discussed the performance of randomized online algorithms

1 Comparing Online Algorithms

Last time, we looked at how to choose pages to replace such that the number of page-ins needed is minimized.

The bad news for today is: comparing different algorithms is hard.

Theorem: If A is a deterministic, c-competitive online algorithm, then $c > k$ (where k is the number of pages in the cache).

So, according to this theorem, we can show provably bad performance for every deterministic algorithm we try.

Proof Idea: Note that if an algorithm A is c-competitive, for every input sequence $P = p_1, p_2, ...p_n$, $A(P)$ causes fewer faults than $c * OPT(P)$ (Where OPT is the optimal algorithm).

We need to show that for every deterministic online algorithm, there exists a terrible sequence $p_1, p_2, ...p_n$ that will cause a fault on every page request. We also need to show that the optimal solution does pretty well on this sequence.

Proof: We can find a sequence that is bad for A, but good for OPT.

Assume A and OPT have the same initial pages in the cache, WLOG number these pages 1,2,3,...k.

1. Request a new page, $k + 1$=a_1. A faults and evicts some page, call it a_2.

2. Request page a_2. A evicts some page, a_3.

3. Request page a_3.

... Repeat this process ad nauseam.

So, A faults on every page request. A is deterministic, so we can always figure out which page to pick next to make this happen.

This gives us a total number of different pages $k + 1$.

How many times does OPT fault for this sequence? Consider the first and second faults like this:

OPT	a_1	a_2	a_3	...	a_j	a_{j+1}
Faults	*					*
	First fault					Next fault

a_j is, WLOG, the last element before the next fault occurs.

Claim: $j \geq k+1$

Proof: OPT evicts the page in cache that is requested furthest in the future. Because our set of requests comes from the set of pages that were originally in the cache, plus a_1, there are only $k+1$ different pages. So, the next k pages can be kept in the cache.

OPT faults at most once every $k+1$ requests. A faults at every request. This implies that the number of faults for A on this sequence is greater than k times the number of faults for OPT on this sequence:

$f_A(a_1, a_2, ...a_n) \geq k * f_{OPT}(a_1, a_2, ...a_n)$

Note that the only property of A we used is that it is deterministic.

So, we have shown that, for is a deterministic, c-competitive online algorithm A, then $c > k$.

Theorem: LRU (Least Recently Used replacement algorithm) is k-competitive:

Proof: Let $p_1 p_2 ... p_n$ be an arbitrary sequence of page requests.

Partition this sequence into contiguous subsequences (phases) such that LRU faults on the first page of the phase and the phase contains exactly k different pages.

Example: Let's call the pages A, C, B, D and use a cache of size $k = 3$.

LRU	A	C	D	C	C	B	C	A	D	A	A
Faults	*	?	?			*	?	?	*	?	
Phase	1	—	—	—	—	2	—	—	3	—	...

Where * is a page fault. There could also be additional page misses within a phase, but not more than 2, as there can only be a fault on the first occurrence of a page in the phase. Note that this is true because of properties of LRU.

So, LRU faults at most k times per phase.

OPT must have the first page of a phase in the cache at the beginning of a phase. Since the remainder of the phase plus the first page of the next phase consists of k different pages, OPT must fault at least once during these requests.

So, we have that:

If $a =$ the number of phases, wher f is the number of faults:

$f_{LRU}(p_1, p_2, ...p_n) \leq k * a$

$f_{OPT}(p_1, p_2, ...p_n) \geq a - 1$

And we can see that LRU is k-competitive.

Our first theorem showed that a deterministic online algorithm for page replacement can be at most k-competitive. So, what about non-deterministic algorithms?

2 Randomized Online Algorithms

Cat and Mouse:

We can model the paging problem as a cat and mouse problem, where the mouse represents the page that is not in the cache, and the cat represents the page requests being sent in, like so:

The mouse hides in one of m hiding places. Every time step, the cat looks in one of the m places. If the mouse is there, the mouse must move to a different hiding spot.

The total cost is the number of times the mouse moves.

The optimal algorithm, OPT, represents the minimum number of times the mouse must move if the mouse can see the future and know where the cat will look.

Example: For hiding spots labelled 1234:

If the cat probes 12341234, the optimal cost is 2.

If the cat probes 12113412, the optimal cost is 1.

This corresponds to paging with $m-1$ spots in the casche, and m total pages. The location of the cat's probe represents the page being requested, and the mouse represents the page not found in the cache. If the mouse has to move, we have encountered a page fault.

We will find out that the deterministic mouse doesn't do very well.

Deterministic Mouse:

From the previous section, we know that for the deterministic mouse m, there is a sequence of cat probes s such that the mouse $cost_m(s) \geq cost_{OPT}(s)$.

So, the deterministic mouse can achieve no better than $(m-1)$-competitive compared to the optimal mouse.

Random Marking Mouse:

This algorithm executes the following:

```
1. Start at random location
2. When a cat probes a location, the mouse marks the spot the cat probed
3. When the cat probes the mouse's location:
    4. Move to a random, unmarked spot
    5. If the mouse is at the last unmarked spot, clear the marks
```

Note that clearing the marks in step 5 corresponds to starting a new "phase"

Claim: The expected cost of RMM over the algorithm's randomized choices is subject to the following inequality:

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

Proof: Initially, probability that mouse is at any spot $\frac{1}{m}$ because the mouse starts in a random location. This implies that the first probe finds the mouse with probability $\frac{1}{m}$.

Whether the mouse is found or not, the mouse is at each of the $m-1$ unmarked spots with probability $\frac{1}{m-1}$. We can see that if it had to run to one of them, it is now in one of those $m-1$ spots, or if it was not in the one we just checked, we can discount it and there are only $m-1$ spots.

Now, we move onto the second probe. Let us consider this to be the next probe to an unmarked spot, because a probe to a spot we've already probed won't incur any costs for the OPT or the RMM, so we can safely ignore it. This probe will find the mouse with probability $\frac{1}{m-1}$.

For the third probe, we can see that the probability of finding the mouse will be $\frac{1}{m-2}$. So, the overall probability of finding the mouse at each probe follows the pattern $\frac{1}{m-1}, \frac{1}{m-2}, \frac{1}{m-3}$... and so on.

So, how many times would we expect to find the mouse during this process?

E[# of times mouse is found] = E[$\sum_{i=1} x_i$] = E[x_1] + E[x_2] + ...

Indicator Random Variables:

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

So, by linearity of expectation values, we can see that:

# Times mouse is found during a phase $= \frac{1}{m} + \frac{1}{m-1} + \frac{1}{m-1} + ... + 1$

This is the harmonic series from 1 to m, which is equal to $\log(m)$.

So, the expected number of times the mouse is found during a phase is equal to $\log(n)$.

The number of times OPT is found during a phase is at least 1.

So, RMM is $\log(n)$ competitive with OPT. This is an improvement on our deterministic algorithms!