CS787: Advanced Algorithms	**Scribe:** Tuan Dinh
Lecture 12: Competitive Analysis for Caching	**Date:** 03/05/2019

12.1 Competitive Ratio

An online algorithm ALG is said to achieve competitive ratio r if there exists some constant c such that for any input sequence σ (of any length),

$$\mathrm{ALG}(\sigma) \leq r\,\mathrm{OPT}(\sigma) + c$$

If our objective is maximization, the condition becomes:

$$\mathrm{ALG}(\sigma) \geq \frac{1}{r}\,\mathrm{OPT}(\sigma) - c$$

For the randomized algorithm, the condition becomes:

$$\mathbb{E}[\mathrm{ALG}(\sigma)] \leq r\,\mathrm{OPT}(\sigma) + c$$

12.2 Caching/Paging

Consider each cache has a fixed number of pages, we are interested in 2 events:

- Cache miss: access page is not in cache - cost 1
- Cache hit: access page is in cache - cost 0

We need an eviction policy to decide what to evict when cache is full, and this is an online algorithm because we don't know which future accesses will be.
Our objective is to minimize the total cache miss cost.
Example: Given a 3-page cache and 5 access pages as the sequence: 1, 2, 1, 4, 3, 5, 1, 1, 2, 3, 5, 4. The beginning state of the cache is : 1, 2, 3.
We first have a cache miss at 4^{th} position, and we to decide which one we have to evict.
The optimal cost is 4. ■

For this example, we have an offline optimal algorithm: evicts furthest-in-future page. This is a greedy algorithm but it guarantees the optimal cost. Intuitively, for any eviction policy, at a cache miss step, we can exchange the chosen page to the furthest position. This doesn't increase the cost with just 1 movement step.

In practice, we have several well-known caching schemes: LRU (least recently used), LRF (least frequently used), FWF (flush when full), FIFO (first in first out), LIFO (last in first out). We will show, in the later parts, that a randomized version of LRU achieves the optimal worst case solution, this also explains why LRU outperforms other methods in practice. Furthermore, other caching methods are almost identical in competitive analysis (for the worst case).

12.2.1 Setting

- k: size of cache
- k + 1 access pages
- σ: Pages $1 \dots k$ in cache

Given this setting, we provide a bad sequence for LFU as below:

$$\sigma = \underbrace{1 \dots 1}_{\text{m times}} \underbrace{2 \dots 2}_{\text{m times}} \dots \underbrace{(k-1) \dots (k-1)}_{\text{m times}} \underbrace{k(k+1)}_{\text{m times}}$$

For this sequence,

- The optimal cost is 1 (evicts any page from 1 to k-1)
- The cost of LFU is $2m - 1$ because it keeps switching page k and page k + 1.

This shows how bad LFU is. However, one can ask a question: if the past doesn't reflect the future, what can algorithm do? In the next sections, we will try to derive the lower bound for this caching problem.

Now, fix any deterministic algorithm A. Adversary constructs input σ by picking at step i, the page that is not in cache at the step in algorithm A. So,

- $A(\sigma) = |\sigma|$
- OPT $\leq \frac{|\sigma|}{k}$ suffers a miss at most every k steps

From this, we have following theorems:

Theorem 12.2.1 *No deterministic algorithm can achieve a competitive ratio less than k for caching.*

Theorem 12.2.2 *LRU has a competitive ration of k.*

Fix σ, define 'phases' as follows:

- Phase 1: begins at 1st access
- Phase i: $(i > 1)$ begins at the $(k+1)^{\text{th}}$ distinct access after phase (i-1) begins

So, each phase has k distinct accesses, whenever we see a new page, just start a new phase.
Example: Cache has k = 3, and 5 access pages as following:

$$\underbrace{4,1,2,1}_{1}, \underbrace{3,5,1}_{2}, \underbrace{2,2,4,1}_{3}, \underbrace{3,5,3,5,3,3,1}_{4}, \underbrace{2,4}_{5}$$

■

Claim 12.2.3 *LRU suffers at most k cache misses in any phase.*

Proof: Every page accessed in phase suffers a miss at most at its first access in phase. This is the reason of defining phases as above. ■

Claim 12.2.4 *For any i, OPT suffers at least 1 cache miss among the 2nd through last access in phase i or first access in phase (i+1).*

Proof: It's obvious from the phase partition.

$$4, \underbrace{1, 2, 1, 3}_{1}, \underbrace{5, 1, 2}_{2}, \underbrace{2, 4, 1, 3}_{3}, \underbrace{5, 3, 5, 3, 3, 1, 2}_{4}, \underbrace{4}_{5}$$

■

With m being the number of phases,

- LRU costs no greater than mk
- OPT costs at least $m - 1$

Hence, the competitive ratio is at most k. Q.E.D

A simpler version of LRU getting the same CR is 1-bit LRU, which simply keeps track of each page by a single bit. For eviction, it arbitrarily evict a page for the new access page. We can achieve the same results as of LRU with identical analysis. In the next section, we will examize a different version of 1-bit LRU.

12.3 Randomized 1-bit LRU or Marker

Algorithm

- At the beginning, all bits are 0
- When a page is accessed:
  - If in cache, set bit to 1
  - if not in cache,
    - If all bits are 1, reset all bits to 0
    - Evict a uniformly random page with bit being 0
    - Bring new page in and set its bit to 1

Notice that this is a randomized algorithm, hence the adversary doesn't have access into the sequences generated.

Theorem 12.3.1 *No randomized algorithm can achieve* $CR < \log k$

Consider (k+1) access pages and σ is a set of uniformly random page at every step. Thus, no algorithm can exploit this sequence. Hence, we have:

- $ALG(\sigma) = \frac{|\sigma|}{k+1}$: every step, $P(miss) = \frac{1}{k+1}$
- $\mathbb{E}[OPT(\sigma)] \leq \frac{|\sigma|}{k \log k}$

Hence, the gap ratio is $\Omega(\log k)$

In the OPT case, the problem is actually the well-known **coupon collector** problem: given n coupons, get a uniform randomized coupon at every step, the question is after how many steps have we collected at least 1 copy of each coupon ? The answer for this problem is $O(n \log n)$.