**CS 270: Combinatorial Algorithms and Data Structures** Spring 2023

Lecture 15 – March 7th, 2023

*Prof. Jelani Nelson* *Scribe: Vaibhav Agrawal, Eric Nyugen*

## Continuing Online Algorithms

Today looking at mostly paging: cache replacing policy.

# 1 Deterministic Paging

We consider the paging problem for a cache of size $k$. If a page is already in the cache, we can access it for free. Otherwise, we will incur cost by retrieving it from disk.

**Definition 1.1** (Page Fault)**.** A page fault occurs when the cache misses, which means the page we were requesting was not in the cache. If the cache is full, we will need to pick a page to evict to make room for the requested page. We define the cost of a page fault cost = 1.

## 1.1 Last Time: List Update Problem

In the list update problem, we maintain a linked list of items. We had an operation access(x), which incurs cost of $i$, where i is the position of x in the list. We can view the paging problem as a list update problem but with a modified cost function, where

$$cost(i) = \begin{cases} 1, & \text{if } i > k. \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

After any access(x), we move item $x$ to the front as the first $k$ items in the list represents what is in the cache. Also, we can assume some exchanges are free (for eviction).

## 1.2 Heuristics for Paging

Listed are heuristics for paging that choose which page to evict in the case the cache is full.

- **Least Recently Used** or **LRU**: The page that was least accessed will be evicted.
- **First In First Out** or **FIFO**: For every page in the cache, we record the timestep when they first entered the cache. The page who has been there the longest (has the earliest timestep) will be evicted.
- **Least Frequently Used** or **LFU**: We track a counter for each page. Each time a page is accessed, add to its counter. We then evict the page with the smallest counter.
- **Belady's Algorithm (1966)**: We evict the page that will be requested furthest in the future. This is the optimal algorithm.

**Resource augmentation:** $cost(LRU_k) \leq \frac{k}{k-h+1} OPT_n$

**Lemma 1.2.** *No deterministic algorithm can have competitive ratio less than k.*

*Proof.* Imagine there are k+1 different memory address in my machine. Given any algorithm A, consider it currently has k things in its cache. It will bring something in and evict something. Then I will request the evicted page. Do this is a loop. This leads to m requests $\implies$ m page faults for A.

Look at OPT. It evicts the person that is requested farthest in the future. So every time it faults it does not fault any more for the next k+1 times. So Opt faults $\leq \lceil \frac{m}{k} \rceil$ times. □

**Definition 1.3.** 1-bit LRU

1. unmark all pages
2. request(x)
   (a) if $x \notin$ cache If all pages are marked then unmark all. Always: Evict arbitrary unmarked pages and finally insert $x$ into cache
   (b) mark x

**Theorem 1.4.** *LRU/FIFO are k-competitive.*

*Proof.* Will show **1-bit LRU** is k-competitive. LRU is a way of implementing 1-bit LRU so we just need to proof this.

Divide time into phases:

1. everytime we execute step 1 starts new phase.
2. in each phase, 1-bit LRU has $\leq k$ page faults.
3. OPT has $\geq 1$ fault per phase.

□

# 2 Randomized Paging

## 2.1 Attack Model for Randomized Algorithms

- **Omniscient adversary**: This adversary can see the future, including the outcomes of all our calls to random(). For example, it knows our future coin tosses. From the adversary's perspective, our randomized algorithm is essentially deterministic. No algorithm can beat k-competitive against this.
- **Adaptive adversary**: This adversary cannot see the future, but they can see our past. They know what random decisions we made up until now. So, they know what is in our cache. No algorithm can beat k-competitive against this either.
- **Oblivious adversary**: This adversary only knows the code for our algorithm. It does not know future outcomes for calling random(), and does not know what is in our cache. Then,

we claim that $2H_k$-competitive is possible where $H_k$ is the kth harmonic number, defined as $H_k = \sum_{j=1} \frac{1}{j}$. We also bound $H_k \approx ln(k) + O(1)$.

It is known that $H_k$-competitive is actually possible due to [McGeoch, Sleaor, '91] and less than $H_k$ is not possible.

**Definition 2.1.** Competitive ratio of randomized algorithm $A$ is $C$ if for all sequences $\sigma$:

$$\mathbb{E}[cost(A(\sigma))] \leq C \cdot OPT(\sigma) + O(1) \tag{2}$$

## 2.2 Randomized Marking Algorithm

The randomized marking algorithm $Mark$ is an implementation of 1-bit LRU, but randomized. The modification to the algorithm is that we evict a uniformly random page, instead of an arbitrary page.

**Definition 2.2.** A page is clean if it was not requested in the last phase and not requested so far in the current phase.

**Definition 2.3.** A page is stale if it was requested in the last phase but not yet in the current phase.

For future analysis, we define $D_i :=$ the number of pages in $OPT$'s cache at the beginning of phase $i$, which are not in $Mark$'s cache. Equivalently, $D_{i+1} :=$ the number of pages in $OPT$'s cache at the end of phase $i$, which are not in $Mark$'s cache. We also define $L_i :=$ the number of clean requests in phase $i$.

**Claim 2.4.** $OPT \geq \frac{1}{2} \sum_i (L_i + D_{i+1} - D_i) \geq \frac{1}{2} \sum_i L(i)$
This series telescopes. $D_0$ is 0 and $D_i + 1$ is nonnegative, so we ultimately get an upper bound of $\frac{1}{2} \sum_i L(i)$.

*Proof.*

$$\text{Cost of } OPT \text{ in phase } i \geq max\{L_i - D_i, D_i + 1\}. \tag{3}$$

Let's say we just entered phase $i$. In this phase, there will be $L_i$ clean requests. For $Mark$, all clean page requests will end up being page faults. Since there exists $L_i$ clean requests that are not in $Mark$'s cache at the beginning of the phase, it follows that there exists $\geq L_i - D_i$ requests that are not in $OPT$'s cache at the beginning of the phase.
$D_{i+1}$ is the number of pages in $OPT$'s cache at the end of phase $i$, which are not in $Mark$'s cache. Everything requested in phase $i$ is in Mark's cache, which is $k$ different pages. These $k$ pages also had to be serviced by $OPT$ but $OPT$ had to service an additional $D_{i+1}$ that were left over somewhere. Therefore, $OPT$ had $\geq D_{i+1}$ page faults. □

**Claim 2.5.** $\mathbb{E}[cost(Mark(\sigma)] \leq H_k \sum_i L_i$

*Proof.* Suppose that we have seen $c$ clean requests and $s$ stale requests so far in this phase. At the beginning of the phase, our cache had $k$ stale pages. As we serviced $s$ stale requests, $s$ pages are definitely in our cache and have been marked. However, the remaining $k - s$ stale pages might not

be in our cache, as $c$ of them have been clobbered by clean requests. Then, the probability that a new stale request is not in our cache is $\leq \frac{c}{k-s}$

$$\begin{aligned}
\mathbb{E}(cost(Mark(\sigma)) \text{ in phase i} &\leq L(i) + \sum_{s=0}^{k-L(i)-1} \frac{c}{k-s} \\
&= L(i) + \sum_{j=L(i)+1}^{k} \frac{c_j}{j} \\
&\leq L(i) + L_i \cdot H_k - L_i \\
&= L_i \cdot H_k
\end{aligned}$$

□

**Claim 2.6.** Randomized marking algorithm $Mark$ is $2H_k$-competitive.

*Proof.* Earlier, we have proven that

$$\mathbb{E}[cost(Mark(\sigma)] \leq H_k \sum_i L_i \tag{4}$$

and

$$OPT \geq \frac{1}{2} \sum_i L(i) \tag{5}$$

Then by combining the two,

$$\mathbb{E}[cost(Mark(\sigma)] \leq 2H_k \cdot OPT \tag{6}$$

□

**Claim 2.7.** It is impossible to beat $H_k$-competitive.

*Proof.* At every point in time, request a uniform random page amongst pages {1, ..., k+1}. For any algorithm $A$, $\mathbb{E}(cost(A(\sigma))) = \frac{m}{k+1}$ for $m$ requests. The Coupon Collector problem implies that

$$OPT \leq \frac{m}{k \cdot H_k} \tag{7}$$

□

**Next time:** Online Primal-dual. Similar to LP duality to get a good competitive ratio.

# References


[1] Daniel Sleator, Robert Tarjan. Amortized Efficiency of List Update and Paging Rules. *ACM*, 28(2):202–208, 1985.

[2] Lyle McGeoch, Robert Tarjan. A Strongly Competitive Randomized Paging Algorithm. *Algorithmica*, 6(6):816–825, 1991.