Lecture 21: A Randomized Online Algorithm for Paging\textsuperscript{1}

1 Brief Review

Definition 1.0.1 (Randomized Online Algorithm) A randomized online algorithm $A$ is a probability distribution over deterministic online algorithms $A_x$ where $x$ is the sequence of $A$’s coin tosses over the course of the algorithm. **The determinism is kept by arguing that all sets of randomly generated coin tosses are known to $A$.**

Definition 1.0.2 (Oblivious Adversary) An oblivious adversary knows $A$ but has no access to the coin tosses.

**Note:** (An adaptive adversary furthers this. It also knows $A$, but changes its strategy by its observations of the coin tosses.)

Definition 1.0.3 ($\alpha$-competitive for Randomized Algorithms) A randomized online algorithm $A$ is $\alpha$-competitive against an oblivious adversary if there exists a constant $c$ such that for all input sequences $\sigma$:

$$E[C_{A_x}(\sigma)] \leq \alpha C_{\text{min}}(\sigma) + c$$

where min is the optimal offline algorithm, and the expectation is over the set of possible coin toss sequences $x$.

\textsuperscript{1}These notes are partially based on lecture notes scribed by Jeff Tustin in 2002
2 Randomized Marking Algorithm

The randomized marking algorithm $M$ begins with all pages in memory marked. When there is a request for page $p$:

- If $p$ is in memory, return it.
- Otherwise:
  1. If all pages in memory are marked, then unmark them all
  2. Swap $p$ with a uniformly selected (based on coin toss) unmarked page in memory **(a page fault occurs)**
- Mark $p$ **(that is now in memory)**

2.1 Unmarking Requests

**Definition 2.1.1 (Unmarking Request)** For any sequence of input requests $\sigma$ to $M$, we say that a request $\sigma_i$ is an unmarking request if the following are true:

- When $\sigma_i$ arrives, all pages are marked
- $\sigma_i$ is a request for a page that is not in memory

In other words, at some point all pages in memory are marked and if a new page is requested that is not in memory, all pages are unmarked.

2.2 Phases

$\sigma$ can be divided into phases as follows:

**Phase 0:** Everything up to and including the first unmarking request.

**Phase $i$:** The substring from the $i^{th}$ unmarking request to the $i + 1^{st}$
Phase $n$: The last unmarking phase

Let $S_i$ be the set of all pages in memory before phase $i$ begins.

1. A phase ends when $k$ pages are marked. At the start, just after the unmarking step, no pages are marked. Therefore, $k$ distinct pages are marked in a phase. **(note that $k$ is also the total number of pages that can be marked in memory)**

2. A page is marked once it is accessed in a phase. Hence, that page stays in memory until the end of the phase (i.e. we pay for page $q$ at most once in that phase).

The cost of $\sigma$ is equal to the total cost of all phases.

**Claim 2.2.1** Regardless of the coin tosses, a phase is always $k$ distinct page requests.

**Discussion:** The definition of a phase is independent of the coin tosses of $M!$ In other words, a phase ends when exactly $k$ distinct page accesses have been made. $S_i$ also does not depend on the coin tosses since it is the $k$ distinct page accesses in phase $i - 1$. Focus on the $k$ distinct page accesses in phase $i$. For each page, count only the 1st time it is accessed (only time it causes a page fault).

**Definition 2.2.2 (Clean Page Request) A request for a page that does not belong to $S_i$.**

All algorithms incur a cost for clean requests. **(i.e., this request has never been asked for, and must be faulted to memory)**

**Definition 2.2.3 (Dirty Page Request) A request for a page that does belong to $S_i$. **(i.e., this request was previously asked for, but may or may not still be in memory)**
**Question 1:** So what is the expected cost of a dirty request? $\sigma_j$ is a dirty request to page $p$ at time $j$ within phase $i$. Assume there have been $s$ dirty + $c$ clean requests so far in this phase. $p$ is accessed for the 1$\text{st}$ time which implies $p$ belongs to $U$, the set of pages at time $j$ from $S_i$ that have not been marked in phase $i$.

**Question 2:** What is the probability that $p$ is still in memory after $c$ clean + $s$ dirty requests? $p$ is one of $|U| = k - s - c$ pages in $L_i$ where $L_i$ is the set of $k - s$ pages in $S_i$ that have not been requested so far in phase $i$.

All pages in $L_i$ are equally likely to be in memory with probability $1 - f$, and thrown out with probability $f$. Hence,

$$k - s - c = |U| = E[|U|] = \sum_{q \in L_i} (1 - f) = (1 - f)(k - s)$$

Thus,

$$f = \frac{c}{k - s}$$

Therefore the expected cost of the $S + 1$st dirty request is $\frac{c}{k - s}$

If there are $l_i$ clean requests total in phase $i$, there are $k - l_i$ dirty requests. The number of clean requests preceding any dirty request is less than $l_i$. Any dirty request is certainly less than $l_i$, if there are at most $l_i$ clean requests total. Thus, we can bound the expected cost of all dirty requests in a phase by

$$D = \frac{l_i}{k} \frac{l_i}{k - 1} + \cdots + \frac{l_i}{k - (k - l_i - 1)}$$
Which leads us to the expected cost of dirty plus clean requests:

\[
\begin{align*}
\leq & \quad D + l_i = l_i \left(1 + \frac{1}{k} + \frac{1}{k-1} + \frac{1}{k-2} + \cdots + \frac{1}{k-(k-l_i-1)}\right) \\
\leq & \quad l_i H_k
\end{align*}
\]

Therefore, the expected cost of \( M \) on phase \( i \) of \( \sigma \) is \( \leq l_i H_k \)

3 Analysis

3.1 Comparison to Offline Algorithm

In this section we will compute how competitive \( M \) is to an offline algorithm. We begin by bounding the total cost of any offline algorithm \( A \) on a phase, and then simulate \( A \) and \( M \) on the same input sequence \( \sigma \).

A potential function \( \Phi_i \) is the number of pages in \( A \)'s memory that are not in \( M \)'s memory just before phase \( i \) begins. Note: Phases are not dependent on \( A \)'s coin tosses.

\( M \) receives \( l_i \) clean requests in phase \( i \). By the definition of clean, they are not in \( M \)'s memory at the start of phase \( i \). At least \( l_i - \Phi_i \) of these pages are also not in \( A \)'s memory at the start of phase \( i \). Thus \( C_i(A) \geq l_i - \Phi_i \)

Lemma 3.1.1 \( C_i(A) \geq l_i - \Phi_i \)

By definition, \( A \) has \( \Phi_{i+1} \) pages at the end of phase \( i \) that are not in \( M \)'s memory. \( M \) has a set \( P_i \) of \( \Phi_{i+1} \) pages at the end of phase \( i \) that are not in \( A \)'s memory. However, each page in \( M \)'s memory at the end of phase \( i \) was accessed in phase \( i \). Thus all pages in \( P_i \) must have been in \( A \)'s memory sometime in phase \( i \), but were rejected.

Lemma 3.1.2 \( C_i(A) \geq \Phi_{i+1} \)
By combining lemmas 1 and 2:

\[
2C_i(A) \geq \Phi_{i+1} + l_i - \Phi_i \\
C_i(A) \geq 0.5(l_i + \Phi_{i+1} - \Phi_i) \\
C_i(A) \geq 0.5(\sum l_i)
\]

Therefore, \( M \) is \( 2H_k \)-competitive (like \( 2 \log k \)).

*Final remarks:* Randomizing defangs the adversary!