COMP 7720 - Online Algorithms

Caching (Paging) Problem

Shahin Kamali

University of Manitoba
Problem Definition

- There are two types of memory: a fast ‘cache’ of size $k$, and a slow memory of unbounded size.
  - The input is an online sequence of requests to pages of size 1.
- To serve a request to page $x$, it should be in the cache.
  - In case $x$ is not in the cache, a fault of cost 1 happens.
  - In case $x$ is in the cache, a hit of cost 0 happens.
  - The goal is to minimize the total number of faults.
- To bring $x$ to the cache, we might need to evict a page.
  - A caching algorithm is defined through its eviction policy.

Cost (number of faults): 5

$$\sigma = a \ b \ c \ b \ a \ d \ c \ e$$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>
Least-Recently-Used (LRU)

- LRU algorithm: if eviction is necessary, evict the least recently used item.

Cost (number of faults):

$$\sigma = a \ b \ c \ b \ a \ d \ c \ e \ f \ a$$

\[ \begin{array}{cccc}
  a & b & c & d \\
  a & e & c & d \\
  f & e & c & d \\
\end{array} \quad \begin{array}{cccc}
  a & e & c & d \\
  f & e & c & d \\
  f & e & c & a \\
\end{array} \]
First-In-First-Out (FIFO)

- FIFO algorithm: if eviction is necessary, evict the oldest page in the cache (the one that came earlier).

Cost (number of faults): 7

\[ \sigma = a\ b\ c\ b\ a\ d\ c\ e\ f\ a \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>a</td>
<td>d</td>
</tr>
</tbody>
</table>
Flash-When-Full (FWF)

- FWF algorithm: if eviction is necessary, evict all pages in the cache (flash).

Cost (number of faults): 7

\[ \sigma = a \ b \ c \ b \ a \ d \ c \ e \ f \ a \]

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>c</th>
<th>d</th>
<th></th>
<th>e</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td></td>
<td></td>
<td>e</td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td></td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An Offline Algorithm

- Furthest-In-Future: Evict the page whose next request is furthest in the future among all pages in the cache.

Cost (number of faults): 6

\[ \sigma = a \, b \, c \, b \, a \, d \, c \, e \, f \, a \, c \, d \, c \, f \, a \, b \, a \, e \]

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>e</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>f</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>f</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>
```
Optimal Caching Algorithm

Theorem

Furthest-In-Future (FIF) is the optimal offline algorithm for Caching.

- Idea: we can modify any optimal algorithm $O_{FF}$ to work similar to FIF without increasing its cost.
- Assume on an access to $z$, $O_{FF}$ evicts $y$ while $x$ is furthest in future.
- Change $O_{FF}$ so that instead of $y$, $x$ is evicted.
  - We skip the details; a case analysis is required.
# Caching Algorithms & Competitive Ratio

## Theorem

For a cache of size $k$, no deterministic caching algorithm can have a competitive ratio better than $k$.

- Consider any online algorithm $A$
- Create an adversarial sequence of length $n$ on $k + 1$ pages so that $A$ faults on every single request.
  - At any time, at least one page is outside of the cache $\rightarrow$ adversary asks for that!
  - The cost of $A$ will be $n$.

- After the first $k$ “cold” misses, if FIF misses at one request, it hits in the next $k - 1$ requests.
  - Assume FIF evicts page $x$ for a request to $z$; so all $k + 1$ pages except $x$ are in the cache.
  - The next fault happens on a request to $x$.
  - But we know all $k - 1$ pages (all pages in the cache except $x$) have been requested before the next request to $x$.
  - In FIF, for each fault, there are at least $k - 1$ hits.
  - The cost of FIF is $k + \frac{n - k}{k} = \frac{n}{k} + O(k)$.

On an adversarial sequence of length $n$ on $k + 1$ pages:

- $A$ has a cost of $n$
- FIF has a cost of at most $\frac{n}{k} + O(k)$

The ratio between the cost of $A$ and FIF is at least $k$, assuming $k \in o(n)$. 

| COMP 7720 - Online Algorithms | Caching (Paging) Problem | 8 / 31 |
**Competitive Ratio of LRU**

**Theorem**

*LRU has a competitive ratio of at most* $k$.

- Use a **phase partitioning** technique.
- Define a phase as a sequence $\sigma_i, \sigma_{i+1}, \ldots, \sigma_{i+m}$ so that requests in this range involve $k$ different pages
  - The next request $\sigma_{i+m+1}$ is different from all these $k$ requests.

- What is the cost of LRU **per phase**?
  - $k$ different pages; LRU incurs at most $k$ faults

- What is the cost of OPT **per phase**?
  - Each phase + next item has $k+1$ distinct pages
  - OPT has to pay a cost of 1 per phase!

- The ratio between LRU and OPT is at most $k$ **per phase**

$$c.r.(LRU) = \frac{LRU(phase_1) + \ldots + LRU(phase_N)}{OPT(phase_1) + \ldots + OPT(phase_N)} \leq \max_i \frac{LRU(phase_i)}{OPT(phase_i)} \leq k$$
Other Algorithms with C.R. \( k \)?

- In the proof, we just used the fact that LRU has a cost of at most \( k \) for each phase.
  - For any subsequence formed by requests to \( k \) pages, LRU incurs a cost of at most \( k \)
- Can we extend this proof to other algorithms?
A marking algorithm maintains a bit (‘mark’) for each page in the cache.

- Start with all pages unmarked.
- Upon a hit, mark the page (if it is not already marked).
- Upon a fault, if eviction is required, evict an unmarked page.
  - If all pages in the cache are marked, all of them are unmarked first!
- When an item is brought to the cache it is marked

\[
\sigma = a\ b\ c\ b\ e\ f\ d\ a
\]

<table>
<thead>
<tr>
<th></th>
<th>f</th>
<th>a</th>
<th>e</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Marking Family of Algorithms (cntd.)

Theorem

Any deterministic marking algorithms $M$ has competitive ratio $k$.

- What is the cost of $M$ per phase?
  - It starts the phase with all pages unmarked
  - At the end of the phase, all $k$ pages of the phase are marked
  - On the first request to $x$, it becomes marked
    - $x$ remains in the cache until the end of the phase
    - $M$ incurs a cost of 1 for $x$ throughout the phase
  - **For each phase, $M$ incurs a cost of at most $k$**
  - Recall that $\text{OPT}$ has to pay a cost of 1 per phase!

$$\sigma = \underbrace{a b c b a d c}_{\text{phase 1}} \underbrace{e f a c}_{\text{phase 2}} \underbrace{d c d f a b a e \ldots}_{\text{phase 3}}$$

$k = 4$
Marking Algorithms Remarks

- LRU and Flash-When-Full are marking algorithms
  - They have competitive ratio $k$
- FIFO is Not a marking algorithm
  - Yet, it has a competitive ratio of $k$. 

COMP 7720 - Online Algorithms  Caching (Paging) Problem
Randomized Paging Algorithms

- Random Algorithm: in case an eviction is necessary, evict a page selected uniformly at random.
- Random has a competitive ratio of $k$
- Is it good?
MARK Algorithm

- MARK Algorithm is a randomized marking algorithm.
- In case an eviction is necessary, evict an unmarked page selected uniformly at random from all unmarked pages.
  - If all pages are marked, unmark all of them.

\[ \sigma = a\ b\ c\ b\ e\ f\ d\ a\ c\ e\ b \]

| f | c | a | e✓ |
Theorem

MARK has a competitive ratio of at most $2H_k$

- $H_k$ is the $k$'th harmonic number

$$H_k = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k}$$

- For any $k$, we have $\ln k < H_k \leq 1 + \ln k$.
  - So $H_k \in \Theta(\log k)$

- No randomized algorithm can have a competitive ratio better than $H_k$
There are two types of memory: a fast ‘cache’ of size $k$, and a slow memory of unbounded size.

- The input is an online sequence of requests to pages of size 1.
- To serve a request to page $x$, it should be in the cache.
  - In case $x$ is not in the cache, a fault of cost 1 happens.
  - In case $x$ is in the cache, a hit of cost 0 happens.
  - The goal is to minimize the total number of faults.
- To bring $x$ to the cache, we might need to evict a page.
  - A caching algorithm is defined through its eviction policy.
Caching Problem: A Review

- Latest-In-Future (LIF) is the optimal offline algorithm.
- No deterministic algorithm has a competitive ratio better than $k$.
- An algorithm is marking if it maintains a ‘mark’ for each page.
  - After a request to $x$ mark it.
  - Always evict an unmarked page (if all marked, first unmark all pages and then evict one)
- Any deterministic marking algorithm has a competitive ratio of $k$.
  - Least-Recently-Used (LRU), and Flash-When-Full (FWF) both have competitive ratio $k$.
- Fist-In-First-Out (FIFO) also has a competitive ratio of $k$. 
Caching Problem: A Review

- A randomized algorithm which randomly evict a page has a competitive ratio of $k$.

- A marking algorithm that evicts an unmarked page uniformly at random has a competitive ratio of $H_k$
  
  - $H_k = 1 + 1/2 + 1/3 + \ldots + 1/k$
  
  - For large values of $k$, we have $H_k \approx \ln(k) \in \Theta(\log k)$.

- In fact, no randomized algorithm can achieve a better competitive ratio (i.e., $o(\log k)$)
**Question:** How many bits of advice are sufficient to achieve an optimal algorithm?

- $n$: the length of input sequence (number of requests).
- $k$: the size of the cache

An algorithm has to make at most $n$ decisions about the page to be evicted.

- One decision per fault to indicate which one of the $k$ pages in the cache should be evicted.

- At most $O(n \log k)$ bits are sufficient!
Caching & Advice

What does the advice encode?

What is the size of advice? Assume Opt makes \( m \leq n \) faults for the optimal algorithm. For each fault, the advice indicates which page should be evicted. There are \( k \) pages in the cache, and the evicted page can be indicated in \( \Theta(\log k) \) bits. The total number of bits will be \( m \cdot O(\log k) \in O(n \log k) \).

How the algorithm works, provided by these bits of advice? It just mimics Opt; whenever there is a fault, it reads the advice to see which page should be evicted.

Why the algorithm has a competitive ratio of 1 (optimal here)? It works exactly like Opt.
Theorem

There is an online algorithm that, provided with \( O(n \log k) \) bits of advice, can achieve an optimal solution.

- It is a naive solution :'-)
- Can we achieve an optimal solution with a smaller number of bits of advice?
  - For many problems, the answer is no!
  - For caching problem, we can indeed do better.
Assume $\text{Opt}$ brings a page $x$ to the cache at time $t$.

- Either $\text{Opt}$ evicts $x$ before the next access to $x \rightarrow x$ is mortal.
- $\text{Opt}$ keeps $x$ in the cache until the next access to $x \rightarrow x$ is resident.
Caching & Advice (cntd.)

Assume $OPT$ brings a page $x$ to the cache at time $t$.

- Either $OPT$ evict $x$ before the next access to it $\rightarrow x$ is mortal.
- $OPT$ keeps $x$ in the cache until the next access to $x$ $\rightarrow x$ is resident

$$\sigma = a \ b \ c \ b \ a \ d \ c \ e \ f \ a \ c \ d \ c \ f \ a \ b \ a \ e$$

$a, c$ are residents    $b$ is mortal

\[
\begin{array}{ccc}
  a & \quad & d & \quad & c \\
\end{array}
\]
Caching & Advice (cntd.)

- If \( \text{OPT} \) has a hit for request \( x \) \( \Rightarrow \) \( x \) has been resident in cache since its last access to \( x \).

- If \( \text{OPT} \) has a fault for request \( x \) \( \Rightarrow \) either it is the first access to \( x \) or \( x \) has been mortal after its previous access (so that it is evicted at some point).

- Consider an algorithm \( \text{ResMor} \) that evicts a mortal page if an eviction is required.
  - \( \text{ResMor} \) always has the same resident pages as \( \text{OPT} \) in its cache
  - The mortal pages might be different.

- \( \text{OPT} \) and \( \text{ResMor} \) have the same cost
  - Assume \( \text{OPT} \) has smaller cost \( \Rightarrow \) there is a request to \( x \) that is a hit by \( \text{OPT} \) and a miss for \( \text{ResMor} \) \( \Rightarrow \) \( x \) is resident in \( \text{OPT} \) and not in \( \text{ResMor} \) \( \Rightarrow \) they maintain different resident pages (\( \text{ResMore} \) has evicted a resident page at some point) \( \Rightarrow \) contradiction
Caching & Advice (cntd.)

- *ResMor* is an optimal algorithm that, instead of the whole sequence, only needs to know which pages are resident/mortal at each given time.

- Assume with each request, there is one bit of advice that indicates whether the requested page is resident or mortal after the request.

- We can think of *ResMor* as an online algorithm with $n$ bits of advice.
What does the advice encode?

What is the size of advice?

How the algorithm works, provided by these bits of advice?

Why the algorithm has a competitive ratio of 1 (optimal here)? It maintains the same resident pages as Opt; so in case of a hit by Opt there will be a hit by the algorithm.
Advice Complexity of Paging

- With $n$ bits of advice, one can achieve an optimal algorithm.
- With roughly $\log\left(\frac{r+1}{r} \cdot \frac{r}{r+1}\right) \cdot n$ bits, one can achieve a competitive ratio of $r$.
  - With roughly $0.27n$ bits, one can achieve a competitive ratio of 2.
  - With roughly $0.24n$ bits, one can achieve a competitive ratio of 3.
- For a potential project, do a survey on advice complexity of paging, and try to deduce new results!
Belady’s Anomaly

- Naturally, we expect that having more pages results in less faults.
- In some caching algorithms, the number of page-faults might increase when the number of available pages increases.
  - This is called Belady’s anomaly
- FIFO suffers from Belady’s anomaly

\[ \sigma = a \ b \ c \ d \ a \ b \ e \ a \ b \ c \ d \ e \]
Belady’s Anomaly

- Naturally, we expect that having more pages results in less faults.
- In some caching algorithms, the number of page-faults might increase when the number of available pages increases.
  - This is called Belady’s anomaly
- FIFO suffers from Belady’s anomaly

Assume \( k = 4 \). FIFO Cost is: 10

Assume \( k = 3 \).

FIFO Cost is: 9

<table>
<thead>
<tr>
<th>d</th>
<th>e</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
</table>
Anomaly’s Summary

- We see more anomalies in analysis of online algorithms
- Project topic: make a survey on animality of different caching algorithms
  - Do some experiments, try to find anomaly examples by running algorithms on random inputs!