Review & Plan
Today’s objectives

- Caching Problem
  - Optimal offline algorithm
  - Lower bound for deterministic algorithms
  - Marking algorithms & upper bounds
  - Randomized algorithms
  - Caching anomalies
Caching Problem
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): \( \sigma = a \ b \ c \ b \ a \ d \ c \ e \)

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

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

Cost (number of faults):

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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>a</th>
<th>e</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>e</td>
<td>c</td>
<td>d</td>
<td>f</td>
<td>e</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
<td>c</td>
<td>d</td>
<td>f</td>
<td>e</td>
<td>c</td>
<td>a</td>
</tr>
</tbody>
</table>
**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

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

\[
\begin{array}{cccc}
  a & b & c & d \\
  e & b & c & d \\
  e & f & c & d \\
  e & f & a & d \\
\end{array}
\]

\[
\begin{array}{cccc}
  e & b & c & d \\
  e & f & c & d \\
  e & f & a & d \\
\end{array}
\]
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>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<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></td>
<td></td>
<td></td>
<td>a</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></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>e</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>f</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>f</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>
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.

- The cost of $A$ will be $n$.

For any such sequence, 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 potentially $z$) have been request before the next request to $x$.
- In FIF, for each fault, there are at least $k - 1$ hits.
Competitiveness of Deterministic Algorithms

- So, no deterministic algorithm can be better than $k$-competitive.
  - No algorithm is ‘competitive’ in the sense that the competitive ratio depends on the input.
- Yet, a competitive ratio of $k$ is much better than a ratio that depends on $n$.
  - Why?
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**

$$c.r.(LRU) = \frac{LRU(\text{phase } 1) + \ldots + LRU(\text{phase } N)}{OPT(\text{phase } 1) + \ldots + OPT(\text{phase } N)} \leq \max_i \frac{LRU(\text{phase } i)}{OPT(\text{phase } i)} \leq k$$
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.
- 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!

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

<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>
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!

$$\sigma = \underbrace{a \ b \ c \ b \ a \ d \ c}_{\text{phase1}} \ \underbrace{e \ f \ a \ c}_{\text{phase2}} \ \underbrace{d \ c \ d \ f \ a \ b \ a \ e \ \ldots}_{\text{phase3}} \quad k = 4$$
Marking Algorithms & LRU

Theorem

**LRU is a marking algorithm**

- Assume LRU is not marking
  - So, it evicts a marked page $x$ at some phase for a request to $y$
    - Both $x$ and $y$ are among $k$ pages that define the phase
  - In order to evict $x$, it should be least-recently used, i.e., there should be $k - 1$ pages requested after $x$ and before $y$.
    - Adding $x$ and $y$, there will be $k + 1$ pages in the phase → contradiction
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 $k$. 
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✓ |
Competitive ratio of MARK

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 $H_k$
No paging algorithm can have a competitive ratio better than $k$
  - LRU, FIFI, and FWF all have the optimal competitive ratio of $k$

No randomized algorithm can have a competitive ratio better than $H_k \in \Theta(\log k)$.
  - MARK has the optimal competitive ratio of $H_k$. 
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

\[
\begin{array}{cccc}
d & e & b & c \\
\end{array}
\]
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!