Online algorithms

Perform some sequence of operations \( \sigma_1, \sigma_2, \ldots, \sigma_m \).
When performing \( \sigma_i \), the future operations \( \sigma_{i+1}, \ldots, \sigma_m \) are unknown.
- Very broad topic
- Most data structures can be seen this way

Example

**Online paging problem**

![Diagram of cache memory and external memory](image)

Typically, \( k \ll N \)

Sequence of requests

\( \Gamma = \{ \gamma_1, \gamma_2, \ldots, \gamma_m \} \)

For each request \( \gamma_i \):
- If page \( \gamma_i \) is in cache) "cache hit"
  
  Cost = 0, nothing to do

- If page \( \gamma_i \) is not in cache) "cache miss/sFault"

  Move \( \gamma_i \) into cache
  
  Cost = 1

If cache memory is full, need to evict one page from cache.

**Question:** How to decide which page to evict from cache?

- Caching/paging strategy

(Ignore other bookkeeping cost and computation)
\[ K = 3 \]
\[ N = 100 \]

requests:
- \( a \) 1
- \( b \) 1
- \( a \) 0
- \( c \) 1
- \( d \) 1 (evict \( b \))
- \( b \) 1

(\text{In hindsight, \( b \) was not a good decision})

Possible strategies:
1. LRU (evict least recently used)
2. LFU (evict least frequently used)
3. FIFO (first in first out \( \Rightarrow \) evict first inserted)
4. LIFO (last in first out \( \Rightarrow \) evict last inserted)
5. Random (evict a page chosen uniformly at random)

6. LFD (evict page with largest forward distance)

(\text{Offline algorithm})

\[ \text{e.g. } K = 3 \]

\[ \text{LRU} \]

requests: \( \{ a, b, c, d, a, b, c, a, b, c \} \)

\[ \text{evicted: } a, b, c \]

\[ \text{cost of LRU: } \mathbb{Z} \]

\[ \text{Each miss cost: } 1 \]
LFD rejects a b c d a b d a a b c
evidently c

How good are these strategies? Which strategy is best?
How to analyze/compare different strategies?

regret sequence: \( R = r_1, r_2, \ldots, r_m \)
\( K < N \)

Worst-case analysis:
- for an arbitrary algorithm, \( \text{cost} = m \)
- (adversary requests a page not in the cache)
- \( \text{even opt. cost} \leq m \)

Competitive analysis of online algorithms

Let \( \text{Alg}(R) \) denote cost of \( \text{Alg} \) on sequence \( R \).

Let \( \text{OPT}(R) \) denote the optimal cost of serving sequence \( R \).

Knewly entire \( R \) in advance,
Refine computation

Assume:
Initial state of algorithm: cache is full

Def. "competitive ratio" of \( \text{Alg} \) is \( \leq \), if \( \text{Alg}(R) \leq \leq \cdot \text{OPT}(R) + o(1) \)
(\( \leq \) is the maximum ratio between \( \text{Alg}. \) cost and optimum cost)
Theorem. \( \text{LFD}(R) = \text{OPT}(R) \) for all \( R \).

Proof:

- Let \( R_1, \ldots, R_m \) be the requests.
- \( \text{OPT} \to \text{optimal strategy} \)
- \( \text{LFD} \to \text{strategy of LFD algorithm} \)

Proof strategy: We transform \( \text{OPT} \) step-by-step, until it matches \( \text{LFD} \). The transformation will not increase cost.

Suppose \( \text{OPT} \) and \( \text{LFD} \) do the same on \( R_{i-1}, R_i \), do different on \( R_i \).

\[ \text{OPT} \]
\[ R_1 \ R_2 \ R_3 \ldots \ R_i \]
\[ \text{LFD} \]
\[ \text{do nothing} \]

**Case A.** \( R_{i+1} \) is a cache hit (for both algorithms)

- **Case A1.** \( \text{OPT} \) keeps \( p' \) in cache until it is requested.
  - \( \text{OPT} \to \text{OPT}' \)
  - \( p \to p' \) at time \( j \)
  - \( \text{OPT} \) at time \( i+1 \) do nothing

  \( \text{OPT} \) and \( \text{OPT}' \) have same cost.

  - \( \text{OPT}' \) agrees with \( \text{LFD} \) on \( R_1, \ldots, R_{i+1} \)

- **Case A2.** \( \text{OPT} \) replaces \( p' \) with \( g \) before \( j \) (at time \( j' < j \))
  - \( \text{OPT} \to \text{OPT}' \)
  - \( p \to g \) at time \( j' \)
  - \( \text{OPT}' \) at time \( j' \) do nothing

  \( \text{OPT}' \) is valid and cost smaller than cost of \( \text{OPT} \). Contradiction.
Case B: request $r_{i+1}$ is a cache miss

$OPT: P_2 \leftarrow P$

$LED: P_2 \leftarrow P$

Suppose $r_{i+1}$ requests some page $p$

$OPT$ first loads $P_1$ at time $j$ ($X \leftarrow P_1$)

$CON B1$

$OPT$ keeps $P_2$ until time $j$

$OPT \rightarrow OPT'$

at time $i+1$ do $P_2 \leftarrow P$

at time $j$ do $X \leftarrow P_2$

$OPT'$ valid, same cost as $OPT$

$OPT'$ agrees with $LED$ on $r_{i+1}$, $r_{i+1}$ repeats

Summary: $LED$ is optimal.
Proof

- Suppose \( C_i \) is a cache miss

  - load \( r_i \)

  - decide which page to evict next \( k+1 \) requests

    - future requests are \( r_i, r_{i+1}, r_{i+2}, \ldots, r_{i+k+1} \)

      a) if all these \( k+1 \) requests are different,

        then load \( r_{i+k+1} \).

        \( \Rightarrow \) next \( k \) requests will have 0 miss.

      b) if next \( k+1 \) requests have some duplicates

        \( \Rightarrow \) at most \( k \) different requests away then

        \( \Rightarrow \) keep all these \( k \) pages in cache

        \( \Rightarrow \) next \( k+1 \) requests will have 0 miss

For every \( \text{cost} \) \( k \) repeat, we have \( > k \) consecutive \( \text{cost} 0 \) repeats

\( \Rightarrow \) Total cost \( \leq \frac{m}{k} \)

\( \frac{m}{k} \text{ miss, hit, \ldots, hit, miss, hit, \ldots, hit, miss} \)

\( > k \)

\( > k \)

**Theorem B**

\( N = k+1 \)

Let \( \text{Alg} \) be an arbitrary deterministic online paging algorithm.

There is a sequence \( R = r_1, \ldots, r_m \) s.t. \( \text{Alg}(R) > m \).

Proof

Cache size is \( k \), Adversary requests exactly the missing page.

Here we use that \( \text{Alg} \) is deterministic so adversary knows which page is missing

\( \text{Theorem A} \) \( \Rightarrow \) For every deterministic online paging algorithm

\( \text{competitive ratio} \geq k \).

Lower bound on competitive ratio
**Thm.** (LRU) is k-competitive. at a cache miss, evict the page that was not requested for the longest time.

**Claim.** \( \text{LRU}(R) \leq k \cdot \text{OPT}(R) \)

**Proof**

Split the sequence \( R \) into phases \( P_0, P_1, \ldots \). In each phase LRU has exactly k misses (except for last phase).

In each phase LRU cost = k.

Need to show that OPT also makes at least one miss in each phase.

**True in \( P_0 \):** (LRU, OPT start from same initial state)

Look at phase \( P_i \):

- \( t, t+1, t+2, \ldots, t+k \)
- \( \text{Page} \) \( p \)

**Lemma:** Phase \( P_i \) contains requests to \( k \) different pages (other than \( p \)).

**Observe:** After time \( t+1 \), OPT has \( p \) in the cache.

**Lemma:** at least one of the \( k \) different pages requested in \( P_i \) (other than \( p \)) is currently not in the cache.

\( \Rightarrow \) OPT will make a miss.

**Proof of Lemma**

**Case A:** LRU makes k misses on \( k \) different pages (other than \( p \))

\( \Rightarrow \) Done

**Case B:** LRU makes two misses on same page \( q \neq p \)
- \( \text{LRU misses for } q \), where LRU misses
- \( q \) must be evicted between \( \text{Pic} \), \( \text{Piz} \)
- Why did LRU evict \( q \) at time \( j^* \)? (\( j^* \in (i, i2) \))
  \[ \Rightarrow \text{because } \# \text{ other pages were requested in time } \text{interval } (i, j^* ] \]
  \[ \text{ (other than } q \text{) } \]
  \[ \Rightarrow k+1 \text{ diff. requests in time } [i, j^* ] \] (including \( q \))
  \[ \Rightarrow k \text{ diff. requests (other than } p \text{) in this interval. } \]

**Case C.**

LRU makes a miss on \( p \)

After time \( t-1 \), \( p \) is in cache
At time \( j \), \( p \) is a miss
  \[ \Rightarrow \text{at some time } j^* \in (t-1, j) \text{ } \]
  \[ \Rightarrow p \text{ was evicted. } \]
  Why was \( p \) evicted?
  \[ \Rightarrow \text{there were } k \text{ diff. pages (other than } p \text{) that were } \text{ “more recent” } \]
  \[ \Rightarrow \text{these were requested in interval } [t, j^* ] \]
Thin. FIFO is $K$-competitive (proof similar)
This suggests FIFO and LRU similar.
But in practice LRU better than FIFO.

Thin. LFU/LIFO are not $K$-competitive
(or even $f(K)$-competitive for any function $f(.)$)

Proof (LIFO) lets page that was last recently added
Repeated $m$ times

$R = 1, 2, \ldots, K, K+1, (K,K+1)^m$ (length of $R$ is $2m + K + 1$)

$m$ can be arbitrarily large

$$\text{OPT cost} = K + 1$$

$$\text{LIFO cost} = 2m + K + 1$$

We cannot bound $\frac{\text{LIFO}(R)}{\text{OPT}(R)} = \frac{2m + K + 1}{K + 1}$

$\Rightarrow$ LIFO is not competitive.

Randomized algorithm

Thus, RANDOM is $K$-competitive

Proof (one side)

RANDOM is not better than $K$-competitive

$R = (K+1), 1, 2, \ldots, K, (1, 2, \ldots, K)^*$

OPT makes one miss, evict $K + 1$

\[ \text{OPT}(R) = 1 \]
How many misses by \textsc{Random} for \( K \)?

Obs. After \((k+1)\) is evicted, algorithm makes no more miss.

\# misses until \( k+1 \) evicted:

When make a miss, \( \Pr(K+1 \text{ is evicted}) = \frac{1}{K} \)

\( \mathbb{E}[\# \text{misses}] = \frac{1}{K} \text{ pr. tail} \)

\( \mathbb{E}[\# \text{misses}] = \frac{1}{K} \text{ pr. head} \)

Expected cost: \textsc{Random}(\( R \)) = \( K \)

\[ \Rightarrow \frac{\text{Random}}{\text{opt}} \geq K \]

Can we go beyond \( k \)-competitiveness? Answer: \text{YES}

\( O(\log k) \) - competitive paging algorithm.

(Randomized)

Marking algorithm \( \rightarrow O(\log k) \) competitive

- Each \( k \) locations in cache are marked \( 0/1 \)
- Initially all are marked \( 0 \)
- Cache hit \( \Rightarrow \) mark location \( 1 \)
- Cache miss \( \Rightarrow \) load requested page, mark \( 1 \)

Evict randomly chosen page away those that are marked \( 0 \)
- Once all \( k \) locations are marked \( 1 \), we reset all to \( 0 \)

Hybrid between \textsc{Random} lRU
Thus, no randomized Alg. can have competitive ratio $o(\log K)$.