Online algorithms

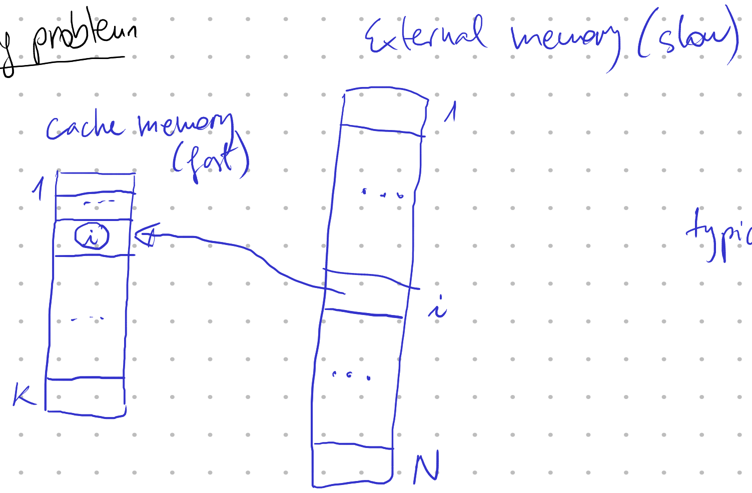
Perform some sequence of operations $op_1, op_2, \dots, op_m$

When performing op_i, the future operations $op_{i+1}, \dots, op_m$ are unknown.

- very broad topic
- most data structures can be seen this way

Example

Online paging problem



typically $k \ll N$

Sequence of requests

$r_1, r_2, \dots, r_m \in \{1, \dots, N\}$

for each request r_i, - if page r_i is in cache, "cache hit", cost = 0, nothing to do

- if page r_i is not in cache, "cache miss / fault" move r_i into cache cost = 1

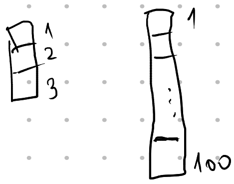
If cache memory is full, need to evict one page from cache.

Question: how to decide which page to evict from cache?

→ caching / paging strategy.

(Ignore other bookkeeping cost and computation)

e.g. $k=3$
$N=100$



		cost	
request(a)	a	1	
request(b)	a, b	1	
request(a)	a, b	0	
request(c)	a, b, c	1	
request(d)	a, d, c	1	(evict b)
request(b)	a, d, b	1	

(In hindsight evicting 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 → evict first inserted)
4. LIFO (last in first out → evict last inserted)
5. Random (evict a page chosen uniformly at random)

online algorithms

...

6. LFD (evict page with longest forward distance)

page which will be requested furthest in future
(if some page never needed in future, evict that one)

offline algorithm

e.g. $k=3$
LRU

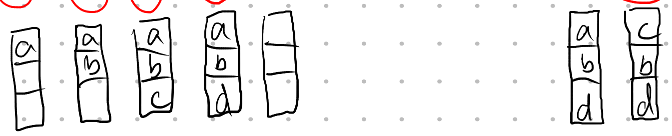
requests:	a	b	c	d	a	b	d	a	a	b	c
	a, –, –	a, b, –	a, b, c	d, b, c	d, a, c	d, a, b	d, a, b	d, a, b	d, a, b	d, a, b	c, a, b
evicted				a	b	c					d

⊛ each miss cost = 1

cost of seq. with LRU is 7.

LFD

requests a b c d a b d a a b c



evicted c a

cost of seq. with LFD is 5.

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

request sequence $R = r_1, r_2, \dots, r_m$

$K < N$

cache size, # pages

Worst-case analysis:

for an arbitrary algorithm cost $= m$

(adversary always requests a page not in the cache)

e.g. $k=3$
request: $1,2,3,4,5,6,7,\dots,m$
cost $= m$
even OPT cost is m

Competitive analysis of online algorithms

Let $Alg(R)$ denote cost of Alg on sequence R.

Let $OPT(R)$ denote the optimal cost of serving sequence R

knowing entire R in advance, infinite computation

"the price of not knowing the future"

Assume:
Initial state of algorithm: cache is full

Def. "competitive ratio" of Alg is c, if $Alg(R) \leq c \cdot OPT(R) + O(1)$

for all sequences R.

(think of c as the maximum ratio between Alg. cost and optimum cost)

Theorem. $LFD(R) = OPT(R)$ for all R.

Proof

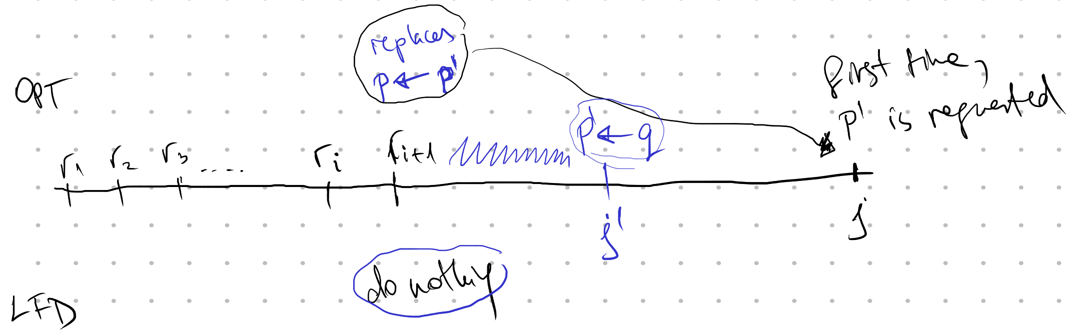
Requests $r_1, \dots, r_m$

OPT $\to$ optimal strategy

LFD $\to$ strategy of LFD algorithm

Proof strategy : We transform OPT step-by-step, until it matches LFD, the transformation will not increase cost.

Suppose that OPT and LFD do the same on $r_1, \dots, r_i$, do different on r_{i+1}



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

OPT replaces some $p \leftarrow p'$ at time $i+1$

LFD does nothing at time $i+1$

Case A_1 OPT keeps p' in cache until it is requested

OPT $\to$ OPT$'$

s.t. $p \leftarrow p'$ postponed to time j

at time $i+1$ do nothing

OPT and OPT$'$ have same cost

but OPT$'$ agrees with LFD on $r_1, \dots, r_{i+1}$

Case A_2 OPT replaces p' with q before time j (at time $j' < j$)

OPT $\to$ OPT$'$

s.t. at time $i+1$ $p \leftarrow q$

at time j' do nothing

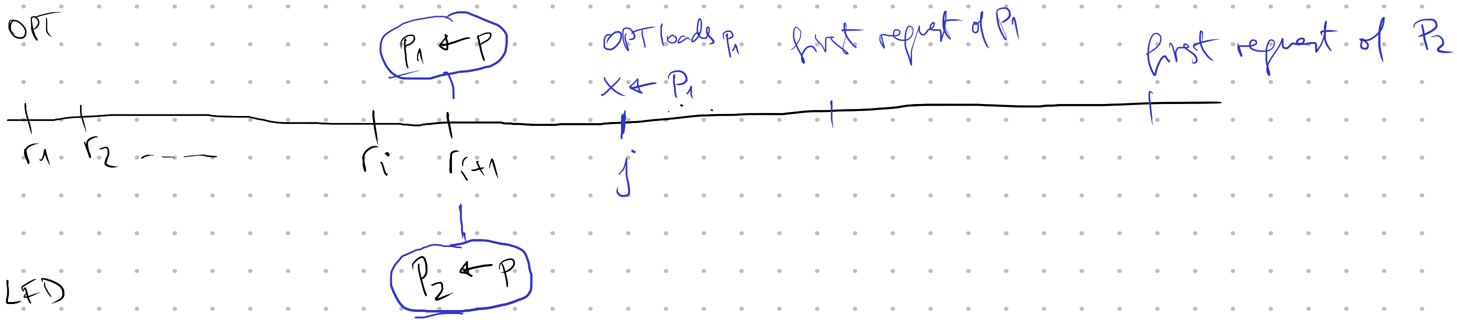
OPT$'$ is valid and cost smaller than cost of OPT

contradiction.

Case B: request r_{i+1} is a cache miss

OPT: $P_1 \leftarrow p$

LFD: $P_2 \leftarrow p$



Suppose r_{i+1} requests some page p

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

Case B1

OPT keeps P_2 until time j

$OPT \rightarrow OPT'$

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

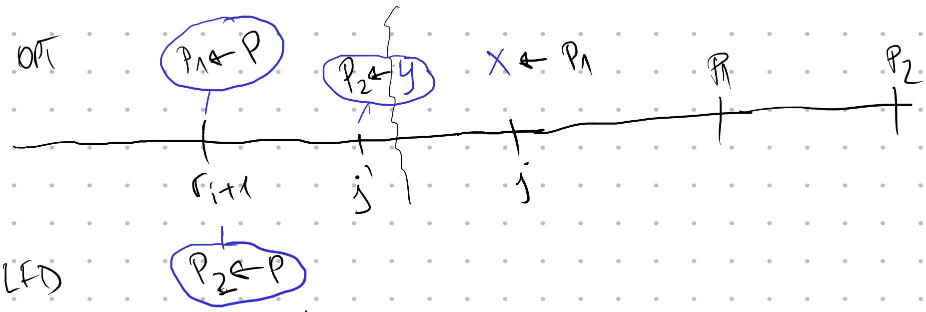
at time j do $X \leftarrow P_2$

OPT' valid, same cost as OPT,

OPT' agrees with LFD on $r_1, \ldots, r_{i+1}$ requests

Case B2

OPT evicts P_2 before time j (at time j')
$P_2 \leftarrow y$



$OPT \rightarrow OPT'$:

at time $i+1$
$P_2 \leftarrow p$

at time j'
$P_1 \leftarrow X$

OPT' valid, same cost as OPT

OPT' agrees with LFD on $r_1, \ldots, r_{i+1}$

Summary: LFD is optimal.

Thm A For $R = r_1, \dots, r_m$, "Impossibility result"

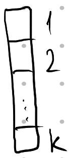
$\underline{N = k+1}$,

$OPT(R) \leq \frac{m}{k}$

Thm A. OPT has low cost on R
Thm B. Arbitrary alg. has high cost on R
$\Big\} \ \frac{Alg(R)}{OPT(R)}$ is large

enough to show for $N = k+1$

Proof



- Suppose $\underline{r_i}$ is a cache miss
- load r_i
- decide which page to evict

future requests are $\overbrace{r_{i+1}, r_{i+2}, \dots, r_{i+k+1}}^{\text{next } k+1 \text{ requests}}$

- a) if all these $k+1$ requests are different, then evict r_{i+k+1}.
  → Next k requests will have 0 miss.

b) if next $k+1$ requests have some duplicates
→ at most k different requests among them
→ keep all these k pages in cache
⇒ Next $k+1$ requests will have 0 miss

For every cost 1 request, we have $\geq k$ consecutive cost 0 requests.

$\Rightarrow$ Total cost $\leq \frac{m}{k}$

miss, $\underbrace{\text{hit}, \dots, \text{hit}}_{\geq k}$, miss, $\underbrace{\text{hit}, \dots, \text{hit}}_{\geq k}$, ...

Thm B $N = k+1$

Let Alg be an $\underline{\text{arbitrary}}$ deterministic online paging algorithm.

There is a sequence $R = r_1, \dots, r_m$ s.t. $Alg(R) = \underline{m}$.

Proof

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

Here we use that Alg is deterministic so adversary knows which page is missing

☒

Thm A
Thm B
$\Big\} \Rightarrow$ For every $\underline{\text{deterministic online}}$ paging algorithm competitive ratio $\geq k$.

[Lower bound on competitive ratio]

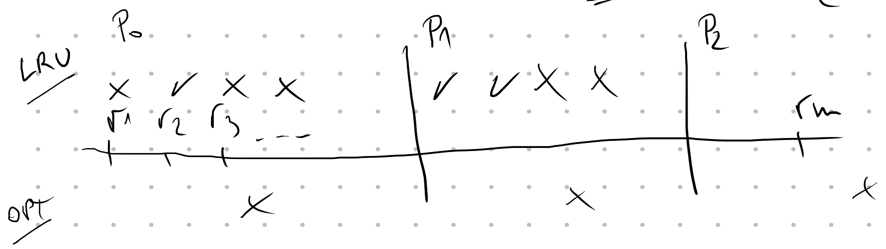
Thm. LRU is k-competitive. → at a cache miss, evict the page that was not requested for the longest time.

Arbitrary $R = r_1, \dots, r_m$

Claim. $LRU(R) \leq k \cdot OPT(R)$ → cache size.

Proof

Split the sequence R into phases $P_0, P_1, \dots$ s.t. in each phase LRU has exactly $\underline{k}$ 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: $r_{t-1} \quad \underbrace{r_t \quad r_{t+1} \quad \dots \quad r_{t+e}}_{P_i}$

r_{t-1} → 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

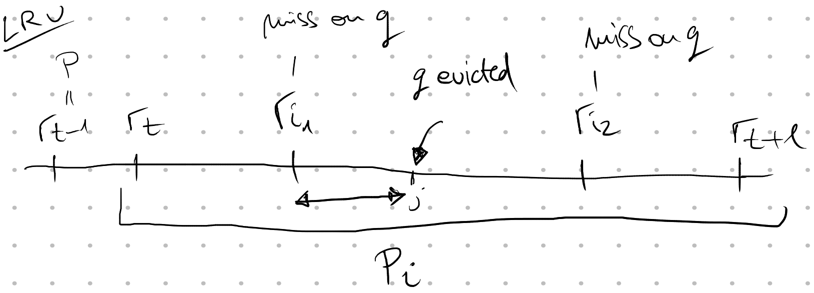
Lemma ⇒ at least one of the k different pages requested in P_i (other than p) is currently not in the cache
⇒ OPT will make a miss.

Proof of Lemma

Case A. If LRU makes k misses on k different pages (other than p)
⇒ Done

Case B LRU makes two misses on same page $q \neq p$

LRU



- r_{i_1}, r_{i_2} two requests for q, where LRU misses
- q must be evicted between r_{i_1}, r_{i_2}
- Why did LRU evict q at time j? $(j \in (i_1, i_2))$

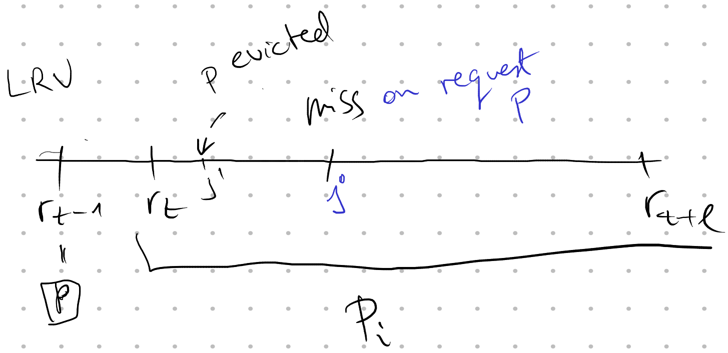
$\Rightarrow$ because k other pages (other than q) were requested in time interval $(i_1, j]$ (these are "more recent" than q)

$\Rightarrow k+1$ diff. requests in time $[i_1, j]$ (including q)

$\Rightarrow k$ diff. requests (other than p) 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

$\Rightarrow$ at some time $j' \in (t-1, j)$ p was evicted.

Why was p evicted?

$\Rightarrow$ there were k diff. pages (other than p) that were "more recent" than p.

$\rightarrow$ these were requested in interval $[t, j']$

$\boxed{}$

Thm. FIFO is k-competitive (proof similar)

This suggests FIFO and LRU similar.

But in practice LRU better than FIFO.

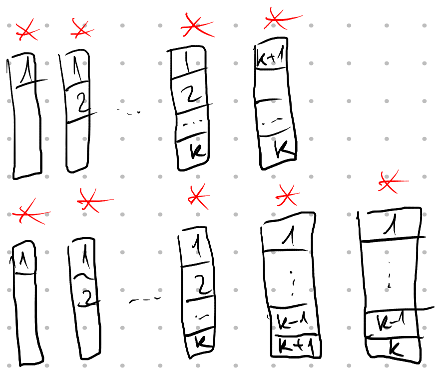
Thm. LFU / LIFO are not k-competitive
(or even $f(k)$-competitive for any function $f(\cdot)$)

Proof (LIFO) — evicts page that was most recently added

$$R = 1, 2, \dots, k, k+1, (k, k+1)^m$$ ⇝ repeated m times

(length of R is $2m + k + 1$) ⇓ m can be arbitrarily large

OPT



LIFO

OPT cost $= k+1$

LIFO cost $= 2m + k + 1$

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

$\Rightarrow$ LIFO is not competitive.

Randomized algorithms
- in expectation w.r. random choices
- adversary knows algorithm, but not random choices. "Oblivious adversary"

Thm. RANDOM is k-competitive

Proof (one side)

RANDOM is not better than k-competitive

$$R = (k+1), 1, 2, \dots, k, (1, 2, \dots, k)^* \qquad OPT(R) = 1$$

OPT makes one miss
evict $k+1$

$(k+1), 1, 2, \ldots k-1, \mid k, 1, 2, \ldots, k, \ldots$

no misses

How many misses by RANDOM for R?

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

misses until $k+1$ evicted:

when make a miss, $\Pr(k+1 \text{ is evicted}) = \frac{1}{k}$ (each page in cache equally likely to be evicted)

$E[\#\text{ misses}] = $ Coinflips $\frac{1}{k}$ pr. tail, $1-\frac{1}{k}$ pr. head | expected # coinflips until first tail

$= k$

Expected cost RANDOM $(R) = k$

$\Rightarrow \frac{\text{RANDOM}}{\text{OPT}} \geq k$

Can we go beyond k-competitiveness? Answer: YES

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

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

hybrid between RANDOM / LRU

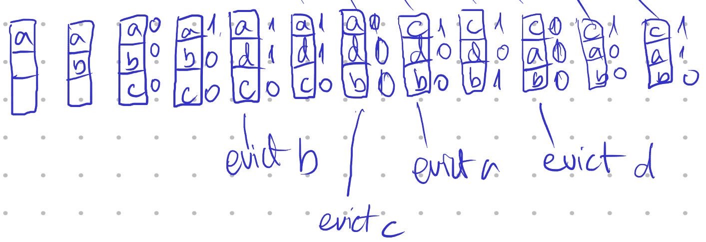
- Each k locations in cache are marked 0/1
- Initially all are marked 0
- Cache hit $\Rightarrow$ mark location 1
- cache miss $\Rightarrow$ load requested page, mark 1

  evict randomly chosen page among those that are marked 0
- Once all k locations are marked 1, we reset all to 0

e.g.

$k=3$

R a b c a d a b c b a c a



Cost = 7

Thm. No randomized Alg can have competitive ratio $o(\log k)$.