Overview

- The paging problem
- Several algorithms
- Resource augmentation analysis
- Randomization
- Types of adversaries

(Absolute) competitive ratio

- Definition for minimization problems:

$$C_{ALG} = \sup_{\sigma} \frac{\mathrm{ALG}(\sigma)}{\mathrm{OPT}(\sigma)}$$

  (we look for the input that results in worst relative performance)

- For maximization problems:

$$C_{ALG} = \sup_{\sigma} \frac{\mathrm{OPT}(\sigma)}{\mathrm{ALG}(\sigma)}$$

- Goal:

  find ALG with minimal C_{ALG}

Paging

- Computers usually have a small amount of fast memory (cache)
- This can be used to store data (pages) that are often used
- Problem when the cache is full and a new page is requested
- Which page should be thrown out (evicted)?

Definitions

- k = size of cache (number of pages)
- We assume that access to the cache is free, since accessing main memory costs much more
- Thus, a cache hit costs 0 and a miss (fault) costs 1
- The goal is to minimize the number of page faults

Paging algorithms

- Last In First Out (LIFO): evict newest page
- First In First Out (FIFO): evict oldest page
- Least Frequently used (LFU): evict page that was requested least often
- Least Recently Used (LRU): evict page that was requested least recently
- Flush When Full (FWF): on a fault, evict all pages
- Longest Forward Distance (LFD): evict page that will be requested the latest

Longest Forward Distance is optimal

We show: any optimal offline algorithm can be changed to act like LFD without increasing the number of page faults.

Inductive claim: given an algorithm ALG, we can create ALG_i such that

- ALG and ALG_i are identical on the first $i-1$ requests
- If request i causes a fault, ALG_i evicts page with longest forward distance
- $\mathrm{ALG}_i(\sigma) \leq \mathrm{ALG}(\sigma)$

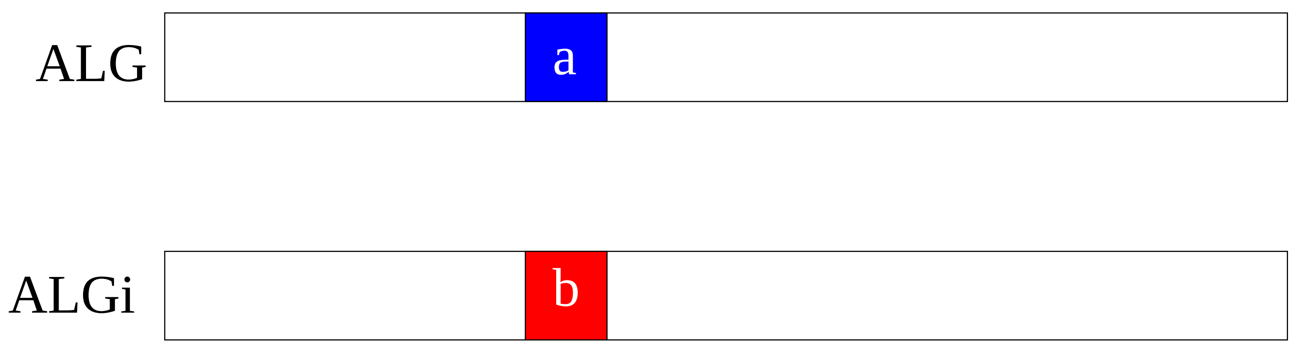
Using the claim

- Start with a given request sequence σ and an optimal offline algorithm ALG
- Use the claim for $i = 1$ on ALG to get ALG_1, which evicts the LFD page on the first request (if needed)
- Use the claim for $i = 2$ on ALG_1 to get ALG_2
- …
- Final algorithm ALG_n is equal to LFD

Proof of the claim

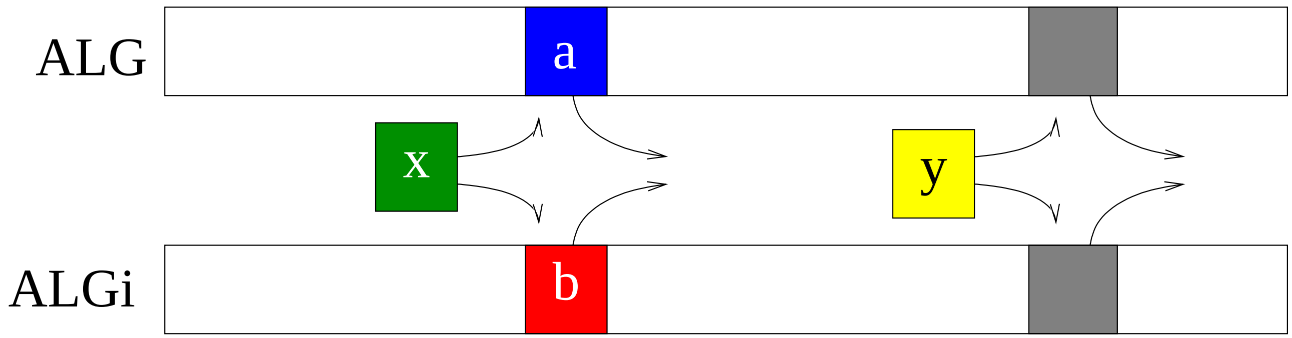
Suppose that after request i, ALG has page a while ALG_i has page $b \neq a$. Remaining pages are the same.

Until now, both algorithms have the same number of faults.



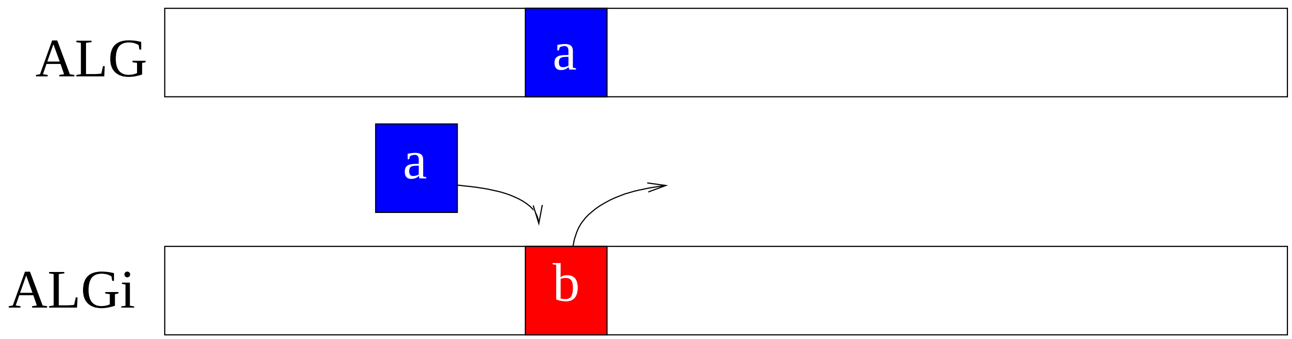
Proof of the claim

Until a is requested, ALG_i does the same as ALG, but it evicts b if ALG evicts a. Then both algorithms again have the same pages in the cache, and we are done.



Proof of the claim

If a is requested before ALG evicts a, ALG_i has a fault. But a was the LFD page, so before this ALG must have had a fault where ALG_i did not. ALG_i now evicts b and loads a.



Comparison of algorithms

- LFD is not online, since it looks forward
- Which is the best online algorithm?
- LIFO is *not* competitive: consider an input sequence

$$p_1, p_2, \dots, p_{k-1}, \underline{p_k, p_{k+1}, p_k, p_{k+1}, \cdots}$$

- LFU is also *not* competitive: consider

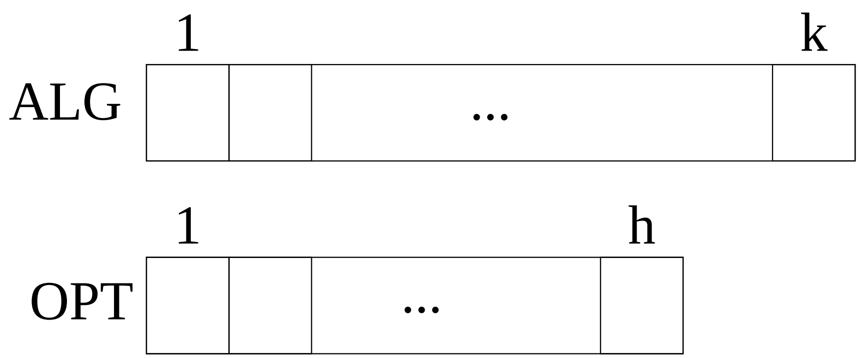
$$p_1^m, p_2^m, \dots, p_{k-1}^m, (p_k, p_{k+1})^{m-1}$$

A general lower bound

- To illustrate the problem, we show a lower bound for *any* online paging algorithm ALG
- There are $k+1$ pages
- At all times, ALG has k pages in its cache
- There is always one page missing: request this page at each step
- OPT only faults *once every k steps*
  $\Rightarrow$ lower bound of k on the competitive ratio

Resource augmentation

- We will compare an online algorithm ALG to an optimal offline algorithm *which has a smaller cache*
- We hope to get more realistic results in this way
- Size of offline cache = $h < k$
- This problem is known as (h, k)-paging



Conservative algorithms

- An algorithm is conservative if it has at most k page faults on any request sequence that contains at most k distinct pages
- The request sequence may be arbitrarily long
- LRU and FIFO are conservative
- LFU and LIFO are not conservative (recall that they are not competitive)

Competitive ratio

Theorem 1. *Any conservative algorithm is* $\frac{k}{k-h+1}$*-competitive*

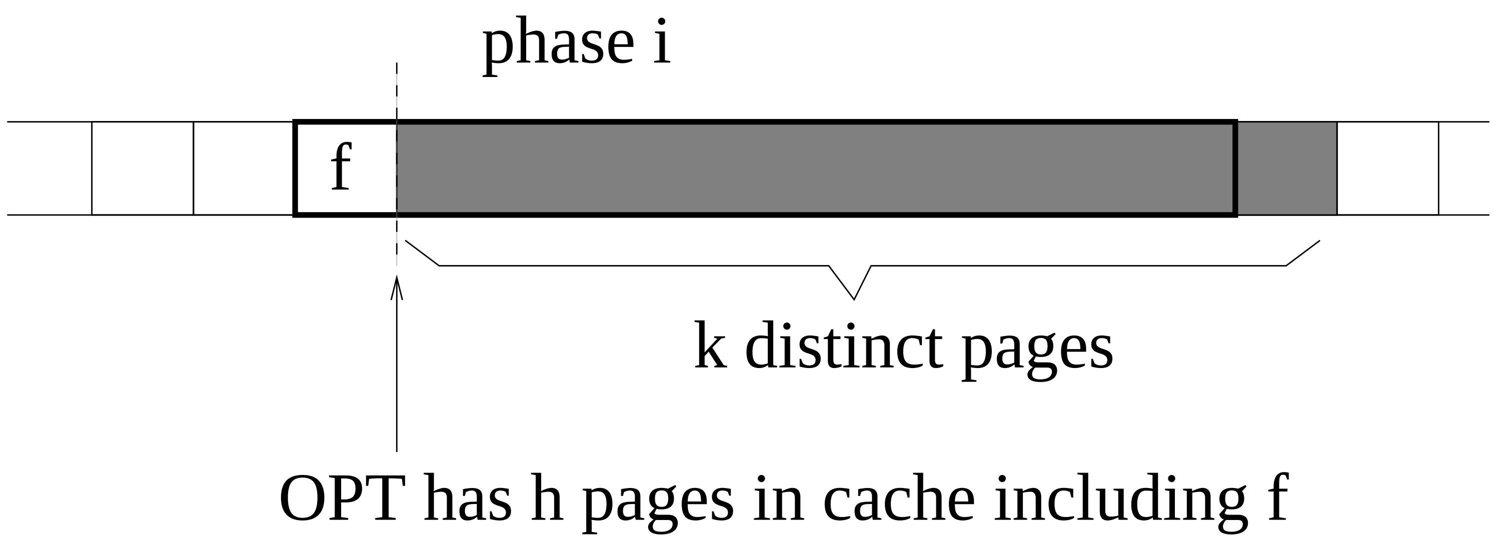
Proof: divide request sequence σ into **phases**.

- Phase 0 is the empty sequence
- Phase $i > 0$ is the maximal sequence following phase $i-1$ that contains at most k distinct pages

Phase partitioning does not depend on algorithm. A conservative algorithm has at most k faults per phase.

Counting the faults of OPT

Consider some phase $i > 0$, denote its first request by f



Thus OPT has at least $k-(h-1) = k-h+1$ faults on the grey requests

Conclusion

- In each phase, a conservative algorithm has k faults
- To each phase except the last one, we can assign (charge) $k-h+1$ faults of OPT
- Thus
$$\mathrm{ALG}(\sigma) \leq \frac{k}{k-h+1} \cdot \mathrm{OPT}(\sigma) + r$$
where $r \leq k$ is the number of page faults of ALG in the last phase
- This proves the theorem

Notes

- For $h = k/2$, we find that conservative algorithms are 2-competitive
- The previous lower bound construction does not work for $h < k$
- In practice, the “competitive ratio” of LRU is a small constant
- Resource augmentation can give better (more realistic) results than pure competitive analysis

Randomized algorithms

- Another way to avoid the lower bound of k for paging is to use a randomized algorithm
- Such an algorithm is allowed to use random bits in its decision making
- Crucial is what the adversary knows about these random bits

Three types of adversaries

- Oblivious: knows only the probability distribution that ALG uses, determines input in advance
- Adaptive online: knows random choices made so far, bases input on these choices
- Adaptive offline: knows random choices in advance (!)

Randomization does not help against adaptive offline adversary

We focus on the oblivious adversary

The MARK Algorithm

- This algorithm marks pages which are requested
- It never evicts a marked page
- When all pages are marked and there is a fault, it unmarks everything (but marks the page which caused the fault)
- Eviction strategy: evict randomly and uniformly chosen page from the set of all unmarked pages
- LRU and FWF are also marking algorithms
- Only difference is in eviction strategies

Competitive ratio of MARK

- Consider the harmonic numbers H_k $(k = 1, \ldots)$

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

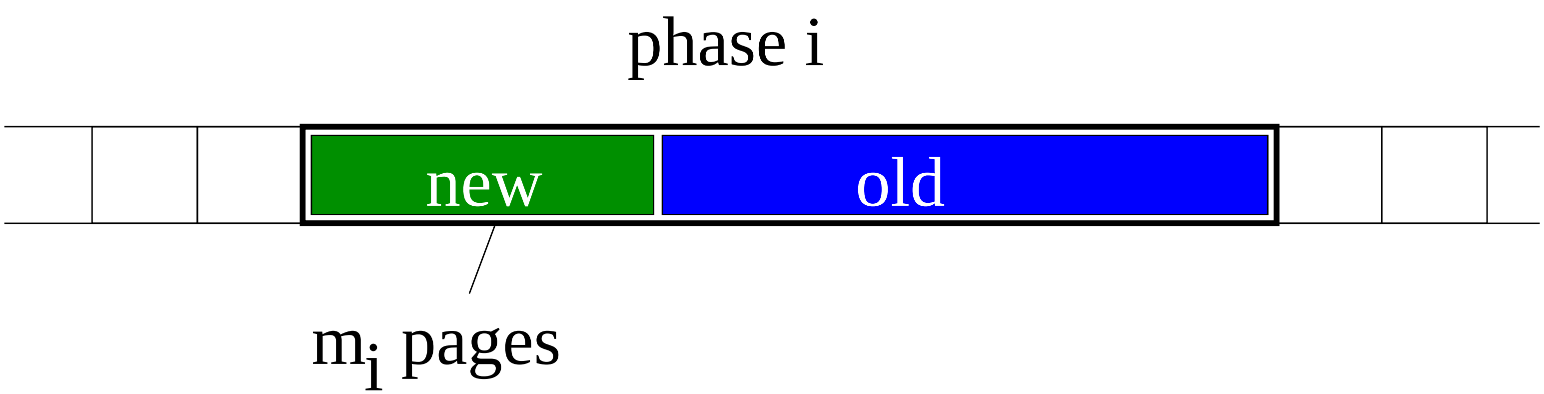
- We have $\ln k < H_k \leq 1 + \ln k$
- We show that MARK is $2H_k$-competitive

Analysis of MARK (1)

- Consider the phase partitioning of an input σ (does not depend on algorithm!)
- Pages in cache at start of phase i are old
- Non-old pages requested in phase i are new
- Let m_i be the number of new pages requested in phase i
- What is the worst order of new pages vs. old pages?

Analysis of MARK (2)

- Worst case is that the new pages come first in a phase
- This means m_i page faults on those pages
- How many faults are there on the $k - m_i$ old pages?



Analysis of MARK (3)

- The jth old page is in the cache at the moment it is first requested with probability

$$\frac{k - m_i - (j-1)}{k-(j-1)} \; .$$

- Explanation:
  - $k - m_i - (j-1) =$ number of old unmarked pages in the cache
  - $k - (j-1) =$ total number of old unmarked pages

Analysis of MARK (4)

- So, the jth old page causes a fault with probability

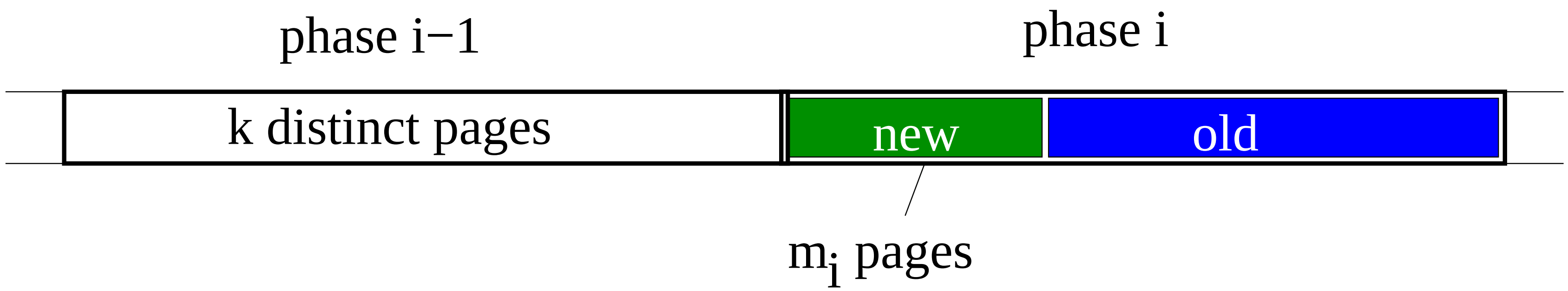
$$1 - \frac{k - m_i - (j-1)}{k - (j-1)} = \frac{m_i}{k - j + 1}.$$

- Expected number of faults is

$$\begin{aligned} m_i + \sum_{j=1}^{k-m_i} \frac{m_i}{k-j+1} &= m_i + m_i(H_k - H_{m_i}) \\ &= m_i(H_k - H_{m_i} + 1) \leq m_i H_k \end{aligned}$$

- Now we still need a lower bound for OPT

Lower bound for OPT



- There are m_i new pages in phase i
- Thus, in phases $i-1$ and i together, $k+m_i$ pages are requested
- OPT makes at least m_i faults in phases i and $i-1$ for any i
- Total number of OPT faults is at least $\frac{1}{2}\sum_i m_i$

Upper bound for MARK

- Expected number of faults in phase i is at most $m_i H_k$ for MARK
- Total expected number of faults is at most $H_k \sum_i m_i$
- OPT has at least $\frac{1}{2}\sum_i m_i$ faults
- Conclusion: MARK is $2H_k$-competitive

Discussion

- The upper bound for MARK holds against an oblivious adversary (the input sequence is fixed in advance)
- Question: is it possible to improve MARK?
- We show that no algorithm can be better than H_k-competitive
- Thus, MARK is optimal apart from a factor of 2
- Note that H_k is much smaller than k

Randomized lower bound

- Idea: use $k+1$ pages
- Keep track of probabilities p_j that page j is not in the cache
- Create the sequence based on these probabilities
- The adversary can do this because it knows the description of the algorithm, and creates the input sequence
- Construction uses phases
- In each phase, ALG will make H_k faults, OPT makes 1 fault

A phase in the lower bound

- Each phase consists of k subphases
- The **adversary** uses a marking algorithm to serve the sequence
- We make no assumptions about the online algorithm!
- At the start of subphase i, there will be $k-i+1$ unmarked pages
- Subphase 1: k unmarked pages (the page which caused the fault that ended the previous phase is marked)

Calculations

- The expected cost of ALG for subphase i will be $1/(k-i+1)$.
- Thus, the total cost for phase is

$$\sum_{i=1}^{k} \frac{1}{k-i+1} = H_k.$$

- Since OPT pays 1 per phase, this proves the lower bound (OPT has no cost as long as unmarked pages exist; this holds until the entire phase ends (with subphase k))

Construction of subphase j

- Each subphase contains
  - some (maybe 0) requests to marked pages
  - one request for an unmarked page (it is then marked!)
- Let M be the set of marked pages at the start of subphase j (so $|M| = j$)
- There are $u = k + 1 - j$ unmarked pages
- Consider $\gamma = \sum_{i \in M} p_i$ (note: p_i is probability that page i is not in the cache)

Subphase j: $\gamma = \sum_{i \in M} p_i$

- If $\gamma = 0$, there exists an unmarked page a with $p_a \geq 1/u$; request this page and end this subphase
- Else, there exists a marked page $m \in M$ with $p_m > 0$.
- Define $\varepsilon = p_m$ and start with a request for page m
- Repeatedly request marked pages as follows:

While (expected cost for ALG is less than $1/u$ and $\gamma > \varepsilon$) request marked page ℓ with maximal p_ℓ

Subphase j: the case $\gamma = \sum_{i \in M} p_i > 0$

While (expected cost for ALG is less than $1/u$ and $\gamma > \varepsilon$)
request marked page ℓ with maximal p_ℓ

- The expected cost of ALG increases in each step of this loop, so the loop terminates
- In fact, if $\gamma > \varepsilon$ then cost increases by at least $\gamma/|M| > \varepsilon/|M|$.

After the loop

While (expected cost for ALG is less than $1/u$ and $\gamma > \varepsilon$)
request marked page ℓ with maximal p_ℓ

- If expected cost for ALG is at least $1/u$, request an arbitrary unmarked page
- Else, $\gamma \leq \varepsilon$
- In this case, request unmarked page b with maximal p_b
- We have $p_b \geq (1-\gamma)/u$
- Cost for ALG is

$$p_m + p_b \geq \varepsilon + \frac{1-\gamma}{u} \geq \varepsilon + \frac{1-\varepsilon}{u} \geq \frac{1}{u}.$$

Result

- No algorithm ALG is better than H_k-competitive against an oblivious adversary
- Against stronger adversaries, this holds a forteriori
- There exists an H_k-competitive algorithm
- It is substantially more complicated than MARK
- Competitiveness for (h, k)-paging is still unknown