Online algs for caching (the paging problem)

C blocks of cache

$a_1, a_2, \ldots, a_n$ access sequence of block numbers

cache control alg. → cost T := # cache misses

↓ online

→ offline → OPT := cost of optimum offline alg.

We want: find online alg. s.t. $T \leq k \cdot OPT$ (k a constant) } alg. is k-competitive

Unfortunately not possible

→ we can force $T \geq \sim C \cdot OPT$.

LRU strategy

↑ least-recently used

We discard the block not accessed for the longest time.

Cheat: Give LRU a larger cache than OPT has → $C_{LRU} > C_{OPT}$.

Theorem (Sleator & Tarjan; weakened):

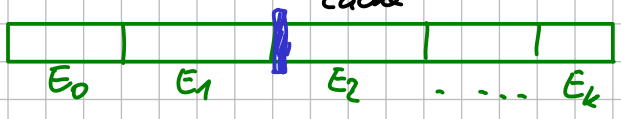
For every $C_{LRU} > C_{OPT} \geq 1$ and every access sequence:

$$T_{LRU} \leq \frac{C_{LRU}}{C_{LRU} - C_{OPT}} \cdot T_{OPT} + C_{OPT}$$

← for case OPT starts with a non-empty cache

for $C_{LRU} = 2C_{OPT}$: $T_{LRU} \leq 2 \cdot T_{OPT} + C_{OPT}$

Proof: Split access seq. to epochs:



Consider epoch E_i for $i > 0$:

For E_i: # misses of LRU = C_{LRU}.

Except E_0: # misses of LRU ≤ C_{LRU}

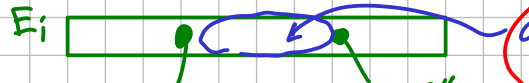
Case 1: All LRU's misses on different blocks

→ accesses to C_{LRU} diff. blocks

→ opt. alg. misses at least $C_{LRU} - C_{OPT}$ times.

in this epoch: $\frac{T_{LRU}}{T_{OPT}} \leq \frac{C_{LRU}}{C_{LRU} - C_{OPT}}$ (*)

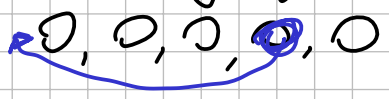
Case 2: LRU misses twice on the same block.



↳ b is at the head of the LRU's list

accesses to at least C_{LRU} different blocks

LRU keeps a list of cached blocks sorted by age



If we have $\frac{T_{LRU}}{T_{OPT}} \leq$ (*) in every E_i, it also holds over the whole seq. (except for E_0).

Return to E_0:

Case 1: The cache is empty at the start.

We have at most C_{LRU} diff. blocks accessed.

⇒ $T_{OPT} \geq T_{LRU}$. } ratio $\frac{T_{LRU}}{T_{OPT}} \leq 1 \leq$ (*)

Case 2: OPT starts with a non-empty cache...

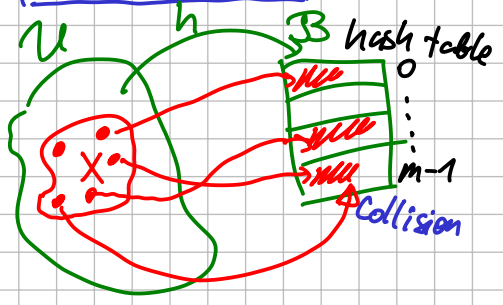
it can save up to C_{OPT} misses.

→ So the assumption of opt. strategy affects only constants.

Conclusion: For algs we studied: #reads $(N, B, M) \in O($#reads $(N, B, M/2))$

So: #reads with LRU, cache $\frac{M}{2} \in O($#reads with LRU for cache size $M) \in O($#reads with OPT, cache size $M/2)$

HASHING



Notation: $\mathcal{U}$ universe, $u := |\mathcal{U}|$

in many cases: $\mathcal{U} = \{0, \dots, u-1\}$, $[u]$

$\mathcal{B}$ set of buckets $= [m]$

h hash function $h: \mathcal{U} \to \mathcal{B}$] evaluated in const. time

$X \subseteq \mathcal{U}$ a finite set stored in the D.S.

$n := |X|$

Complexity: worst-case, amortized, average-case

cost of hashing ↑

↳ no order-based operations (Min, Succ, ...)

In every bucket, we have a linked list of items (a chain) → hashing with chaining → time per op. is $O(\text{chain length})$ → we hope for short chains on average

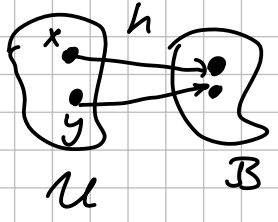
↳ worst-case $\Theta(n)$

$\frac{n}{m}$ ↑ we choose $m \sim n$

Def: A family $\mathcal{H}$ of functions from $\mathcal{U}$ to $\mathcal{B}$ is c-universal for $c > 0$ iff

$\forall x, y \in \mathcal{U}, x \neq y : \Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq c/m$.

for perfectly random h it's exactly $\frac{1}{m}$

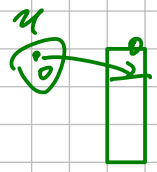


We usually want: ① $h \in \mathcal{H}$ can be picked randomly in $O(1)$ time

② $h(x)$ can be evaluated in $O(1)$ time

↳ solution: $\mathcal{H} := \{h_a \mid a \in A\}$ (a parameter), pick $a \in A$] it can happen that $h_a = h_{a'}$ → consider $\mathcal{H}$ a multi-set.

Def: $\mathcal{H}$ is universal iff $\exists c > 0$: $\mathcal{H}$ is c-universal.



Lemma: Given $x_1, \dots, x_n, y \in \mathcal{U}$ all distinct and $\mathcal{H}$ is c-universal:

$\mathbb{E}_{h \in \mathcal{H}}[\#_i : h(x_i) = h(y)] \leq \frac{cn}{m}$.

($\#_i$: $h(x_i) = h(y)$ is A)

Proof: Use indicators $A_1, \dots, A_n$: $A_i = \begin{cases} 0 \\ 1 & h(x_i) = h(y) \end{cases}$

$\mathbb{E}[A_i] = \cancel{0 \cdot \Pr[A_i = 0]} + 1 \cdot \Pr[A_i = 1]$

$\Pr[h(x_i) = h(y)] \leq c/m$

$\mathbb{E}[A_i] \leq c/m$.

$A = \sum_i A_i$ linearity of $\mathbb{E}$ $\mathbb{E}[A] = \sum_i \mathbb{E}[A_i] \leq \frac{cn}{m}$.

(n terms; $\mathbb{E}[A_i] \leq c/m$)

Always: $\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

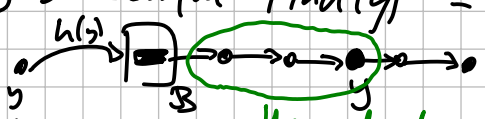
Use the lemma to analyze expected complexity of hashing with chaining with hash func. chosen u.a.r. from c-universal family.

$X = \{x_1, \dots, x_n\}$

① unsuccessful Find(y) → by Lemma $\mathbb{E}[\text{length of chain in } h(y)] \in O(n/m)$

② successful Insert(y) →

③ successful Find(y) ≤ succ. Insert



④ unsucc. Ins. → succ. Find] these already were there at the time of Insert

⑤ Delete → Find

if we maintain $m \in \Omega(n)$, this is $O(1)$.

use flexible arrays & rehash