CSE 421

Greedy Algorithms / Caching Problem

Yin Tat Lee

Last Lecture

- Greedy Analysis Strategies
  - Greedy algorithm stays ahead: Induction
  - Structural: Come up with a lower bound
  - Exchange argument: transform OPT to greedy
- Minimize “Maximum Lateness”
  - You should do homework according to deadline.
  - We prove it using exchange argument.
- Students asked how about minimize “Total Lateness”
  - I said I don’t know ☹. The answer is NP-hard. ☹ ☹
- I said that one can ignore the proof for termination for reduction proof
  - I forget there can be complicated reduction in this class.
  - So, please still include the proof.
  - Usually just 1 line like “BFS is linear time”

Optimal Caching/Paging

Memory systems

- Many levels of storage with different access times
- Smaller storage has shorter access time
- To access an item it must be brought to the lowest level of the memory system

Type	Latency	Capacity
Registers	0.25 ns	36 KB
L1 Cache	1 ns	192 KB
L2 Cache	3 ns	1.5 MB
L3 Cache	14 ns	15 MB
DRAM (DDR4)	66 ns	32 GB
SDD	0.15 ms	480 GB
Internet	7 ms	

My computer at home (2018)

Optimal Caching/Paging

Memory systems

- Many levels of storage with different access times
- Smaller storage has shorter access time
- To access an item it must be brought to the lowest level of the memory system

Consider the problem between 2 levels

- Main memory with n data items
- Cache can hold $k < n$ items
- Assume no restrictions about where items can be
- Suppose cache is full initially
  - Holds k data items to start with

Optimal Offline Caching

Caching

- Cache with capacity to store k items.
- Sequence of m item requests $d_1, d_2, \cdots, d_m$.
- Cache hit: item already in cache when requested.
- Cache miss: item not already in cache when requested: must bring requested item into cache, and evict some existing item, if full.

Goal

- Eviction schedule that minimizes number of evictions.

Example: $k = 2$, initial cache $= a, b$,
requests: a, b, c, b, c, a, a, b.

Optimal eviction schedule: 2 cache misses.

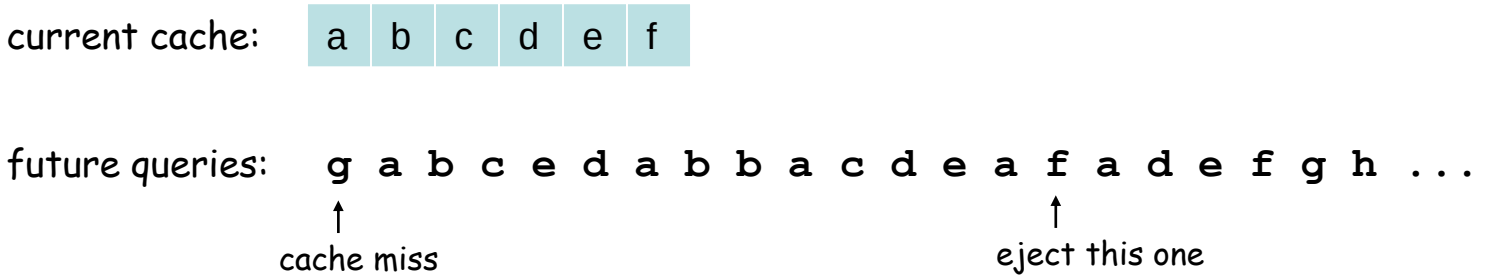
requests	cache	
a	a	b
b	a	b
c	c	b
b	c	b
c	c	b
a	a	b
a	a	b
b	a	b

Optimal Offline Caching: Farthest-In-Future

Which item we should evict?

Farthest-in-future

- Evict item in the cache that is not requested until farthest in the future.



Theorem

- [Bellady, 1960s] FIF is an optimal eviction schedule.

Exchange Argument

- We can swap choices to convert other schedules to Farthest-In-Future without losing quality

Reduced Eviction Schedules

Definition

- A reduced schedule is a schedule that only inserts an item into the cache in a step in which that item is requested.

Intuition

- Can transform an unreduced schedule into a reduced one with no more cache misses.

a	a	b	c
a	a	x	c
c	a	d	c
d	a	d	b
a	a	c	b
b	a	x	b
c	a	c	b
a	a	b	c
a	a	b	c

an unreduced schedule

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

a reduced schedule

Reduced Eviction Schedules

Claim

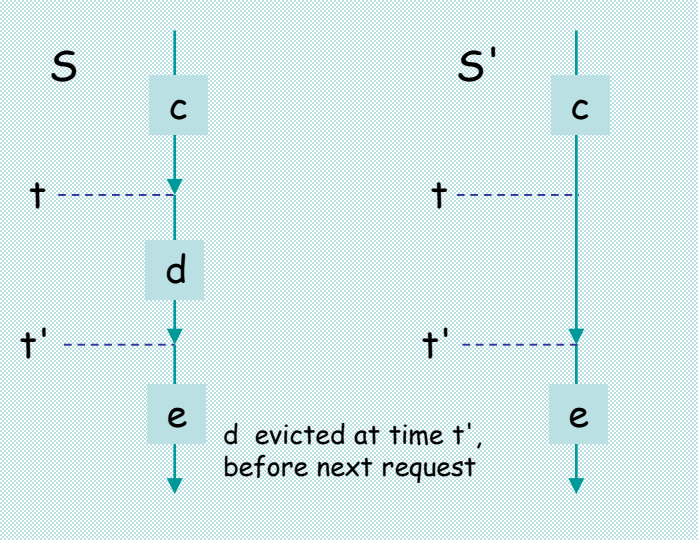
- Given any unreduced schedule S, can transform it into a reduced schedule S' with no more cache misses.

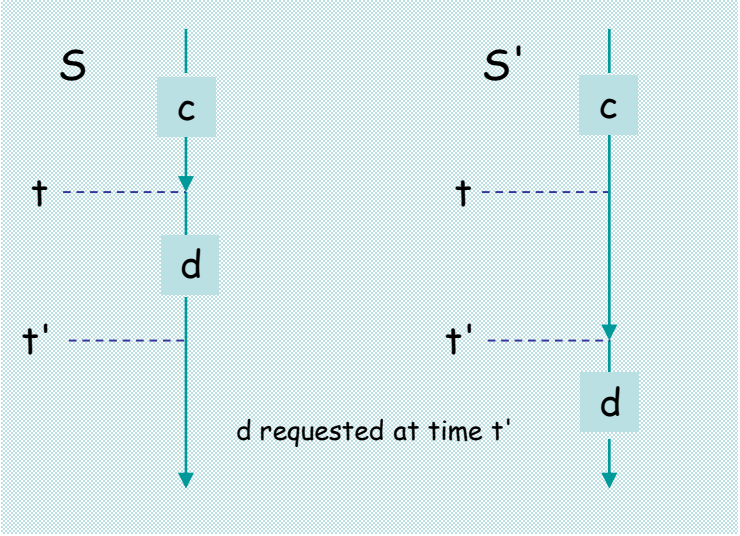
Proof (by induction on number of unreduced items)

- Suppose S brings d into the cache at time t, without a request.
- Let c be the item S evicts when it brings d into the cache.

  Case 1: d evicted at time t', before next request for d.

  Case 2: d requested at time t' before d is evicted. ▪





Farthest-In-Future: Analysis

Theorem

- FIF is optimal eviction algorithm.

Proof. (by induction on number or requests j)

Invariant: There exists an optimal reduced schedule S that makes the same eviction schedule as S_{FIF} through the first $j + 1$ requests.

Let S be reduced schedule that satisfies invariant through j requests. We produce S' that satisfies invariant after $j + 1$ requests.

- Consider $(j + 1)^{st}$ request $d = d_{j+1}$.
- Since S and S_{FIF} have agreed up until now, they have the same cache contents before request $j + 1$.

Case 1: (d is already in the cache).

$S' = S$ satisfies invariant. (used S is reduced here)

Case 2: (d is not in the cache and S and S_{FIF} evict the same element).

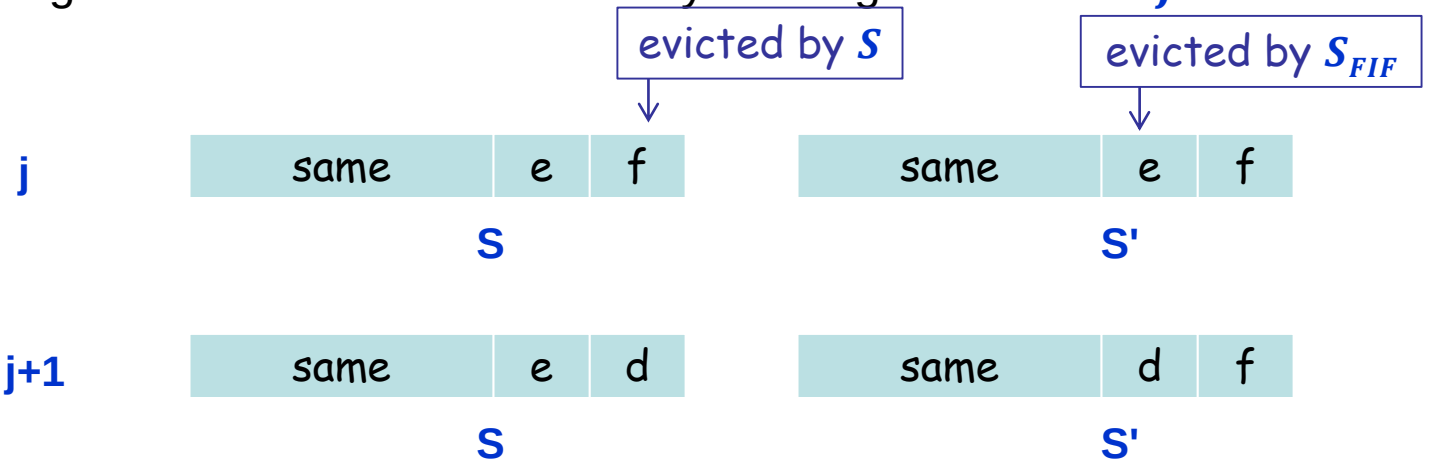
$S' = S$ satisfies invariant.

Farthest-In-Future: Analysis

Proof. (continued)

Case 3: (d is not in the cache; S_{FIF} evicts e; S evicts $f \neq e$).

- begin construction of S' from S by evicting e instead of f

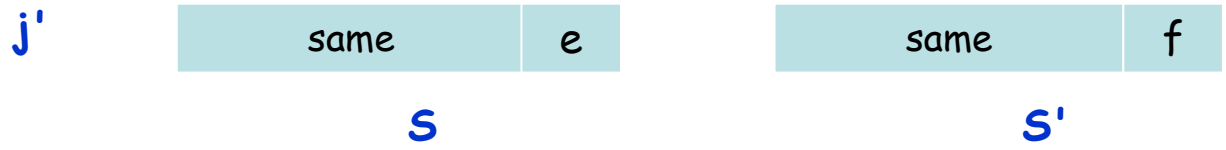


- now S' agrees with S_{FIF} on first $j + 1$ requests; we show that having element f in cache is no worse than having element e
  - Continue building S' to be the same as S until forced to be different

Farthest-In-Future: Analysis

Proof. (continued)

Let j' be the first time after $j + 1$ that S and S' must take a different action, and let g be item requested at time j'.



Case 3a: $g = e$.

Can't happen: e was evicted by Farthest-In-Future so there must be a request for f before e.

Case 3b: $g = f$.

Element f can't be in cache of S, so let e' be the element that S evicts.

- if $e' = e$, S' accesses f from cache; now S and S' have same cache
- if $e' \neq e$, S' evicts e' and brings e into the cache; now S and S' have the same cache

Note: S' is no longer reduced, but can be transformed into a reduced schedule that agrees with S_{FIF} through step $j + 1$

Farthest-In-Future: Analysis

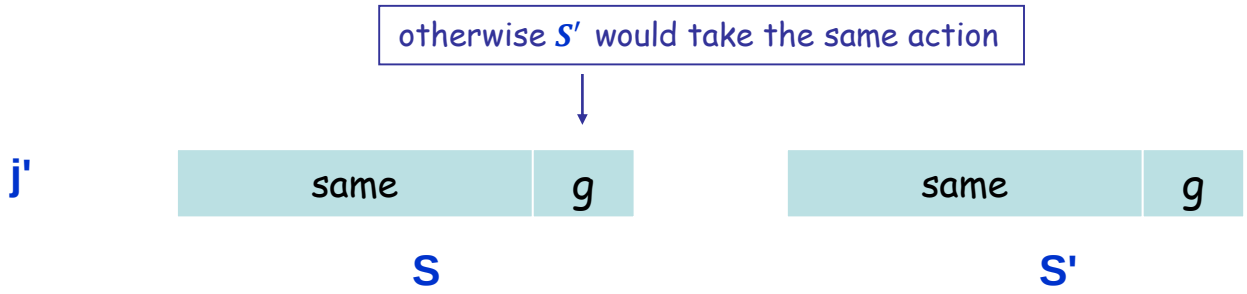
Proof. (continued)

Let j' be the first time after $j+1$ that S and S' must take a different action, and let g be item requested at time j'.

Case 3c: $g \neq e$ and $g \neq f$.

S must evict e.

Make S' evict f; now S and S' have the same cache. ▪



In each case can now extend S' using rest of S at no extra cost.

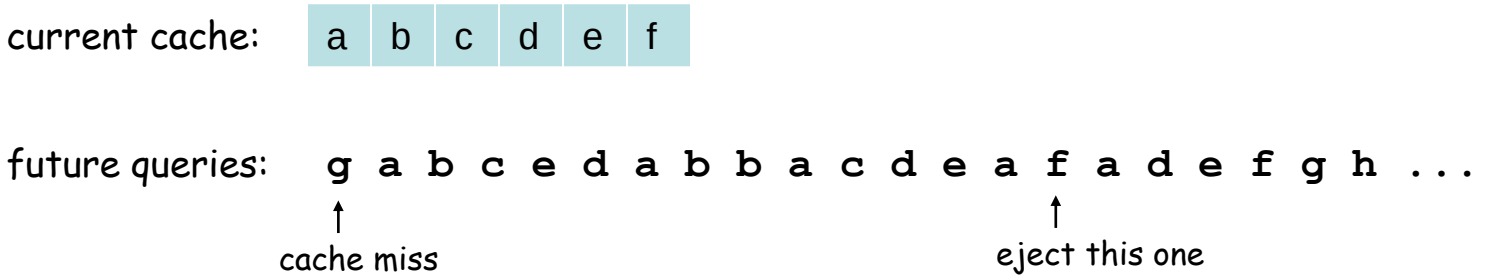
S' is optimal, reduced, and agrees with S_{FIF} for $j+1$ steps

Optimality of S_{FIF} follows by induction.

Alternative proof for $n = k + 1$

Farthest-in-future

- Evict item in the cache that is not requested until farthest in the future.



When $n = k + 1$,
between the cache miss and the farthest-item in the future,
"**g a b c e d a b b a c d e a f**"
contains all the item.
Hence, any algorithm must miss once.
Exercise: What's the problem of this proof for $n > k + 1$.

Online Caching

- Online vs. offline algorithms.
  - Offline: full sequence of requests is known a priori.
  - Online (reality): requests are not known in advance.
  - Caching is among most fundamental online problems in CS.

- LIFO. Evict page brought in most recently.
- LRU. Evict page whose most recent access was earliest.

FIF with direction of time reversed!

- Theorem. FIF is optimal offline eviction algorithm.
  - Provides basis for understanding and analyzing online algorithms.
  - LRU is k-competitive. [Section 13.8]
  - LIFO is arbitrarily bad.

Online Caching

What if the cost of replacing item i in cache by item j depends on d_{ij}?

K-server problem

From Wikipedia, the free encyclopedia

The **k-server problem** is a problem of theoretical computer science in the category of online algorithms, one of two abstract problems on metric spaces that are central to the theory of competitive analysis (the other being metrical task systems). In this problem, an online algorithm must control the movement of a set of *k servers*, represented as points in a metric space, and handle *requests* that are also in the form of points in the space. As each request arrives, the algorithm must determine which server to move to the requested point. The goal of the algorithm is to keep the total distance all servers move small, relative to the total distance the servers could have moved by an optimal adversary who knows in advance the entire sequence of requests.

Unsolved problem in computer science:

? *Is there a k-competitive algorithm for solving the k-server problem in an arbitrary metric space?*

(more unsolved problems in computer science)

The problem was first posed by Mark Manasse, Lyle A. McGeoch and Daniel Sleator (1990). The most prominent open question concerning the *k*-server problem is the so-called *k*-server conjecture, also posed by Manasse et al. This conjecture states that there is an algorithm for solving the *k*-server problem in an arbitrary metric space and for any number *k* of servers that has competitive ratio exactly *k*. Manasse et al. were able to prove their conjecture when $k = 2$, and for more general values of *k* when the metric space is restricted to have exactly $k+1$ points. Chrobak and Larmore (1991) proved the conjecture for tree metrics. The special case of metrics in which all distances are equal is called the *paging problem* because it models the problem of page replacement algorithms in memory caches, and was also already known to have a *k*-competitive algorithm (Sleator and Tarjan 1985). Fiat et al. (1990) first proved that there exists an algorithm with finite competitive ratio for any constant *k* and any metric space, and finally Koutsoupias and Papadimitriou (1995) proved that Work Function Algorithm (WFA) has competitive ratio $2k - 1$. However, despite the efforts of many other researchers, reducing the competitive ratio to *k* or providing an improved lower bound remains open as of 2014. The most common believed scenario is that the Work Function Algorithm is *k*-competitive. To this direction, in 2000 Bartal and Koutsoupias showed that this is true for some special cases (if the metric space is a line, a weighted star or any metric of $k+2$ points).

In 2011, a randomized algorithm with competitive bound $\tilde{O}(\log^2 k \log^3 n)$ was found.[1]

Anna Karlin

James R. Lee