

Analysis of Algorithms, I

CSOR W4231

Eleni Drinea
Computer Science Department

Columbia University

The greedy principle: cache maintenance

- 1 Cache maintenance (the offline problem)
- 2 An optimal greedy algorithm for the offline problem:
Farthest-into-Future (FF)
- 3 Proof of optimality of FF
- 4 The online problem

- 1 Cache maintenance (the offline problem)
- 2 An optimal greedy algorithm for the offline problem:
Farthest-into-Future (*FF*)
- 3 Proof of optimality of *FF*
- 4 The online problem

- ▶ **Caching**: the process of storing a small amount of data in a fast memory, so as to reduce the amount of time spent interacting with slow memory.
 - ▶ **Goal**: when attempting to download a page, it should be in cache.
- ⇒ Need a **cache maintenance** algorithm to determine what to keep and what to **evict** from the cache.

The input

Input

- ▶ n , the number of pages in the main memory
- ▶ k , the size of the **cache** memory
- ▶ a sequence of m requests r_1, r_2, \dots, r_m for memory pages

Example:

- ▶ #main memory pages $n = 3$
- ▶ set of main memory pages = $\{a, b, c\}$
- ▶ cache size $k = 2$
- ▶ #requests $m = 7$
- ▶ sequence of requests: a, b, c, b, c, a, b

The model

- ▶ To **service a request**, the corresponding page **must be in the cache**.
 - ⇒ After the first k requests for distinct pages the cache is full.
- ▶ A request is received and serviced within the same time step.
- ▶ **Cache miss**: a request for a page that is not in the cache.
 - ⇒ If there is a cache miss, we must **evict** a page from the cache to bring in the requested page.

Remark 1.

*The expensive operation is the **eviction**; every cache miss incurs an eviction.*

Our objective

At each time step $1 \leq t \leq m$, we must decide which page (if any) to evict from the cache.

Definition 1 (Scheduling algorithm).

A **schedule** is a sequence of eviction decisions so that all m requests are serviced at time m . An algorithm that provides such a schedule is a **scheduling** algorithm.

Goal: find the schedule that **minimizes** the total number of cache misses.

Example 2.

- ▶ # pages in main memory: $n = 3$
- ▶ cache size: $k = 2$
- ▶ sequence of $m = 7$ requests: a, b, c, b, c, a, b

time t :	1	2	3	4	5	6	7
requests:	$a,$	$b,$	$c,$	$b,$	$c,$	$a,$	b

Example 2.

- ▶ # pages in main memory: $n = 3$
- ▶ cache size: $k = 2$
- ▶ sequence of $m = 7$ requests: a, b, c, b, c, a, b

time t :	1	2	3	4	5	6	7
requests:	$a,$	$b,$	$c,$	$b,$	$c,$	$a,$	b
eviction schedule S :	$-,$	$-,$	$a,$	$-,$	$-,$	$c,$	$-$
cache contents:	$\{a\}$	$\{a, b\}$	$\{b, c\}$	$\{b, c\}$	$\{b, c\}$	$\{b, a\}$	$\{b, a\}$

- ▶ $-$ stands for “no eviction”
- ▶ $S = \{-, -, a, -, -, c, -\}$ evicts a at time 3, c at time 6
- ▶ S incurs 2 cache misses (*can't do better*)

Offline and online problems

- ▶ **Offline** problem: the entire sequence of requests $\{r_1, r_2, \dots, r_m\}$ is part of the **input** (known at time $t = 0$).
- ▶ **Online** problem (more natural): requests arrive one at a time; r_t must be serviced at time t , **before** future requests r_{t+1}, \dots, r_m are seen.
- ▶ A **scheduling algorithm for the online problem** can only base its eviction decision at time t on
 1. the requests it has seen so far,
 2. the eviction decisions it has made so far.
- ▶ The optimal offline algorithm provides a lower bound on the performance of **any** online algorithm.

Today

- 1 Cache maintenance (the offline problem)
- 2 An optimal greedy algorithm for the offline problem:
Farthest-into-Future (FF)**
- 3 Proof of optimality of FF
- 4 The online problem

The Farthest-into-Future (FF) rule

Definition 3 (Farthest-into-Future (FF) rule).

When the page requested at time i is not in the cache, evict from the cache the page that is needed the farthest into the future and bring in the requested page.

Notation: we will denote the schedule produced by this algorithm S_{FF} .

Example: the schedule S in Example 2 is the schedule produced by FF .

Definition 4 (Reduced schedule).

A reduced schedule brings a page in the cache at time t only if

1. the page is requested at time t ; *and*
2. the page is not already in the cache.

Remark 2.

1. *In a sense, a reduced schedule performs the least amount of work at every time step.*
2. *FF is a reduced schedule.*

There is an optimal *reduced* schedule

Fact 5.

We can transform a non-reduced schedule into a reduced one that is at least as good, that is, incurs at most the same number of evictions.

Remark 3.

- ▶ *The expensive memory operation is the eviction: so we should be minimizing #evictions and not #cache misses.*
 - ▶ *By Definition 4, #cache misses = #evictions in reduced schedules.*
 - ▶ *By Fact 5, we can focus solely on reduced schedules to find the optimal schedule.*
- ⇒ *Thus our original goal of minimizing #cache misses (rather than #evictions) is justified.*

Proof of Fact 5

- ▶ Let S' be a schedule that is not reduced and solves an instance of cache maintenance.
- ▶ We will construct a reduced schedule S that incurs at most as many evictions as S' .
 - ▶ Suppose that at time i , there is a request $r_i \neq a$ but S' evicts a page from the cache to bring in page a , although a is **not** requested at time i .
 - ▶ Then S *pretends* to bring in a but in fact does nothing: at the **first** time step $j > i$ such that $r_j = a$, S brings in a .
 - ⇒ We can *charge* the cache miss of S at time j to the eviction of S' at the **earlier** time i .
- ▶ Thus S performs at most as many evictions as S' .

Example 6.

- ▶ # pages in main memory: $n = 4$
- ▶ cache size: $k = 3$
- ▶ sequence of $m = 9$ requests: $a, b, c, d, b, c, a, d, b$
- ▶ $r_t =$ request at time t
- ▶ $C_S(t) =$ contents of the cache of schedule S at end of time t

Non-reduced schedule S

t	1	2	3	4	5	6	7	8	9
r_t	a	b	c	d	b	c	a	d	b
bring	a	b	c	d	c	d	b	-	-
evict	-	-	-	c	d	b	c	-	-
$C_S(t)$	a	{a, b}	{a, b, c}	{a, b, d}	{a,b,c}	{a, c, d}	{a,b,d}	{a, b, d}	{a, b, d}

Example 6.

- ▶ # pages in main memory: $n = 4$
- ▶ cache size: $k = 3$
- ▶ sequence of $m = 9$ requests: $a, b, c, d, b, c, a, d, b$
- ▶ r_t = request at time t
- ▶ $C_S(t)$ = contents of the cache of schedule S at end of time t

Non-reduced schedule S

t	1	2	3	4	5	6	7	8	9
r_t	a	b	c	d	b	c	a	d	b
bring	a	b	c	d	c	d	b	-	-
evict	-	-	-	c	d	b	c	-	-
$C_S(t)$	a	{a, b}	{a, b, c}	{a, b, d}	{a,b,c}	{a, c, d}	{a,b,d}	{a, b, d}	{a, b, d}

Reduced schedule S'

t	1	2	3	4	5	6	7	8	9
r_t	a	b	c	d	b	c	a	d	b
bring	a	b	c	d	-	c	-	d	b
evict	-	-	-	c	-	d	-	b	c
$C_{S'}(t)$	a	{a, b}	{a, b, c}	{a, b, d}	{a,b,d}	{a, b, c}	{a,b,c}	{a, c, d}	{a, b, d}

Blue entries denote cache misses.

Red entries denote evictions when no cache miss occurred.

S evicts pages d, b, c at times 5, 6, 7, even though these pages are not requested at these times (no cache misses). S' performs the exact same evictions that S performed at times 5, 6, 7 at the later times 6, 8, 9 respectively, when these pages were actually requested (hence cache misses were incurred).

Today

- 1 Cache maintenance (the offline problem)
- 2 An optimal greedy algorithm for the offline problem:
Farthest-into-Future (*FF*)
- 3 Proof of optimality of FF**
- 4 The online problem

Claim 1.

Let S_i be a reduced schedule that makes the same eviction decisions as S_{FF} up to time i , that is, up to request i . Then there is a reduced schedule S_{i+1} that

1. makes the same eviction decisions as S_{FF} up to time $t = i + 1$, that is, up to request $i + 1$;
2. S_{i+1} incurs no more total cache misses than S_i .

Proposition 1.

The schedule S_{FF} provided by the Farthest-into-Future algorithm is optimal.

Proof of Proposition 1: case $i = 0$

Notation

- ▶ $cm(S)$ = total #cache misses of schedule S
- ▶ S^* is an optimal reduced schedule
- ▶ Schedule S follows schedule S' up to request i if S makes the same eviction decisions as S' up to the i -th request

$i = 0$: trivially, S^* follows S_{FF} up to request $i = 0$. By Claim 1, we can construct a reduced schedule S_1 such that

1. S_1 follows S_{FF} up to request $i = 1$,
2. $cm(S_1) \leq cm(S^*)$.

Proof of Proposition 1: case $i > 0$

Notation: $cm(S)$ = total #cache misses of schedule S

- ▶ $i = 1$: now S_1 is a reduced schedule that follows S_{FF} up to request $i = 1$. By Claim 1, we can construct a reduced schedule S_2 such that
 1. S_2 follows S_{FF} up to request $i = 2$,
 2. $cm(S_2) \leq cm(S_1)$.

- ▶ $i = 2$: now S_2 is a reduced schedule that follows S_{FF} up to request $i = 2$. By Claim 1, we can construct a reduced schedule S_3 such that
 1. S_3 follows S_{FF} up to request $i = 3$,
 2. $cm(S_3) \leq cm(S_2)$.

Proof of Proposition 1: $S_m = S_{FF}$

Notation: $cm(S) = \text{total \#cache misses of schedule } S$

- ▶ Applying the claim for every $3 \leq i \leq m - 1$, we obtain a reduced schedule S_m that
 1. follows S_{FF} up to time m ,
 2. $cm(S_m) \leq cm(S_{m-1})$.

Tracing back all the inequalities, we obtain $cm(S_m) \leq cm(S^*)$.

- ▶ Finally, since S_m follows S_{FF} up to time m , $S_{FF} = S_m$.

Hence $cm(S_{FF}) = cm(S_m) \leq cm(S^*)$.

Thus S_{FF} is optimal.

Before we prove Claim 1, we slightly simplify notation to obtain the following claim.

Claim 2.

Let S be a reduced schedule that makes the same eviction decisions as S_{FF} up to time $t = i$, that is, up to request i . Then there is a reduced schedule S' such that

- 1. S' makes the same eviction decisions as S_{FF} up to time $t = i + 1$, that is, up to request $i + 1$;*
- 2. S' incurs no more total cache misses than S .*

Proof of Claim 2: a case-by-case analysis

Notation:

- ▶ $cm(S) = \text{total \#cache misses of schedule } S$
- ▶ $C_i(S) = \text{contents of the cache of schedule } S \text{ at the end of time step } i$

Since S and S_{FF} have made the same scheduling decisions up to time i , the following statements hold at the end of time step i .

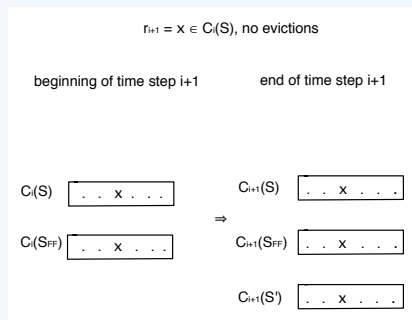
1. The contents of their caches are identical, that is,

$$C_i(S) = C_i(S_{FF}).$$

2. **So far**, S has the same number of cache misses as S_{FF} .

Case 1: request $r_{i+1} = x \in C_i(S)$

1. If page x requested at time $i + 1$ is in $C_i(S)$, then
 - ▶ $x \in C_i(S_{FF})$ (recall that $C_i(S) = C_i(S_{FF})$);
 - ▶ no cache miss for either schedule.



- ▶ Set $S' = S$; then
 1. S' follows S_{FF} up to time $i + 1$ (S does!);
 2. $cm(S') \leq cm(S)$.

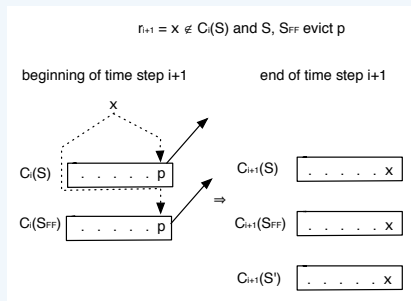
Case 2: $r_{i+1} = x \notin C_i(S)$ (cont'd)

2. If page x requested at time $i + 1$ is not in $C_i(S)$, then

- ▶ $x \notin C_i(S_{FF})$ (recall that $C_i(S) = C_i(S_{FF})$);
- ▶ both schedules must bring x in, hence incur a cache miss.

2.1: If S and S_{FF} both evict the same page p , set $S' = S$:

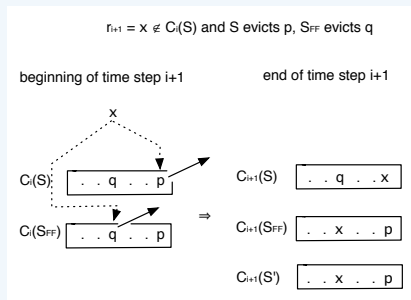
1. S' follows S_{FF} up to time $i + 1$ (S does!),
2. $cm(S') \leq cm(S)$.



Case 2.2: $r_{i+1} = x \notin C_i(S)$

2.2: If S evicts p but S_{FF} evicts q :

- ▶ By construction of S_{FF} , q must be requested later in the future than p (recall the *Farthest-into-Future* rule).
- ▶ At the end of time step $i + 1$, the cache contents for the two schedules will differ in exactly one item.



Case 2.2: S evicts p , S_{FF} evicts q

At the end of time step $i + 1$

- ▶ the cache of S contains q ;
- ▶ the cache of S_{FF} contains p ;
- ▶ the remaining $k - 1$ items in both caches are the same;
- ▶ thus

$$C_{i+1}(S_{FF}) = C_{i+1}(S) - \{q\} + \{p\}.$$

- ▶ Since we want S' to *follow* S_{FF} up to time $i + 1$, S' evicts q from its cache as well. Hence

$$C_{i+1}(S') = C_{i+1}(S_{FF}) = C_{i+1}(S) - \{q\} + \{p\}.$$

Roadmap for case 2.2: S evicts p , S_{FF} evicts q

Notation: $cm(S)$ = total #cache misses of schedule S

- ▶ At the end of time step $i + 1$,
 - ▶ the cache contents of S, S' differ in exactly one item;
 - ▶ S' follows S_{FF} up to time $i + 1$;
 - ▶ #cache misses of $S =$ #cache misses of S' .

- ▶ **Goal:** Ensure that S' does not incur more misses than S for $i + 1 < t \leq m$, so that $cm(S') \leq cm(S)$.

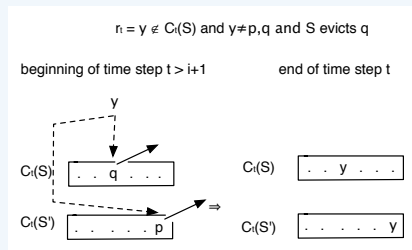
- ▶ **Idea:** Set $S' = S$ as soon as the cache contents of S, S' are the same again.
 1. Make $C_t(S')$ equal $C_t(S)$ at the earliest $t > i + 1$ possible, while not incurring unnecessary misses.
 2. Once $C_t(S') = C_t(S)$, set $S' = S$. \Rightarrow If S' has not incurred more misses than S between steps $i + 2$ and t , then $cm(S') \leq cm(S)$ and the claim holds.

Case 2.2.1: $r_t = x \notin \{p, q\}$, $x \notin C_t(S)$, S evicts q

For all $t > i + 1$, S' follows S **until** one of the following happens for the first time:

2.2.1: $r_t = y \notin \{p, q\}$, **and** $y \notin C_t(S)$, **and** S evicts q .

Since $C_t(S)$ and $C_t(S')$ only differ in p, q , then $y \notin C_t(S')$.
Set S' to evict p and bring in y . Then $C_t(S') = C_t(S)$!

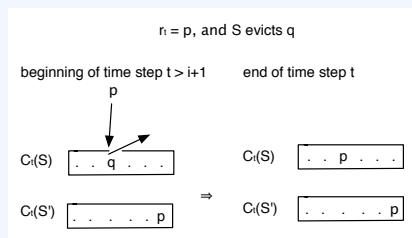


Set $S' = S$ henceforth: S' follows S_{FF} up to time $i + 1$ and $cm(S') \leq cm(S)$.

Case 2.2.2.1: $r_t = p$, S evicts q

2.2.2: $r_t = p$

2.2.2.1: If S evicts q , $C_t(S) = C_t(S')$!

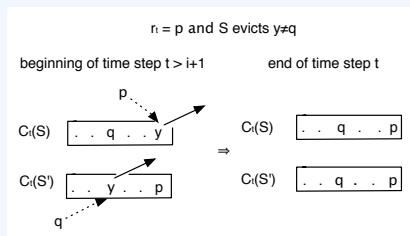


Set $S' = S$ henceforth: S' follows S_{FF} up to time $i + 1$ and $cm(S') < cm(S)$.

Case 2.2.2.2: $r_t = p$, S evicts $y \neq q$

2.2.2: $r_t = p$

2.2.2.2: If S evicts $y \neq q$ from its cache, then S' evicts y as well and brings in q . Then $C_t(S') = C_t(S)$.



Set $S' = S$ henceforth: S' follows S_{FF} up to time $i + 1$ and $cm(S') \leq cm(S)$.

2.2.2.2: S' is no longer *reduced*

- ▶ S' is no longer **reduced**: q was brought in when there was no request for q at time t (recall that $r_t = p$).
- ▶ Fortunately, we can use Fact 1 to transform S' into a reduced schedule \bar{S} that
 - ▶ incurs at most the same total #evictions as S' ;
 - ▶ still follows S_{FF} up to time $i + 1$: all the real evictions of the reduced \bar{S} will happen **after** time $i + 1$.
- ▶ Hence we return \bar{S} as the schedule that satisfies Claim 2.

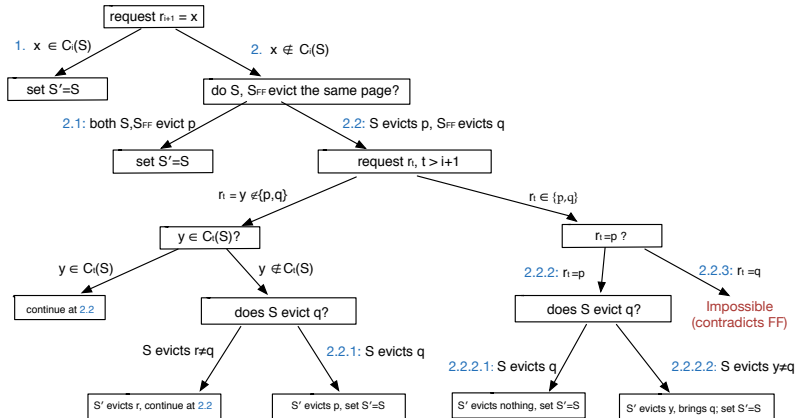
2.2.3: $r_t = q$

Can't happen!

S_{FF} evicted q and not p , hence q appears farther in the future than p .

Hence one of the cases [2.2.1](#), [2.2.2](#) will happen first.

Complete roadmap



Today

- 1 Cache maintenance (the offline problem)
- 2 An optimal greedy algorithm for the offline problem:
Farthest-into-Future (*FF*)
- 3 Proof of optimality of *FF*
- 4 The online problem

The online problem

- ▶ **Offline** problem: the entire sequence of requests $\{r_1, r_2, \dots, r_m\}$ is part of the **input** (known at time $t = 0$).
- ▶ **Online** problem (more natural): requests arrive one at a time; r_t must be serviced at time t , **before** r_{t+1}, \dots, r_m are seen.
- ▶ An **online scheduling algorithm** can only base its eviction decision at time t on
 1. the requests it has seen so far;
 2. the eviction decisions it has made so far.
- ▶ The optimal offline algorithm provides a lower bound on the performance of **any** online algorithm.

The Least Recently Used principle

- ▶ The **Least Recently Used (LRU)** principle: evict the page that was requested the **longest ago**.
- ▶ **Intuition:** a running program will generally keep accessing the things it's just been accessing (**locality of reference**).
- ▶ Essentially Farthest-into-Future (FF) reversed in time.
- ▶ LRU behaves well on average inputs.
- ▶ However an adversary can devise a specific sequence of online requests that will cause LRU to perform very badly compared to the optimal offline algorithm (*how?*).

Worst-case input to LRU

Example

- ▶ #pages in main memory: $n = 3$
- ▶ size of the cache: $k = 2$
- ▶ sequence of online requests

$$\underbrace{a, b, c}_1, \underbrace{a, b, c}_2, \dots, \underbrace{a, b, c}_M$$

- ⇒ LRU: **every** request starting at time $t = 3$ is a miss, hence $3M - 2$ misses.
- ⇒ *FF*: no more than $3M/2$ misses.

Competitive ratio

- ▶ More generally, if we have a sequence of nM online requests as above, for some integer M , LRU will incur $nM - k$ misses, while FF will incur no more than $\lceil nM/k \rceil$ misses.
- ▶ Hence LRU may perform up to a factor of k times worse than FF.
- ▶ **(Online analysis) competitive ratio:** the worst-case ratio between the performance of the online algorithm and the performance of the optimal offline algorithm.