

Algorithm Design and Analysis

CSE
565

LECTURE 6

Greedy Algorithms

- Problems
 - Scheduling to Minimize Lateness
 - Optimal Caching
- Analysis
 - Exchange Argument

Sofya Raskhodnikova

9/12/2007

S. Raskhodnikova; based on slides by K. Wayne

L1.1

Scheduling to Minimize Lateness

Minimizing lateness problem.

- Single resource processes one job at a time.
- Job j requires t_j units of processing time and is due at time d_j .
- If j starts at time s_j , it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max\{0, f_j - d_j\}$.
- Goal: schedule all jobs to minimize **maximum lateness** $L = \max \ell_j$.

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Possible Greedy Algorithms

Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time t_j .
- [Earliest deadline first] Consider jobs in ascending order of deadline d_j .
- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time t_j .

	1	2
t_j	1	10
d_j	100	10

counterexample

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

	1	2
t_j	1	10
d_j	2	10

counterexample

Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

```
Sort n jobs by deadline so that  $d_1 \leq d_2 \leq \dots \leq d_n$ 
t ← 0
for j = 1 to n
  ◁ Assign job j to interval [t, t + tj]
  sj ← t, fj ← t + tj
  t ← t + tj
output intervals [sj, fj]
```



No Idle Time

Observation. There exists an optimal schedule with no idle time.



Observation. The greedy schedule has no idle time.

Inversions

Definition. An **inversion** in a schedule S is a pair of jobs i, j such that: $d_i < d_j$ but j is scheduled before i .

Observation. Greedy schedule has no inversions.

Observation. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

Claim. Swapping two consecutive, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

Analysis of Greedy: Exchange Argument

Theorem. Greedy schedule S is optimal.

Proof. Let S^* be an optimal schedule.

- Can assume S^* has no idle time.
- If S^* has no inversions, then $S = S^*$.
- If S^* has an inversion, we'll reduce the number of inversions:
 - let i, j be an adjacent inversion.
 - swapping i and j does not increase the maximum lateness and strictly decreases the number of inversions
 - this contradicts definition of S^*

Greedy Analysis Strategies

Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.

Structural. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

Other greedy algorithms. Kruskal, Prim, Dijkstra, Huffman, ...

Optimal Offline Caching

Caching.

- Cache with capacity to store k items.
- Sequence of m item requests d_1, d_2, \dots, 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 cache misses.

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

Optimal eviction schedule: 2 cache misses.

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

Optimal Offline Caching: Farthest-In-Future

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

current cache: $a \ b \ c \ d \ e \ f$

future queries: $g \ a \ b \ c \ d \ e \ a \ b \ b \ a \ c \ d \ e \ a \ f \ a \ d \ e \ f \ g \ h \ \dots$

↑
cache miss

↑
evict this one

Theorem. [Bellady, 1960s] FF is optimal eviction schedule. Algorithm and theorem are intuitive; proof is subtle.

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	b	c
a	a	c	b

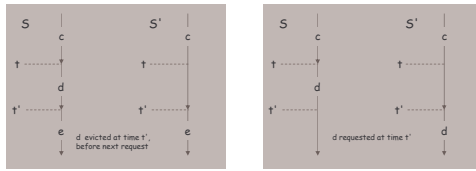
a reduced schedule

Reduced Eviction Schedules

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

Pf. (by induction on number of unreduced items) doesn't enter cache at requested time

- 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. •



Case 1

Case 2

Farthest-In-Future: Analysis

Theorem. FF is optimal eviction algorithm.

Pf. (by induction on number of requests j)

Invariant: There exists an optimal reduced schedule S that coincides with S_{FF} on the first $j+1$ requests.

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

- Consider $(j+1)^{th}$ request $d = d_{j+1}$.
- Since S and S_{FF} 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.
- Case 2: (d is not in the cache and S and S_{FF} evict the same element). $S' = S$ satisfies invariant.

Farthest-In-Future: Analysis

Pf. (continued)

- Case 3: (d is not in the cache; S_{FF} evicts e ; S evicts $f \neq e$).
- begin construction of S' from S by evicting e instead of f



- now S' agrees with S_{FF} on first $j+1$ requests; we show that having element f in cache is no worse than having element e

Farthest-In-Future: Analysis

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

must involve e or f (or both)



- Case 3a: $g = e$. Can't happen with Farthest-In-Future since 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_{FF} through step $j+1$

Farthest-In-Future: Analysis

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

must involve e or f (or both)



otherwise S' would take the same action

- Case 3c: $g \neq e, f$. S must evict e .
Make S' evict f ; now S and S' have the same cache. •



Caching Perspective

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.

FF with direction of time reversed!

Theorem. FF is optimal offline eviction algorithm.

- Provides basis for understanding and analyzing online algorithms.
- LRU is k -competitive. [Section 13.8]
- LIFO is arbitrarily bad.