Objectives

Greedy Algorithms

Greedy Algorithms

At each step

- Decision: Take as much as you can get
  - Feasible – satisfy problem's constraints
  - Locally optimal – best local choice among available feasible choices
  - Irrevocable – after decided, no going back

Scheduling to Minimizing Lateness

Single resource processes one job at a time

Job j requires t_j units of processing time and is due at time d_j

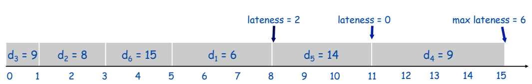
If j starts at time s_j, it finishes at time $f_j = s_j + t_j$

Lateness: $\ell_j = \max\{ 0, f_j - d_j \}$

Goal: schedule all jobs to minimize maximum lateness $L = \max \ell_j$

Ex:

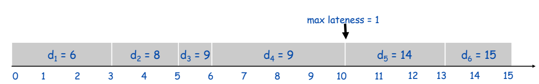
	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

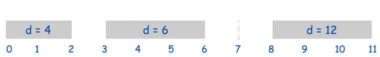
```
Sort n jobs by deadline so that d1 ≤ d2 ≤ … ≤ dn
t = 0
for j = 1 to n
   Assign job j to interval [t, t + tj]
   sj = t
   fj = t + tj
   t = t + tj
output intervals [sj, fj]
```

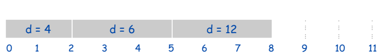


What can we say about this algorithm/its results?

Minimizing Lateness: No Idle Time

Observation. There exists an optimal schedule with no idle time





Observation. The greedy schedule has no idle time

Proving Optimality

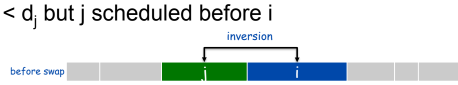
Goal: Prove greedy algorithm produces optimal solution

Approach: **Exchange argument**

- Start with an optimal schedule Opt
- Gradually modify Opt
  - Preserving its optimality
    - How do we measure optimality in this case?
- Transform into a schedule identical to greedy's schedule

Minimizing Lateness: Inversions

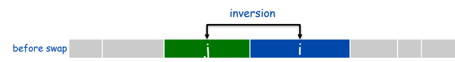
Def. An inversion in schedule S is a pair of jobs i and j such that:
$d_i < d_j$ but j scheduled before i



Can Greedy's solution have any inversions?

Minimizing Lateness: Inversions

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



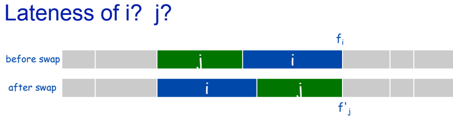
Observation. Greedy schedule has no inversions

Minimizing Lateness: Inversions

Claim. Swapping two adjacent jobs with the same deadline does not increase the max lateness

Pf Sketch. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards

- Lateness of other jobs?
- Lateness of i? j?

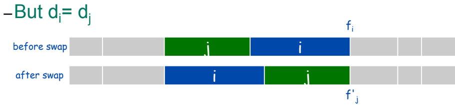


Minimizing Lateness: Inversions

Claim. Swapping two adjacent jobs with the same deadline does not change the max lateness

Pf. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards

- Lateness remains the same for all other jobs:
  - $\ell'_k = \ell_k$ for all $k \neq i, j$
- Lateness of i before is $f_i - d_i = t_i + t_j - d_i$
- Lateness of j after is $f_j' - d_j = t_i + t_j - d_j$
  - But $d_i = d_j$



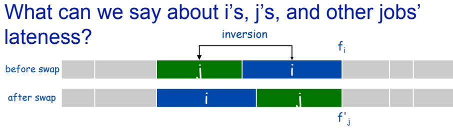
Minimizing Lateness: Inversions

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

- How do we know inversions are adjacent?

Pf Setup. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards

- What can we say about i's, j's, and other jobs' lateness?



By def of inversion, $d_i < d_j$

Minimizing Lateness: Inversions

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

Pf. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards

- $\ell'_k = \ell_k$ for all $k \neq i, j$
- $\ell'_i \leq \ell_i$
- If job j is late:

$$\begin{aligned} \ell'_j &= f'_j - d_j && \text{(definition)} \\ &= f_i - d_j && (j \text{ finishes at time } f_i) \\ &\leq f_i - d_i && (i < j) \\ &\leq \ell_i && \text{(definition)} \end{aligned}$$

Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy schedule S is optimal

Pf idea. Convert Opt to Greedy

- Does opt schedule have idle time?
- What if opt schedule has no inversions?
- What if opt schedule has inversions?

Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy schedule S is optimal

Pf. Define S* to be an optimal schedule that has the fewest number of inversions, and let's see what happens

- Can assume S* has no idle time
- If S* has no inversions, then S = S*
- If S* has an inversion, 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.

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

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.

OPTIMAL CACHING

Motivating Caching

On an airplane, where do you keep the stuff that

- You need to use most often/have fastest access to?
  - How large is that space?
- Where do you keep the stuff that you want access to during the flight?

Caching

Memory: smaller capacity but fast access

Disk: larger capacity but slower access

Other examples of caches

- Web browser cache
- DNS cache
- DB cache

Optimal Offline Caching

Cache with capacity to store *k* items

Sequence of *m* item requests $d_1, d_2, \ldots, d_m$

Cache hit: item already in cache when requested

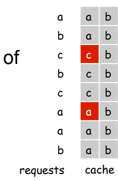
Cache miss: item not already in cache when requested

- Must bring requested item into cache
- Evict an 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



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

Ideas for Eviction Selection Criteria?

Optimal Offline Caching: Farthest-In-Future

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

current cache: a b c d e f

future queries: g a b c e d a b b a c d e a f a d e f g h ...

cache miss

Example:

- g is requested but not in the cache
- Which element should we eject from the cache?

Optimal Offline Caching: Farthest-In-Future

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

current cache: a b c d e f

future queries: g a b c e d a b b a c d e a f a d e f g h ...

cache miss eject this one

Theorem. [Bellady, 1960s] FF is optimal eviction schedule

Pf. Algorithm and theorem are intuitive; proof is subtle

- Better than least frequently used?

Reduced Eviction Schedules

Def. A reduced schedule is a schedule that only inserts an item into the cache when that item is requested

- No bringing in an item ahead of time; minimal amt of work per step

→ Why might we want/have an unreduced schedule?

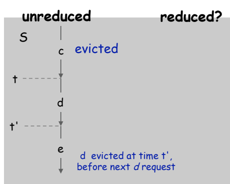
an unreduced schedule   a reduced schedule

Reduced Eviction Schedules

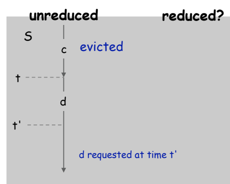
Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more cache misses

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



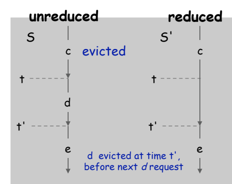
Case 2

Reduced Eviction Schedules

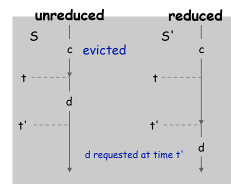
Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more cache misses

Pf. (by induction on number of unreduced items)

- 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 Sketch

- Let S_{FF} be schedule by Farthest-in-Future
- Let S* be optimal schedule
  - Fewest possible cache misses
- Transform S* into S_{FF}
  - One eviction decision at a time
  - Not increasing number of cache misses

Feedback on Problem Sets

Overall, did well

- More lenient grading because I'm figuring out what I want/ expect

Looking for a little more of your work/thinking

- To understand what you were thinking
  - Problem misunderstanding or otherwise
- Comments and/or descriptive variable names
- Some background on your approach, outside of algorithm
  - Picture

Brief description of why algorithm has that running time