Greedy Algorithms - 2

CSIE 2136 Algorithm Design and Analysis, Fall 2018

https://cool.ntu.edu.tw/courses/61

Hsu-Chun Hsiao

National Taiwan University

Announcement

Homework assignments

- Mini-hw6 due next week
- HW2 due in 2 weeks

Please remember to put references in HW

Feedback about COOL?

Review lecture next week

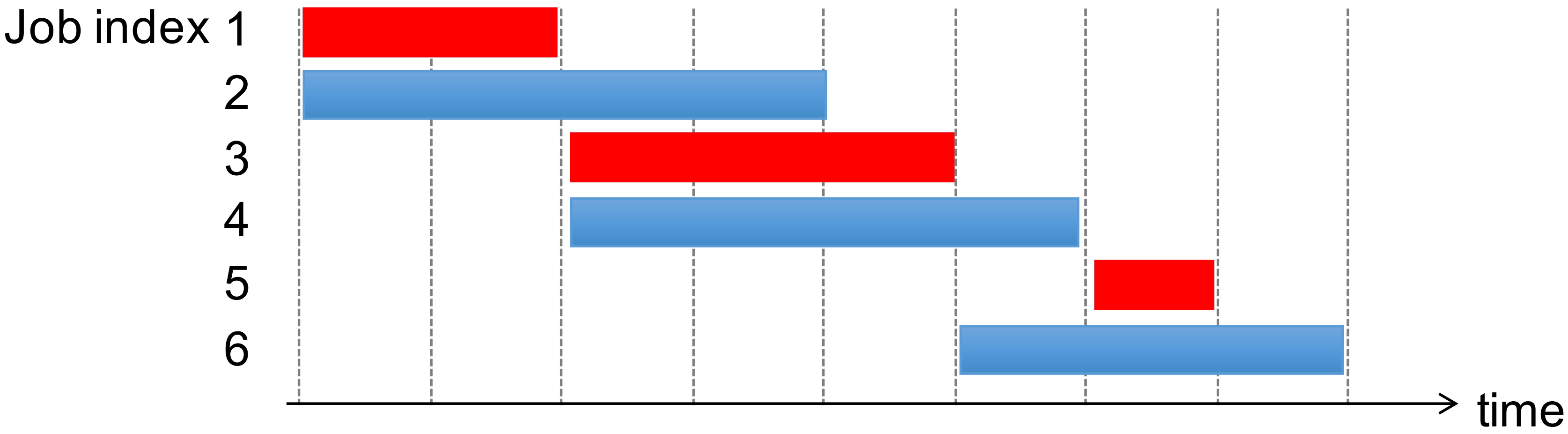
Interval Scheduling

Textbook Chapter 16.1

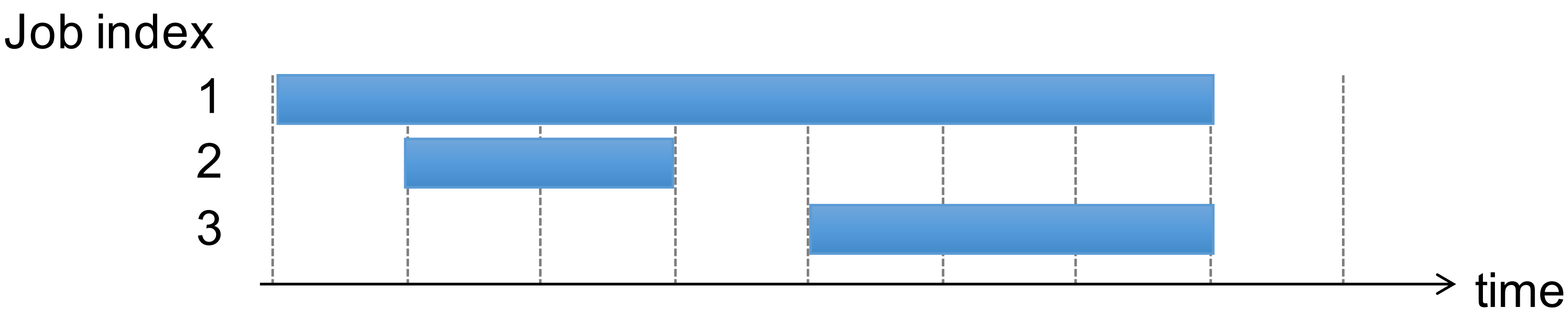
Chapter 4.1 in Algorithm Design by Kleinberg & Tardos

Interval scheduling (區間調度)

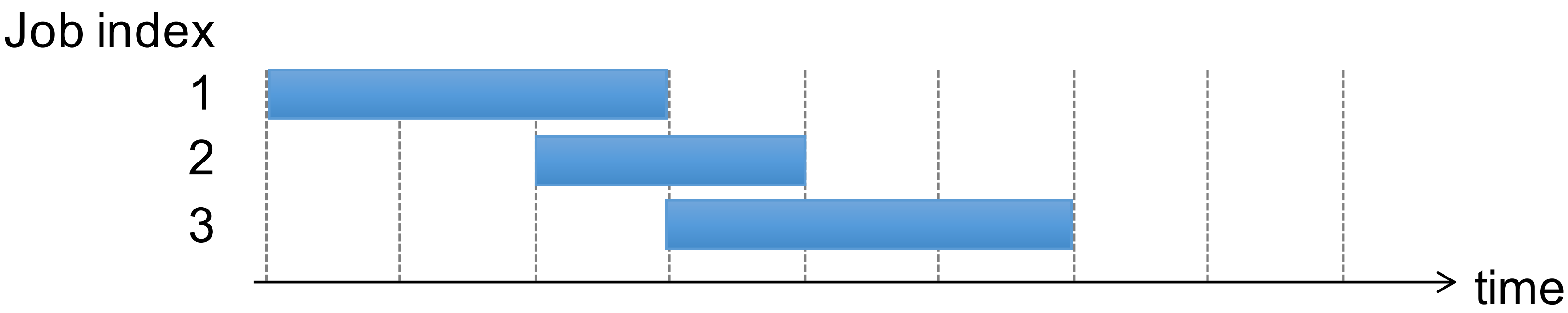
- Given a set of job requests with start times and finish times, find the maximum number of compatible jobs
  - E.g., 給定每門課的時間，這一天最多可以上幾門課？
- A special case of weighted interval scheduling, but solve it using DP is an overkill
- What should be the greedy choice here?
  - Earliest start time, shortest interval, fewest number of non-compatible requests, earliest finish time…?



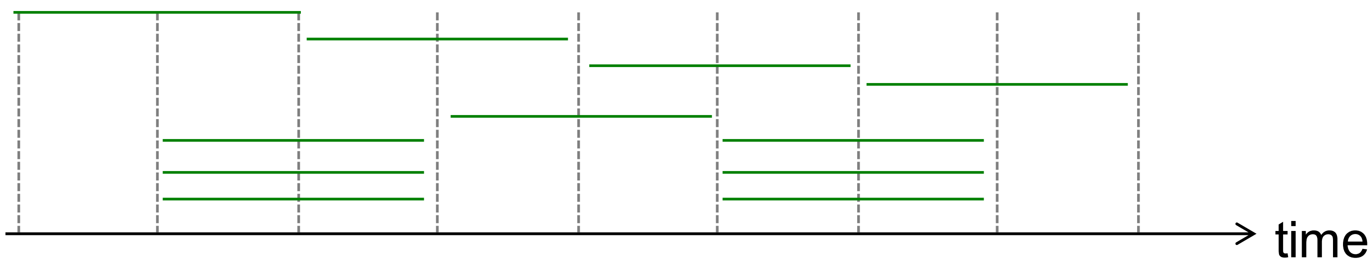
Counterexample to earliest-start-time first:



Counterexample to shortest-interval first:



Counterexample to choosing fewest number of non-compatible requests:



Interval scheduling

Greedy choice: earliest-finish-time-first

- Intuition: leave the resource available for as many jobs that follow it as possible

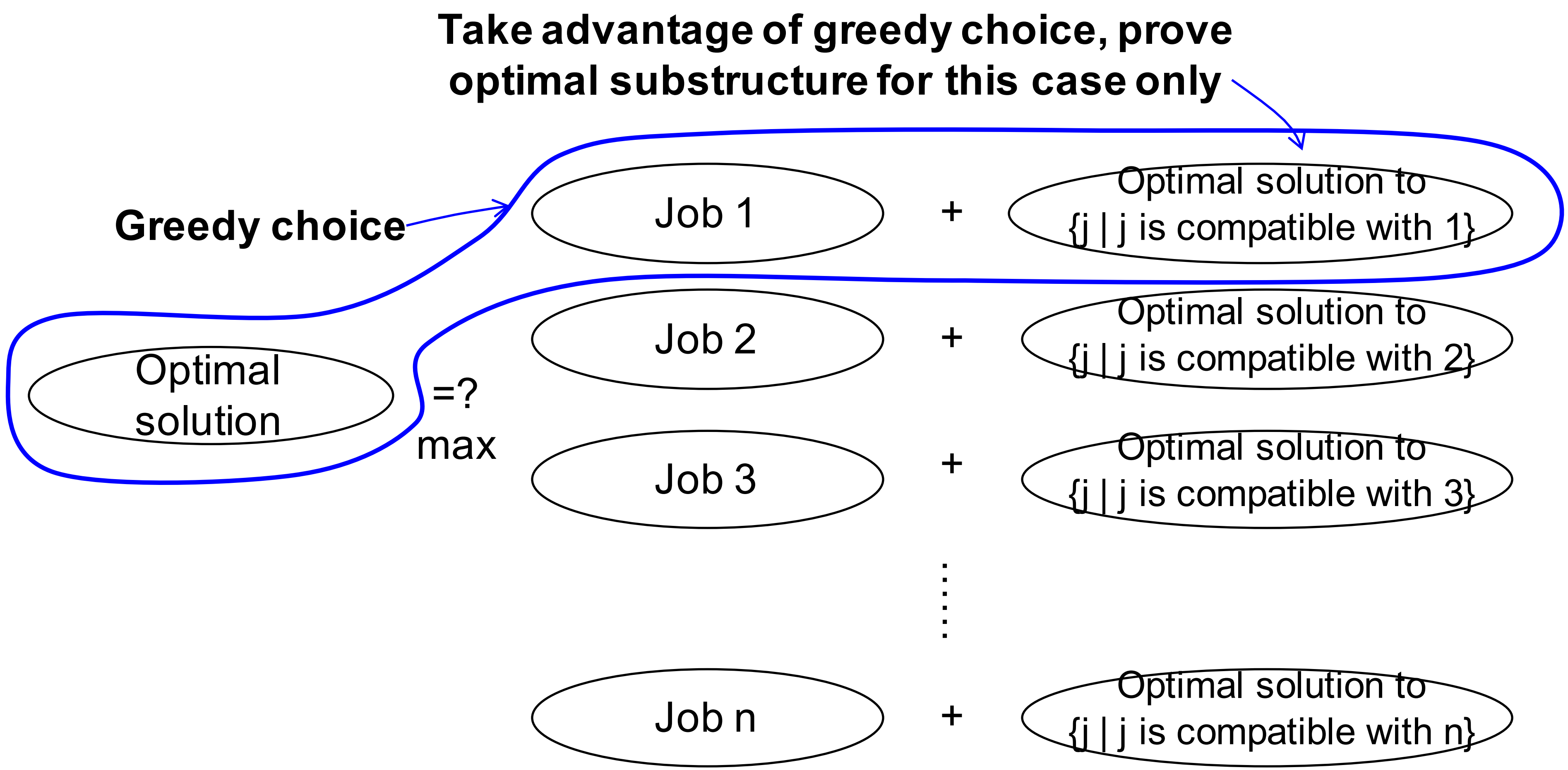
Practice: explain why the resulting set is compatible

```
Input: n, s[1..n], f[1..n]

Interval-Scheduling(n, s[], f[]):
    Sort jobs by finish time such that f[1]≤f[2]≤ …≤f[n]
    A = {1}
    k = 1 //the largest index in A so far
    for i = 2 to n
        if s[i] ≥ f[k]
            A = A∪{i}
            k = i
    return A //indices of selected jobs
```

Running time = O(nlogn) given unsorted jobs

Greedy choice and subproblems



Proof of greedy choice property

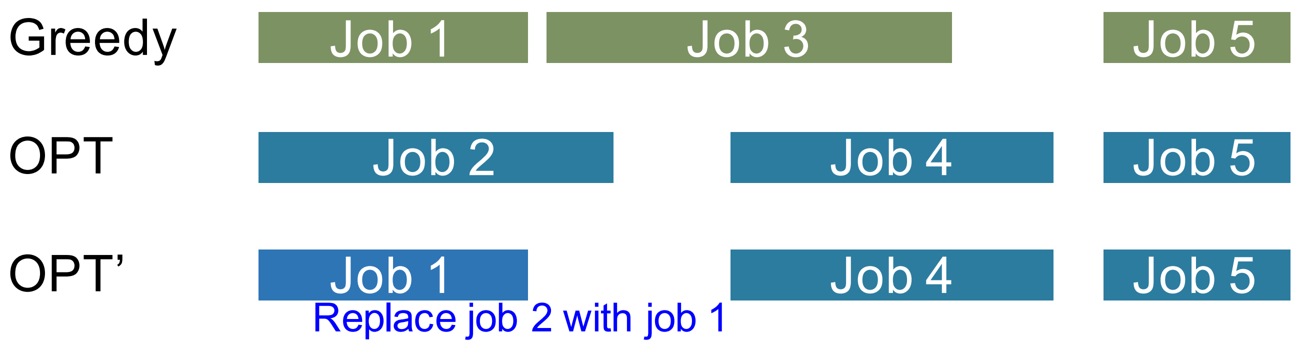
Given n jobs and their finish times f[i], f[1]≤f[2]≤ …≤f[n]

Prove that ∃ an optimal solution containing job 1

<u>Proof by the exchange argument:</u>

Key idea: suppose OPT is an optimal solution. Modify OPT into another optimal solution containing job 1.

If OPT contains job 1, done



Proof of greedy choice property

Proof by the exchange argument (cont'd):

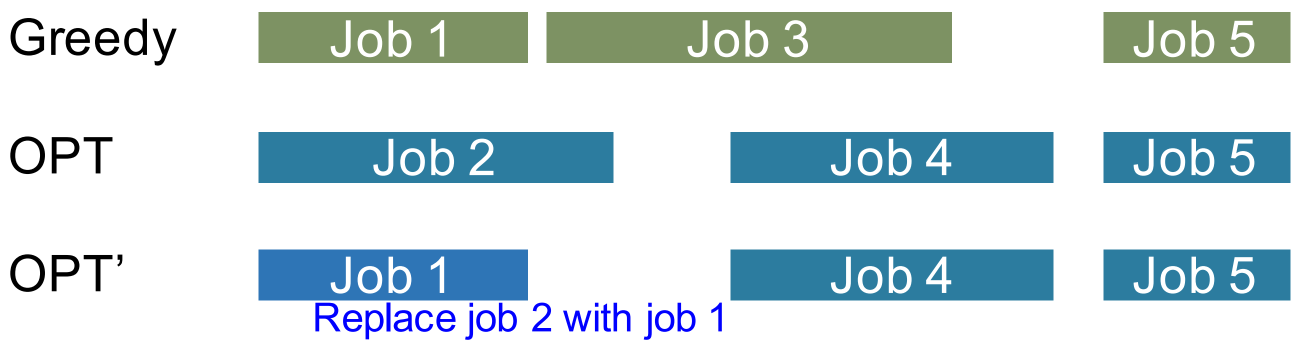
If not, let $x_1, x_2, \ldots, x_q$ be the job indices in OPT from low to high

- => job x_1 is compatible with $x_2, \ldots, x_q$, that is, $f[x_1] \le s[x_j]$ for all j in 2…q

Let $OPT' = OPT \backslash \{x_1\} \cup \{1\}$, $|OPT'| = |OPT| = q$

$f[1] \le f[x_1] \le s[x_j]$ for all j in 2…p, so OPT' is also a compatible set

OPT' is an optimal solution containing job 1



Proof of optimal substructure

Prove that if OPT is an optimal solution to jobs 1, 2,…, n, then OPT\{1} is also an optimal solution to jobs i, i+1,…,n, where i is the smallest index s.t. f[1] ≤s[i]

Proof by contradiction:

Suppose OPT\{1} is not optimal to jobs i…n

- => $\exists$ OPT' s.t. OPT' is optimal to jobs i…n and |OPT'| > |OPT| - 1
- => OPT' $\cup$ {1} is an optimal solution to jobs 1…n and |OPT' $\cup$ {1}| = |OPT'| + 1 > |OPT|
- => contradiction

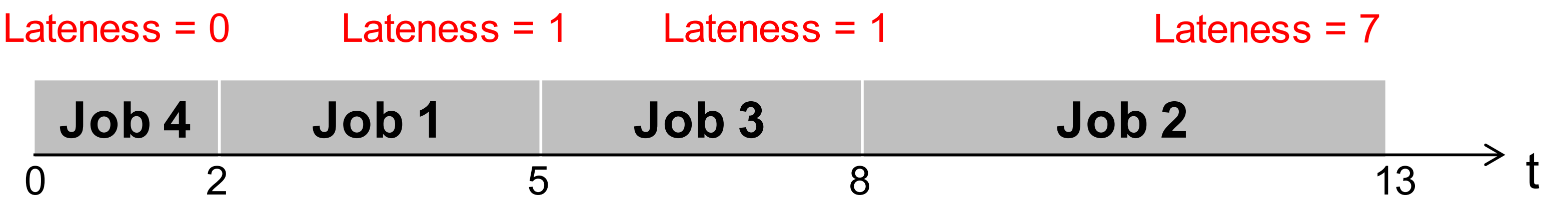
Scheduling to minimize lateness

Scheduling to minimize lateness

Given a set of jobs with processing times and deadlines, schedule **all** jobs to **minimize the maximum lateness** (only one job can be processed at a time)

Example:

Job	1	2	3	4
Processing time	3	5	3	2
Deadline	4	6	7	8



Maximum lateness of all jobs in this schedule is 7

Scheduling to minimize lateness

Given a set of jobs with processing times and deadlines, schedule all jobs to **minimize the maximum lateness** (only one job can be processed at a time)

- t_j = processing time of job j
- d_j = deadline of job j
- Denote by s(H, j) and f(H, j) the start and finish time of job j in a schedule H, thus f(H, j) - s(H, j) = t_j

Lateness of job j in schedule H = L(H,j) = max{0, f(H,j) - d_j}

Maximum lateness of schedule H = L(H) = $\max_j$ L(H,j)

Goal: find a schedule H that minimizes L(H)

Possible greedy choices

Shortest-processing-time-first without idle time?

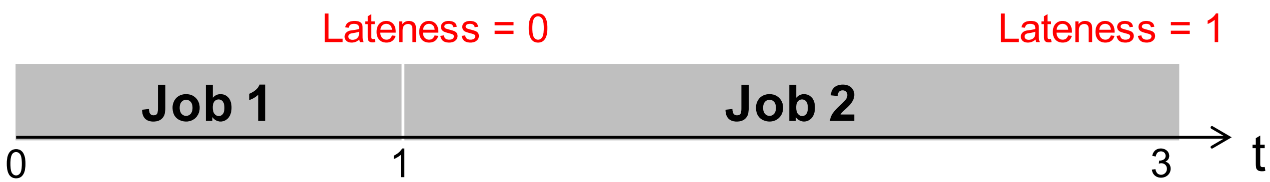
Earliest-deadline-first without idle time?

Practice: Show that any schedule with idle time is not optimal

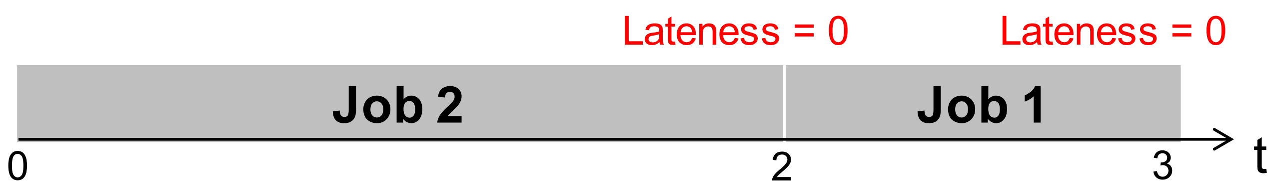
Counterexample to shortest processing time first

Job	**1**	**2**
Processing time	1	2
Deadline	10	2

Shortest processing time first
(Max lateness = 1)



An optimal solution
(Max lateness = 0)

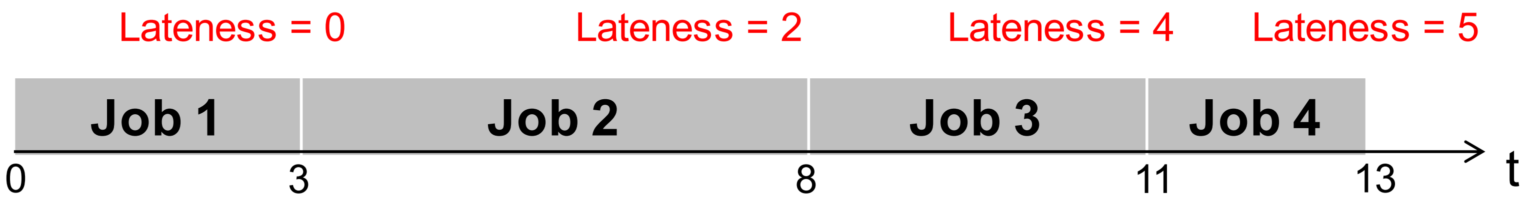


Minimizing lateness

<u>Greedy choice</u>: earliest-deadline-first without idle time

Example:

Job	1	2	3	4
Processing time	3	5	3	2
Deadline	4	6	7	8



Maximum lateness of all jobs in this schedule is 5

Minimizing lateness

Greedy choice: earliest-deadline-first without idle time

```
Input: n, t[1..n], d[1..n]

Minimize-lateness(n, t[], d[]):
    Sort jobs by deadlines such that d[1]≤d[2]≤ …≤d[n]
    ct = 0 //current time
    for j = 1 to n
        Assign job j to interval ct, ct+t[j]
        s[j] = ct
        f[j] = s[j] + t[j]
        ct = ct + t[j]
    return s[], f[]
```

Running time = O(nlogn) given unsorted jobs

Proof of greedy choice property

Given n jobs and their deadlines d_i, $d_1 \leq d_2 \leq \ldots \leq d_n$

Prove $\exists$ an optimal scheduling that processes job 1 first

<u>Proof by exchange argument</u>

<u>Key idea</u>: suppose OPT is an optimal solution. Modify OPT into another optimal scheduling that processes job 1 first.

If OPT processes job 1 first, done

If not, suppose job 1 is the i^{th} being processed

Let OPT' = OPT but with the i-1^{th} and i^{th} swapped

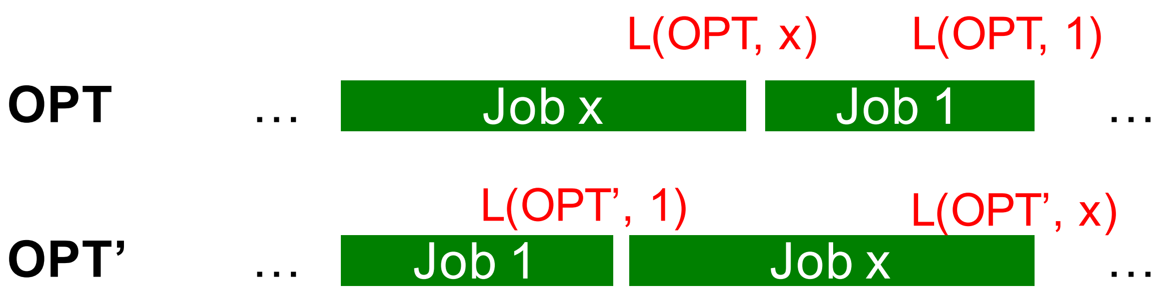
Prove that L(OPT') ≤ L(OPT)

Proof of greedy choice property

Prove that L(OPT’) ≤ L(OPT)

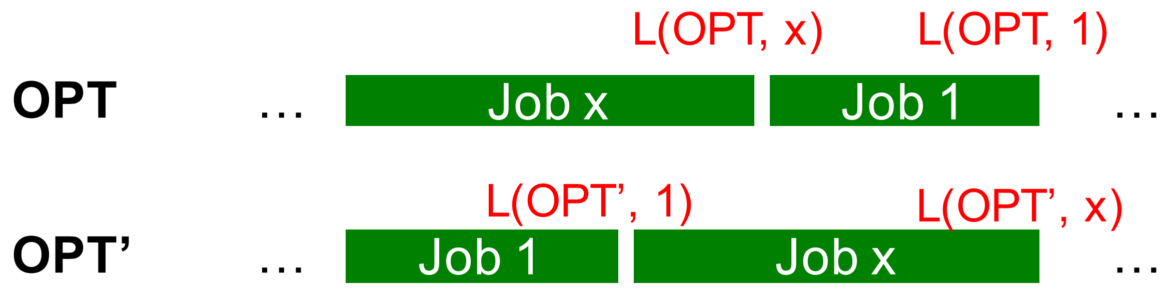
<=> Prove max{L(OPT’, 1), L(OPT’, x)} ≤ max{L(OPT, x), L(OPT, 1)}

<=> Since L(OPT’, 1) ≤ L(OPT, 1), prove that L(OPT’, x) ≤ L(OPT, 1)



Proof of greedy choice property

Prove that $L(OPT', x) \leq L(OPT, 1)$



<u>If job x is not late in OPT'</u>:
$L(OPT', x) = 0$

<u>If job x is late in OPT'</u>:

$$\begin{aligned} L(OPT', x) &= f(OPT', x) - d_x \\ &= f(OPT, 1) - d_x \\ &\leq f(OPT, 1) - d_1 \\ &= L(OPT, 1) \end{aligned}$$

Can we generalized this property?
Prove that there is no "inversion"

Proof of no inversions

Given n jobs and their deadlines d_i, $d_1 \le d_2 \le \ldots \le d_n$

Prove that $\exists$ an optimal scheduling without *inversions*

- Jobs x and y are inverted if $d_x > d_y$ but x is scheduled before y

<u>Proof by exchange argument</u>:

If OPT has no inversions, done

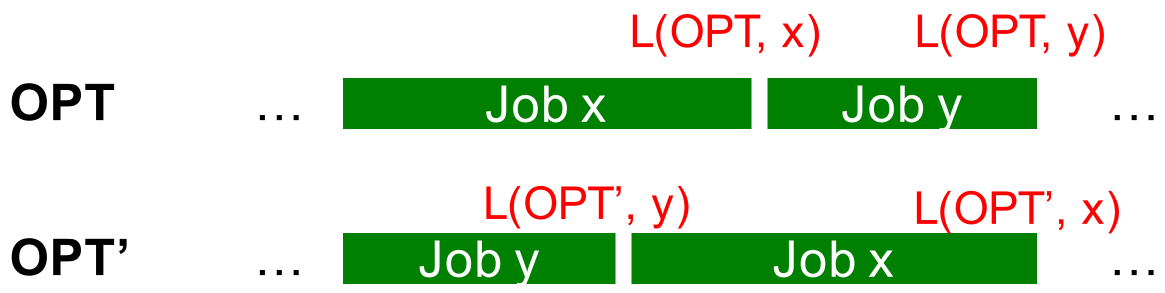
If not, suppose in OPT i-1th and ith jobs are inverted

Let OPT' = OPT but with the i-1th and ith swapped

Prove that L(OPT') ≤ L(OPT)

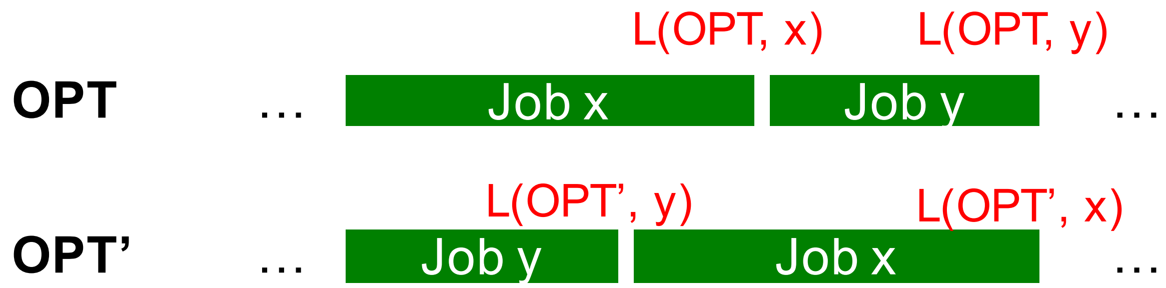
<=> Prove that max{L(OPT', y), L(OPT', x)} ≤ max{L(OPT, x), L(OPT, y)}

<=> Since L(OPT', y) ≤ L(OPT, y), prove that L(OPT', x) ≤ L(OPT, y)



Proof of no inversions

Prove that L(OPT’, x) ≤ L(OPT, y) when $d_y < d_x$



<u>If job x is not late in OPT’</u>:

L(OPT’, x) = 0

<u>If job x is late in OPT’:</u>

$$
\begin{aligned}
L(OPT', x) &= f(OPT', x) - d_x \\
&= f(OPT, y) - d_x \\
&\le f(OPT, y) - d_y \\
&= L(OPT, y)
\end{aligned}
$$

This immediately proves that earliest-deadline-first is optimal!
條條大路通羅馬 ☺

Matroid and Greedy Methods

Ch. 16.4

(optional)

Matroid (擬陣)

A combinatorial structure that generalizes the concept of linear independence

A ***matroid*** is an ordered pair $M = (S, \mathcal{I})$ satisfying the following conditions.

1. S is a finite set.
2. $\mathcal{I}$ is a nonempty family of subsets of S, called the ***independent*** subsets of S, such that if $B \in \mathcal{I}$ and $A \subseteq B$, then $A \in \mathcal{I}$. We say that $\mathcal{I}$ is ***hereditary*** if it satisfies this property. Note that the empty set $\emptyset$ is necessarily a member of $\mathcal{I}$.
3. If $A \in \mathcal{I}$, $B \in \mathcal{I}$, and $|A| < |B|$, then there exists some element $x \in B - A$ such that $A \cup \{x\} \in \mathcal{I}$. We say that M satisfies the ***exchange property***.

More Terminology

Extension: Given a matroid $M = (S, I)$, we call an element $x \notin A$ an ***extension*** of $A \in I$ if we can add x to A while preserving independence.

Maximal: If A is an independent subset in a matroid M, we say that A is ***maximal*** if it has no extensions.

Practice

Theorem 16.6 All maximal independent subsets in a matroid have the same size.

Weighted Matroid

We say that a matroid $M = (S, \mathcal{I})$ is ***weighted*** if it is associated with a weight function w that assigns a strictly positive weight $w(x)$ to each element $x \in S$. The weight function w extends to subsets of S by summation:

$$w(A) = \sum_{x \in A} w(x)$$

Many problems for which a greedy approach provides optimal solutions can be formulated in terms of **finding a maximum-weight independent subset in a weighted matroid**.

- maximum-weight independent subset = "optimal"
- Does not cover Huffman coding and interval scheduling

Greedily find an optimal subset on a weighted matroid

GREEDY(M, w)

$A = \emptyset$
sort $M.S$ into monotonically decreasing order by weight w
for each $x \in M.S$, taken in monotonically decreasing order by weight $w(x)$
    if $A \cup \{x\} \in M.\mathcal{I}$
        $A = A \cup \{x\}$
return A

Works for any weighted matroid!
Time complexity = O(n lgn + n f(n)), where f(n) is the time for checking independence

More Proofs

Lemma 16.7 Matroids exhibit the greedy-choice property

Lemma 16.10 Matroids exhibit the optimal-substructure property

Theorem 16.11 Correctness of the greedy algorithm on matroids

A task-scheduling problem as a matroid

Ch. 16.5

Scheduling unit-time tasks with deadlines and penalties

Given a set of unit-time tasks with deadlines and penalties, schedule **all** tasks for a single processor to **minimize total penalties**

Example:

Task (S)	1	2	3	4	5	6	7
Deadline	4	2	4	3	1	4	6
Penalty	70	60	50	40	30	20	10

Less-penalty-first strategy: Penalty = 90
Is this optimal?

Scheduling unit-time tasks with deadlines and penalties

Given a set of unit-time tasks with deadlines and penalties, schedule **all** tasks for a single processor to **minimize total penalties**

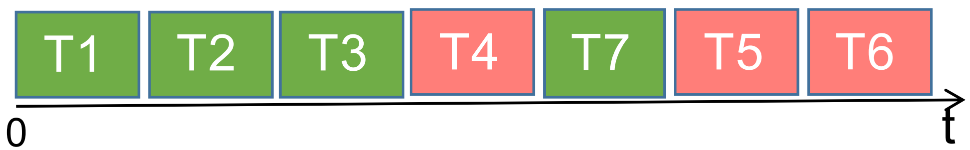
- $S = \{T_1, T_2, \ldots, T_n\}$, n unit-time tasks
- d_j = deadline of task j
- w_j = penalty of doing task j *after* the deadline

<u>**Goal**</u>: find a schedule (a permutation) of S that minimizes total penalty

Observations

<u>**Observation 1**</u> Given any schedule for a set of unit-time tasks *S*, we can always transform it into an **early-first form** without changing the penalty.

Task (S)	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Deadline	4	2	4	3	1	4	6
Penalty	70	60	50	40	30	20	10

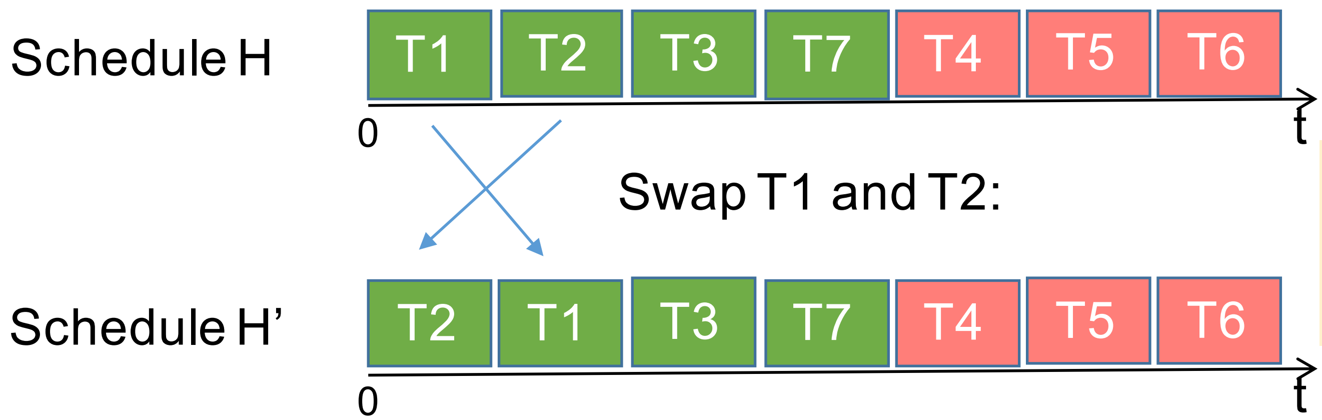


Swap T4 and T7:

Observations

<u>**Observation 2**</u> Given any schedule for a set of unit-time tasks S, we can always rearrange the *early* tasks into an order of **monotonically increasing deadlines** without changing the penalty.

Task (S)	1	2	3	4	5	6	7
Deadline	4	2	4	3	1	4	6
Penalty	70	60	50	40	30	20	10



After swapping T_1 and T_2, we still ensure that $f(H', T_1) = f(H, T_2) < d_2 < d_1$

Modeling as a matriod

Based on these observations, the problem of finding an optimal schedule is reduced to finding a set A of tasks to be *early* in the optimal schedule.

Matriod? We can view this as a weighted matriod $M = (S, I)$ where

- S is the set of tasks
- weights are the penalties
- I are the set of all *independent* sets of tasks
- Minimize penalty of *late* tasks = maximize penalty of *early* tasks
- Still need to prove the **hereditary** & **exchange** properties!

How to check whether a set is independent?

Lemma 16.12
For any set of tasks A, the following statements are equivalent.

1. The set A is independent.
2. For $t = 0, 1, 2, \ldots, n$, we have $N_t(A) \le t$.
3. If the tasks in A are scheduled in order of monotonically increasing deadlines, then no task is late.

A set *A* is **independent** if there exists a schedule for A such that no tasks are late.

$N_t(A)$ = number of tasks in *A* whose deadline is *t* or earlier.

Practice Prove that (1), (2), (3) are equivalent.

Theorem 16.13

If S is a set of unit-time tasks with deadlines, and I is the set of all independent sets of tasks, then the corresponding system $M = (S, I)$ is a matroid.

<u>Proof</u>

Hereditary: every subset of an independent set of tasks is still independent

Exchange property:

1. Find the largest k s.t. $N_t(B) \leq N_t(A)$

$N_t(.)$	0	1	2	3	4	5	6	7
Set A	0	1	2	2	2	2	2	2
Set B	0	0	1	2	2	2	3	3

Set A: T2 (d_2=2), T5 (d_5=1)

Set B: T2 (d_2=2), T4 (d_4=3), T7 (d_7=6)

2. Let x be a task in B - A whose deadline is k +1
3. A U {x} is still independent because…

What did we learn about greedy algorithms?

Greedy algorithms are easy to design one, hard to prove correctness.

Unlike DP, a greedy algorithm makes a greedy choice before solving the resulting subproblem.

Greedy-choice property: Making locally optimal (greedy) choices leads to a globally optimal solution

Optimal substructure: An optimal solution to the problem contains within it optimal solutions to subproblems