Algorithm Design and Analysis
Greedy Algorithms (2)

http://ada.miulab.tw slido: #ADA2021

Yun-Nung (Vivian) Chen

國立臺灣大學
National Taiwan University

Outline

- Greedy Algorithms
- Greedy #1: Activity-Selection / Interval Scheduling
- Greedy #2: Coin Changing
- Greedy #3: Huffman Codes
- Greedy #4: Fractional Knapsack Problem
- Greedy #5: Breakpoint Selection
- Greedy #6: Task-Scheduling
- Greedy #7: Scheduling to Minimize Lateness

Algorithm Design Strategy

- Do not focus on “specific algorithms”
- But “some strategies” to “design” algorithms

- First Skill: Divide-and-Conquer (各個擊破/分治)
- Second Skill: Dynamic Programming (動態規劃)
- Third Skill: Greedy (貪婪法則)

Greedy #4:
Fractional Knapsack Problem

Textbook Exercise 16.2-2

Knapsack Problem

- Input: n items where i-th item has value v_i and weighs w_i (v_i and w_i are positive integers)
- Output: the <u>maximum value</u> for the knapsack with capacity of W

- Variants of knapsack problem
  - 0-1 Knapsack Problem: 每項物品只能拿一個
  - Unbounded Knapsack Problem: 每項物品可以拿多個
  - Multidimensional Knapsack Problem: 背包空間有限
  - Multiple-Choice Knapsack Problem: 每一類物品最多拿一個
  - Fractional Knapsack Problem: 物品可以只拿部分

Knapsack Problem

- Input: n items where i-th item has value v_i and weighs w_i (v_i and w_i are positive integers)
- Output: the maximum value for the knapsack with capacity of W

- Variants of knapsack problem
  - 0-1 Knapsack Problem: 每項物品只能拿一個
  - Unbounded Knapsack Problem: 每項物品可以拿多個
  - Multidimensional Knapsack Problem: 背包空間有限
  - Multiple-Choice Knapsack Problem: 每一類物品最多拿一個
  - **Fractional Knapsack Problem: 物品可以只拿部分**

Fractional Knapsack Problem

- Input: n items where i-th item has value v_i and weighs w_i (v_i and w_i are positive integers)
- Output: the maximum value for the knapsack with capacity of W, where we can take **any fraction of items**

- Greedy algorithm: at each iteration, choose the item with the highest $\frac{v_i}{w_i}$ and continue when $W - w_i > 0$

Step 1: Cast Optimization Problem

Fractional Knapsack Problem
Input: n items where i-th item has value v_i and weighs w_i
Output: the max value within W capacity, where we can take **any fraction of items**

- Subproblems
  - `F-KP(i, w)`: fractional knapsack problem within w capacity for the first i items
  - Goal: `F-KP(n, W)`

Step 2: Prove Optimal Substructure

Fractional Knapsack Problem
Input: n items where i-th item has value v_i and weighs w_i
Output: the max value within W capacity, where we can take **any fraction of items**

- Suppose OPT is an optimal solution to `F-KP(i, w)`, there are 2 cases:
  - Case 1: full/partial item i in OPT
    - Remove w' of item i from OPT is an optimal solution of `F-KP(i - 1, w – w')`
  - Case 2: item i not in OPT
    - OPT is an optimal solution of `F-KP(i - 1, w)`

Step 3: Prove Greedy-Choice Property

> **Fractional Knapsack Problem**
> Input: n items where i-th item has value v_i and weighs w_i
> Output: the max value within W capacity, where we can take **any fraction of items**

- Greedy choice: select the item with the highest $\frac{v_i}{w_i}$
- Proof via contradiction ($j = \underset{i}{\operatorname{argmax}} \frac{v_i}{w_i}$)
  - Assume that there is no OPT including this greedy choice
    - If $W \leq w_j$, we can replace all items in OPT with item j
    - If $W > w_j$, we can replace any item weighting w_j in OPT with item j
  - The total value must be equal or higher, because item j has the highest $\frac{v_i}{w_i}$

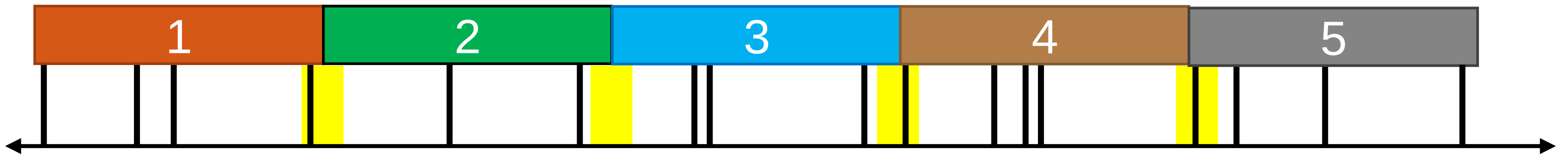
Do other knapsack problems have this property?

Greedy #5: Breakpoint Selection

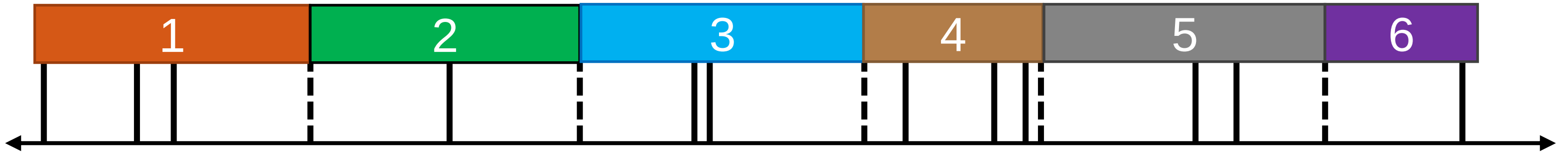
Breakpoint Selection Problem

- Input: a planned route with $n+1$ gas stations $b_0, \ldots, b_n$; the car can go at most C after refueling at a breakpoint
- Output: a refueling schedule ($b_0 \rightarrow b_n$) that minimizes the number of stops

Ideally: stop when out of gas



Actually: may not be able to find the gas station when out of gas



- Greedy algorithm: go as far as you can before refueling

Step 1: Cast Optimization Problem

Breakpoint Selection Problem
Input: $n+1$ breakpoints $b_0, \ldots, b_n$; gas storage is C
Output: a refueling schedule (b_0→b_n) that minimizes the number of stops

- Subproblems
  - `B(i)`: breakpoint selection problem from b_i to b_n
  - Goal: `B(0)`

Step 2: Prove Optimal Substructure

Breakpoint Selection Problem
Input: $n+1$ breakpoints $b_0, \ldots, b_n$; gas storage is C
Output: a refueling schedule ($b_0 \rightarrow b_n$) that minimizes the number of stops

- Suppose OPT is an optimal solution to `B(i)` where j is the largest index satisfying $b_j - b_i \leq C$, there are $j-i$ cases
  - Case 1: stop at b_{i+1}
    - OPT\\{b_{i+1}} is an optimal solution of `B(i + 1)`
  - Case 2: stop at b_{i+2}
    - OPT\\{b_{i+2}} is an optimal solution of `B(i + 2)`

  ⋮
  - Case $j-i$: stop at b_j
    - OPT\\{b_j} is an optimal solution of `B(j)`

$$B_i = \min_{i<k\leq j}(1 + B_k)$$

Step 3: Prove Greedy-Choice Property

Breakpoint Selection Problem
Input: $n + 1$ breakpoints $b_0, \dots, b_n$; gas storage is C
Output: a refueling schedule ($b_0 \rightarrow b_n$) that minimizes the number of stops

- Greedy choice: go as far as you can before refueling (select b_j)
- Proof via contradiction
  - Assume that there is no OPT including this greedy choice (after b_i then stop at b_k, $k \neq j$)
    - If $k > j$, we cannot stop at b_k due to out of gas
    - If $k < j$, we can replace the stop at b_k with the stop at b_j
  - The total value must be equal or higher, because we refuel later ($b_j > b_k$)

$$B_i = \min_{i<k\leq j}(1 + B_k) \Rightarrow B_i = 1 + B_j$$

Pseudo Code

Breakpoint Selection Problem
Input: $n+1$ breakpoints $b_0, \ldots, b_n$; gas storage is C
Output: a refueling schedule (b_0→b_n) that minimizes the number of stops

```
BP-Select(C, b)
  Sort(b) s.t. b[0] < b[1] < … < b[n]
  p = 0
  S = {0}
  for i = 1 to n - 1
    if b[i + 1] - b[p] > C
      if i == p
        return “no solution”
      A = A U {i}
      p = i
return A
```

$$T(n) = \Theta(n \log n)$$

Greedy #6: Task-Scheduling

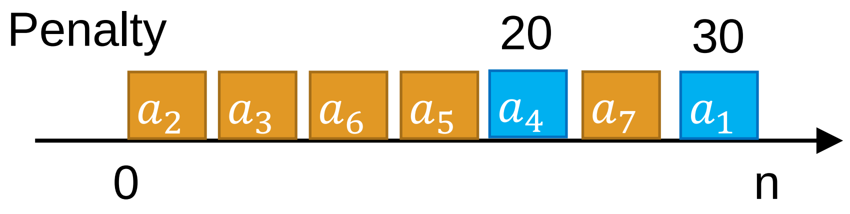
Textbook Exercise 16.2-2

Task-Scheduling Problem

- Input: a finite set $S = \{a_1, a_2, \ldots, a_n\}$ of n ***unit-time tasks***, their corresponding integer deadlines $d_1, d_2, \ldots, d_n$ $(1 \le d_i \le n)$, and nonnegative penalties $w_1, w_2, \ldots, w_n$ if a_i is not finished by time d_i

Job	1	2	3	4	5	6	7
Deadline (d_i)	1	2	3	4	4	4	6
Penalty (w_i)	30	60	40	20	50	70	10

- Output: a schedule that minimizes the total penalty

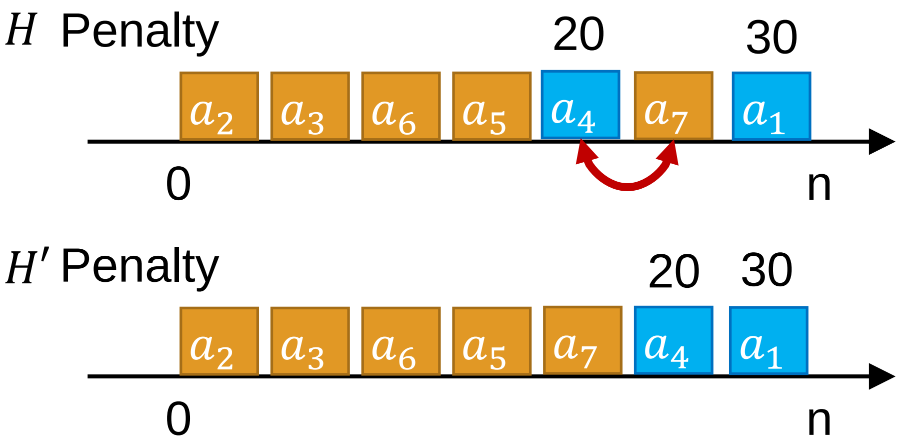


Task-Scheduling Problem

Task-Scheduling Problem
Input: n tasks with their deadlines $d_1, d_2, \ldots, d_n$ and penalties $w_1, w_2, \ldots, w_n$
Output: the schedule that minimizes the total penalty

- Let a schedule H is the OPT
  - A task a_i is <u>late</u> in H if $f(H, i) > d_i$
  - A task a_i is <u>early</u> in H if $f(H, i) \le d_i$
  - We can have an **early-first** schedule H' with the same total penalty (OPT)

Task	1	2	3	4	5	6	7
d_i	1	2	3	4	4	4	6
w_i	30	60	40	20	50	70	10



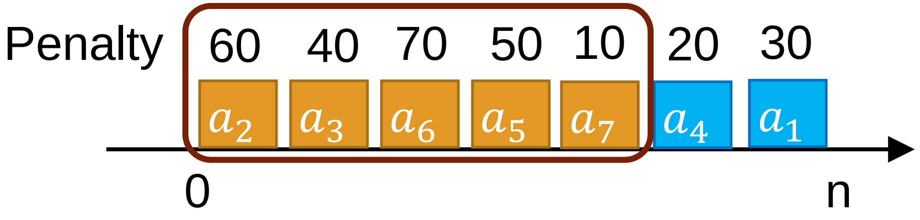
If the late task proceeds the early task, switching them makes the early one earlier and late one still late

Possible Greedy Choices

Task-Scheduling Problem
Input: n tasks with their deadlines $d_1, d_2, \dots, d_n$ and penalties $w_1, w_2, \dots, w_n$
Output: the schedule that minimizes the total penalty

- Rethink the problem: "*maximize the total penalty for the set of early tasks*"

Task	1	2	3	4	5	6	7
d_i	1	2	3	4	4	4	6
w_i	30	60	40	20	50	70	10

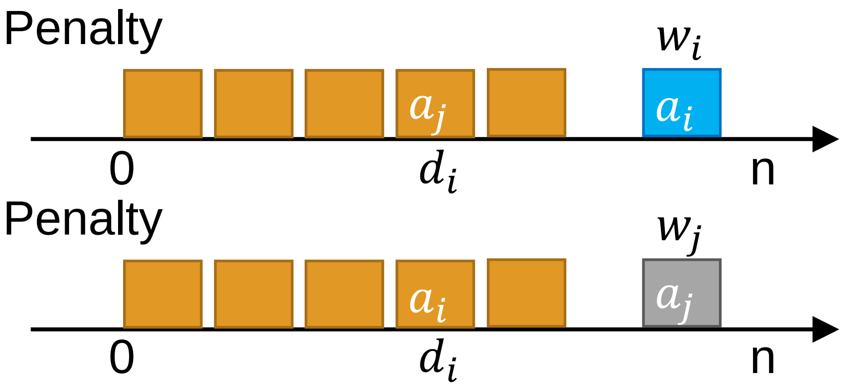


- Greedy idea
  - Largest-penalty-first w/o idle time?
  - Earliest-deadline-first w/o idle time?

Prove Correctness

Task-Scheduling Problem
Input: n tasks with their deadlines $d_1, d_2, \dots, d_n$ and penalties $w_1, w_2, \dots, w_n$
Output: the schedule that minimizes the total penalty

- Greedy choice: select the **largest-penalty** task into the early set if *feasible*
- Proof via contradiction
  - Assume that there is no OPT including this greedy choice
    - If OPT processes a_i after d_i, we can switch a_j and a_i into OPT'
  - The maximum penalty must be equal or lower, because $w_i \geq w_j$



$w_i \geq w_k$ for all a_k in the early set

Prove Correctness

Task-Scheduling Problem
Input: n tasks with their deadlines $d_1, d_2, \dots, d_n$ and penalties $w_1, w_2, \dots, w_n$
Output: the schedule that minimizes the total penalty

- Greedy algorithm

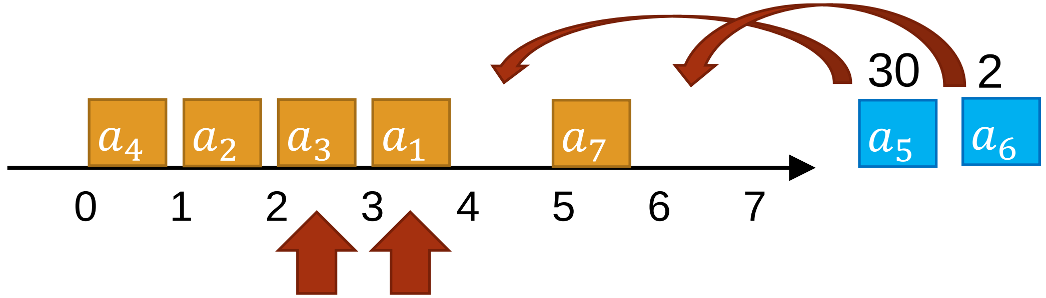
```
Task-Scheduling(n, d[], w[])
  sort tasks by penalties s.t. w[1] ≥ w[2] ≥ … ≥ w[n]
  for i = 1 to n
    find the latest available index j <= d[i]
    if j > 0
      A = A ∪ {i}
      mark index j unavailable
return A // the set of early tasks
```

$$T(n) = O(n^2)$$

Practice: reduce the time for finding the latest available index

Example Illustration

Job	1	2	3	4	5	6	7
Deadline (d_i)	4	2	4	3	1	4	6
Penalty (w_i)	70	60	50	40	30	20	10



Total penalty = 30 + 20 = 50

Practice: how about the greedy algorithm using “earliest-deadline-first”

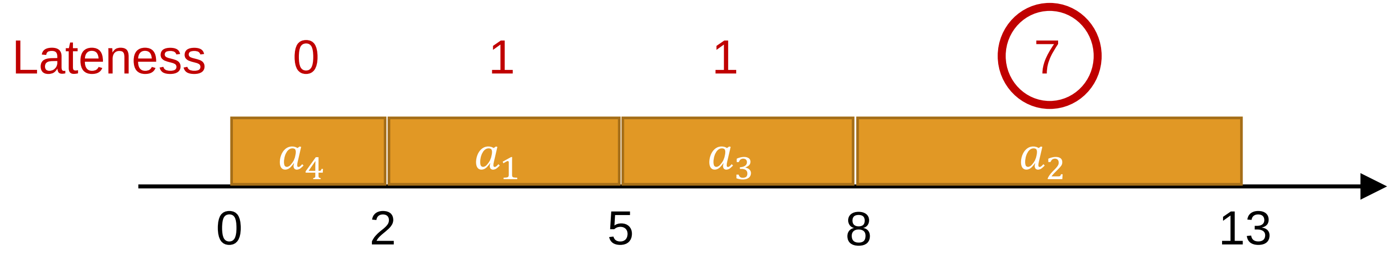
Greedy #7: Scheduling to Minimize Lateness

Scheduling to Minimize Lateness

- Input: a finite set $S = \{a_1, a_2, \dots, a_n\}$ of n tasks, their processing time $t_1, t_2, \dots, t_n$, and integer deadlines $d_1, d_2, \dots, d_n$

Job	1	2	3	4
Processing Time (t_i)	3	5	3	2
Deadline (d_i)	4	6	7	8

- Output: a schedule that minimizes the maximum lateness



Scheduling to Minimize Lateness

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \ldots, t_n$, and deadlines $d_1, d_2, \ldots, d_n$
Output: the schedule that minimizes the maximum lateness

- Let a schedule H contains $s(H,j)$ and $f(H,j)$ as the start time and finish time of job j
  - $f(H,j) - s(H,j) = t_j$
  - Lateness of job j in H is $L(H,j) = \max\{0, f(H,j) - d_j\}$
- The goal is to minimize $\max_j L(H,j) = \max_j \{0, f(H,j) - d_j\}$

Possible Greedy Choices

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \dots, t_n$, and deadlines $d_1, d_2, \dots, d_n$
Output: the schedule that minimizes the maximum lateness

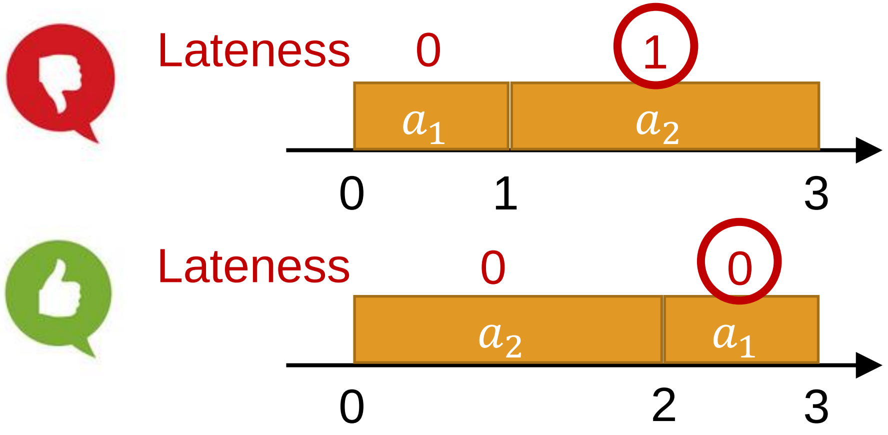
- Greedy idea
  - Shortest-processing-time-first w/o idle time?
  - Earliest-deadline-first w/o idle time?

Practice: prove that any schedule w/ idle is not optimal

Possible Greedy Choices

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \dots, t_n$, and deadlines $d_1, d_2, \dots, d_n$
Output: the schedule that minimizes the maximum lateness

- Idea
  - **Shortest-processing-time-first w/o idle time?**



Job	1	2
Processing Time (t_i)	1	2
Deadline (d_i)	10	2

Possible Greedy Choices

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \dots, t_n$, and deadlines $d_1, d_2, \dots, d_n$
Output: the schedule that minimizes the maximum lateness

- Idea
  - **Earliest-deadline-first w/o idle time?**
- Greedy algorithm

```
Min-Lateness(n, t[], d[])
  sort tasks by deadlines s.t. 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[]
```

$$T(n) = \Theta(n \log n)$$

Prove Correctness – Greedy-Choice Property

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \dots, t_n$, and deadlines $d_1, d_2, \dots, d_n$
Output: the schedule that minimizes the maximum lateness

- Greedy choice: first select the task with the earliest deadline
- Proof via contradiction
  - Assume that there is no OPT including this greedy choice
    - If OPT processes a_1 as the i-th task (a_k), we can switch a_k and a_1 into OPT'
  - The maximum lateness must be equal or lower → $L(\mathrm{OPT}') \leq L(\mathrm{OPT})$

exchange argument

Prove Correctness – Greedy-Choice Property

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \ldots, t_n$, and deadlines $d_1, d_2, \ldots, d_n$
Output: the schedule that minimizes the maximum lateness

- $L(\text{OPT'}) \leq L(\text{OPT})$

$$\iff \max(L(\text{OPT'}, 1), L(\text{OPT'}, k)) \leq \max(L(\text{OPT}, k), L(\text{OPT}, 1))$$

$$\iff \max(L(\text{OPT'}, 1), L(\text{OPT'}, k)) \leq L(\text{OPT}, 1)$$

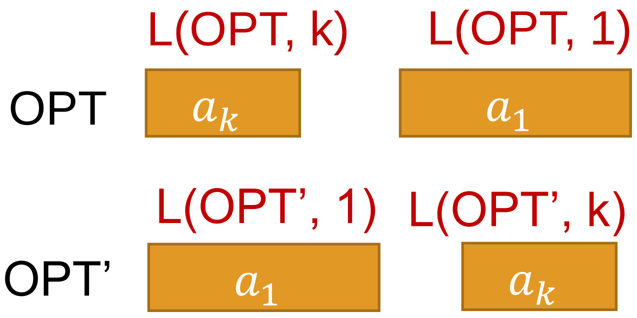
$$\iff L(\text{OPT'}, k) \leq L(\text{OPT}, 1) \because L(\text{OPT'}, 1) \leq L(\text{OPT}, 1)$$

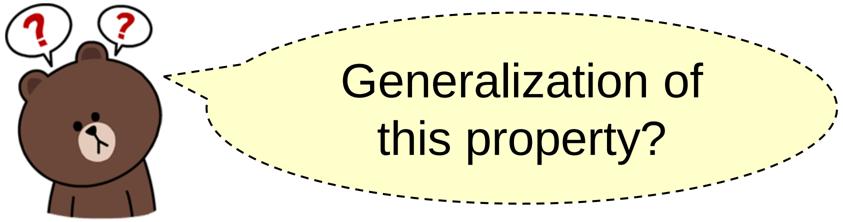
If a_k is not late in OPT':

$$L(\text{OPT'}, k) = 0$$

If a_k is late in OPT':

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





Prove Correctness – No Inversions

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \dots, t_n$, and deadlines $d_1, d_2, \dots, d_n$
Output: the schedule that minimizes the maximum lateness

- There is an optimal scheduling w/o *inversions* given $d_1 \le d_2 \le \cdots \le d_n$
  - a_i and a_j are *inverted* if $d_i < d_j$ but a_j is scheduled before a_i
- Proof via contradiction
  - Assume that OPT has a_i and a_j that are inverted
  - Let OPT' = OPT but a_i and a_j are swapped
  - OPT' is equal or better than OPT → $L(\mathrm{OPT}') \le L(\mathrm{OPT})$

Prove Correctness – No Inversions

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \ldots, t_n$, and deadlines $d_1, d_2, \ldots, d_n$
Output: the schedule that minimizes the maximum lateness

- $L(\text{OPT'}) \le L(\text{OPT})$

$$\iff \max(L(\text{OPT'}, i), L(\text{OPT'}, j)) \le \max(L(\text{OPT}, j), L(\text{OPT}, i))$$

$$\iff \max(L(\text{OPT'}, i), L(\text{OPT'}, j)) \le L(\text{OPT}, i) \because d_i < d_j$$

$$\iff L(\text{OPT'}, j) \le L(\text{OPT}, i) \because L(\text{OPT'}, i) \le L(\text{OPT}, i)$$

<u>If a_j is not late in OPT':</u>

$$L(\text{OPT'}, j) = 0$$

<u>If a_j is late in OPT':</u>

$$\begin{aligned} L(\text{OPT'}, j) &= f(\text{OPT'}, j) - d_j \\ &= f(\text{OPT}, i) - d_j \\ &\le f(\text{OPT}, i) - d_i \\ &= L(\text{OPT}, i) \end{aligned}$$

Optimal Solution = Greedy Choice + Subproblem Solution

L(OPT, j) L(OPT, i)
OPT a_j …… a_i

L(OPT', i) L(OPT', j)
OPT' a_i …… a_j

The earliest-deadline-first greedy algorithm is optimal

Concluding Remarks

- "Greedy": always makes the choice that looks best at the moment in the hope that this choice will lead to a globally optimal solution
- When to use greedy
  - Whether the problem has optimal substructure
  - Whether we can make a greedy choice and remain only one subproblem
  - Common for optimization problem

Optimal Solution = Greedy Choice + Subproblem Solution

- Prove for correctness
  - Optimal substructure
  - Greedy choice property

Question?

Important announcement will be sent to @ntu.edu.tw mailbox & post to the course website

Course Website: http://ada.miulab.tw
Email: ada-ta@csie.ntu.edu.tw