[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.11.12 목요일)

서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

[주제 5]
Greedy Methods

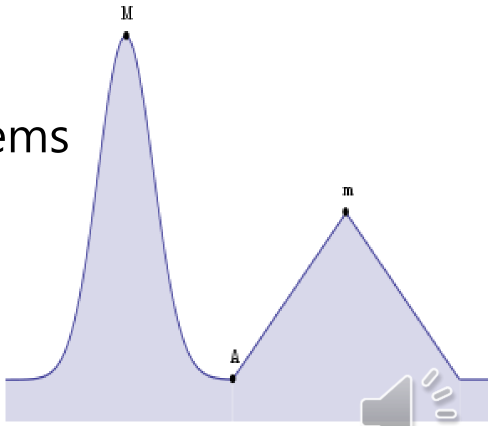
The Greedy Method

- A technique to follow the problem-solving heuristic of making the locally optimal choice at each stage.

- **Strategy**
  - Make the choice that appears best at each moment!
  - ✓ It is hoped to arrive at a globally optimal solution by making a locally optimal choice.

- **Pros and cons**
  - Simple and straightforward to design an algorithm.
  - Does not guarantee the optimal solution to all problems
    - **Local maximum versus global maximum**

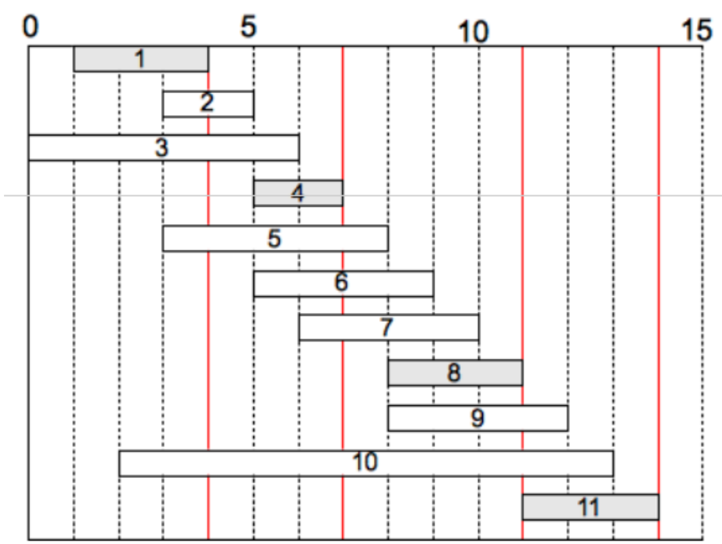
https://en.wikipedia.org/wiki/Greedy_algorithm



Maximum Non-overlapping Intervals

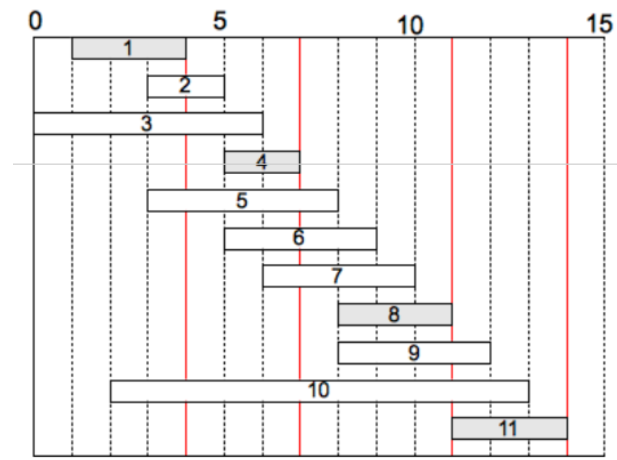
- **Problem:** Let $A = \{a_1, a_2, \cdots, a_n\}$ be a set of n activities, where a_i has start times s_i and finish times f_i $(0 \le s_i < f_i < \infty)$. If selected, activity a_i takes place druing the time interval $[s_i, f_i)$. Two activities a_i and a_j are called *compatible* if the time intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap. Now, select a largest set S of mutually compatible activities.

- **Example** $A = \{a_1, a_2, \cdots, a_n\}$ with $a_i = [s_i, f_i)$ for $0 \le s_i < f_i < \infty$

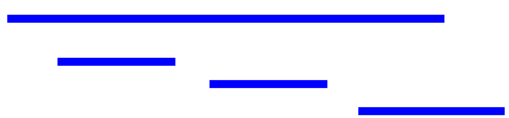


$$A = \{a_1, a_2, \cdots, a_n\} \text{ with } a_i = [s_i, f_i) \text{ for } 0 \le s_i < f_i < \infty$$

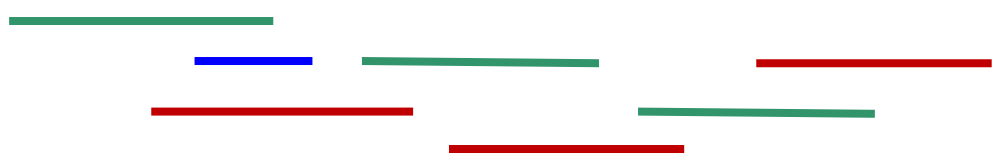
- **Possible strategies for choosing activities**
  - **Longest one first**
  - **Shortest one first**



  - **Earliest start first**

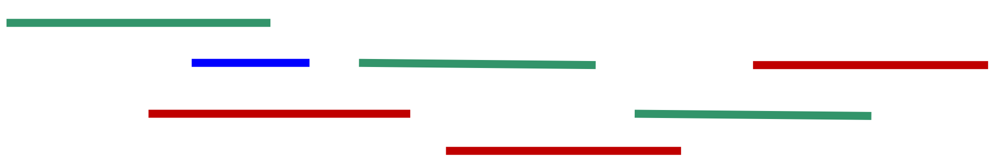


  - **Earliest finish first**



귀류법 (Proof by contradiction)

- **Correctness of "Earliest-finish-first"-based algorithm**
  - **Assertion 1:** For any set A of n activities, there always exists an optimal solution S that contains a_1 with the earliest finish time.
  - **Assertion 2:** If S is an optimal solution for A containing a_1, $S^* = S \setminus \{a_1\}$ is an optimal solution for $A^* = \{a_i \in A \,|\, s_i \geq f_1\}$.
    - Selecting a_1 reduces the problem to finding an optimal solution for activities not overlapping with a_1.

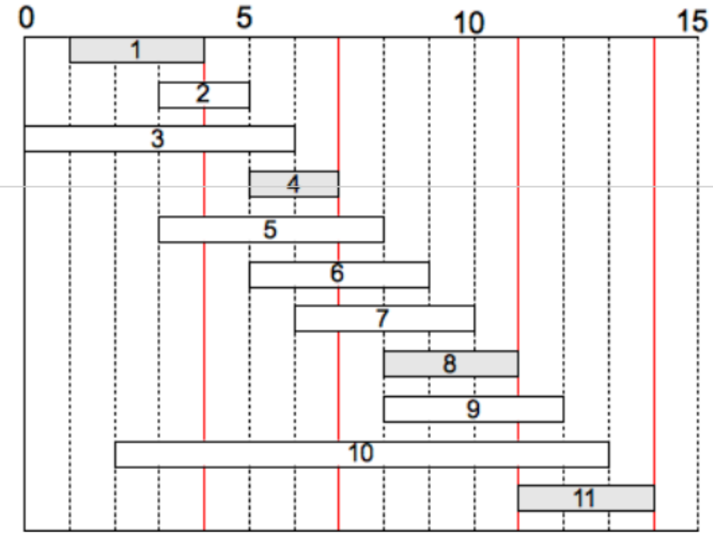


$$A = \{a_1, a_2, \cdots, a_n\} \text{ with } a_i = [s_i, f_i) \text{ for } 0 \leq s_i < f_i < \infty$$

- **Greedy algorithm**

Input: $A = \{a_1, a_2, \cdots, a_n\}$ with $a_i = [s_i, f_i)$ for $0 \le s_i < f_i < \infty$

1. Sort the activities so that $f_1 \le f_2 \le f_3 \le \cdots \le f_{n-1} \le f_n$.
2. $S = \{a_1\};$
3. $k = 1;$
4. **for** $j = 2$ **to** n
5.   **if** $(s_j \ge f_k)$
6. $S = S \cup \{a_j\};$
7. $k = j;$
8. **return** $S;$



→ $O(n \log n + n) = O(n \log n)$ time

Scheduling: Minimizing Total Time in the System

- **Problem**
  - Consider a system in which a server is about to serve n clients. Let $T = \{t_1, t_2, \ldots, t_n\}$ be a set of positive numbers, where $\boldsymbol{t_i}$ is the estimated **time-to-completion** for the ith client. What is the optimal order of service where **the total (wait+service) time in the system is minimized**?
  - Hair stylist with waiting clients, pending operations on a shared hard disk, etc.

- **Example**
  - $T = \{t_1, t_2, t_3\} = \{5, 10, 4\}$

Schedule	Total Time in the System
[1, 2, 3]	5 + (5 + 10) + (5 + 10 + 4) = 39
[1, 3, 2]	33
[2, 1, 3]	10 + (10 + 5) + (10 + 5 + 4) = 44
[2, 3, 1]	43
[3, 1, 2]	☞ 4 + (4 + 5) + (4 + 5 + 10) = 32
[3, 2, 1]	37

- **A *naïve* approach**
  - Enumerate all possible schedules of service, and select the optimal one. → $O(n!)$
- **A greedy approach**
  - **Algorithm:** Sort T in nondecreasing order to get the optimal schedule. → $O(n \log n)$
  - **Correctness:** Does the greedy approach always find a schedule that minimizes the total time in the system?

    귀류법 (Proof by contradiction)

    - Let $S = [s_1, s_2, \ldots, s_n]$ be an optimal schedule, and $C(S)$ be the total time for S.
    - $$\begin{aligned} C(S) &= s_1 + (s_1 + s_2) + (s_1 + s_2 + s_3) + \cdots + (s_1 + s_2 + \cdots + s_n) \\ &= n \cdot s_1 + (n-1) \cdot s_2 + \cdots + 2 \cdot s_{n-1} + 1 \cdot s_n \\ &= \sum_{i=1}^{n} (n+1-i) \cdot s_i = (n+1) \sum_{i=1}^{n} s_i - \sum_{i=1}^{n} i \cdot s_i \end{aligned}$$
    - If they are not scheduled in nondecreasing order, then, for at least one i $(1 \le i \le n-1)$, $s_i > s_{i+1}$.
    - Now consider the schedule $S' = [s_1, s_2, \ldots, s_{i+1}, s_i, \ldots, s_n]$ that is obtained by interchanging s_i and s_{i+1}.
    - Then, $C(S) - C(S') = (i \cdot s_{i+1} + (i+1) \cdot s_i) - (i \cdot s_i + (i+1) \cdot s_{i+1}) = s_i - s_{i+1} > 0$. Therefore, …

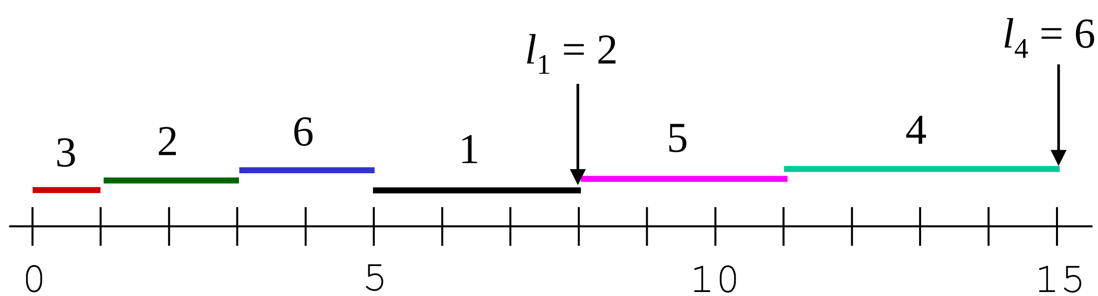
Scheduling: Minimizing Lateness

- **Problem**
  - Let $J = \{1, 2, \ldots, n\}$ be a set of jobs to be served by a single processor.
  - The ith job takes $\boldsymbol{t_i}$ units of processing time, and is due at time $\boldsymbol{d_i}$.
  - When the ith job starts at time s_i, its **lateness** $l_i = \max\{0, s_i + t_i - d_i\}$.
  - **Goal:** Find a schedule S so as to minimize the maximum lateness

    $L = \max\{l_i\}$.

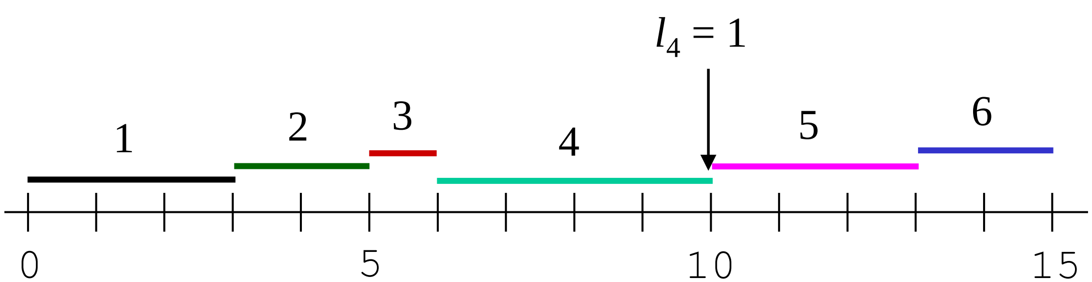
- **Example**
  - $S = \{3, 2, 6, 1, 5, 4\}$ → maximum lateness = 6

Job	t_i	d_i
1	3	6
2	2	8
3	1	9
4	4	9
5	3	14
6	2	15



- **Possible greedy approaches**
  - Sort jobs in nondecreasing order of processing time t_i:
    Shortest Jobs First (?)
    $(t_1 = 2, d_1 = 50),\ (t_2 = 10, d_2 = 11)$
  - Sort jobs in nondecreasing order of slack d_i - t_i:
    Smallest Slack-Time First (?)
    $(t_1 = 1, d_1 = 3),\ (t_2 = 7, d_2 = 8)$
  - Sort jobs in nondecreasing order of deadline d_i:
    Earliest Deadline First (O)
    - ✓ An optimal schedule $S = \{1, 2, 3, 4, 5, 6\}$ → maximum lateness = 1

Job	t_i	d_i
1	3	6
2	2	8
3	1	9
4	4	9
5	3	14
6	2	15



- **Correctness of "Earliest-deadline-first"-based algorithm**
  - **사실**
    1. 만약 주어진 schedule에 **inversion**이 있을 경우, 최소한 연달아 schedule된 두 개의 inversion된 job이 있음.
       - **Inversion**이란 deadline 관점에서 봤을 때 서로 순서가 뒤 바뀐 두 개의 job의 쌍을 말함.
    2. 연달아 있는 inversion 상태의 두 개의 job의 순서를 서로 바꿀 경우, maximum lateness를 증가시키지 않음.

$$[\cdots, d_j, d_i, \cdots] \ (d_i < d_j) \rightarrow [\cdots, d_i, d_j, \cdots]$$

$$< (j,i) \rightarrow (i,j) > (d_i < d_j)$$
$$l'_k = l_k \text{ for all } k \neq i, j$$
$$l'_i \leq l_i$$
$$l'_j = f'_j - d_j = f_i - d_j \leq f_i - d_i = l_i$$

  - **증명**
    1. S를 최소 개수의 inversion을 가지는 최적의 schedule이라 가정.
    2. 만약 S에 inversion이 없다면, 위의 방법으로 구한 schedule과 동일.
    3. 만약 S에 inversion이 있다면, 이 경우 연달아 있는 inversion된 두 job의 순서를 서로 바꾸면, 결과로 발생하는 schedule S'는 maximum lateness를 증가시키지 않음으로 역시 또 다른 최적의 schedule임.
    4. 그러나 S'는 S 보다 inversion의 개수가 적음. 이는 S에 대한 가정에 대한 모순. 따라서 S에는 inversion이 없고 따라서 이는 위의 방법으로 구한 schedule과 동일함.