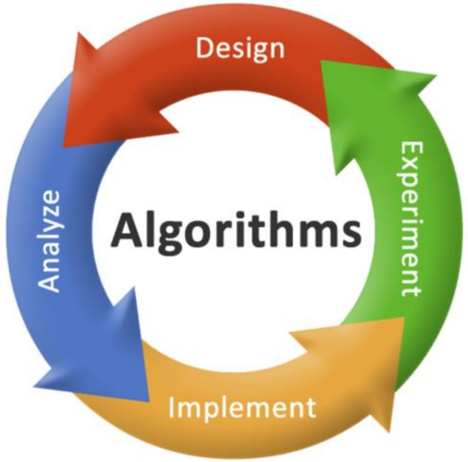


Algorithm Design and Analysis

算法设计与分析

Chapter 3-1: Greedy

张乾坤

贪心算法

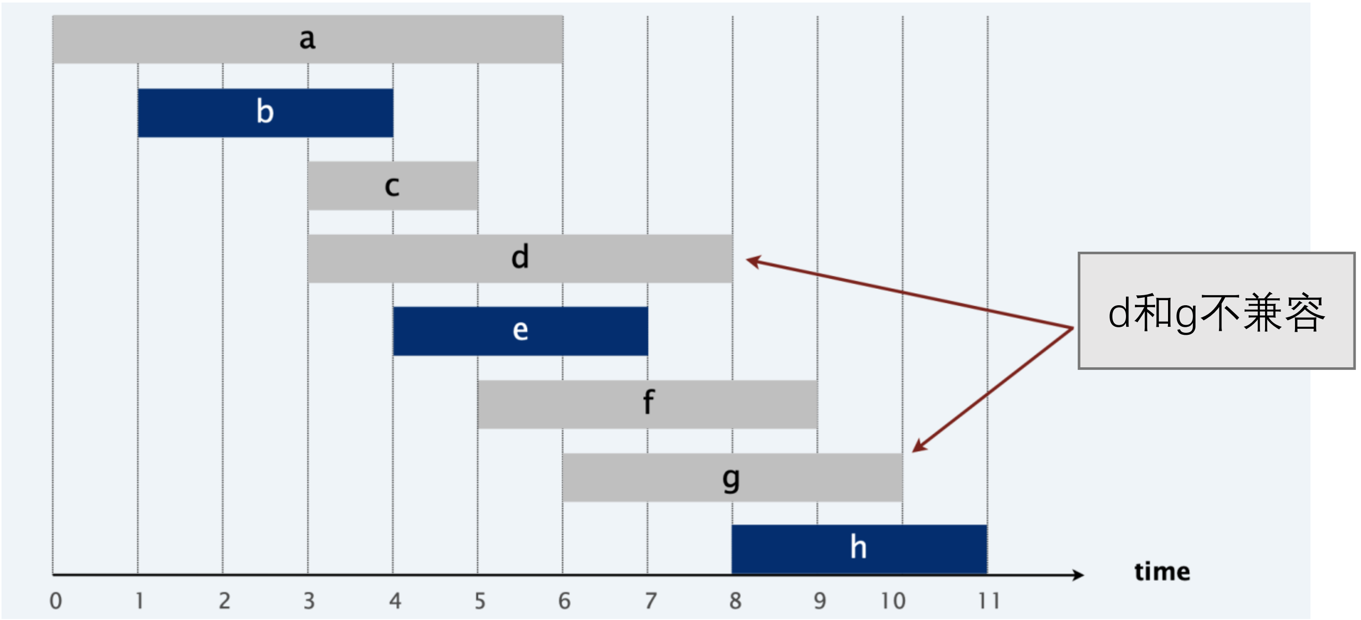
- 间隔调度
- 间隔划分
- 最小化延迟的调度

贪心算法

- 间隔调度
- 间隔划分
- 最小化延迟的调度

区间调度(Interval Scheduling)

- 输入若干个任务，任务j在s_j开始，在f_j结束
- 两个任务是兼容的如果他们的执行时间没有重叠
- 目标：寻找最大的互相兼容的任务子集



区间调度(Interval Scheduling)

- 思考题：考虑按照某种顺序排列任务。在排好序的任务队列中，如果当前任务与已经加入输出任务子集的任务相兼容，则将当前任务加入输出。请问下列的哪种排列方法是最优的？

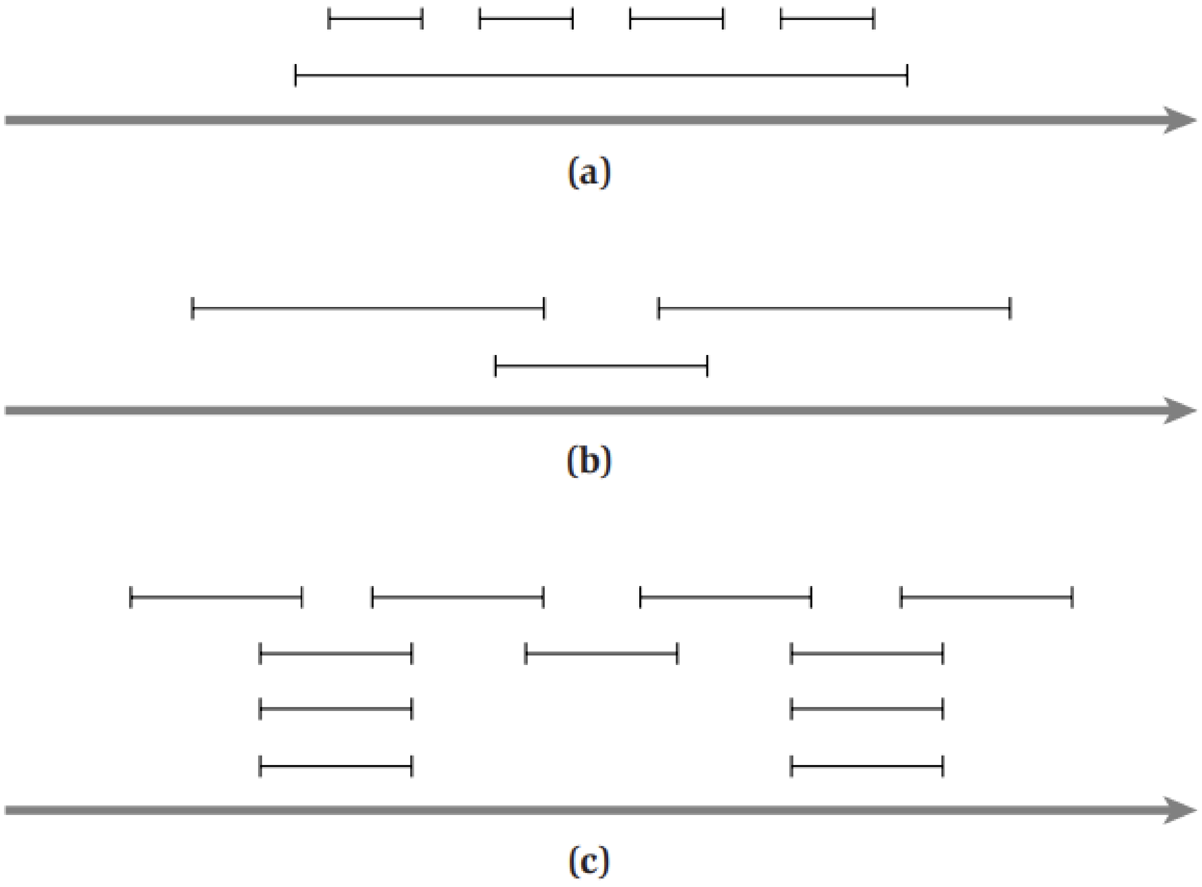
- A. [最早开始时间] 考虑按照递增s_j排列的任务
- B. [最早结束时间] 考虑按照递增f_j排列的任务
- C. [最短任务间隔] 考虑按照递增$f_j - s_j$排列的任务
- D. [最少冲突区间] 选择与其他任务不兼容数量最少的任务

失败的贪心策略

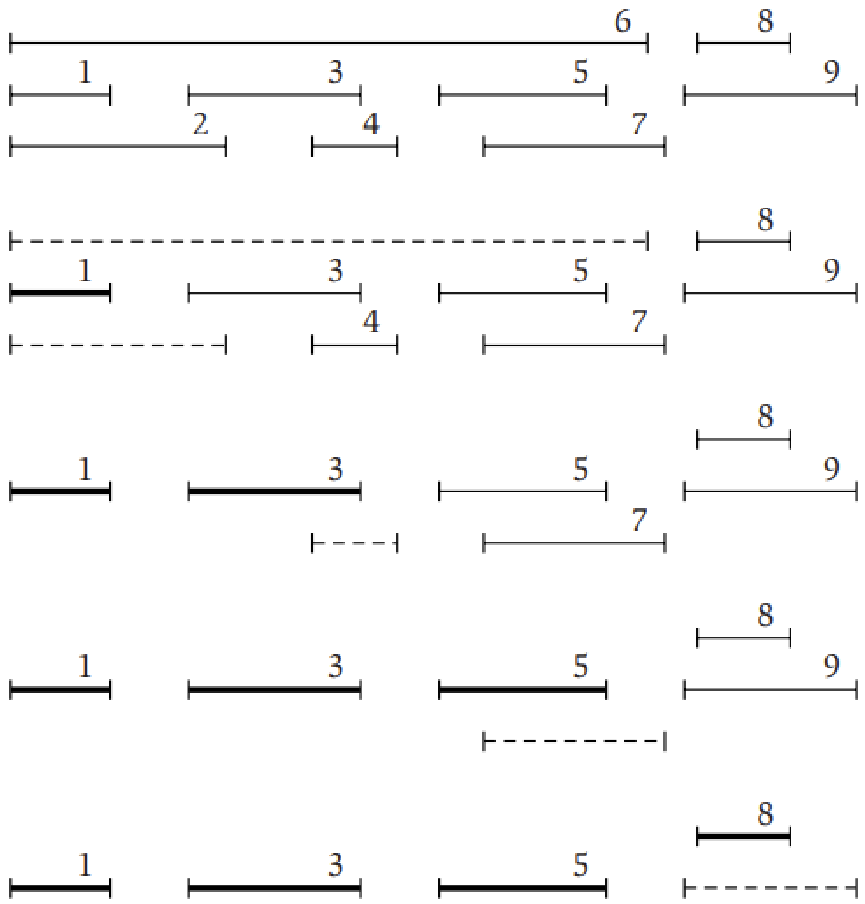
最早开始时间

最短任务间隔

最少冲突区间



区间调度：最早完成时间优先算法



EARLIEST-FINISH-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT jobs by finish times and renumber so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

$S \leftarrow \varnothing$. ← set of jobs selected

FOR $j = 1$ TO n

IF (job j is compatible with S)

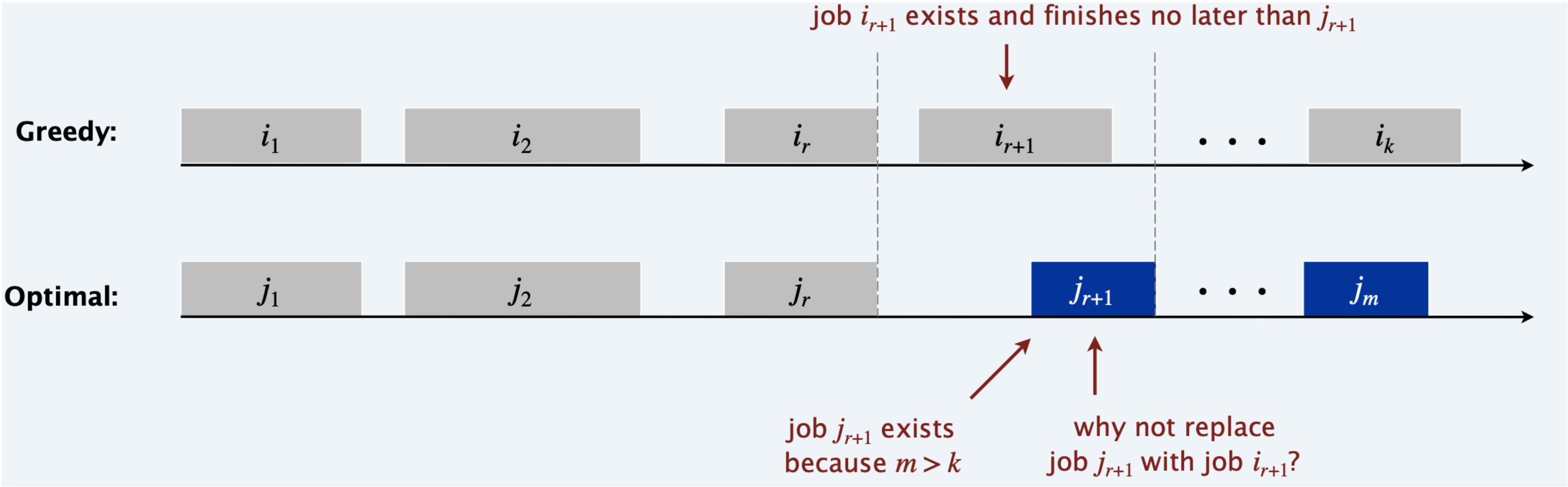
$S \leftarrow S \cup \{ j \}$.

RETURN S.

- 最早完成时间优先算法运行时间为$O(nlogn)$

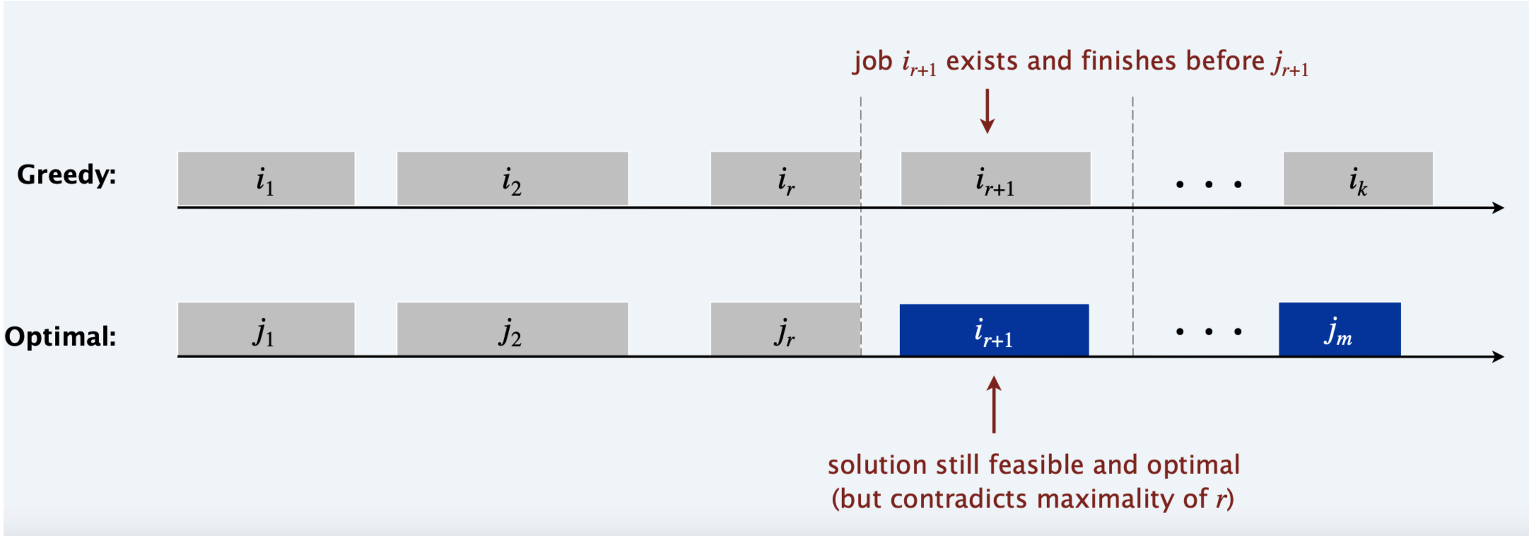
最早完成时间优先是最优策略

- 证明[反证法]:
  - 假设贪心算法不是最优的
  - 用$i_1, i_2, .. i_k$代表被贪心算法选出的任务子集
  - 用$j_1, j_2, \ldots j_m$代表最优解中的任务子集，其中$i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$, r是最大可以实现这个关系的下标



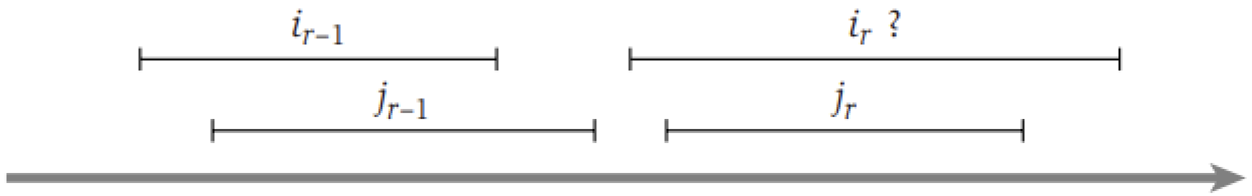
最早完成时间优先是最优策略

- 证明[反证法]:
  - 假设贪心算法不是最优的
  - 用$i_1, i_2, .. i_k$代表被贪心算法选出的任务子集
  - 用$j_1, j_2, \ldots j_m$代表最优解中的任务子集，其中$i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$, r是最大可以实现这个关系的下标



最早完成时间优先是最优策略

- 证明[保持领先]:
  - 设算法任务集$i_1, i_2, .. i_k$，最优解任务集$j_1, j_2, \ldots j_m$，目标证明$k = m$
  - 说明对任意的任务，$f(i_r) \le f(j_r)$[归纳法]
  - $r = 1$显然成立
  - 假设$r - 1$时成立，即 $f(i_{r-1}) \le f(j_{r-1})$



  - 若$f(i_r) > f(j_r)$，算法会选择j_r而不是i_r，会导致什么?

贪心算法（一）

- 思考题：假设每个任务有一个权重，我们的目标是找到具有最大权重的兼容任务子集。那么最早完成时间优先算法还是最优的吗？

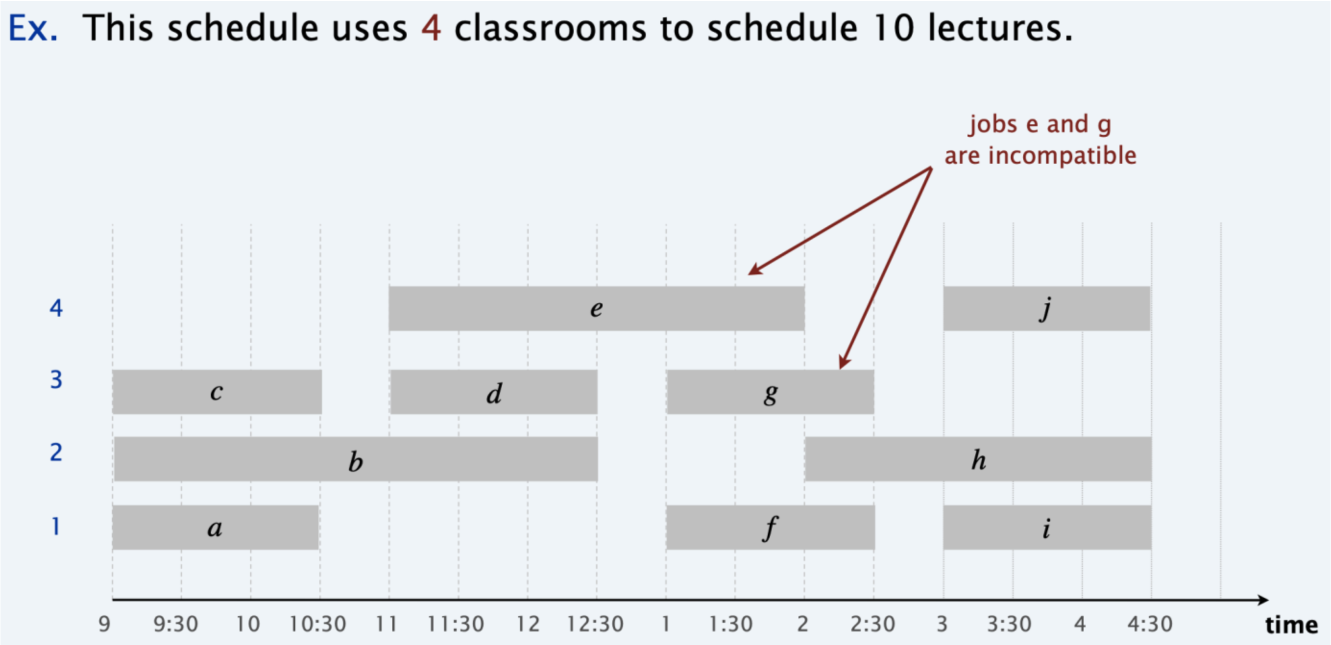
- A. 是的，因为贪心算法总是最优的
- B. 是的，因为相同的证明依旧成立
- C. 不是，因为相同的证明已经不成立了
- D. 不是，因为你可以给与具有最早完成时间的任务不兼容的任务很大的权重

贪心算法（一）

- 硬币交换
- 区间调度
- 区间分割
- 最小化延迟的调度

区间分割（Interval Partitioning）

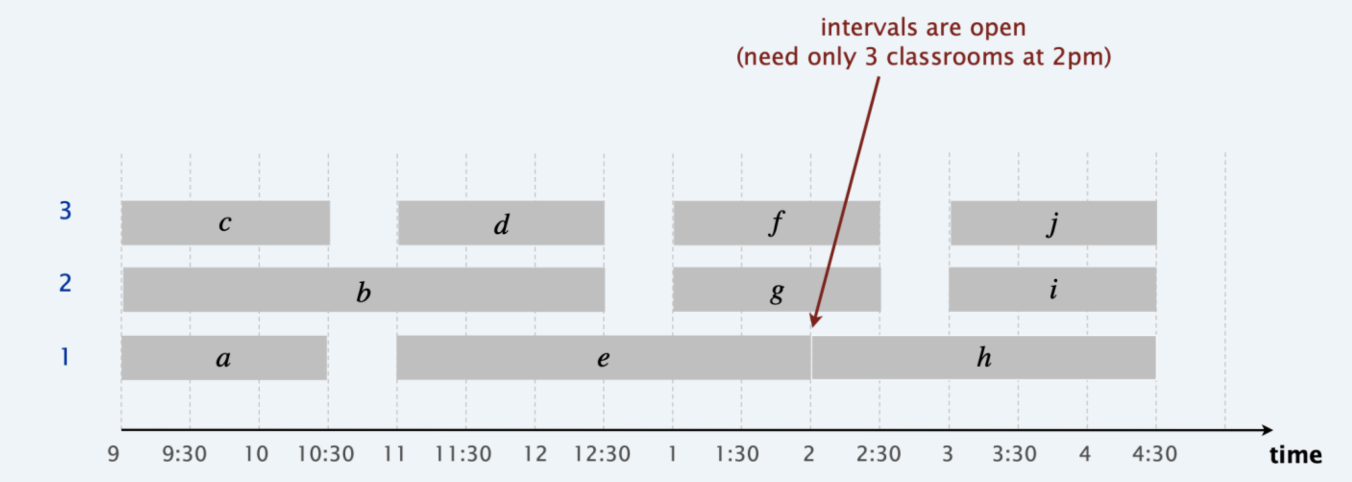
- 课程 j 在 s_j 开始，在 f_j 结束
- 目标：在没有两节课在同一时间在同一教室进行的前提下，用最少的教室来安排所有的可能



区间分割

- 课程 j 在s_j开始，在f_j结束
- 目标：在没有两节课在同一时间在同一教室进行的前提下，用最少的教室来安排所有的可能

Ex. This schedule uses 3 classrooms to schedule 10 lectures.



区间分割

- 思考题：按照某种顺序排列课程，依次将每一节课分到第一间可用的教室（如果无可用教室，就启用一间新的教室），下面的哪些排列方法是最优的？

- A. [最早开始时间] 按照s_j递增的顺序排列课程
- B. [最早结束时间] 按照f_j递增的顺序排列课程
- C. [最短课程间隔] 按照$f_j - s_j$递增的顺序排列课程
- D. 以上都不是最优的

区间分割：最早开始时间优先算法

EARLIEST-START-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT lectures by start times and renumber so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

$d \leftarrow 0$. ⟵ number of allocated classrooms

FOR $j = 1$ TO n

  IF (lecture j is compatible with some classroom)

    Schedule lecture j in any such classroom k.

  ELSE

    Allocate a new classroom $d + 1$.

    Schedule lecture j in classroom $d + 1$.

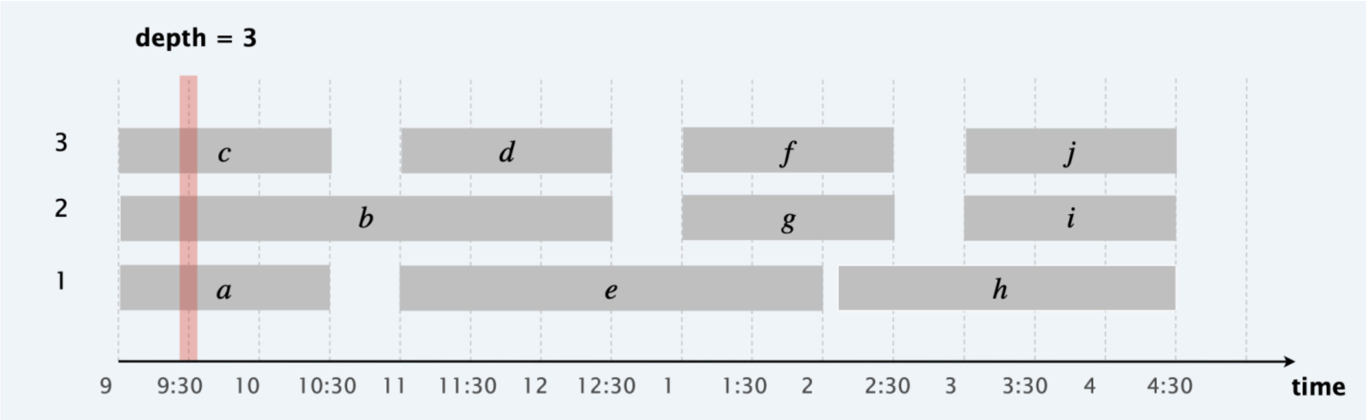
    $d \leftarrow d + 1$.

RETURN schedule.

- 最早开始时间优先算法运行时间为$O(nlogn)$

区间分割：最优解的下界

- 定义：一组开区间的深度是指包含给定点的最大区间的个数
- 需要的教室数量≥深度
- 问题：所需教室的最少数量是否总要等于深度?
- 答案：是的。此外，最早开始时间优先算法找到的教室数量恰好等于所有课程起止时间所对应的区间深度。



最早开始时间优先算法是最优的

- 最早开始时间优先算法不会将不兼容的课程安排在同一间教室
- 证明最优性：
  - 让d = 算法分配的教室数量
  - 教室d会被启用是因为需要将一门课程j安排到新的教室，即课程j与在其余$d-1$间教室中的课程都不兼容
  - 因此，这些课程都会在s_j之后完成
  - 因为按照开始时间排列课程，这些不兼容的课程的开始时间不会晚于s_j
  - 因此，在时刻$s_j+\varepsilon$, 有d门课都在进行
  - 结合需要的教室数量≥深度，所有可行的分配方案都至少使用d间教室

贪心算法（一）

- 硬币交换
- 区间调度
- 区间分割
- 最小化延迟的调度

最小化延迟调度

- 一种资源每一时刻只能处理一个任务
- 任务 j需要t_j的处理时间，且需要在时间d_j前完成
- 如果任务 j在时刻s_j开始，那它的结束时间为$f_j = s_j + t_j$
- 延迟：$\ell_j = \max\{0, f_j - d_j\}$
- 目标：安排和调度所有的任务来最小化最大延迟$L = \max_j \ell_j$

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

最小化延迟调度

- 思考题：按照某种顺序调度所有的任务。下列哪种顺序可以最小化最大延迟？

- A. [最短处理时间] 按照处理时间t_j递增的顺序
- B. [最早截止时间] 按照截止时间d_j递增的顺序
- C. [最小完成提前时间] 按照完成提前时间$d_j - t_j$递增的顺序
- D. 以上都不可以

最小化延迟调度

- 最短处理时间
- 两个任务： $t_1 = 1$和$d_1 = 100$； $t_2 = 10$和$d_2 = 10$

- 最小提前完成时间
- 两个任务： $t_1 = 1$和$d_1 = 2$； $t_2 = 10$和$d_2 = 10$

最小化延迟调度：最早截止时间优先

EARLIEST-DEADLINE-FIRST $(n, t_1, t_2, \ldots, t_n, d_1, d_2, \ldots, d_n)$

SORT jobs by due times and renumber so that $d_1 \leq d_2 \leq \ldots \leq d_n$.

$t \leftarrow 0.$

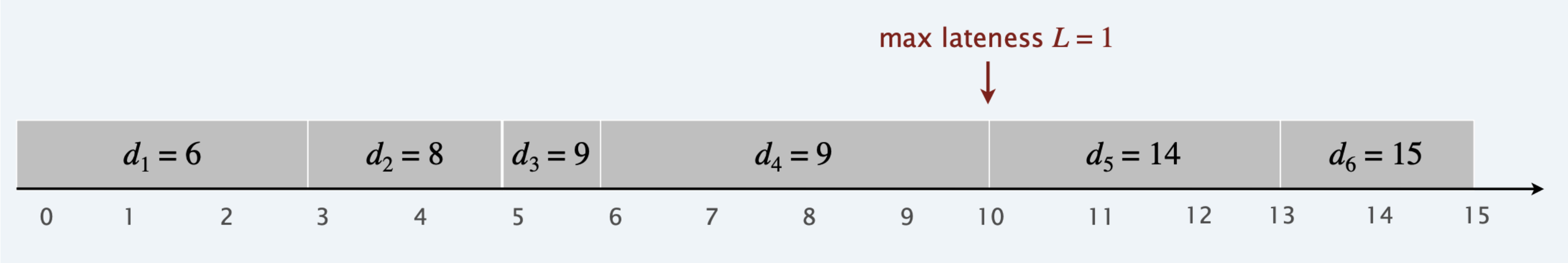
FOR $j = 1$ TO n

Assign job j to interval $[t, t + t_j]$.

$s_j \leftarrow t$; $f_j \leftarrow t + t_j$.

$t \leftarrow t + t_j$.

RETURN intervals $[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]$.



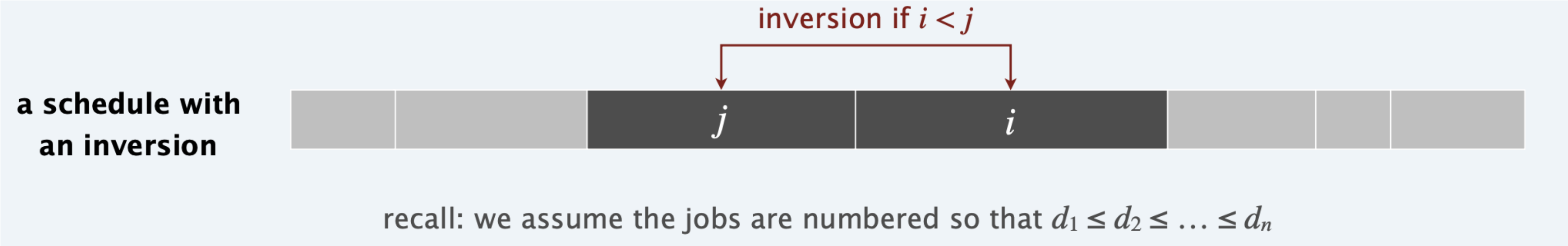
最小化延迟调度：无空闲时间

- 观察1：存在一个无空闲时间的最优调度算法

- 观察2：最早截止时间优先调度无空闲时间。

最小化延迟：倒置

- 定义：给定调度算法S，倒置是指对于一对任务$i, j: d_i < d_j$但是j在i之前被执行



- 观察 3: 最早截止时间优先算法是唯一无空闲时间和倒置的调度。

最小化延迟：倒置

- 观察 4：如果一个没有空闲时间的调度算法存在倒置，那么它有一个相邻倒置（两个倒置任务被连续执行）
- 证明：让i - j作为最相近的倒置

✓假设k是右侧邻接j的任务

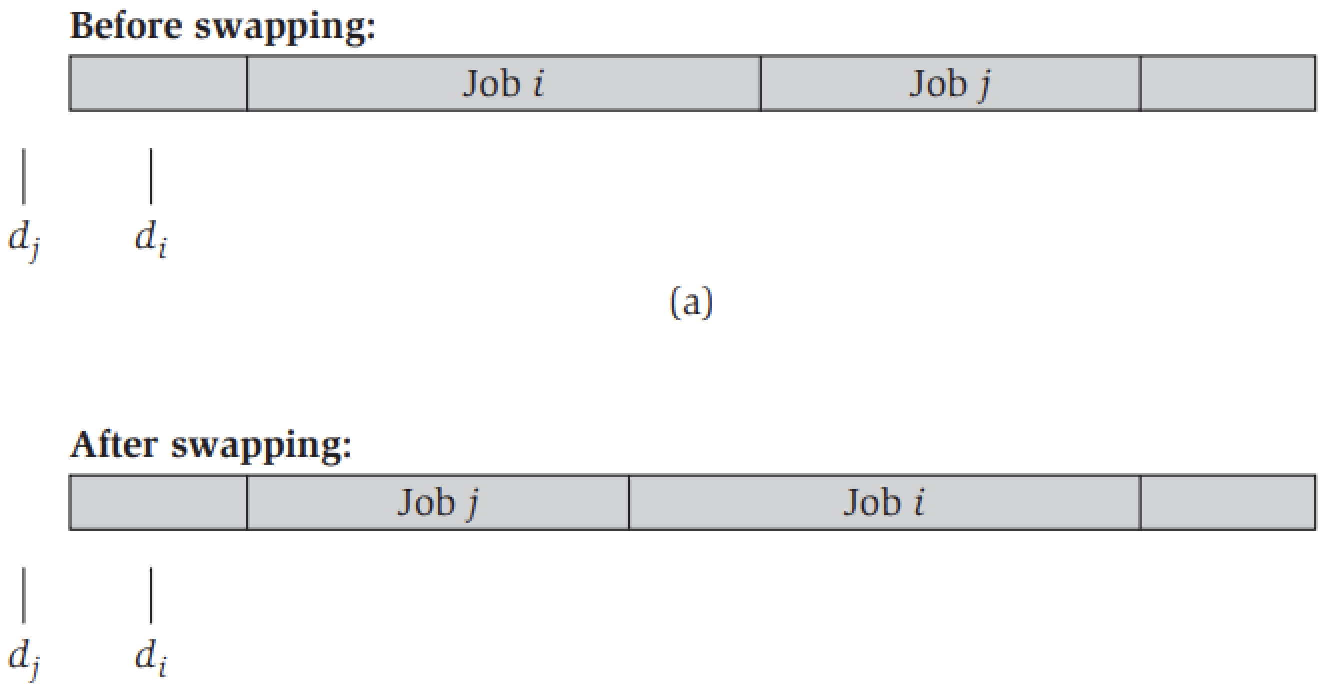
✓情况1:[$d_j > d_k$] 那么$j - k$是一个相邻倒置

✓情况2:[$d_j < d_k$]那么$i - k$是一组更相近的倒置因为$d_i < d_j < d_k$

最小化延迟：倒置

交换论证：

- 交换两个相邻倒置的任务i和j会减少1个倒置且**不会增大最大延迟**

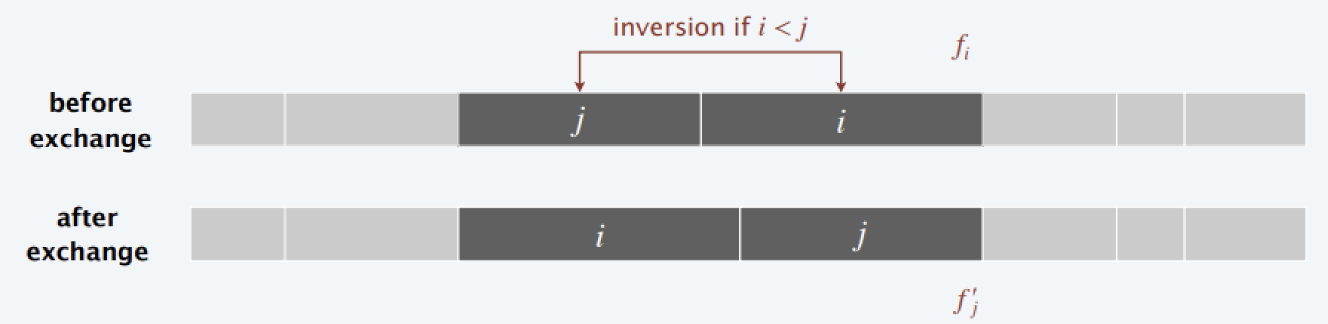


最小化延迟：倒置

交换论证：

- 交换两个相邻倒置的任务i和j会减少1个倒置且**不会增大最大延迟**
- 证明：用ℓ表示交换之前的延迟，用ℓ'表示交换之后的延迟。
  - $\ell'_k = \ell_k$, 对任意的$k \neq i, j$.
  - $\ell'_i \leq \ell_i$
  - 如果任务j超过了截止时间，

$$
\begin{aligned}
\ell'_j &= f'_j - d_j && \longleftarrow \text{definition} \\
&= f_i - d_j && \longleftarrow j \text{ now finishes at time } f_i \\
&\leq f_i - d_i && \longleftarrow i < j \;\Rightarrow\; d_i \leq d_j \\
&\leq \ell_i . && \longleftarrow \text{definition}
\end{aligned}
$$



最早截止时间优先算法是最优的

证明[反证法]

- 假设S^*是最优的且有着最少倒置的调度。
  - 假设S^*没有空闲时间 （观察 1）
  - 情况1[S^*没有倒置]，则$S = S^*$ （观察 3）
  - 情况2[S^*有倒置]
    - ✓ 让i - j表示一组相邻倒置 （观察 4）
    - ✓ 交换i，j会将倒置的数量减少1但不会增加延迟时间
    - ✓ 与S^*有最少倒置这一假设相矛盾

总结：贪心算法分析策略

- 贪心算法保持领先：通过在算法的每一步都证实贪心算法的策略不差于任意算法的策略
- 交换策略：在不降低算法质量的情况下，逐渐将任一解决方案转换为贪心算法所对应的解决方案。