第5讲 贪心算法 (1/2)

罗国杰

gluo@pku.edu.cn

2025年春季学期

算法设计与分析『实验班』

PKU-04833060

贪心法（Greedy Approach）

- 贪心法的设计思想
- 贪心法的正确性证明
- 对贪心法得不到最优解情况的处理
- 贪心法的典型应用
  - 最优前缀码
  - 最小生成树
  - 单源最短路径
- 拟阵相关的贪心法

"Greed, for lack of a better word, is good.
Greed is right. Greed works."

-- Gordon Gekko
(cast: Michael Douglas)
in Wall Street (1987)

活动选择问题

输入：$S = \{1, 2, \ldots, n\}$为n 项活动的集合

s_i, f_i 分别为活动 i 的开始和结束时间

活动 i 与 j 相容 当且仅当 $s_i \geq f_j$ 或 $s_j \geq f_i$

求最大的两两相容的活动集

策略1：（开始时间）排序使 $s_1 \leq s_2 \leq \ldots \leq s_n$，从前向后挑选

策略2：（活动时长）排序使 $f_1 - s_1 \leq f_2 - s_2 \leq \ldots \leq f_n - s_n$，从前向后挑选

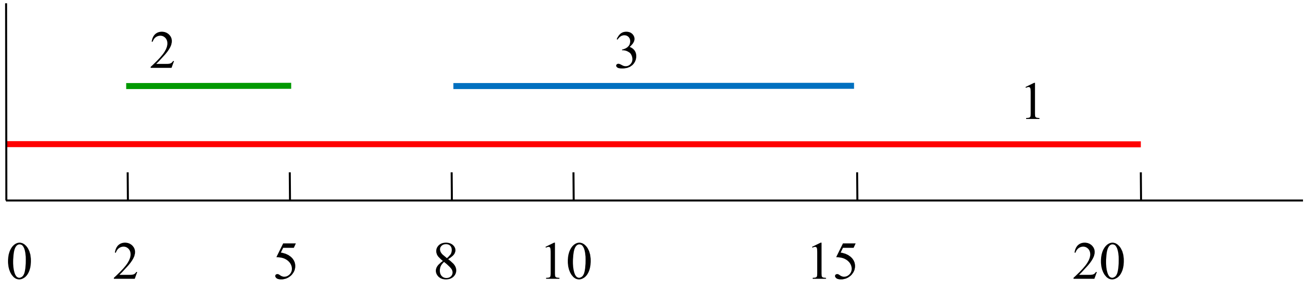
策略3：（结束时间）排序使 $f_1 \leq f_2 \leq \ldots \leq f_n$，从前向后挑选

按策略的排序依次检查活动，若当前活动满足相容性，则放进活动集

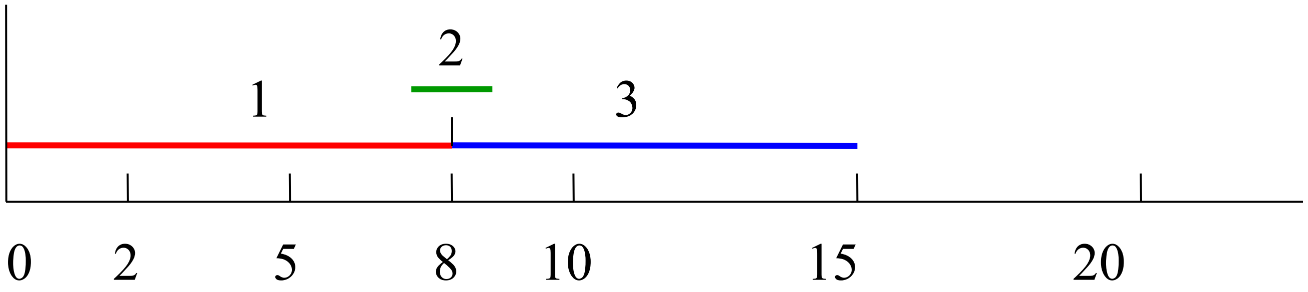
“活动选择”属于 Interval Scheduling 问题

活动选择问题：策略1和策略2的反例

策略1：$S=\{1, 2, 3\}$，$s_1=0, f_1=20, s_2=2, f_2=5, s_3=8, f_3=15$



策略2：$S=\{2, 3, 1\}$，$s_1=0, f_1=8, s_2=7, f_2=9, s_3=8, f_3=15$



活动选择问题： 按结束时间排序

算法 Greedy Select

1. $n \leftarrow length[S]$;
2. $A \leftarrow \{1\}$;
3. $j \leftarrow 1$;
4. for $i \leftarrow 2$ to n
5. do if $s_i \geq f_j$
6. then $A \leftarrow A \cup \{i\}$;
7. $j \leftarrow i$;
8. return A.

最后完成时间 $t = \max \{ f_k : k \in A \}$.

活动选择问题： 实例

输入

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

解为 $A = \{1, 4, 8, 11\}$ $t = 14$

活动选择问题： 正确性证明

证明方法：

(1) 归纳法：证明贪心法得到最优解

叙述一个描述正确性的命题

对算法步数归纳或者对问题规模归纳

(2) 交换论证：在保证最优性不变的前提下，从一个最优解进行逐步替换，最终得到贪心法的解

定理 算法Select 执行到第 k 步, 选择 k 项活动 $i_1= 1, i_2, \dots,$ i_k, 那么存在最优解 A 包含 $i_1=1, i_2, \dots, i_k$

根据定理：算法至多到第 n 步得到最优解

活动选择问题：归纳证明

归纳基础：证明存在包含活动 1 的最优解
归纳步骤：假设按照算法前 k 步选择都存在对应的最优解，证明第k+1步选择也存在对应的最优解

归纳步骤的证明思路

1. 算法第 k 步选择活动 $i_1=1, i_2, \ldots, i_k$，根据归纳假设，存在最优解

$$A = \{ i_1=1, i_2, \ldots, i_k \} \cup B$$

B是剩下的待选活动集 S' 的一个最优解

2. 由归纳基础，存在 S' 的最优解 B' 包含i_{k+1}
3. 由 $|B'|=|B|$ 知 $A' = \{i_1=1, i_2, \ldots, i_k\} \cup B'$ 最优
4. $A'=\{i_1=1, i_2, \ldots, i_k, i_{k+1}\} \cup (B' - \{i_{k+1}\})$最优.

活动选择问题： 归纳基础

设 $S=\{1,2,\ldots,n\}$是活动集，活动按结束时间递增顺序排序。

$k=1$，证明存在最优解包含活动 1。

任取最优解 A，A 中的活动按照截止时间递增的

顺序排列。如果 A 的第一个活动为 j，$j\neq1$，令

$$A' = (A-\{j\})\cup\{1\},$$

由于 $f_1 \leq f_j$，A' 也是最优解，且含有1。

活动选择问题：归纳步骤

假设命题对 k 为真，证明对 $k+1$ 也为真。

算法执行到第 k 步, 选择了活动 $i_1=1, i_2, \ldots, i_k$, 根据归纳假设存在最优解 A 包含 $i_1=1, i_2, \ldots, i_k$,

A中剩下的活动选自集合$S'=\{ i \mid i\in S, s_i \geq f_k\}$,且

$$A = \{ i_1, i_2, \ldots , i_k \}\cup B$$

B是S'的最优解.（若不然, S'的最优解为B^*, B^*的活动比 B 多，那么$B^*\cup\{1, i_2, \ldots , i_k\}$是 S 的最优解，且比 A的活动多, 与 A 的最优性矛盾.）

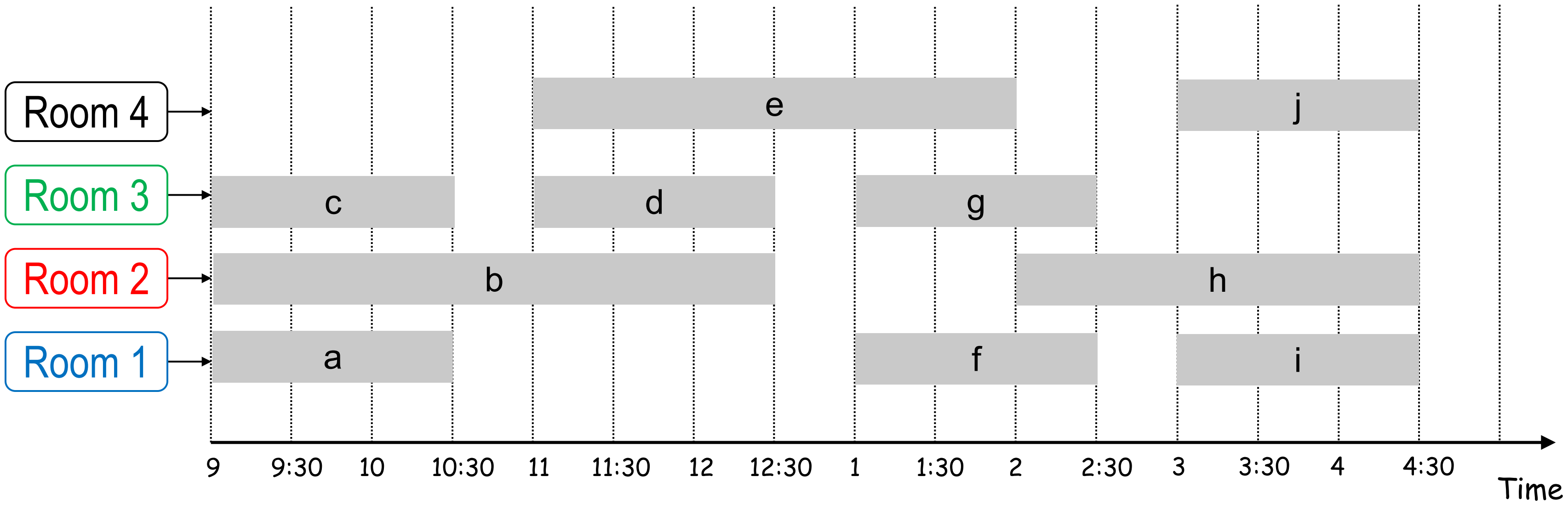
根据归纳基础，存在 S'的最优解B'含有S'中的第一个活动，设为i_{k+1}, 且$|B'|=|B|$, 于是

$$\{i_1,i_2,\ldots,i_k\}\cup B'=\{i_1,i_2,\ldots i_k,i_{k+1}\}\cup(B'-\{i_{k+1}\})$$

也是原问题的最优解.

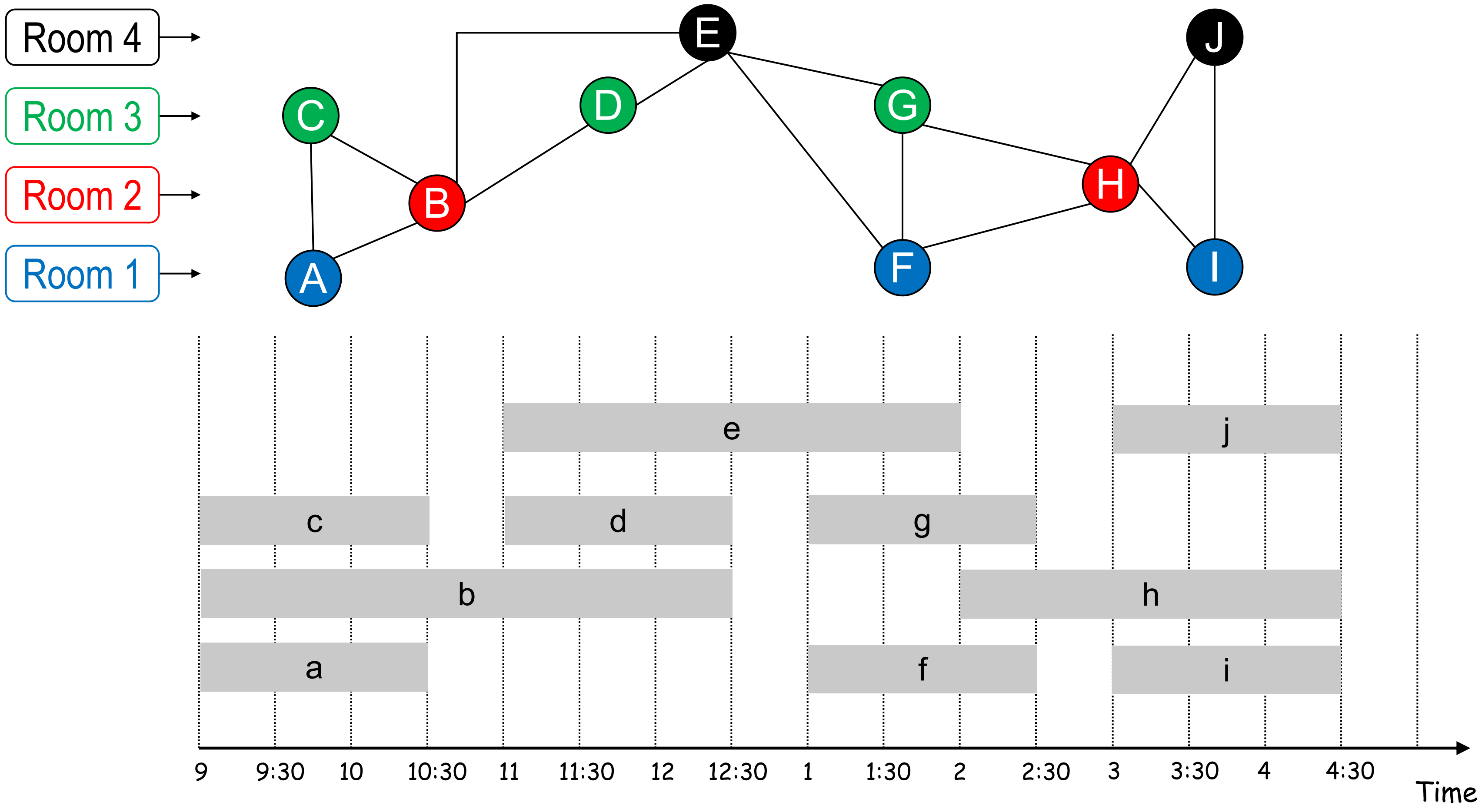
区间划分问题

- 第 j 个讲座开始时间 $s(j)+\varepsilon$、结束时间 $f(j)-\varepsilon$。
- 求：至少需要多少间教室，能够相容地安排所有的讲座
  - 相容——不存在两个讲座在同一时间使用相同的教室



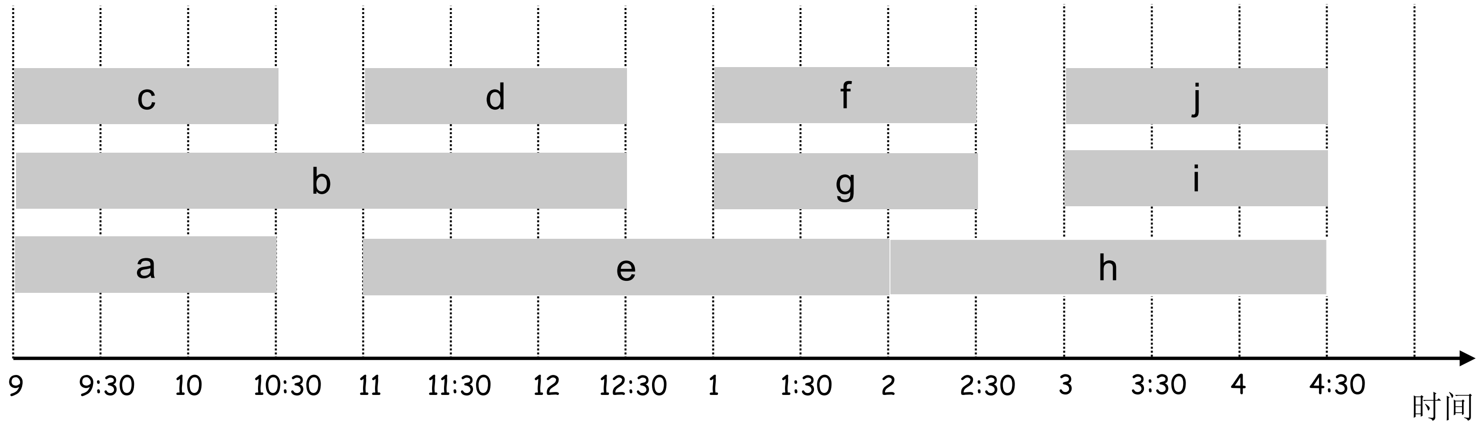
区间划分问题

图染色通常是困难的问题，但对于出自区间相交而构造的图，图染色是简单的。



区间划分问题：更好的调度

- 此方案只需 3 间教室



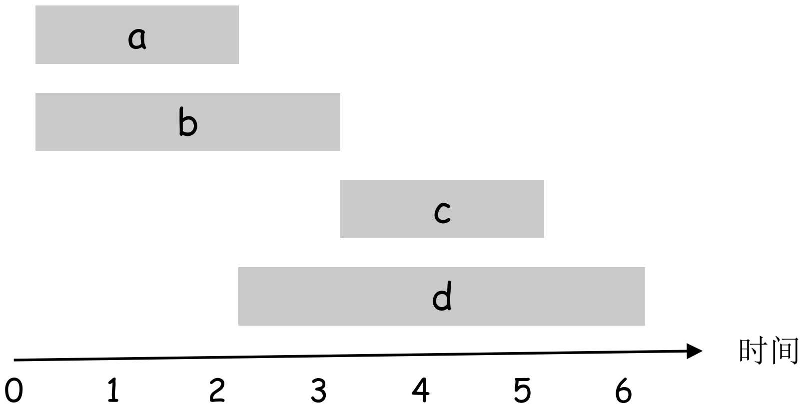
区间划分问题：尝试贪心法

贪心法：考虑按讲座的结束时间升序，安排可用的教室。

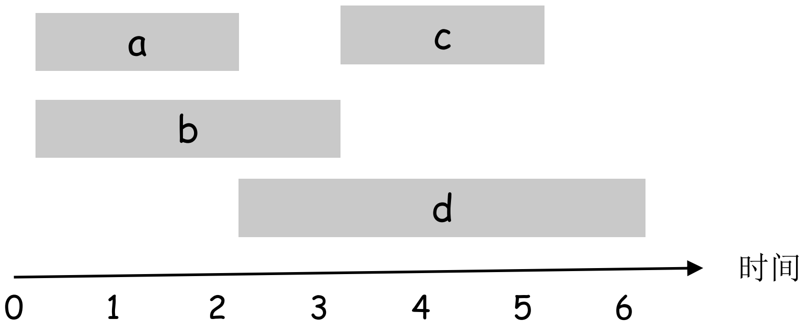
```
Sort intervals by fin[illegible] ≤ f_2 ≤ ... ≤ f_n.
d ← 0

for j = 1 to n {
   if (lect j is compatible with some classroom k, 1 ≤ k ≤ d)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

正确性? 会产生非最优的分配!

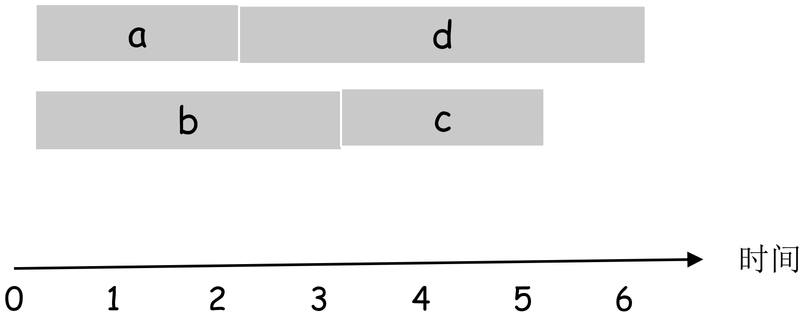
区间划分问题：按结束时间贪心的反例



按结束时间贪心得到：



最优解：



区间划分问题：贪心算法

贪心算法： 考虑按开始时间的升序，安排可用的教室。

```
Sort intervals by starting time so that s₁ ≤ s₂ ≤ ... ≤ sₙ.
d ← 0

for j = 1 to n {
   if (lect j is compatible with some classroom k, 1 ≤ k ≤ d)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

实现：O(n log n) 时间

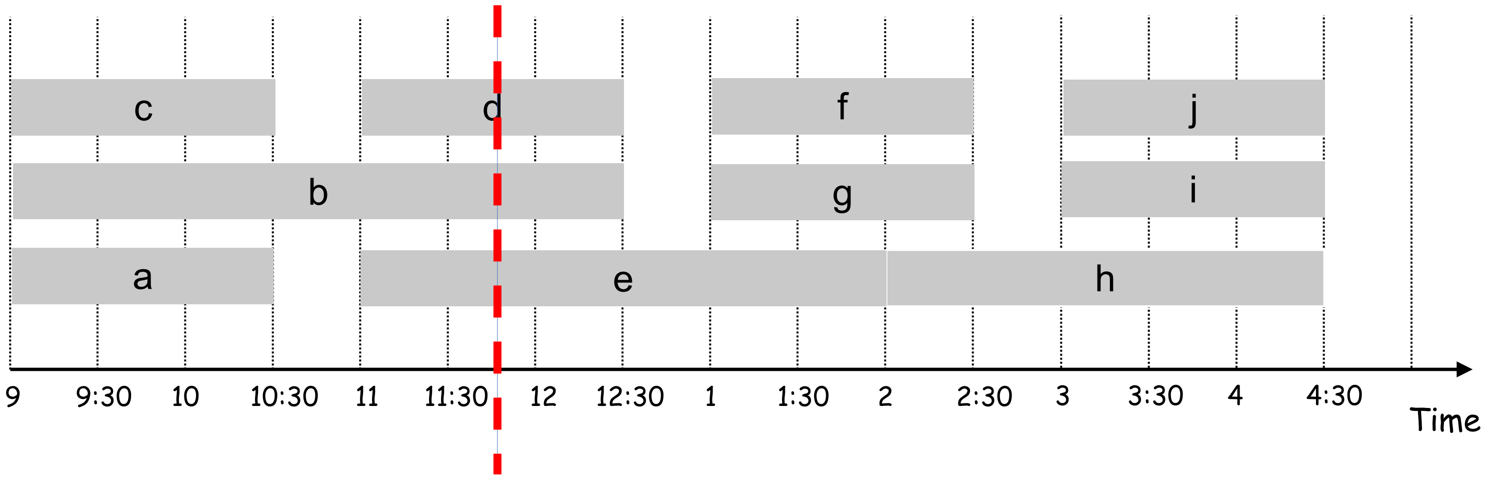
区间划分问题：最优解的结构性下界

定义：开区间集合的深度，是任一时刻所包含的最大区间数目

可行解性质：所需的教室数目 ≥ 深度

（例）以下调度需要3间教室、深度也是 3 ⇒ 调度是最优的

问：是否总是存在等于区间集深度的调度？



区间划分问题：正确性证明

性质：贪心算法得到的调度，同一教室分配的讲座都是相容的。

定理：贪心算法的结果是最优的。

证明：（利用结构化下界）
令 d = 贪心算法所需的教室数目。

我们在调度第 j 个讲座时，分配第 d 个教室，是由于其余 $d-1$ 个教室安排了与第 j 个讲座不相容的讲座。

由于贪心算法按开始时间升序来安排教室，其余 $d-1$ 个教室安排的讲座，开始时间都不会晚于 $s(j)$。

因此，在 $s(j)+\varepsilon$ 时刻至少有 d 个讲座，即：深度 $\geq d$。

可行解性质 $\Rightarrow$ 任意调度使用的教室数 $\geq$ 深度 $\geq d$。
即贪心算法求得需要 d 个教室的调度方案是最优的。■

贪心算法的特点

- 设计要素：
  1. 贪心法适用于组合优化问题.
  2. 求解过程是多步判断过程，最终的判断序列对应于问题的最优解.
  3. 判断依据某种“短视的”贪心选择性质，性质的好坏决定了算法的成败.
  4. 贪心法必须进行正确性证明
- 贪心法的优势：
  - 算法简单，时间和空间复杂性低

贪心法的正确性证明

数学归纳法

1. 叙述一个描述算法正确性的命题 $P(n)$，n 为算法步数或者问题规模
2. 归纳基础：$P(1)$ 或 $P(n_0)$ 为真，n_0 为某个自然数
3. 归纳步骤：
   - $P(k) \Rightarrow P(k+1)$　　第一数学归纳法
   - $\forall k(k<n)\ P(k) \Rightarrow P(n)$　　第二数学归纳法

交换论证

1. 分析算法的解的结构特征
2. 从一个最优解逐步进行结构变换（替换成分、交换次序等）得到一个新的解（结构上与贪心算法的解更接近）
3. 证明：上述变换最终得到算法的解，且变换在有限步结束，每步变换都保持解的最优性不降低.

最优装载 Loading

n 个集装箱1,2,…,n 装上轮船，集装箱 i 的重量 w_i, 轮船装载重量限制为c，无体积限制。问如何装使得上船的集装箱最多？不妨设 $w_i \le c$。

$$\max \sum_{i=1}^{n} x_i$$

$$\sum_{i=1}^{n} w_i x_i \le c$$

$$x_i \in \{0,1\} \quad i = 1,2,\ldots,n$$

贪心法：将集装箱按照从轻到重排序，轻者先装。

最优装载：贪心选择性质证明

命题：对任何规模为n（n是正整数）的输入，上述贪心法都得到最优解.

证明思路 对规模的归纳

- 设集装箱标号按照从轻到重记为$1, 2, \ldots, n$
- 归纳基础：n=1，贪心选择得到最优解（只有1个箱子，显然）
- 归纳假设：假设规模为 $n-1$ 的输入能得到最优解
- 归纳步骤：证明规模为 n 的输入也能得到最优解（下一页）

最优装载：归纳步骤

假设对于 $n-1$ 个集装箱的输入，贪心法都可以得到最优解，考虑 n 个集装箱的输入 $N = \{1, 2, \ldots, n\}$, 其中

$$w_1 \le w_2 \le \ldots \le w_n.$$

由归纳假设，对于$N' = \{2,3,\ldots,n\}$，$c' = c-w_1$, 贪心法 得到最优解 I'. 令 $I = \{1\} \cup I'$，则 I 是关于 N 的最优解.

若不然，若存在包含 1 的关于 N 的最优解 I^*（如果 I^* 中没有1，用 1 替换 I^* 中的第一个元素得到的解也是最优解），且$|I^*| > |I|$；那么 $I^* - \{1\}$是 N' 的解且

$$|I^* - \{1\}| > |I - \{1\}| = |I'|$$

与 I' 的最优性矛盾.

最优装载：说明

- Loading 算法
  复杂性 $T(n)=O(n\log n)$
- Loading 问题 是 0-1背包问题 的特例：
  即$v_i=1$, i=1,2,…,n.
  该问题是 $O(n\log n)$ 时间可解的
  0-1背包问题是NP难的

最小化最坏延迟调度

例 最小化最坏延迟调度

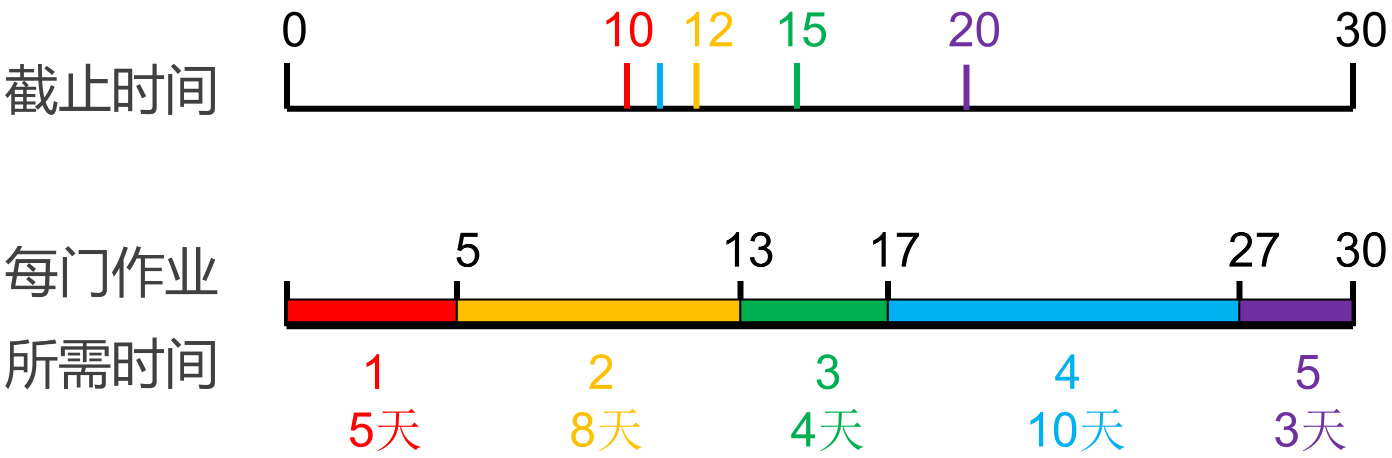
给定作业集合A，$\forall i \in A$，t_i为作业时间，d_i为截至时间，t_i, d_i为正整数. 一个调度是函数 $f: A \rightarrow \mathrm{N}$，$f(i)$为作业 i 的开始时间. 求最坏延迟达到最小的调度，即求f使得

$$\min_{f}\{\max_{i \in A}\{f(i)+t_i-d_i\}\}$$

$$\forall i, j \in A, i \neq j, f(i)+t_i \leq f(j) \text{ or } f(j)+t_j \leq f(i)$$

最小化最坏延迟调度

例1：5门课作业 {1, 2, 3, 4, 5}



问：如何安排作业的完成顺序，使得最大延迟尽可能小

max { -5 , 1 , 2 , 16 , 10 } = 16天

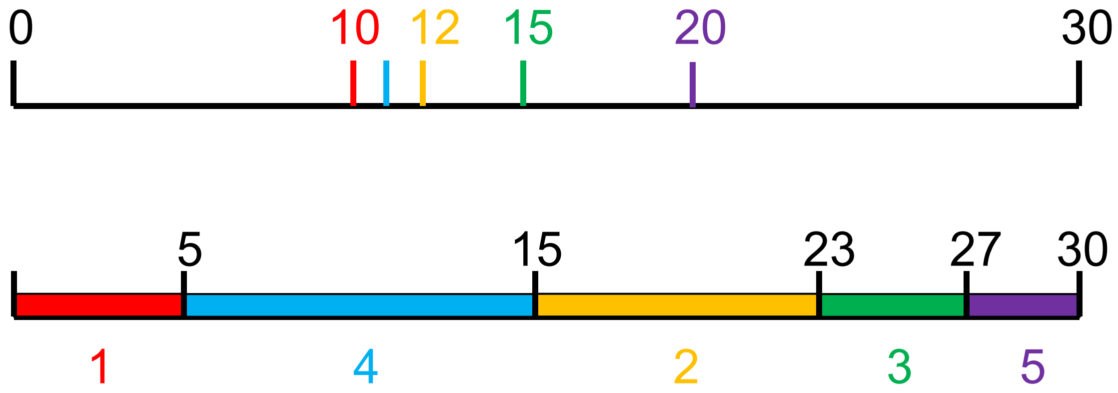
最小化最坏延迟调度

例2，5门课作业 {1, 2, 3, 4, 5}

截止时间

每门作业

所需时间



问：如何安排作业的完成顺序，使得<u>最大延迟</u>尽可能小

max { -5 , 4 , 11 , 12 , 10 } = 12天

最小化最坏延迟调度：贪心策略选择

- 贪心策略1：按照 作业时长 从小到大安排作业
- 贪心策略2：按照 截止时间 – 作业时长 从小到大安排作业
- 贪心策略3：按照 截止时间 从小到大安排作业

- 策略1 对某些实例得不到最优解，反例：
  - 作业1作业时长1天、截止时间第100天
  - 作业2作业时长10天、截止时间第10天
- 策略2 对某些实例得不到最优解，反例：
  - 作业1作业时长1天、截止时间第2天
  - 作业2作业时长10天、截止时间第10天

最小化最坏延迟调度：算法设计

- 算法思想：
  - 按照截止时间从小到大启动作业（没有逆序）
    - 逆序：hw1 截止时间比 hw2 晚、但完成时间比 hw2 早
  - 每完成一项作业即启动下一项作业（没有空闲时间）

最小化最坏延迟调度：算法设计

```
算法  Schedule
输入：A, T, D
输出：f
1.   排序A使得 d1≤d2 ≤ …≤ dn
2.   f(1)←0
3.   i←2
4.   while i≤n do
5.       f(i)←f(i−1)+t(i−1)
           // 任务i在任务i-1结束时开始
6.       i←i+1
```

- 设计思想：按截止时间从早到晚安排任务，没有空闲

最小化最坏延迟调度：正确性证明

- 算法思想：没有逆序+没有空闲时间
- 证明思路：
  - 所有 “没有逆序+没有空闲时间” 的解具有相同的最坏延迟
    - 没有逆序的不同解 → 相同截止时间作业顺序不同 → 不影响最坏延迟
  - 存在 “没有逆序+没有空闲时间” 的最优解
    - 有空闲时间的最优解 → 没有空闲时间的最优解
    - 存在逆序的最优解 → 存在相邻逆序的最优解 → 逆序数少1的最优解 → 没有逆序
  - ⇒ 上述算法所求的 “没有逆序+没有空闲时间” 的解是最优解

最小化最坏延迟调度：交换论证

算法的解的性质：

(1) 没有空闲时间，没有逆序.

(2) 逆序 (i, j): $f(i) < f(j)$ 且 $d_i > d_j$

引理 所有没有逆序、没有空闲时间的调度具有相同的最大延迟.

证：设 f 没有逆序，在 f 中具有相同截止时间 d 的作业 $i_1, i_2, \ldots, i_k$ 必被连续安排。在这 k 个作业中最坏延迟是最后一个作业，被延迟的时间是

$$t_0 + \sum_{j=1}^{k} t_{i_j} - d$$

与 $i_1, i_2, \ldots, i_k$ 的排列次序无关

最小化最坏延迟调度：交换论证

- 证明思想：从一个没有空闲时间的最优解出发，在不改变最优性的条件下，转变成没有逆序的解. 根据引理 1，这个解和算法的解具有相同的最大延迟.

- 证明要点
  1. 相邻逆序的存在性：如果一个最优调度存在逆序，那么存在 $i < n$ 使得 $(i, i+1)$ 构成一个逆序.
  2. 交换相邻的逆序 i 和 j，得到的解的调度仍旧最优.
  3. 每次交换后逆序数减1，至多经过 $n(n-1)/2$ 次交换得到一个没有逆序的最优调度.

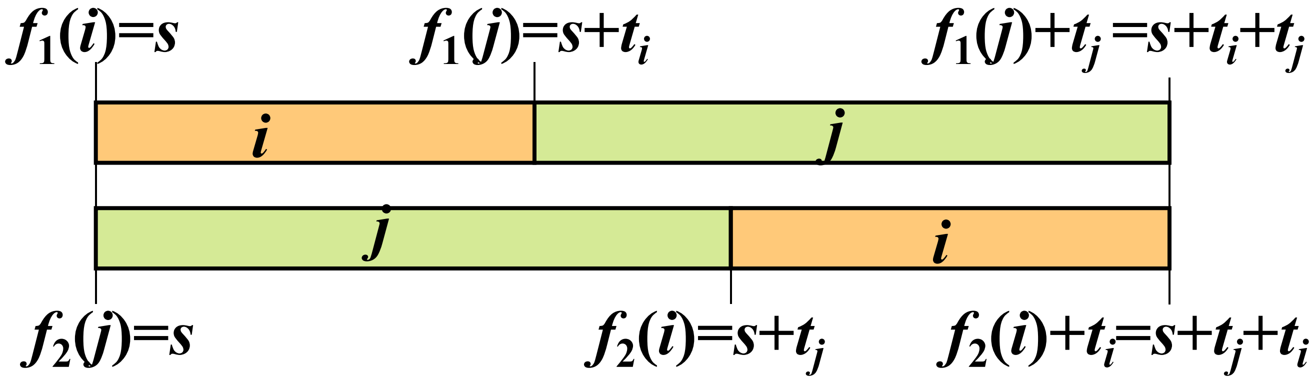
- 定理 在一个没有空闲时间的最优解中，最大延迟是r，如果仅对具有相邻逆序的任务进行交换，得到的解的最大延迟不会超过r。

最小化最坏延迟调度：交换相邻逆序不影响最优性

(1) 交换 i, j 对其他作业的延迟时间没影响

(2) 交换后不增加 j 的延迟

(3) i 在 f' 的延迟delay(f',i)小于 j 在 f 的延迟 delay(f,j), 因此小于f 的最大延迟 r



$$\text{delay}(f',i)=s + t_j+t_i - d_i<\text{delya}(f,j)\le r$$

$$\text{delay}(f,j) = s+t_i+t_j - d_j$$

$$\boxed{d_j< d_i \Rightarrow L_{2i}<L_{1j}}$$

最小化最坏延迟调度

"没有逆序+没有空闲时间"的解是最优解

⇒ 如果有N项作业，N个相同或不同的截止时间：

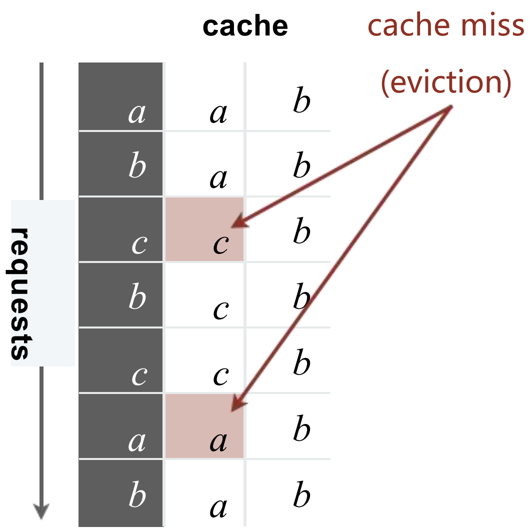
假如使用最小化最坏延迟调度算法（最早截止作业优先，不安排空闲时间）

也要迟交作业，那么任何其他调度方法也不可能在截止日期前完成作业

- 上述问题，假设作业的发布时间都在0时刻。
- 简单扩展：有N项作业，给定发布时刻、完成时长、截至时刻。
- 此时求最小化最坏延迟调度的难度？

最优缓存调度

- 高速缓存
  - 高速缓存的容量是 k 项数据
  - 长度为 m 的访问序列 d1 , d2 , ..., dm
  - 缓存命中：请求访问的数据在缓存中
  - 缓存缺失：请求访问的数据不在缓存中
    - 必须先回收缓存中的一项数据，将待访问的数据放进缓存才能继续访问
- 应用：CPU、RAM、硬盘、网站、浏览器....
- 目标：缓存换进换出的调度策略，最小化回收的次数
- 例：k = 2，初始缓存内容 [a, b]，请求序列：a , b , c , b , c , a , b .
  - 最优缓存调度：2次回收

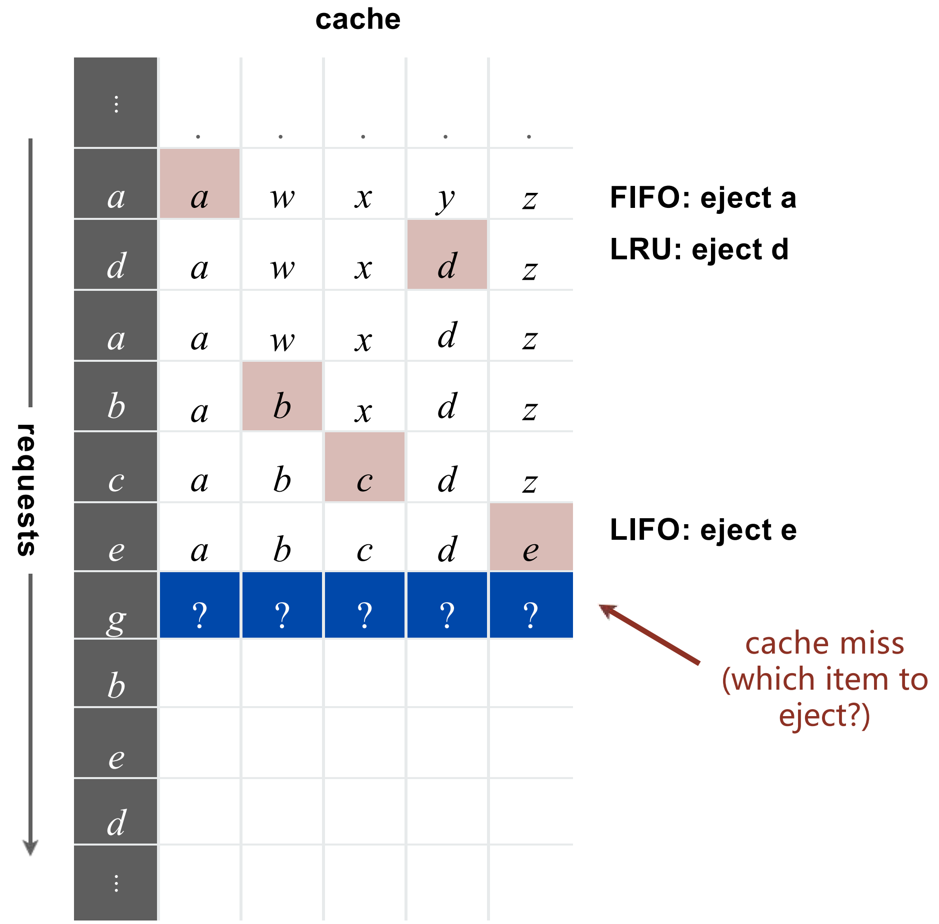


最优缓存调度

- 假如预先知道访问序列，能否求出最优的缓存调度策略?
- 虽然绝大部分情况，我们无法事先得知访问序列，但是:
  - 对于特定应用，序列是已知的
    - 例如代码生成中的寄存器分配问题
  - 用于竞争分析，与最优的离线算法比较在线算法的质量（第14讲）

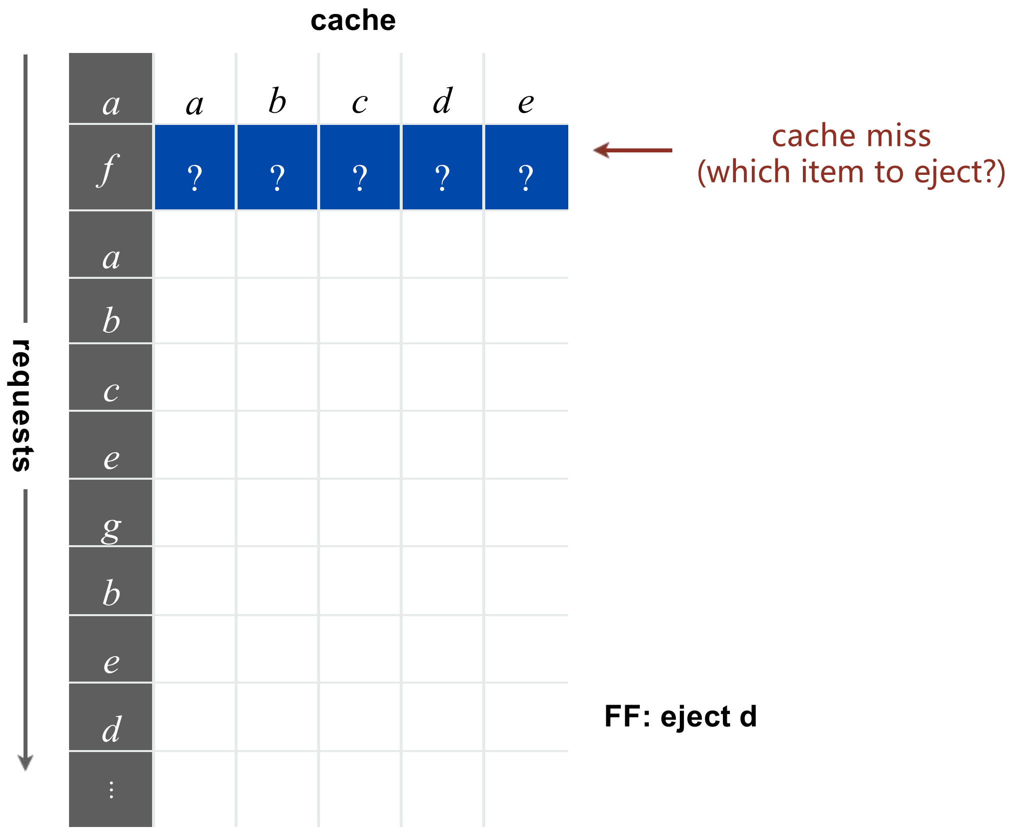
常见的缓存调度策略

- LIFO/FIFO：回收最后/最先进入缓存的数据
- LRU：回收离上次访问最远的数据
- LFU：回收最不常用的数据



最优缓存调度：Furthest-in-Future (FF)

- “千里眼算法”
- 回收未来最迟访问的数据
- 定理：FF 是最优调度
  - [Bélády 1966]
  - 算法和结论很直观
  - 证明需要些技巧



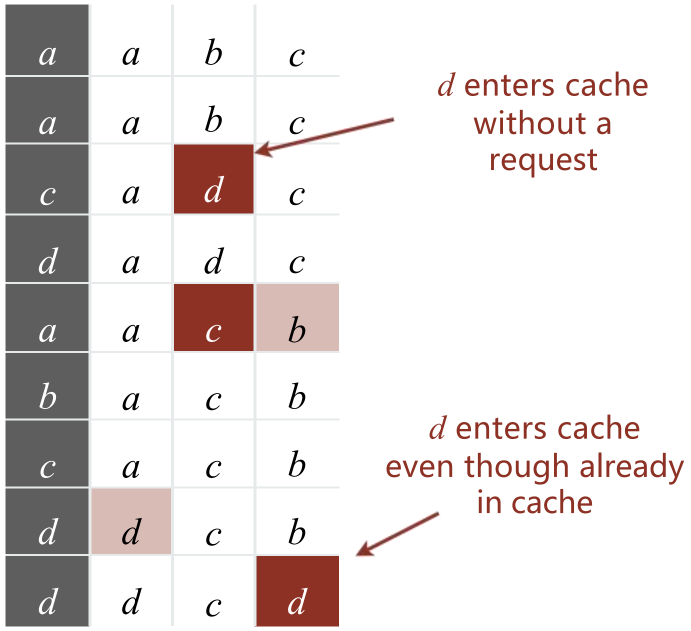
最优缓存调度：Furthest-in-Future 练习

requests	cache			
⋮	.	.	.	.
B	D	B	Y	A
C	D	B	C	A
E	D	E	C	A
F	?	?	?	?
C				
D				
A				
E				
A				
C				
⋮				

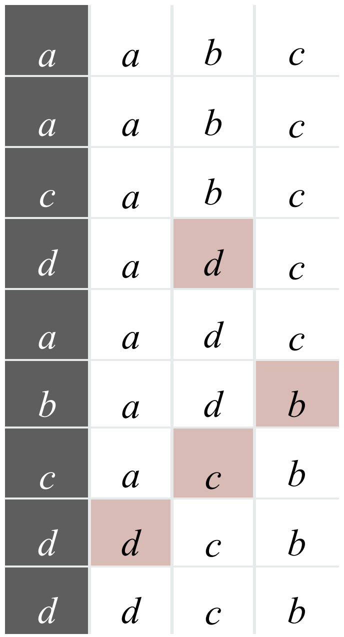
cache miss (which item to eject?)

最优缓存调度：简化的调度（1/6）

- 定义：简化的调度是只有当访问数据 d 发生缓存缺失才将其调入缓存的调度



an unreduced schedule



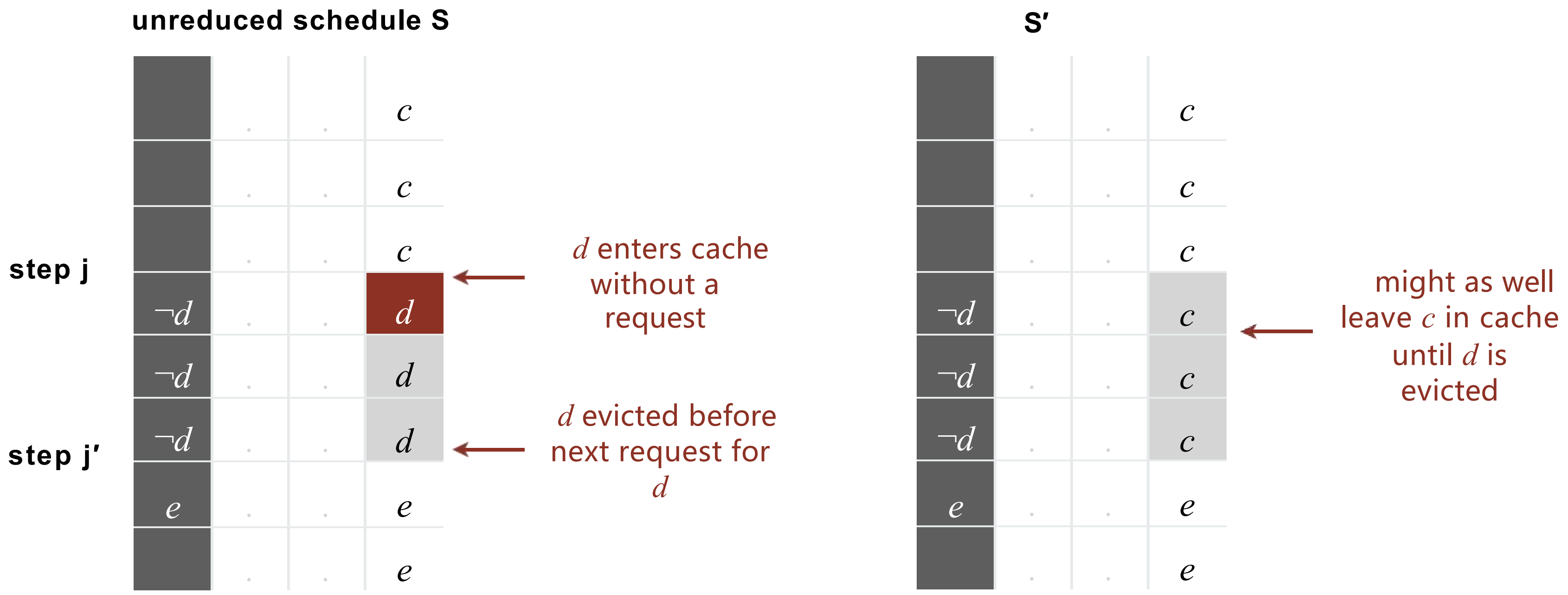
a reduced schedule

最优缓存调度：简化的调度（2/6）

- 性质：给定任意未简化的调度 S，总能变换为简化的调度 S′ 而不增加回收次数
- 证明（按步数 j 用归纳法）
  - 未简化：S 在第 j 步时未发生访问数据 d 的缓存缺失，却将 d 调入缓存
    - 情况1：S 在第 j 步时不访问数据 d，却将 d 调入缓存 （见后续PPT）
    - 情况2：S 在第 j 步时缓存已有数据 d，仍将 d 调入缓存 （见后续PPT）
  - 如果有多个未简化的缓存项，逐一按情况1和情况2处理
    - （解决情况1可能会触发情况2）

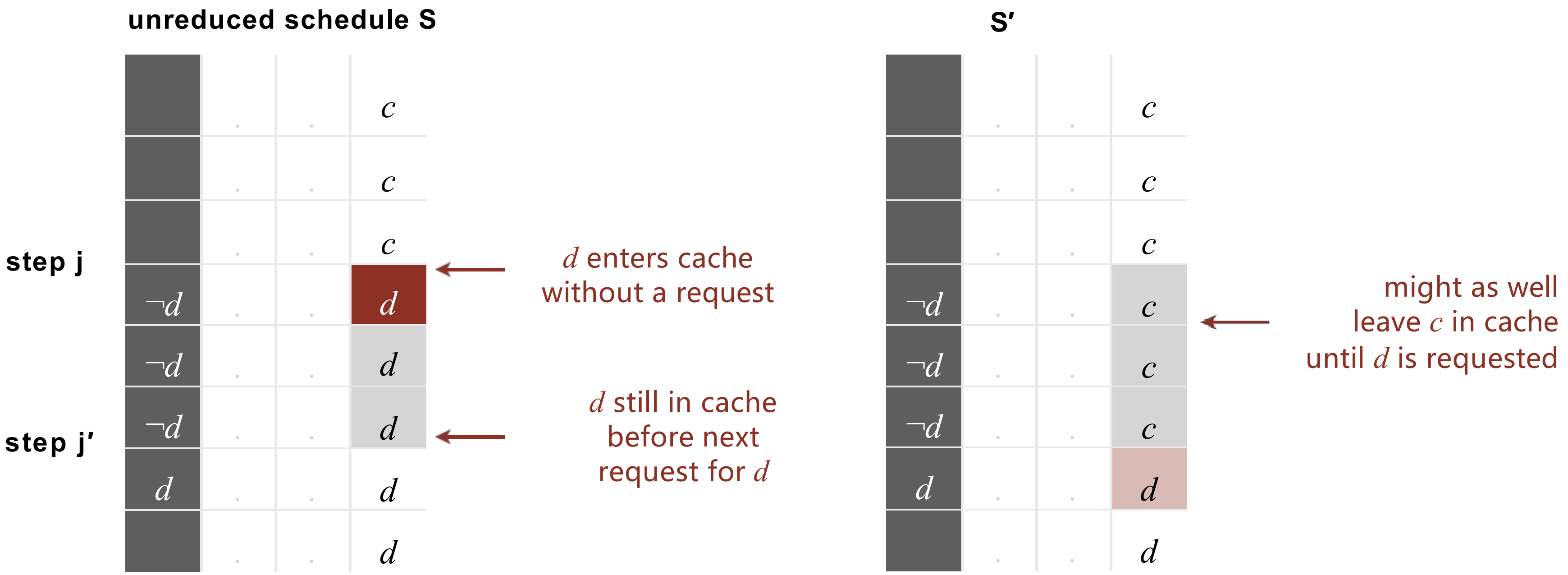
最优缓存调度：简化的调度（3/6）

- 性质：给定任意未简化的调度 S，总能变换为简化的调度 S′ 而不增加回收次数
- 证明（按步数 j 用归纳法）
  - 假设 S 在第 j 步时不访问数据 d，却将 d 调入缓存
  - 记数据 c 为 S 将数据 d 调入缓存时所回收的数据
  - 情况1a：下次访问 d 前 d 已被回收



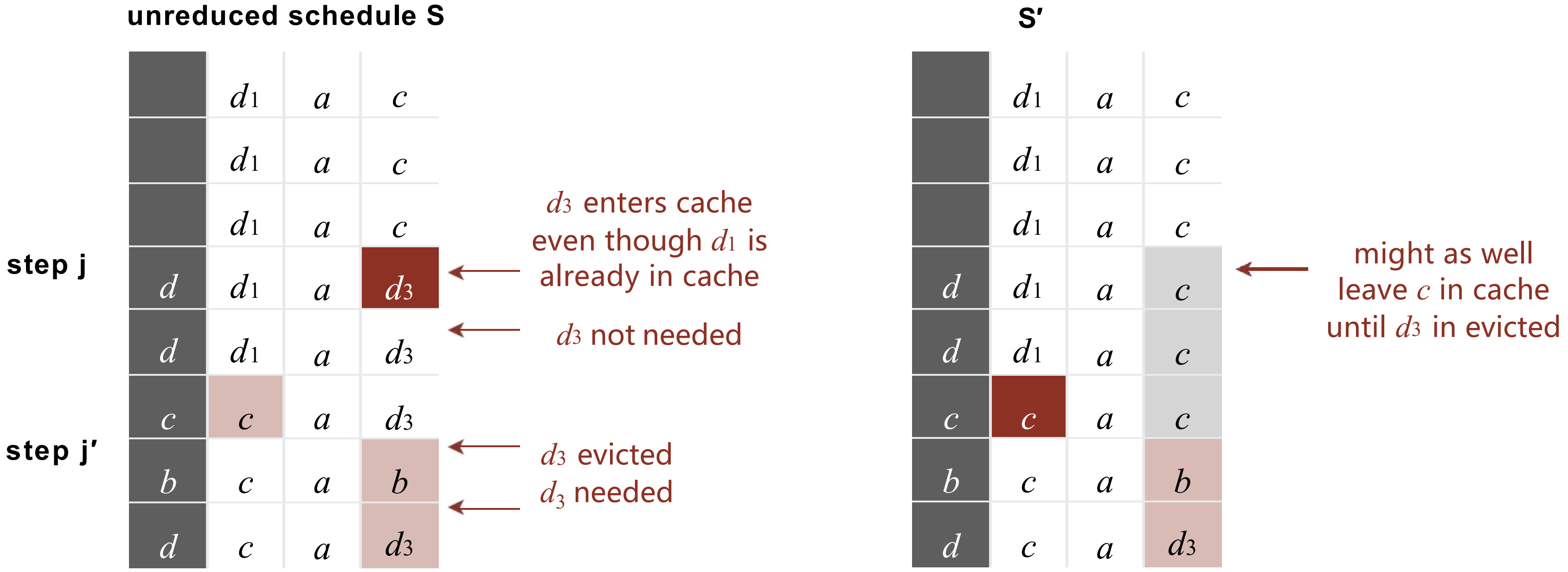
最优缓存调度：简化的调度（4/6）

- 性质：给定任意未简化的调度 S，总能变换为简化的调度 S′ 而不增加回收次数
- 证明（按步数 j 用归纳法）
  - 假设 S 在第 j 步时不访问数据 d，却将 d 调入缓存
  - 记数据 c 为 S 将数据 d 调入缓存时所回收的数据
  - 情况1b：下次访问 d 时 d 未被回收



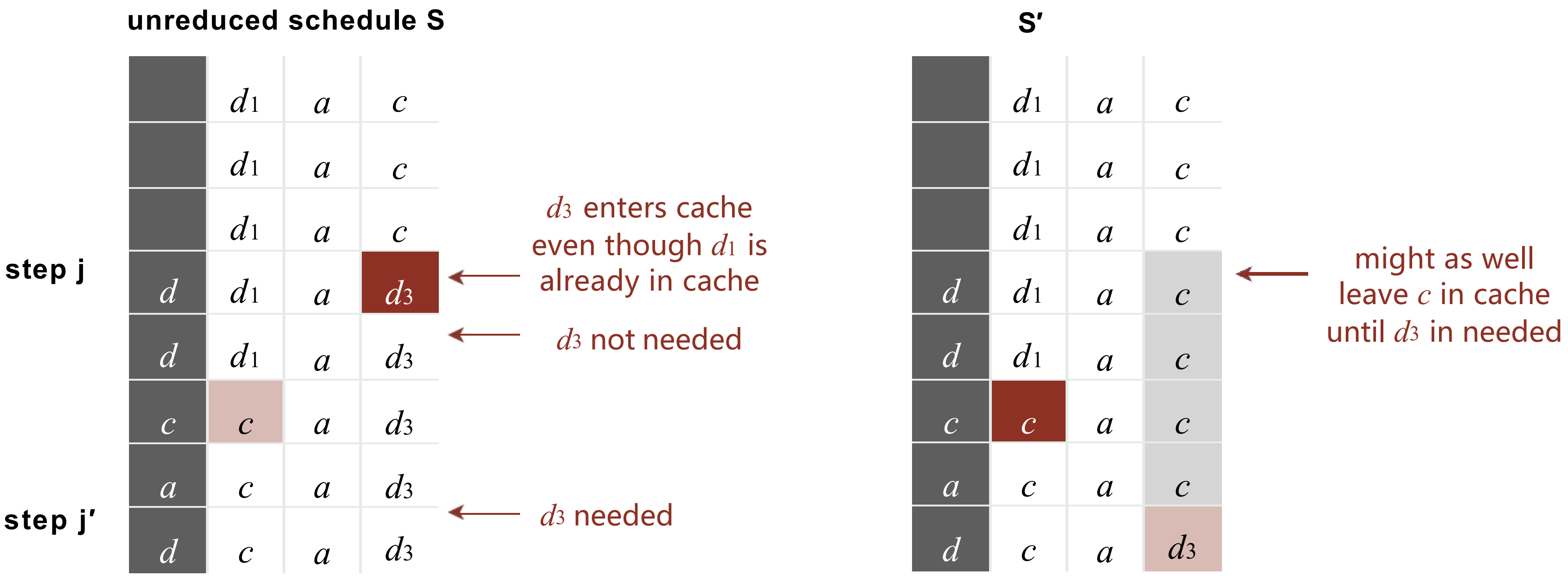
最优缓存调度：简化的调度（5/6）

- 性质：给定任意未简化的调度 S，总能变换为简化的调度 S′ 而不增加回收次数
- 证明（按步数 j 用归纳法）
  - 假设 S 在第 j 步时不访问数据 d，却将 d 调入缓存
  - 记数据 c 为 S 将数据 d 调入缓存时所回收的数据
  - 情况2a：下次访问 d 前 d 已被回收



最优缓存调度：简化的调度（6/6）

- 性质：给定任意未简化的调度 S，总能变换为简化的调度 S’ 而不增加回收次数
- 证明（按步数 j 用归纳法）
  - 假设 S 在第 j 步时不访问数据 d，却将 d 调入缓存
  - 记数据 c 为 S 将数据 d 调入缓存时所回收的数据
  - 情况2b：下次访问 d 时 d 未被回收

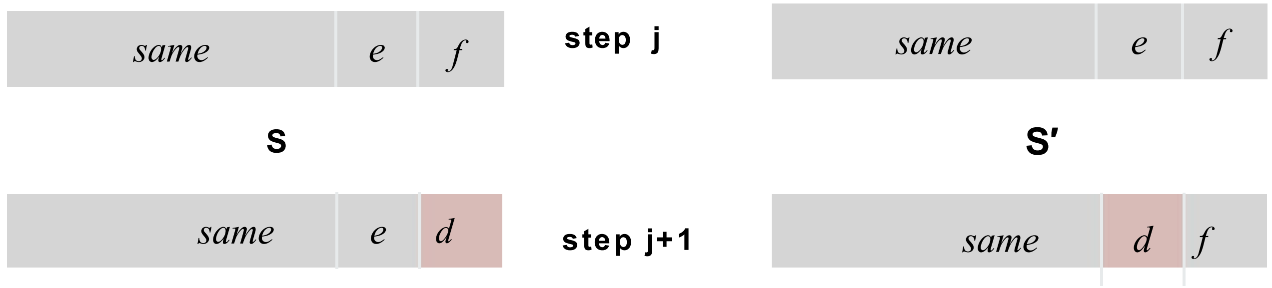


最优缓存调度：分析 Furthest-in-Future

- 定理：Furthest-in-Future 是最优的回收算法
- 证明（归纳法：对于调度 S_{FF}，存在最优的简化调度 S 的前 j 步操作与 S_{FF} 相同）
- 归纳基础：j = 0。
- 归纳假设：令 S 是前 j 步操作与 S_{FF} 相同的最优简化调度
- 归纳步骤：构造最优简化调度 S'，使其前 j+1 步操作与 S_{FF} 相同
  - 令 d 是第 j+1 步请求的数据。
  - 由于 S 和 S_{FF} 前 j 步操作相同，他们在第 j+1 步前有相同的缓存内容。
  - 情况1：d 已在缓存中，令 S’ = S 即可。
  - 情况2：d 不在缓存中，且 S 和 S_{FF} 在第 j+1 步回收相同的缓存项，令 S ’ = S 即可。
  - 情况3：d 不在缓存中，S_{FF} 在第 j+1 步回收 e，S 在第 j+1 步回收 f≠e。（见下页）

最优缓存调度：分析 Furthest-in-Future

- 情况3：d 不在缓存中，S_{FF} 在第 j+1 步回收 e，S 在第 j+1 步回收 f≠e。
  - 用 S 构造 S’，使 S’ 在第 j+1 步回收 e 而不是回收 f。



  - 现在 S’ 和 S_{FF} 在前 j+1 步相同；只需证明保留 f 不比保留 e 差。
  - S’ 和 S 在第 j+1 步后第一次不相同的行为，要么是 S 回收 e，要么发生数据 e 或 f 的访问。

最优缓存调度：分析 Furthest-in-Future

- 设 S’ 和 S 在第 j+1 步后、第 j’ 步首次产生不相同的行为。
- 令 g 是第 j’ 步访问的数据。

- 情况3a： g = e。
  - 不可能发生。由 furthest-in-future 的规则，数据 e 的访问会晚于数据 f 的访问。
- 情况3b： g = f。
  - 数据 f 不在 S 的缓存中；令 e’ 是 S 回收的数据。
  - 如果 e’ = e，S 将 e 换成 f，而 S’ 从缓存访问 f；S 和 S' 拥有相同的缓存内容。
  - 如果 e‘ ≠ e，构造 S’ 在第 j’ 步回收 e’，将 e 放入缓存；此时 S 和 S’ 拥有相同的缓存内容。
    - S' 可能不是简化的调度，但能够转化为简化的调度，且前 j+1 步与 S_{FF} 相同。
  - 令 S’ 和 S 在余下步骤执行相同的操作即可。

最优缓存调度：分析 Furthest-in-Future

- 设 S’ 和 S 在第 j+1 步后、第 j’ 步首次产生不相同的行为。
- 令 g 是第 j’ 步访问的数据。

- 情况3c： g ≠ e, f。且 S’ 回收 e。
  - 令 S’ 回收 f。

  - 此时 S 和 S’ 拥有相同的缓存内容。
  - 令 S’ 和 S 在余下步骤执行相同的操作即可。■

最优缓存调度：缓存视角

- 在线 vs. 离线算法
  - 离线：事先知道所有的访问序列
  - 在线（线上）：事先不知道访问内容。
  - 缓存是计算机科学里最基础的在线问题
- LIFO：回收最近访问的数据。
  - 性能非常糟糕
- LRU：回收最久未使用的数据
  - 与 Furthest-in-Future 在时间上反向
- 定理：Furthest-in-Future 是离线的最优回收算法
  - 理解和分析在线算法的基础
  - LRU 的竞争比是 k：对于任意的访问序列 σ，$LRU(\sigma) \le k \cdot FF(\sigma) + k$。（第14讲）

贪心算法的性质 & 小结

- Optimal substructure
  - 问题最优解包含子问题最优解
- Greedy choice property
  - 以局部最优选择构造全局最优解

贪心法常用的证明要点：

- 尝试论证贪心算法每步产生的选择、或者最终的解，不次于其他算法。
- “结构化”下界：尝试构造答案的下界，论证每个可行解都不低于该下界；当贪心算法能产生达到该下界的解时，则是最优的。
- 交换论证：将其他可能的最优解，通过逐步变换，在保持最优性的情况下，变成贪心算法所求解的形式。