贪心算法

算法比较

- **Divide-and-conquer（分治策略）**

Break up a problem into sub-problems, solve each sub-problem independently, and combine solution to sub-problems to form solution to original problem.

- **Dynamic programming（动态规划）**

Break up a problem into a series of overlapping sub-problems, and build up solutions to larger and larger sub-problems.

- **Greedy（贪心）**

Build up a solution incrementally, myopically optimizing some local criterion.

贪心策略

- **A "greedy algorithm" is a "non-backtracking algorithm in which irrevocable decisions of global significance are made on the basis of local information".**

  ----G. B. McMahon

贪心策略

- **A "greedy algorithm" is a "non-backtracking algorithm in which irrevocable decisions of global significance are made on the basis of local information".**

  ----G. B. McMahon

- 所做的每一步选择都必须满足：
  - 可行的：必须满足问题的约束
  - 局部最优：是当前步骤中所有可行性选择中最佳的局部选择
  - 不可取消：选择一旦做出，在算法的后面步骤中就无法改变了

贪心策略

- **A "greedy algorithm" is a "non-backtracking algorithm in which irrevocable decisions of global significance are made on the basis of local information".**

  ----G.B.McMahon

- 贪心算法的特点：
  - 容易设计
  - 容易分析运行时间
  - 难于证明正确性
  - 精确算法 & 近似算法

贪心算法举例

- 部分背包问题
- 赫夫曼编码问题（**Huffman** 算法）
- 最短路径问题（ **Dijkstra**算法）
- 最小生成树问题
  - **Kruskal**算法
  - **Prim**算法
- 区间调度问题（活动选择问题）
- 区间划分问题
- 最小延迟调度问题

主要内容

- 贪心算法举例
  - 区间调度问题（活动选择问题）
    - 单资源多请求
    - 每个请求有开始时间和完成时间
    - 满足最大数目的请求
  - 区间划分问题
    - 多资源多请求
    - 用尽可能少的资源满足所有请求
  - 最小延迟调度问题
    - 单资源多请求
    - 每个请求有截止时间和处理时间，开始时间不固定
    - 最小化最大延迟
- 贪心算法的理论基础——拟阵

贪心算法举例

- 区间调度问题（活动选择问题）
  - 单资源多请求
  - 每个请求有开始时间和完成时间
  - 满足最大数目的请求
- 区间划分问题
  - 多资源多请求
  - 用尽可能少的资源满足所有请求
- 最小延迟调度问题
  - 单资源多请求
  - 每个请求有截止时间和处理时间，开始时间不固定
  - 最小化最大延迟

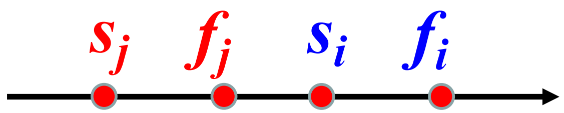
贪心算法举例

- 区间调度问题（活动选择问题）
  - 单资源多请求
  - 每个请求有开始时间和完成时间
  - 满足最大数目的请求
- 区间划分问题
  - 多资源多请求
  - 用尽可能少的资源满足所有请求
- 最小延迟调度问题
  - 单资源多请求
  - 每个请求有截止时间和处理时间，开始时间不固定
  - 最小化最大延迟

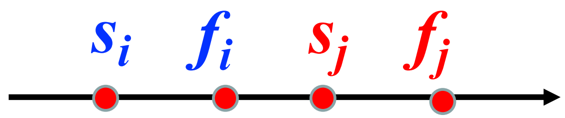
区间调度（Interval scheduling）问题

- 输入
  - $S=\{1, 2, \ldots, n\}$为n 项工作的集合，s_i, f_i分别为第 i 项工作的开始和结束时间
- 输出
  - 最大的两两相容的工作集 A

其中，工作 i 与 j 相容 $\Leftrightarrow$ $s_i \geq f_j$ 或 $s_j \geq f_i$



$s_i \geq f_j$



$s_j \geq f_i$

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

相容工作集：如 {1, 4, 8}，{2, 7}

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

相容工作集：如 {1, 4, 8}， {2, 7}

不相容活动集：如 {1, 2}, {7, 8}

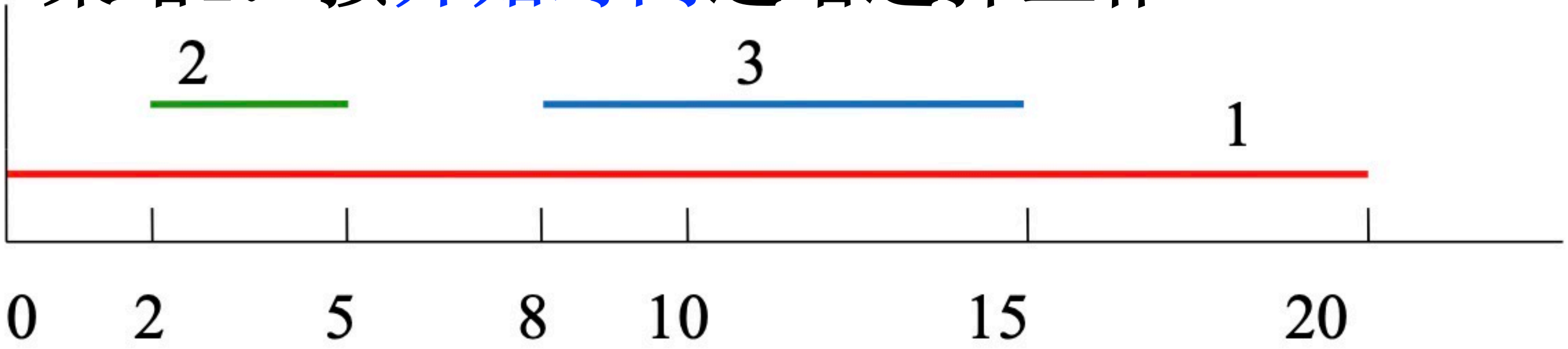
贪心策略：必须满足相容性条件

- 策略1：按开始时间递增选择工作
- 策略2：按结束时间递增选择工作
- 策略3：按活动进行时间递增选择工作

问题：这三种策略能否得到最大的两两相容的工作集？

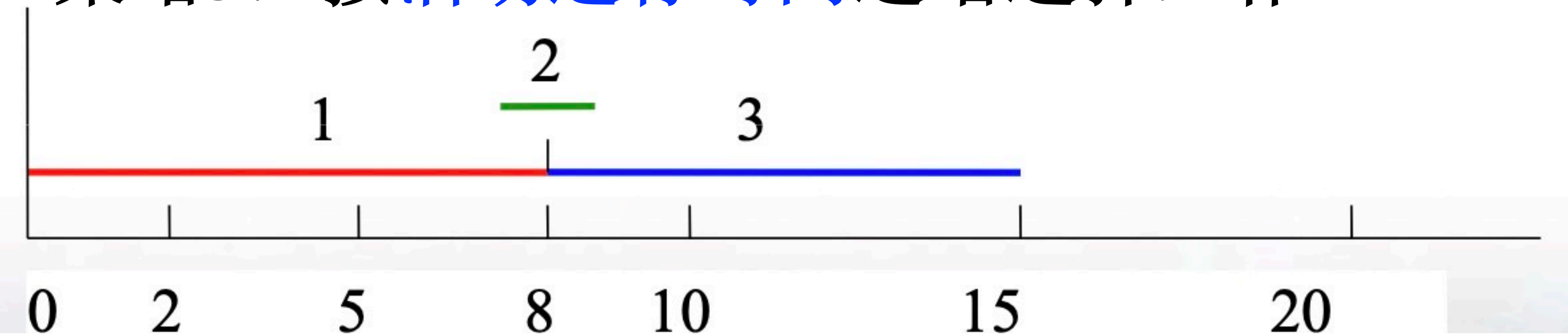
反例

- 策略1：按开始时间递增选择工作



算法输出：{1}，最优解：{2, 3}

- 策略3：按活动进行时间递增选择工作



算法输出：{2}，最优解：{1, 3}

贪心算法

策略2：按结束时间递增选择工作

尽早释放资源

Algorithm Greedy Interval Scheduling

Input: A set S of jobs, s_i, f_i for each fob in S

Output: A subset A of compatible jobs

```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn;      O(nlog n)
A ← ∅ ;                                // set of jobs selected
for j = 1 to n do
    if job j is compatible with A then
        A ← A ∪ {j};                                          O(n)
return A;
```

设上一个加入A的工作为 j'，则判断是否 $s_j \geq f_{j'}$

复杂度：$O(n\log n)$

问题：策略2能否得到最大的两两相容的工作集？

算法正确性

定理：贪心算法返回一个最优的集合*A*。

证明思想：

假设最优解为 *O*, 贪心算法返回的解为 *A*,

只需证明 $|O|=|A|$.

主要思想：贪心算法的解 *A*“领先”于最优解 *O*.

把贪心算法构造的部分解与最优解 *O* 初始一段进行比较，并且证明贪心算法以一步接一步的方式做得更好。

定理：贪心算法返回一个最优的集合A。

证明： 假设贪心算法返回工作集合A，且按工作添加顺序为 $i_1, \ldots, i_n$， 最优解为 工作序列 $j_1, \ldots, j_m$， 此时 $m \geq n$. 只需证明$m=n$。

首先证明以下结论：$f_{i_k} \leq f_{j_k}, k=1,\ldots, n$。

贪心算法的解领先最优解

f_{i_1}	f_{i_2}	$\cdots$	$f_{i_{k-1}}$	f_{i_k}	$\cdots$	f_{i_n}			
f_{j_1}	f_{j_2}	$\cdots$	$f_{j_{k-1}}$	f_{j_k}	$\cdots$	f_{j_n}	$f_{j_{n+1}}$	$\cdots$	f_{j_m}

对k 进行归纳证明：

(1) $k=1$时， 由于贪心算法的策略是按结束时间递增选择工作， 显然有， $f_{i_1} \leq f_{j_1}$.

(2) 假设 $f_{i_{k-1}} \leq f_{j_{k-1}}$， 下面证明$f_{i_k} \leq f_{j_k}$成立.

由于$f_{j_{k-1}} \leq s_{j_k}$， 因此$f_{i_{k-1}} \leq s_{j_k}$， 即工作 j_k 与工作i_{k-1} 相容, 按贪心算法策略， 必有 $f_{i_k} \leq f_{j_k}$.

定理：贪心算法返回一个最优的集合A。

证明： 假设贪心算法返回工作集合A，且按工作添加顺序为 $i_1, \dots, i_n$， 最优解为 工作序列 $j_1, \dots, j_m$， 此时 $m \geq n$.
只需证明$m=n$。
已证结论： $f_{i_k} \leq f_k$, $l=1,\dots, n$ 。

f_{i_1}	f_{i_2}	$\cdots$	$f_{i_{k-1}}$	f_{i_k}	$\cdots$	f_{i_n}			
f_{j_1}	f_{j_2}	$\cdots$	$f_{j_{k-1}}$	f_{j_k}	$\cdots$	f_{j_n}	$f_{j_{n+1}}$	$\cdots$	f_{j_m}

（反证法）设 $m>n$。

考虑 j_{n+1}： 由于$s_{j_{n+1}} \geq f_{j_n}$（即j_{n+1}在 j_n结束后开始），

且$f_{i_n} \leq f_{j_n}$， 得$s_{j_{n+1}} \geq f_{i_n}$，得j_{n+1}必然与i_n相容

而贪心算法此时已在 i_n处停止。矛盾。

故假设不成立，即$m=n$。证毕。

区间调度问题的扩展

- 加权区间调度问题

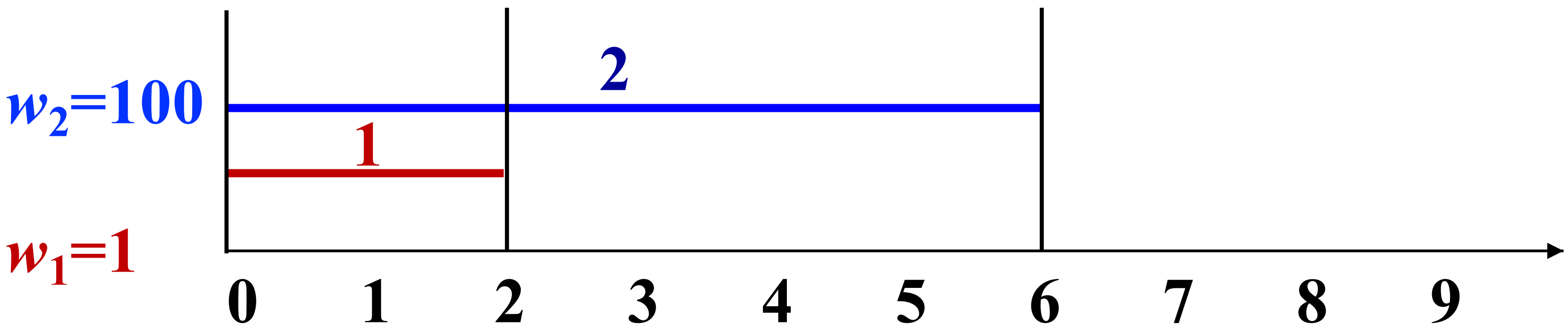
- 输入
  - $S=\{1, 2, \ldots, n\}$为n 项工作的集合，s_i, f_i, w_i分别为第 i 项工作的开始时间，结束时间和权重
- 输出
  - 最大加权的两两相容的工作集 A

其中，工作 i 与 j 相容 $\Leftrightarrow$ $s_i \geq f_j$ 或 $s_j \geq f_i$

- 每项工作的权重均为 1 时，即为区间调度问题

区间调度问题的扩展

- 贪心算法不适用加权区间调度问题



贪心算法的解：{1}

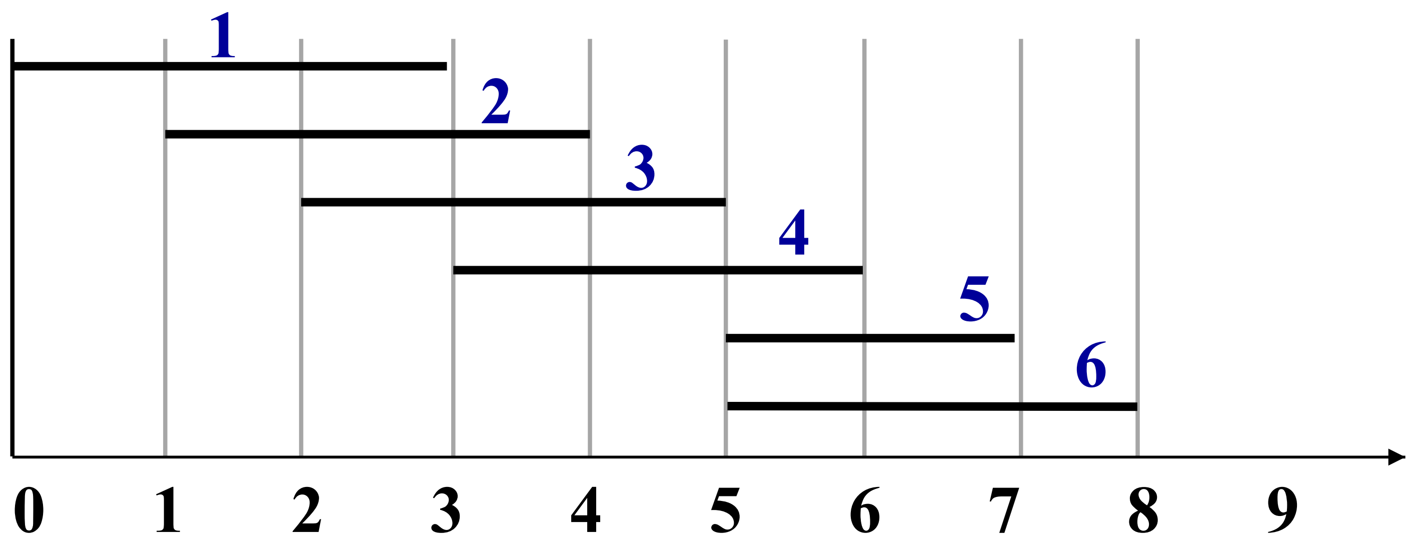
最优解：{2}

- 动态规划算法

加权区间调度问题的动态规划算法

- 设工作S={ 1, 2, …, n}已按结束时间递增进行排序，即$f_1 \le f_2 \le \ldots \le f_n$
- 对任意工作 $j \in S$，令$p(j)$为在 j 之前且与工作 j 相容的所有工作 i 中最大的 i，即

$$p(j) = \max\{ i \mid i<j \text{ 且工作 } j \text{ 与工作 } i \text{ 相容}\}$$



$p(1)=0$

$p(2)=0$

$p(3)=0$

$p(4)=1$

$p(5)=3$

$p(6)=3$

加权区间调度问题的动态规划算法

- 设工作$S=\{1, 2, \ldots, n\}$已按结束时间递增进行排序，即$f_1 \le f_2 \le \ldots \le f_n$
- 对任意工作 $j \in S$，令$p(j)$为在 j 之前且与工作 j 相容的所有工作 i 中最大的 i，即

$$p(j) = \max\{ i \mid i<j \text{ 且工作 } j \text{ 与工作 } i \text{ 相容}\}$$

- 递推式：

令$\{1, 2, \ldots, j\}$的最优解为 O_j，对应的最优值为OPT(j).

(1) 若$j \in O_j$，则 $\mathrm{OPT}(j) = w_j + \mathrm{OPT}(p(j))$

(2) 若$j \notin O_j$，则 $\mathrm{OPT}(j) = \mathrm{OPT}(j-1)$

$O(n)$

因此，得递推式为：

$$\mathrm{OPT}(j) = \max\{ w_j + \mathrm{OPT}(p(j)), \mathrm{OPT}(j-1)\}, j=1, 2, \ldots, n$$

贪心算法举例

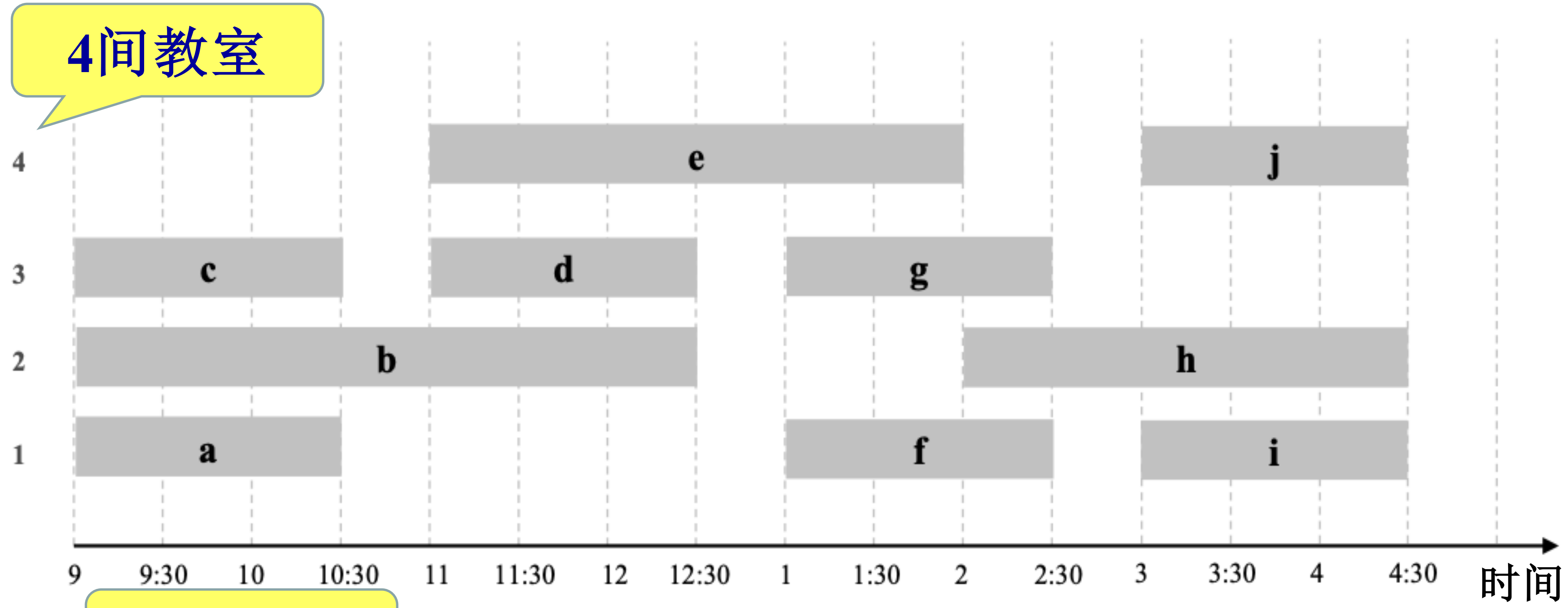
- 区间调度问题（活动选择问题）
  - 单资源多请求
  - 每个请求有开始时间和完成时间
  - 满足最大数目的请求
- 区间划分问题
  - 多资源多请求
  - 用尽可能少的资源满足所有请求
- 最小延迟调度问题
  - 单资源多请求
  - 每个请求有截止时间和处理时间，开始时间不固定
  - 最小化最大延迟

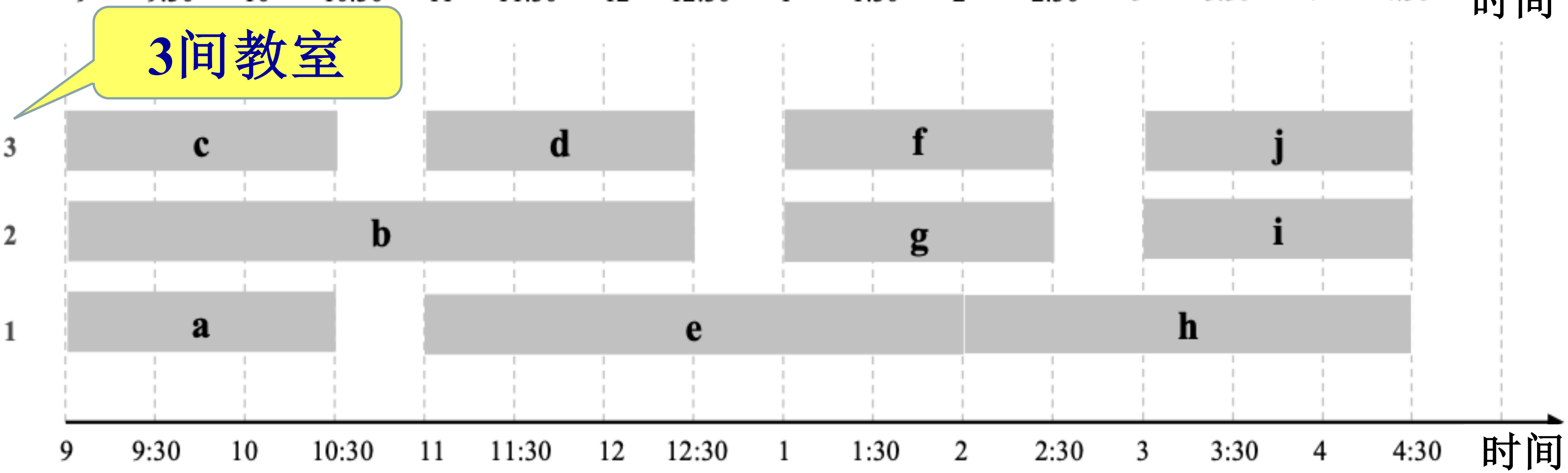
区间划分问题

输入：n门课，每门课 j 有开始时间 s_j 与结束时间 f_j

目标：找到**最小数目的教室**使得可以容纳这 n门课，且没有两门课会安排在同一个教室的同一个时间。

区间划分问题

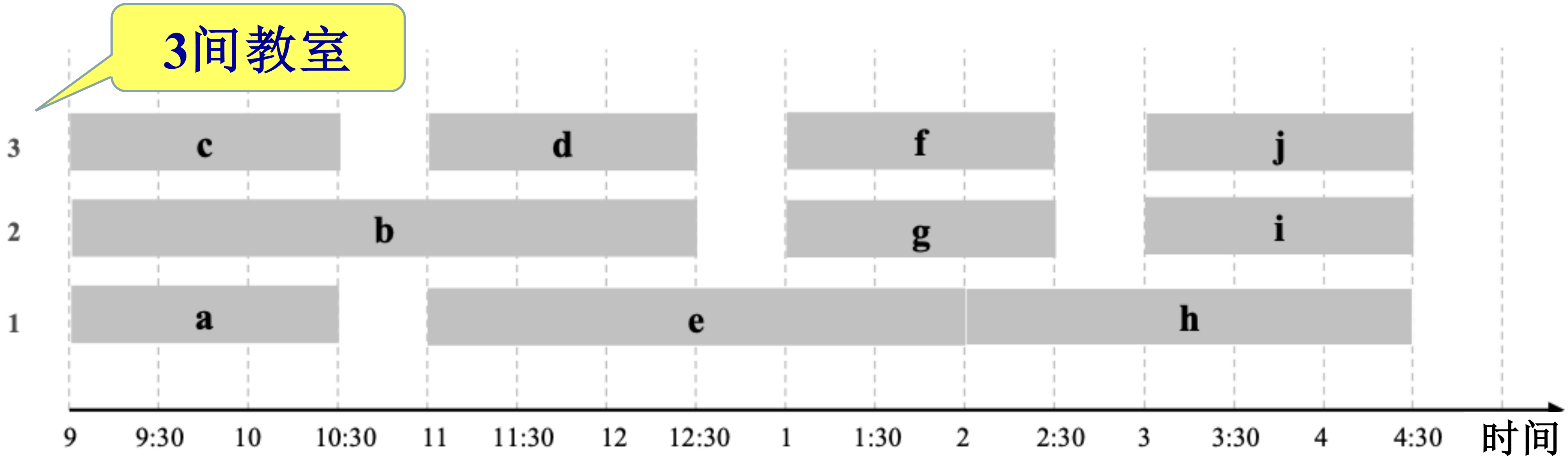

4间教室
4
3
2
1
e
j
c
d
g
b
h
a
f
i
9
9:30
10
10:30
11
11:30
12
12:30
1
1:30
2
2:30
3
3:30
4
4:30
时间


3间教室
3
2
1
c
d
f
j
b
g
i
a
e
h
9
9:30
10
10:30
11
11:30
12
12:30
1
1:30
2
2:30
3
3:30
4
4:30
时间

区间划分问题

输入：n门课，每门课j有开始时间s_j与结束时间f_j

目标：找到**最小数目的教室**使得可以容纳这n门课，且没有两门课会安排在同一个教室的同一个时间。



- 区间集合**深度**：通过时间线上任何一点的最大区间数
- 所需的**教室数目**必须**至少是区间集合的深度**

- 问题：是否存在一个课程安排，使得教室数目等于区间集合的深度？

Algorithm Interval Partitioning Greedy Algorithm

Sort intervals by starting time so that $s_1 \le s_2 \le \dots \le s_n$;
$d \leftarrow 0$; // number of allocated classrooms
for $j = 1$ ***to*** n **do**
  if *lecture j is compatible with some classroom k* **then**
    schedule lecture j in classroom k;
  else
    allocate a new classroom $d + 1$;
    schedule lecture j in classroom $d + 1$;
    $d \leftarrow d + 1$;
return d;

- 对每个教室 k，维护最后加入课程的完成时间
- 用优先队列维护所有教室

- 时间复杂度 $O(n\log n)$

算法正确性

定理：贪心算法一定找到最小数目教室

证明：假设算法输出的教室间数为 d，且当开放教室 d 时，需要安排课程 j。

因此，课程 j 与前面 d-1个教室都不相容。

不失一般性，记前面d-1个教室安排的课程中与课程 j 不相容的课程记为1, …, d-1，则有

$$f_i > s_j,\ i = 1, \ldots, d\text{-}1。$$

因此，至少存在 d 个时间区间有重叠，即通过时间线上s_j至少有d个区间。故区间集合的深度 $\geq d$。

又由于$d \geq$所需教室数目$\geq$区间集合的深度。

因此，d即为区间集合深度，即d为最小教室数目。

贪心算法举例

- 区间调度问题（活动选择问题）
  - 单资源多请求
  - 每个请求有开始时间和完成时间
  - 满足最大数目的请求
- 区间划分问题
  - 多资源多请求
  - 用尽可能少的资源满足所有请求
- 最小延迟调度问题
  - 单资源多请求
  - 每个请求有截止时间和处理时间，开始时间不固定
  - 最小化最大延迟

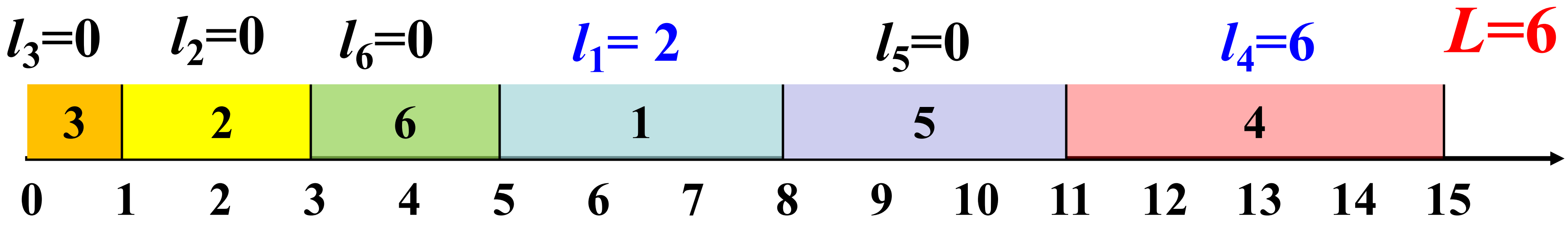
最小延迟调度问题

输入：n个任务，其中

- 单个资源一次只能处理一个任务
- 任务 j 需要 t_j 单位的处理时间，且截止时间为 d_j
- 如果任务 j 开始时间为 s_j, 则完成时间为 $f_j = s_j + t_j$
- 任务 j 的延迟定义为 $l_j = \max\{0, f_j - d_j\}$

目标：对n个任务进行调度，使得最大延迟$L=\max l_j$ 最小

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



最小延迟调度问题

输入：n个任务，其中

- 单个资源一次只能处理一个任务
- 任务 j 需要 t_j 单位的处理时间，且截止时间为 d_j
- 如果任务 j 开始时间为 s_j, 则完成时间为 $f_j = s_j + t_j$
- 任务 j 的延迟定义为 $l_j = \max\{0, f_j - d_j\}$

目标：对n个任务进行调度，使得最大延迟$L=\max l_j$ 最小

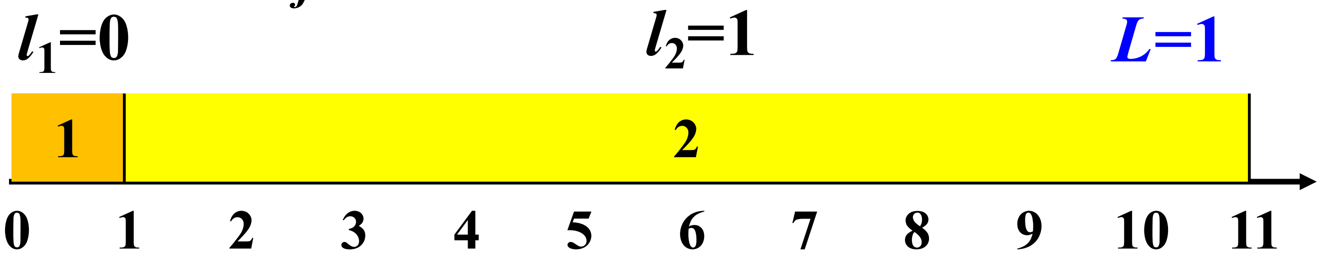
■ 贪心策略

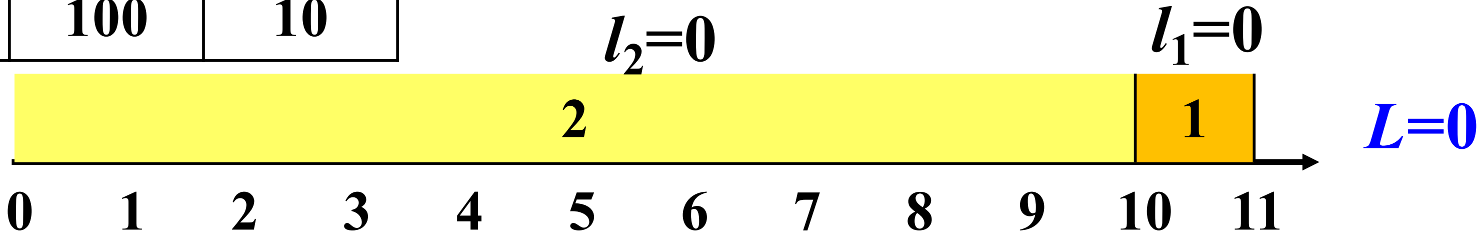
- 策略1：最短处理时间 t_j 优先（反例）
- 策略2：最早截止时间 d_j 优先
- 策略3：最小松弛时间 $d_j - t_j$ 优先（反例）

反例

- 策略1：最短处理时间 t_j 优先

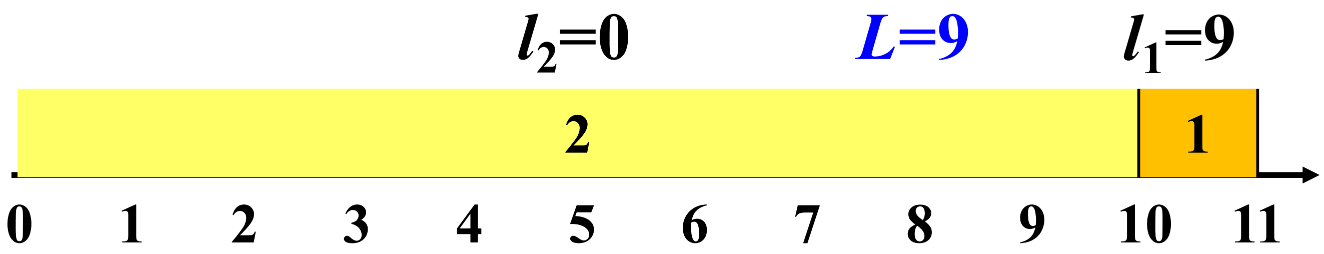
任务	1	2
t_j	1	10
d_j	100	10

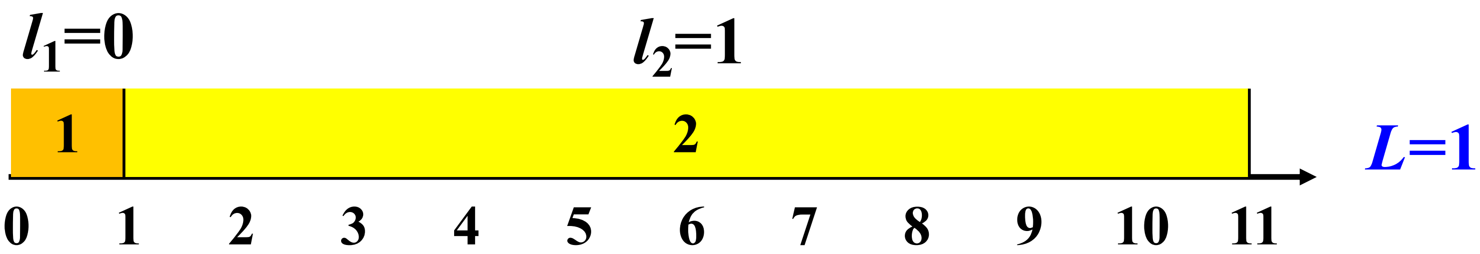




- 策略3：最小松驰时间 $d_j - t_j$ 优先

任务	1	2
t_j	1	10
d_j	2	10





贪心算法

- **策略2：最早截止时间 d_j 优先**

Algorithm Greedy Minimizing Lateness

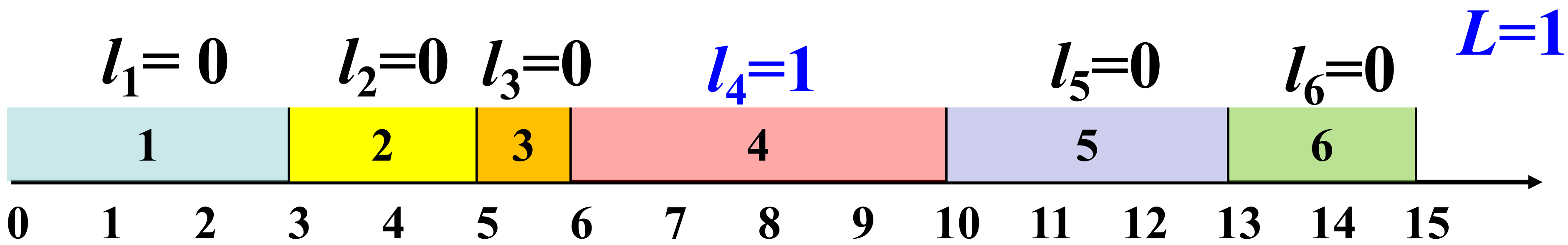
Sort n jobs by deadline so that $d_1 \leq d_2 \leq \ldots \leq d_n$;
$t \leftarrow 0$;
for $j = 1$ ***to*** n **do**
    Assign job j to interval $[t, t + t_j]$;
    $s_j \leftarrow t, f_j \leftarrow t + t_j$;
    $t \leftarrow t + t_j$;
return intervals $[s_j, f_j]$;

- 时间复杂度 ***O*(*n*log *n*)**

贪心算法运行实例

- 策略2：最早截止时间 d_j 优先

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



正确性证明

- 假设任务序列 1, 2, …, n，满足截止时间 $d_1 \le d_2 \le \ldots \le d_n$

定理 1：贪心算法返回的调度 S 一定为最优调度。

证明方法：交换论证（exchange argument）

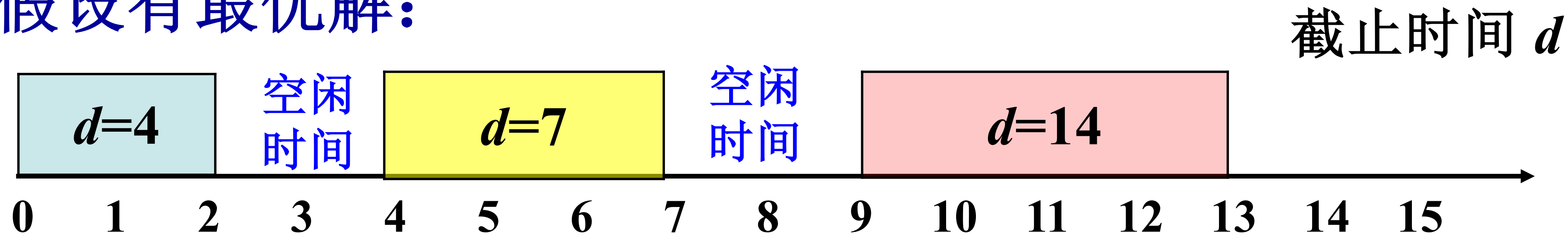
设 S^* 为最优调度。

证明思想：逐步修改 S^*，在每一步保持最优性，最终将它置换为贪心算法找到的调度 S。

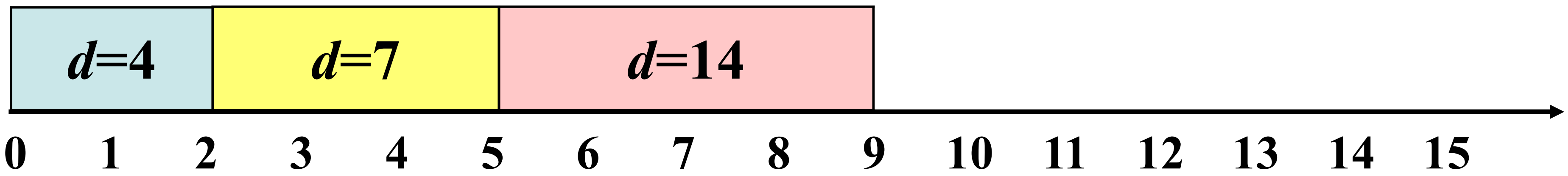
正确性证明

- 假设任务序列 1, 2, …, n，满足截止时间 $d_1 \le d_2 \le \ldots \le d_n$
- 观察1：一定存在一个没有空闲时间的最优调度

假设有最优解：



显然以下也是最优解：



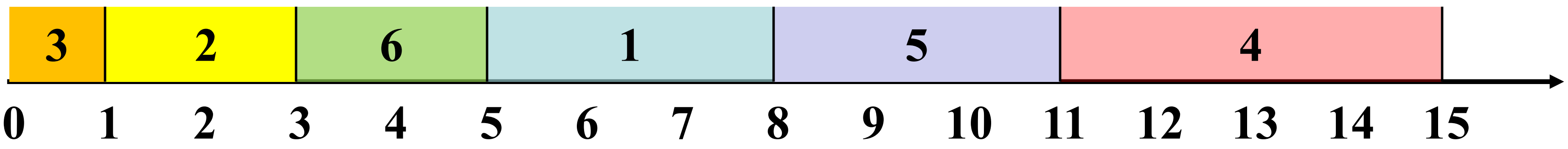
- 观察2：贪心算法返回的调度一定没有空闲时间

正确性证明

- 假设任务序列 1, 2, …, n，满足截止时间 $d_1 \le d_2 \le \dots \le d_n$
- 逆序：给定一个调度 S，如果一对任务i，j 满足:
  - $i < j$，且　　(i 的截止时间在 j 之前)
  - j 在 i 之前被调度，

则称 i, j 构成一个逆序$< i, j >$。

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



逆序：$<2, 3>$, $<1, 3>$, $<1, 2>$, $<1, 6>$, $<5, 6>$, $<4, 6>$, $<4, 5>$

正确性证明

- 假设任务序列 1, 2, …, n，满足截止时间 $d_1 \le d_2 \le \ldots \le d_n$
- 逆序：给定一个调度 S，如果一对任务i，j 满足:
  - $i<j$，且 (i 的截止时间在 j 之前)
  - j 在 i 之前被调度，

则称 i, j 构成一个逆序$<i, j>$。

是否可以通过把最优解的逆序一一消除，产生贪心算法的调度？

- 贪心算法返回的调度不包含逆序。
- 如果一个（没有空闲时间的）调度包含逆序，则一定包含一对相邻的任务构成逆序。

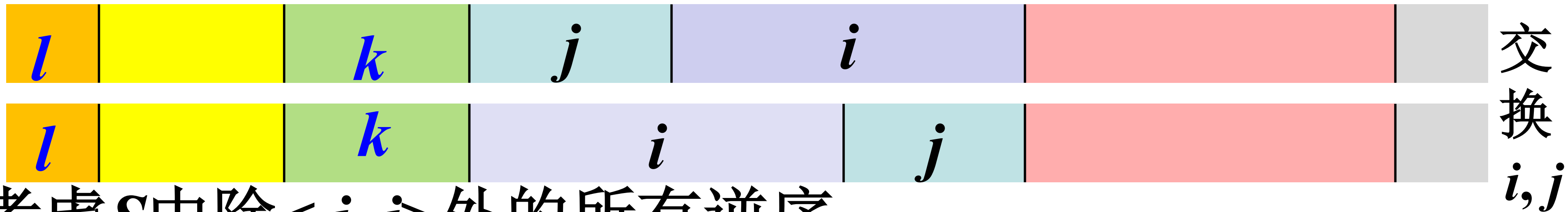
		j	$j_1>j$	$j_2>j$	…	$j_l>j$	$k<j$				i	

从 j 开始往右，设第1个比 j小的任务为 k（即 $k<j$），则k与前一个任务形成逆序。

正确性证明

- 假设任务序列 1, 2, …, n，满足截止时间 $d_1 \le d_2 \le \dots \le d_n$

引理 1：假设任务 i 和 j（$i<j$）在调度 S 中相邻，且构成逆序，则<u>交换 i 和 j 后</u>，S 中逆序数目减 1 且最大延迟不会增大。



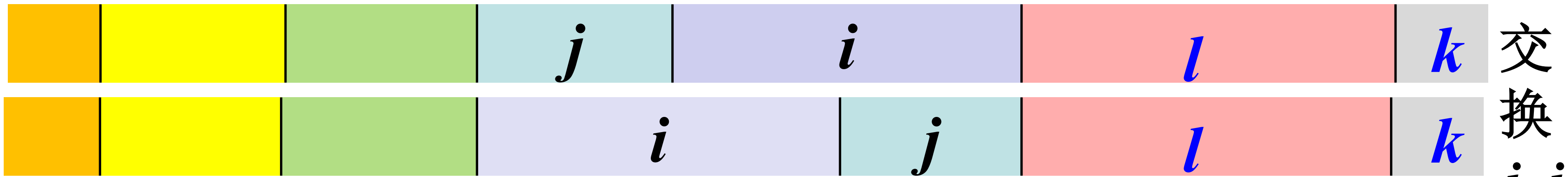
考虑 S 中除 $<i, j>$ 外的所有逆序：

(1) 任务 k, l（$k<l$）构成逆序，且均在任务 j 之前执行

正确性证明

- 假设任务序列 1, 2, …, n，满足截止时间 $d_1 \le d_2 \le \ldots \le d_n$

引理 1：假设任务 i 和 j（$i < j$）在调度 S 中相邻，且构成逆序，则<u>交换 i 和 j 后</u>，S 中逆序数目减 1 且最大延迟不会增大。



考虑 S 中除 $<i, j>$ 外的所有逆序：

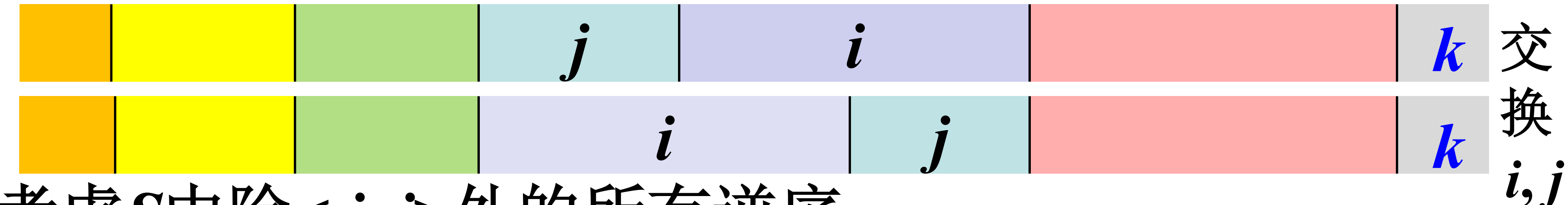
(1) 任务 k, l（$k<l$）构成逆序，且均在任务 j 之前执行

(2) 任务 k, l（$k<l$）构成逆序，且均在任务 i 之后执行

正确性证明

- 假设任务序列 1, 2, …, n，满足截止时间 $d_1 \leq d_2 \leq \ldots \leq d_n$

> 引理 1：假设任务 i 和 j（$i<j$）在调度 S 中相邻，且构成逆序，则交换 i 和 j 后，S 中逆序数目减 1 且最大延迟不会增大。



考虑 S 中除 $<i, j>$ 外的所有逆序：

(1) 任务 k, l（$k<l$）构成逆序，且均在任务 j 之前执行

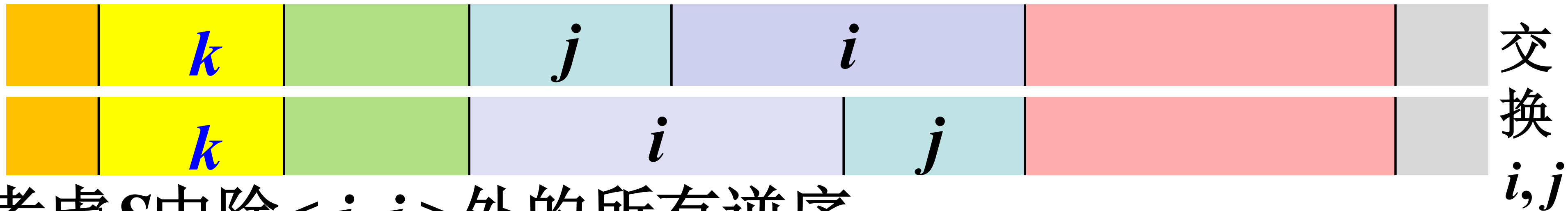
(2) 任务 k, l（$k<l$）构成逆序，且均在任务 i 之后执行

(3) 任务 k 与 j ($k<j$) 或 i ($k<i$) 构成逆序

正确性证明

- 假设任务序列 1, 2, …, *n*，满足截止时间 $d_1 \le d_2 \le \ldots \le d_n$

引理 1：假设任务 *i* 和 *j*（$i<j$）在调度 *S* 中相邻，且构成逆序，则<u>交换 *i* 和 *j* 后</u>，*S* 中逆序数目减 1 且最大延迟不会增大。



考虑*S*中除$<i, j>$外的所有逆序：

(1) 任务 *k*, *l*（$k<l$）构成逆序，且均在任务 *j* 之前执行

(2) 任务*k*, *l*（$k<l$）构成逆序，且均在任务 *i* 之后执行

(3) 任务 *k* 与 *j*（$k<j$）或 *i*（$k<i$）构成逆序

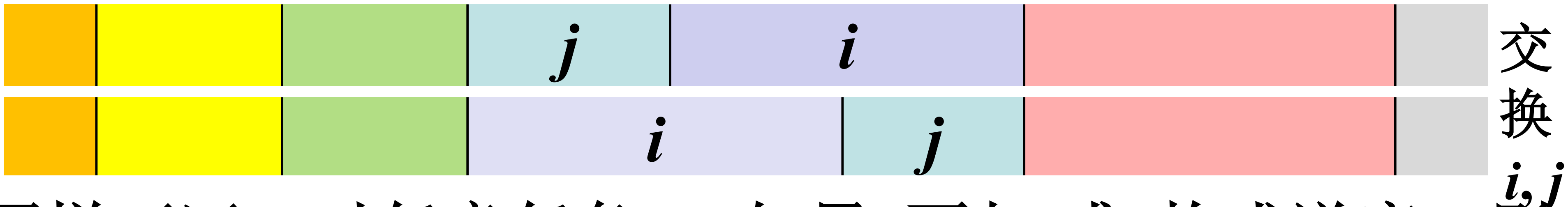
(4) 任务 *k* 与 *j*（$k>j$）或 *i*（$k>i$）构成逆序

以上逆序在交换 *i* 和 *j* 后仍为逆序。

正确性证明

- 假设任务序列 1, 2, …, n，满足截止时间 $d_1 \le d_2 \le \ldots \le d_n$

引理 1：假设任务 i 和 j（$i<j$）在调度 S 中相邻，且构成逆序，则<u>交换 i 和 j 后</u>，S 中逆序数目减 1 且最大延迟不会增大。



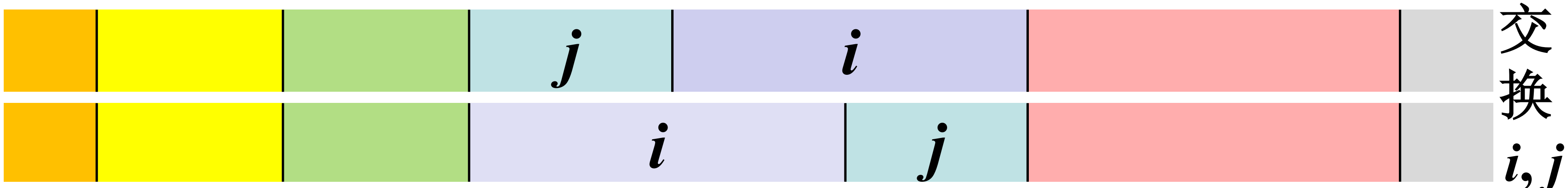
同样可证：对任意任务 k，如果 k 不与 i 或 j 构成逆序，则交换 i 与 j 后，k 仍不与 i 或 j 构成逆序。

得 S 中逆序数目减 1.

正确性证明

- 假设任务序列 1, 2, …, n，满足截止时间 $d_1 \le d_2 \le \ldots \le d_n$

> 引理 1：假设任务 i 和 j（$i < j$）在调度 S 中相邻，且构成逆序，则交换 i 和 j 后，S 中逆序数目减 1 且最大延迟不会增大。



对任意任务 k，令其原延迟为 l_k，交换 i, j后延迟为l'_k，完成时间为f'_k

- 若$k \ne i, j$，则 $l_k = l'_k$
- 考虑任务i，有：$l'_i \le l_i$
- 考虑任务j，有：$l'_j = f'_j - d_j = f_i - d_j \le f_i - d_i = l_i$

$f'_j = f_i$

$d_i \le d_j$

因此，最大延迟不会增大。

正确性证明

- 假设任务序列 1, 2, …, n，满足 $d_1 \le d_2 \le \dots \le d_n$

定理 1：贪心算法返回的调度 S 一定为最优调度。

证明：设 S^* 为最优调度，且不包含空闲时间。

(1) 若 S^* 中没有逆序，显然有$S^* = S$。

(2) 若 S^* 中有逆序，则最多有C_n^2个逆序.

因此至多经过C_n^2次交换相邻任务够成的逆序，得到一个不具有逆序的调度，记为S。

显然，S 是贪心算法返回的调度（不考虑截止时间相同的任务的顺序）。

由引理 1知， S 的最大延迟一定不超过S^*的最大延迟。

由 S^* 为最优调度知，S 一定为最优调度。

最小延迟调度的扩展

输入：*n*个任务，其中

- 单个资源一次只能处理一个任务
- 任务 j 有最早处理时间r_i， t_j 单位的处理时间，且截止时间为 d_j
- 如果任务 j 开始时间为 s_j, 则完成时间为 $f_j = s_j + t_j$
- 任务 j 的延迟定义为 $l_j = \max\{0, f_j - d_j\}$

问题：是否能对*n*个任务进行调度，使得每个任务在截止时间完成？

最大延迟 $L = \max l_j = 0$

- 是 NP-完全问题
  - 目前仍未有多项式时间算法

主要内容

- 贪心算法举例
  - 区间调度问题（活动选择问题）
  - 区间划分问题
  - 最小延迟调度问题
- 贪心算法的理论基础——拟阵
  - 什么是拟阵（**Matroids**）？
  - 拟阵的通用贪心解法
  - 拟阵通用贪心解法的正确性
  - 基于拟阵的贪心算法举例

主要内容

- 贪心算法的理论基础
  - 什么是拟阵（Matroids）？
  - 拟阵的通用贪心解法
  - 拟阵通用贪心解法的正确性
  - 基于拟阵的贪心算法举例

线性相关与线性无关

◆ 线性相关的两个性质

✓ 如果$X=\{x_1, x_2,\ldots, x_k\}$是一个线性无关向量组，则对 X 的任意子集 X'也是线性无关的.

✓ 如果 $X=\{x_1, x_2,\ldots, x_r\}$和 $Y=\{y_1, y_2,\ldots, y_m\}$是两个线性无关向量组且 $m>r$，则必存在一个 $y_i \in Y$，使得

$$X \cup \{y_i\}\text{是一个线性无关向量组.}$$

1935年，美国数学家哈斯勒·惠特尼（**Hassler Whitney**）把以上两条性质进行了抽象推广，提出了拟阵概念.

拟 阵

■ 一个拟阵是一个满足如下条件的有序对 $M = (S, I)$

(1) S是非空有限集；

(2) I 是 S 的子集的一个非空族，这些子集称为S的独立子集，即 I 满足遗传性质：若$B \in I$，且$A \subseteq B$，则$A \in I$；

（若B是S的独立子集，则B的任意子集均是S的独立子集

(3) I 满足交换性质，即若$A \in I$, $B \in I$且 $|A| < |B|$，则存在某个元素 $x \in B - A$，使得$A \cup \{x\} \in I$.

例（矩阵拟阵）： 给定实数域上的矩阵 T，令

- S是T的列的集合，且
- $A \in I$ 当且仅当 A 中的列是线性无关的，

则 (S, I) 是一个拟阵.

例（图拟阵）：给定无向图$G=(V, E\neq\emptyset)$，定义

- S_G 为 图 G 的边集 E,
- I_G 是 G 的无循环边集的非空族：如果 A是 E 的子集，则 $A\in I_G$ 当且仅当A是无圈的，即图$G_A=(V, A)$ 构成一个森林.

则 $M_G=(S_G, I_G)$ 是一个拟阵.

证明：$M_G=(S_G, I_G)$满足拟阵的3个条件.

(1) 因为 S_G 为图G的边集，显然非空;

(2) I_G满足遗传性质：由于从 S_G的一个无循环边集中去掉若干条边不会产生循环，即森林的子集还是森林，

因此S_G的无循环边集族 I_G 具有遗传性质.

例（图拟阵）：给定无向图$G = (V, E\neq \emptyset)$，定义

- S_G为 图 G 的边集 E，
- I_G是 G 的无循环边集的非空族：如果 A是 E 的子集，则 $A\in I_G$ 当且仅当A是无圈的，即图$G_A=(V, A)$ 构成一个森林.

则 $M_G = (S_G, I_G)$ 是一个拟阵.

证明：(3) I_G 满足交换性质：设$G_A=(V, A)$ 和 $G_B =(V, B)$ 是图G的两个森林，且$|B|>|A|$.

有结论：由于有 k 条边的森林恰由 n-k 棵树组成，其中n是该森林的节点数，因此 G_B中的树比G_A中的少.

（假设森林 $F = (V_F, E_F)$包含了 t 棵树，其中第 i 棵树包含 v_i个顶点和 e_i条边，

则有 $|E_F|=\sum_{i=1}^{t} e_i=\sum_{i=1}^{t}(v_i-1)=\sum_{i=1}^{t} v_i-t =|V_F|-t$，得 $t = |V_F| - |E_F|$.

由于$|B|>|A|$，得 $|V|- |B|< |V|- |A|$，即 森林 G_B包含的树比 G_A中的少。）

例（图拟阵）：给定无向图$G=(V, E\neq \emptyset)$，定义

- S_G为 图 G 的边集 E,
- I_G是 G 的无循环边集的非空族：如果 A是 E 的子集，则 $A\in I_G$ 当且仅当A是无圈的，即图$G_A=(V, A)$ 构成一个森林.

则 $M_G=(S_G, I_G)$ 是一个拟阵.

证明：(3) I_G满足交换性质： 设$G_A=(V, A)$ 和 $G_B=(V, B)$ 是图G的两个森林，且$|B|>|A|$.

有结论：由于有 k 条边的森林恰由 n-k 棵树组成，其中 n是该森林的节点数，因此 G_B中的树比G_A中的少.

则G_B中存在一棵树T，其顶点在森林G_A的不同的棵树中.

由于T是连通的，故 T 中必有一条边 (u, v)满足u, v分别在G_A的两棵树中. 因此将 (u, v) 加入 A 不会产生循环.

所以 I_G满足交换性质.

综上所述, $M_G=(S_G, I_G)$ 是一个拟阵.

拟阵的重要概念和性质（1）

- 独立子集的扩展：给定拟阵 $M=(S, I)$，对于 I 中的独立子集 $A\in I$， 若 S 有一元素 $x\notin A$，使得将 x 加入 A后仍保持独立性，即 $A\cup\{x\}\in I$，则称 x为A的一个扩展.

例：对于图拟阵 M_G， 如果 A是一个边独立集(森林)，则边 e 是 A 的一个扩展当且仅当 e不在A中且将e加入 A 中不会形成圈.

- 对拟阵 M中的一个独立子集 A，如果A不存在扩展，则称A是最大独立子集，或拟阵的基.

定理 1 拟阵 M 中所有最大独立子集具有相同大小.

定理 1 拟阵 M 中所有最大独立子集具有相同大小.

证明：(反证法) 假设 A，B是M的两个最大独立子集，且$|A|<|B|$.

则由交换性质可知，存在 $x \in B-A$，使得 $A\cup\{x\}$是一个独立子集，与A是最大独立子集矛盾.

例：假设G是一个连通无向图．考虑G的图拟阵 M_G.

则 M_G 的最大独立子集必定是一棵边数为$|V|$-1，连接了G的所有顶点的树，即G的生成树。

拟阵的重要概念和性质（2）

- **加权矩阵**：如果一个拟阵 $M=(S, I)$ 关联一个加权函数w，使得对于任意$x\in S$，有$w(x)>0$，则称拟阵 M 为加权拟阵.

- 通过求和，可将权重函数 w 扩展到 S 的任意子集A:

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

例： 考虑 G的图拟阵 M_G. 如果以$w(e)$表示G中边e的长度，则$w(A)$ 为边集A中所有边长的总长度.

主要内容

- 什么是拟阵(**Matroids**)?
- 拟阵的通用贪心解法
- 拟阵通用贪心解法的正确性
- 基于拟阵的贪心算法举例

加权拟阵上的贪心算法

- 许多用贪心算法求解的问题可以表示为加权拟阵的最大权独立子集问题

给定加权拟阵 $M = (S, I)$，计算 S 的具有最大权值 $w(A)$的独立子集 $A \in I$，称为拟阵M的最优子集.

- 由于S中任一元素x的权$W(x)$是正的，因此，最优子集也一定是最大独立子集 (不存在可扩展元素).

问题：如何计算最大独立子集？

✓ 独立子集的扩展

加权拟阵上的贪心算法框架

输入：具有正权函数 w 的加权拟阵$M = (S, I)$

输出：M的最优子集A.

GREEDY(M, w)

1. $A=\emptyset$
2. sort S into monotonically *decreasing* order by weight w
3. for each $x\in S$, taken in monotonically decreasing order by weight $w(x)$
4. if $A\cup\{x\}\in I$
5. $A= A\cup\{x\}$
6. return A

$O(n\log n)$

$O(nf(n))$

- 复杂度：$O(n\log n + nf(n))$，其中
  - $n = |S|$
  - $O(f(n))$为每次检查 $A\cup\{x\}$ 是否为独立子集的时间

主要内容

- 什么是拟阵(Matroids)?
- 拟阵的通用贪心解法
- 拟阵通用贪心解法的正确性
- 基于拟阵的贪心算法举例

GREEDY算法的正确性

定理 2 如果 $M=(S, I)$ 是具有正权函数 w 的加权拟阵，则调用GREEDY(M, w) 返回一个最优子集.

GREEDY(M, w)

1. $A=\emptyset$
2. sort S into monotonically *decreasing* order by weight w
3. for each $x \in S$, taken in monotonically decreasing order by weight $w(x)$
4. if $A \cup \{x\} \in I$
5. $A = A \cup \{x\}$
6. return A

- 第一个选中的独立子集 $\{x\}$一定会包含于一个最优子集A中
- 第一个被舍弃的元素，永远不可能用于构造最优子集
- 归纳利用以上两个结论

引理 1 （拟阵的贪心选择性质）设$M=(S, I)$ 是具有正权函数w的带权拟阵，且S中元素依权值从大到小排列．又设 $x\in S$是 S 中第一个使得 $\{x\}$ 是独立子集的元素，则存在 S的最优子集 A 使得 $x\in A$.

证明: (1) 若不存在$x\in S$ 使得$\{x\}$是独立子集，则引理1是平凡的.

(2) 设 B是一个非空的最优子集.

由于$B\in I$，且 I 具有遗传性质，故 B中所有单个元素子集 $\{y\}$ 均为独立子集.

又由于 x 是 S中的第一个单元素独立子集，

故对任意的 $y\in B$，均有：$w(x) \geq w(y)$.

(a)若 $x\in B$，则只要令$A=B$，定理得证；

引理 1 （拟阵的贪心选择性质）设$M=(S, I)$是具有正权函数w的带权拟阵，且S中元素依权值从大到小排列．又设 $x\in S$是 S 中第一个使得 $\{x\}$ 是独立子集的元素，则存在 S的最优子集 A 使得 $x\in A$.

(b) 若$x\notin B$，按如下方法构造包含元素x的最优子集A.

(i) 首先，设$A=\{x\}$，此时A是一个独立子集.

若$|B|=|A|=1$，则定理得证.

(ii) 若$|B|>|A|$，反复利用拟阵 M的交换性质，从B中选择一个新元素加入A中并保持A的独立性，直到$|A|=|B|$.

此时，必有一元素 $y\in B$且$y\notin A$,使得

$$A=B-\{y\}\cup\{x\}\text{且}w(x)\geq w(y)$$

且满足：$w(A) = w(B)-w(y)+w(x) \geq w(B)$

由于B是一个最优子集，所以$w(B) \geq w(A)$.

因此$w(A)= w(B)$，即A也是一个最优子集，且$x\in A$.

GREEDY算法的正确性

定理 2 如果 $M=(S, I)$ 是具有正权函数 w 的加权拟阵，则调用GREEDY(M, w) 返回一个最优子集.

- 第一个选中的独立子集 $\{x\}$ 一定会包含于一个最优子集A中
- 第一个被舍弃的元素，以后也永远不可能用于构造最优子集
- 归纳利用以上两个结论

GREEDY算法的正确性

定理 2 如果 $M=(S, I)$ 是具有正权函数 w 的加权拟阵，则调用GREEDY(M, w) 返回一个最优子集.

- 第一个选中的独立子集 $\{x\}$一定会包含于一个最优子集A中
- 第一个被舍弃的元素，以后也永远不可能用于构造最优子集
- 归纳利用以上两个结论

引理2：设$M = (S, I)$ 是拟阵．若S中元素 x 不是空集$\emptyset$的可扩展元素，则 x 也不可能是 S中任一独立子集A的可扩展元素.

证明：(反证法) 设 $x\in S$不是$\emptyset$的一个可扩展元素，但它是 S 的某独立子集A的一个可扩展元素，即

$$A\cup\{x\}\in I.$$

由 I 的遗传性质可知 $\{x\}$ 是独立的.

这与 x 不是空集 $\emptyset$ 的一个可扩展元素相矛盾.

- 算法Greedy在初始化独立子集A时舍弃的元素可以永远舍弃.

GREEDY算法的正确性

定理 2 如果 M＝(S, I) 是具有正权函数 w 的加权拟阵，则调用GREEDY(M, w) 返回一个最优子集.

- 第一个选中的独立子集 $\{x\}$一定会包含于一个最优子集A中
- 第一个被舍弃的元素，以后也永远不可能用于构造最优子集
- 归纳利用以上两个结论

GREEDY算法的正确性

定理 2 如果 M＝(S, I) 是具有正权函数 w 的加权拟阵，则调用GREEDY(M, w) 返回一个最优子集.

- 第一个选中的独立子集 $\{x\}$一定会包含于一个最优子集A中
- 第一个被舍弃的元素，以后也永远不可能用于构造最优子集
- 归纳利用以上两个结论

引理3（拟阵的最优子结构性质）设 x 是求带权拟阵 $M=(S, I)$ 最优子集的贪心算法GREEDY所选择的S中的第一个元素．那么，原问题可简化为求带权拟阵 $M'=(S', I')$的最优子集问题，其中：

- $S'=\{y \mid y\in S$且$\{x, y\}\in I\}$，即 y是 x的可扩展元素$\}$
- $I'=\{B \mid B\subseteq S-\{x\}$ 且 $B\cup\{x\}\in I\}$
- M'的权函数是M的权函数在S'上的限制（称M'为M关于元素x的收缩）．

证明：首先，由M'的定义可得：若 A是 M的包含元素 x的最大独立子集，则 $A'=A-\{x\}$ 是M' 的一个独立子集．
反之，M' 的任一独立子集 A' 产生 M 的一个独立子集 $A=A'\cup\{x\}$．
在这两种情形下均有：$W(A)=W(A')+W(x)$．
因此 M的包含元素 x的最优子集包含 M'的一个最优子集，反之亦然．

定理 2 如果 $M=(S, I)$ 是具有正权函数 w 的加权拟阵，则调用GREEDY(M, w) 返回一个最优子集.

证明：(1)由引理1得，如果第一次加入A的元素是x，则必存在包含元素x的一个最优子集．因此，Greedy第一次选择是正确的.

(2)由引理2知，选择 x 时被舍弃的元素不可能被再选中，即它们不可能是任一最优子集的元素．因此，这些元素可以永远舍弃.

(3) 由引理3可知，Greedy选择了元素 x后，原问题简化为求拟阵M'的最优子集问题.

由于对 $M'=(S', I')$中的任一独立子集 $B\in I'$，均有$B\cup\{x\}$在M中是独立的（由M'的定义可知）.

因此，GREEDY选择了元素 x后，后续求解将演变为求拟阵$M'=(S', I')$的最优子集问题.

由归纳法可知：其后继步骤求出M'的一个最优子集，从而算法GREEDY最终求出的是M的一个最优子集.

加权拟阵最优子集的贪心算法框架如下：

输入：具有正权函数 W 的 $\boxed{\text{加权拟阵} M = (S, I)}$

输出：M的最优子集A.

GREEDY(M, w)

1. $A=\emptyset$
2. sort S into monotonically *decreasing* order by weight w
3. for each $x \in S$, taken in monotonically decreasing order by weight $w(x)$
4. if $A \cup \{x\} \in I$
5. $A = A \cup \{x\}$
6. return A

- 复杂度：$O(n\log n + nf(n))$，其中
  - $n = |S|$
  - $O(f(n))$为每次检查 $A\cup\{x\}$ 是否为独立子集的时间

主要内容

- 什么是拟阵(Matroids)?
- 拟阵的通用贪心解法
- 拟阵通用贪心解法的正确性
- 基于拟阵的贪心算法举例
  - 最小生成树问题
  - 单位时间任务调度问题

最小生成树问题

输入：无向连通图 $G = (V, E)$ 及边权函数 w，使得G中的每一条边 $(u,v)\in E$ 有权值 $w(u,v)$.

输出：G的最小生成树T，即边权和最小的生成树.

- 可转化为一个拟阵中找出最大独立子集的问题

例（图拟阵）：给定无向图$G = (V, E\neq \emptyset)$，定义

- S_G为 图 G 的边集 E,
- I_G是 G 的无循环边集的非空族：如果 A是 E 的子集，则 $A\in I_G$当且仅当A是无圈的，即图$G_A=(V, A)$ 构成一个森林.

则 $M_G = (S_G, I_G)$ 是一个拟阵.

Kruskal 算法

输入：无向连通图 $G = (V, E)$ 及边权函数 w

（构造出与最小生成树对应的权函数，转化为最大独立集问题：对任意边$e \in E$, 令$w'(e)=w_0-w(e)>0$）

输出：最小生成树T

1. $A \leftarrow \emptyset$;
2. 对E中所有边按照边权值$w'(e)$的大小递减排序；
3. 按边权值大小递减顺序，对于E中每条边 (u,v),
4. 若$A \cup \{(u, v)\}$不构成回路
5. $A \leftarrow A \cup \{(u, v)\}$;
6. return A

如何判断？

- $A \cup \{(u, v)\}$不构成回路当且仅当$A \cup \{(u, v)\}$是一个森林
- 如果u, v位于A中同一棵树内， $A \cup \{(u, v)\}$会出现循环
- 如果u, v不位于A中同一棵树内， $A \cup \{(u, v)\}$不会出现循环
- 问题：如何判断u, v是否在A中的同一棵树内？
  使用不相交集（**disjoint set**）把树表示为集合

不相交集

设U为集合， $n = |U|$. 对任意 $u,v \in U$:

- **Create-Set(u):** 创建一个包含节点u的集合
  - O(1)
- **Find-Set(u):** 找到包含元素u的集合
  - O(log n)
- **Union(u, v):** 合并包含u与v的两个集合
  - O(log n)

Kruskal's Algorithm

Input: A graph $\boldsymbol{G}$, a matrix $\boldsymbol{w}$ representing the weights between vertices in G
Output: MST of G
Sort E in increasing order by weight w;// $O(|E| \log |E|)$
// After sorting $E = \langle \{u_1, v_1\}, \{u_2, v_2\}, ..., \{u_{|E|}, v_{|E|}\} \rangle$
$A \leftarrow \{\}$;
for $u \in V$ **do**
    Create-Set(u);// $O(|V|)$
end
for $e_i \in E$ **do**
    // $O(|E| \log |V|)$
    if *Find-Set*(u_i)$\neq$*Find-Set*(v_i) **then**
        add $\{u_i, v_i\}$ to A;
        Union(u_i, v_i);
    end
end
return A;

Create-Set(x):

```
x.parent ← x;
```

Find-Set(x):

- 找到x所在树的根

```
while x ≠ x.parent do
|   x ← x.parent;
end
```

Union(x, y)

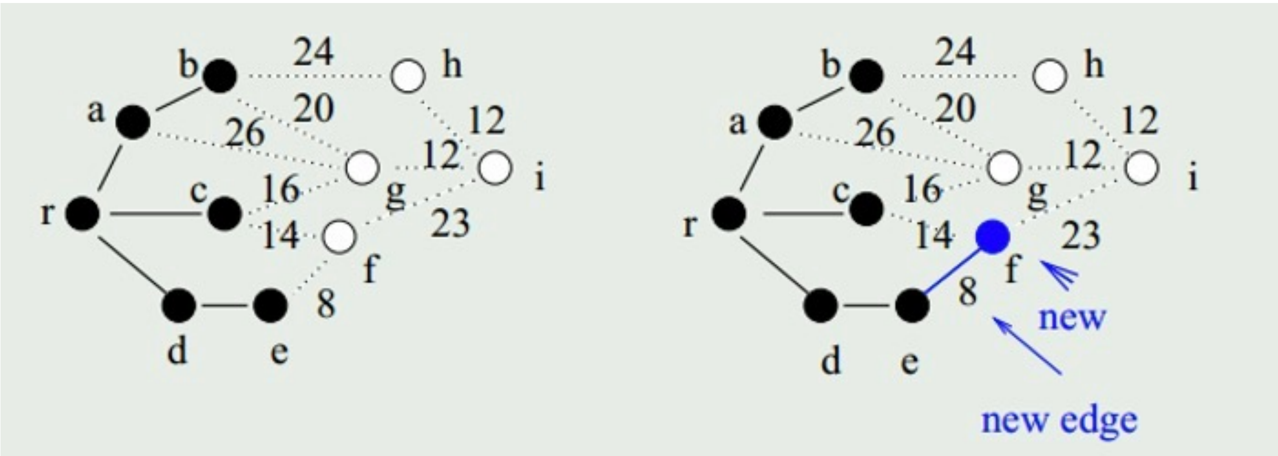
Input: Two elements $\boldsymbol{x}, \boldsymbol{y} \in U$
Output: None

$a \leftarrow$Find-Set(x);
$b \leftarrow$Find-Set(y);
if $a.height \le b.height$ **then**
  if *a.height is equal to b.height* **then**
    b.height++;
  end
  a.parent $\leftarrow b$;
end
else
  b.parent $\leftarrow a$;
end

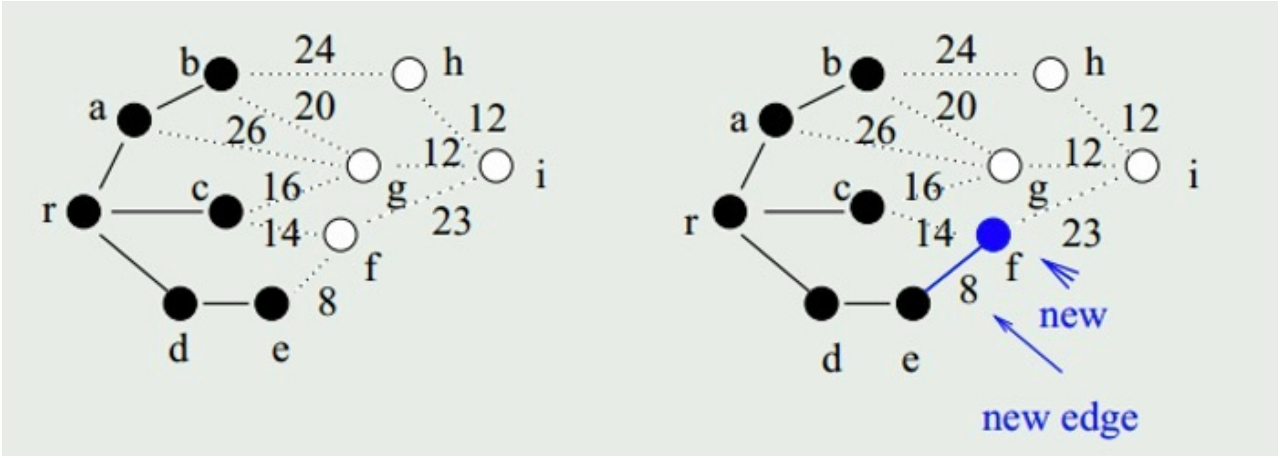
总是把高的树的根作为矮的树的根的父节点

另一个贪心算法：Prim 算法

- **Kruskal's算法**
  - 每次加一条边，维护一个森林
- **Prim算法**
  - 每次加一条边，维护一棵树
  - 算法思想不满足拟阵（不满足遗传性质）



另一个贪心算法：Prim 算法



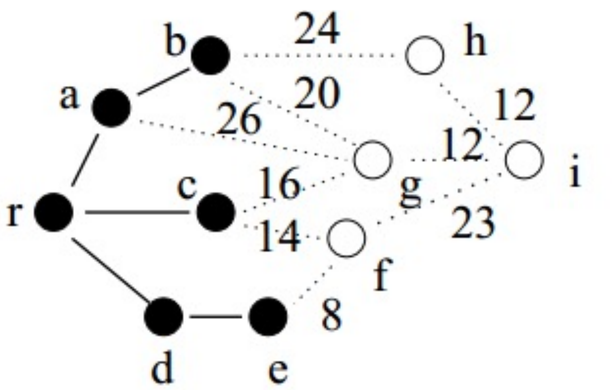
算法 Prim(G,E,W)

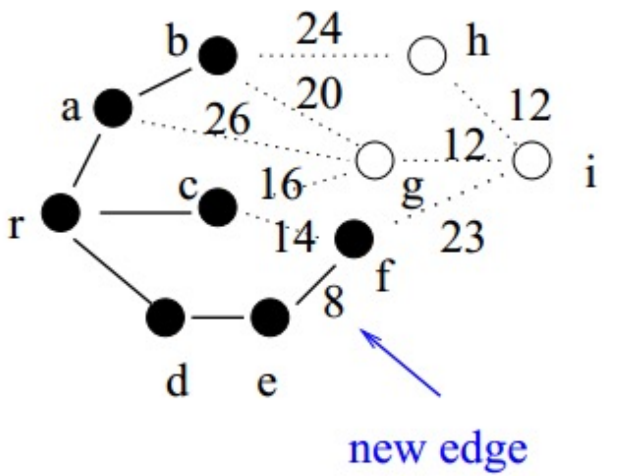
1. $S \leftarrow \{1\}$
2. **while** $V-S \neq \varnothing$ **do**
3. 　从$V-S$中选择j使得j到S中顶点的边权最小
4. 　$S \leftarrow S \cup \{j\}$

优先队列

优先队列：(*u*, key[*u*])

- $u \in$ V-S,
- key[*u*]：从u到*S*中节点的边中权重最小边的权重
- pred[*u*]：以上边在*S*中的端点





key[f] = 8, pred[f] = e

key[i] = infinity, pred[i] = nil

key[g] = 16, pred[g] = c

key[h] = 24, pred[h] = b

→ f has the minimum key

key[i] = 23, pred[i] = f

After adding the new edge and vertex f, update the key[v] and pred[v] for each vertex v adjacent to f

Prim算法

Prim(G,w,r)

```
Input: A graph G, a matrix w representing the weights between vertices
       in G, the algorithm will start at root vertex r
Output: None
Let color[1...|V|], key[1...|V|], pred[1...|V|] be new arrays;
for u ∈ V do
    color[u] ← WHITE,key[u] ← +∞;// Initialize
end
key[r] ← 0, pred[r] ← NULL;// Start at root vertex
Q ← new PriQueue(V);// put vertices in Q
while Q is nonempty do
    u ← Q.Extract-Min();// lightest edge
    for v ∈ adj[u] do
        if (color[v] ← WHITE)&&(w[u, v] < key[v]) then
            key[v] ← w[u, v];// new lightest edge
            Q.Decrease-Key(v, key[v]);
            pred[v] ← u;
        end
    end
    color[u] ← BLACK;
end
```

Prim算法

Input: A graph $\boldsymbol{G}$, a matrix $\boldsymbol{w}$ representing the weights between vertices in G, the algorithm will start at root vertex $\boldsymbol{r}$
Output: None

```
Let color[1...|V|], key[1...|V|], pred[1...|V|] be new arrays;
for u ∈ V do
    color[u] ← WHITE, key[u] ← +∞; // O(V)
end
key[r] ← 0, pred[r] ← NULL;
Q ← new PriQueue(V); // O(V)
while Q is nonempty do
    u ← Q.Extract-Min(); // O(logV)
    for v ∈ adj[u] do
        if (color[v] ← WHITE)&&(w[u,v] < key[v]) then
            key[v] ← w[u,v];
            Q.Decrease-Key(v, key[v]); // O(logV)
            pred[v] ← u;
        end
    end
    color[u] ← BLACK;
end
```

$$O((V+E)\log V) = O(E \log V)$$

主要内容

- 什么是拟阵(Matroids)?
- 拟阵的通用贪心解法
- 拟阵通用贪心解法的正确性
- 基于拟阵的贪心算法举例
  - 最小生成树问题
  - 单位时间任务调度问题

拟 阵

- 一个拟阵是一个满足如下条件的有序对 $M=(S, I)$

(1) S是非空有限集；

(2) I 是 S 的子集的一个非空族，这些子集称为S的独立子集，即 I 满足遗传性质：若$B\in I$，且$A\subseteq B$，则$A\in I$；

（若B是S的独立子集，则B的任意子集均是S的独立子集

(3) I 满足交换性质，即若$A\in I$, $B\in I$且 $|A|<|B|$，则存在某个元素 $x\in B-A$，使得$A\cup\{x\}\in I$.

加权拟阵上的贪心算法框架

输入：具有正权函数 w 的加权拟阵$M = (S, I)$

输出：M的最优子集A.

GREEDY(M, w)

1. $A=\emptyset$
2. sort S into monotonically *decreasing* order by weight w
3. for each $x \in S$, taken in monotonically decreasing order by weight $w(x)$
4. if $A \cup \{x\} \in I$
5. $A = A \cup \{x\}$
6. return A

$O(n\log n)$

$O(nf(n))$

- 复杂度：$O(n\log n + nf(n))$，其中
  - $n = |S|$
  - $O(f(n))$为每次检查 $A \cup \{x\}$ 是否为独立子集的时间

单位时间任务调度问题

- 单位时间任务：需要一个单位的时间来运行的任务
- 调度：给定一个单位时间任务的有限集合 S，

对 S 的一个调度即为 S 的一个排列，它规定了各任务的执行顺序.

  - 第一个任务开始于时间0，结束于时间1
  - 第二个任务开始于时间1，结束于时间2，… …

例： 任务集合 $S=\{a_1, a_2, \ldots, a_n\}$

S 的一个调度 $a_{i1}\ a_{i2}\ \ldots\ a_{in}$， $a_{ij} \in S, j \in [1, n]$

具有截止时间和误时惩罚的单位时间任务的调度问题

输入：

(1) n个单位时间任务的集合 $S=\{\ a_1, a_2, \ldots, a_n\ \}$；

(2) 任务a_i 的截止时间 d_i, $1\leq i\leq n$, $1\leq d_i\leq n$，即要求任务i 在时间 d_i之前结束；

(3) 任务 a_i 的误时惩罚 w_i, $1\leq i\leq n$, 即任务 i 未在规定时间d_i之前结束将招致 w_i 惩罚；若按时完成则无惩罚.

输出：

S的最优调度，即为总误时惩罚最小的调度.

基本概念

- 给定一个调度，定义：
  - 迟(late)任务：一个任务在截止时间后完成
  - 早(early)任务：一个任务在截止时间前完成
  - 早任务优先形式：早的任务总是在迟的任务之前

例. 早任务优先调度：

早任务 a_{i1} a_{i2} … a_{ik} $a_{i,k+1}$ … a_{ip} … a_{in} 迟任务

截止时间： $d_{ij} \ge j,\ j=1,\ldots,k;\ d_{ij} < j,\ j=k+1,\ldots,n$

- 任意一个调度总可以置换成早任务优先的形式

a_{i1}	a_{i2}	…	a_{ij}	…	$a_{i,k+1}$	…	a_{ip}	…	a_{in}

$d_{ij}<j<p$ $\qquad$ $d_{ip} \ge p>j$

交换 a_{ip} 与 a_{ij} 的位置，a_{ip} 仍是早任务， a_{ij} 仍是迟任务

问题：如何转化为拟阵？

- 独立任务集：称一个任务的集合A是独立的，如果存在A的一个调度，使得没有一个任务是迟的.

设 $N_t(A)$是任务集A中所有截止时间小于等于 t 的任务数，t =1,2,…, n.

引理4 对于 S 的任一任务子集 A，下面各命题是等价的.

(1) 任务子集 A 是独立子集.

(2) 对于 t = 1, 2, …, n， $N_t(A) \le t$.

截止时间小于等于t的任务数最多有 t 个

(3) 若A中任务依其截止时间非递减排列，则A中所有任务都是早的.

引理4 对于 S 的任一任务子集 A，下面各命题是等价的.

(1) 任务子集 A 是独立子集.

(2) 对于 $t = 1, 2, \ldots, n$，$N_t(A) \le t$.

截止时间小于等于t的任务数最多有 t 个

(3) 若A中任务依其截止时间非递减排列，则A中所有任务都是早任务.

证明：(1)→(2)： (反证法) 假设任务子集 A是独立的，且存在某个 t 使得 $N_t(A) > t$,

则A中有多于 t 个任务要在时间 t 之前完成，

故A中必有迟任务，这与A是独立任务子集相矛盾.

因此，对所有t =1,2,…, n，$N_t(A) \le t$;

(2)→(3)：将A中任务按截止时间的非递减排列，则(2)中不等式意味着排序后A中截止时间为 t 的任务前面，需要调度的任务数少于 t. 故排序后A中所有任务都是早任务;

(3) →(1)：由独立子集的定义可得.

定理3 设S是带有截止时间的单位时间任务集，I 是 S的所有独立任务子集构成的集合．则有序对(S, I)是拟阵.

证明：(1)首先，独立任务集的子集显然也是独立子集．故 I 满足遗传性质.

存在一个调度，使得没有一个任务是迟的

(2)设A、B为独立任务子集且$|B|>|A|$，下面证明 I 满足交换性质.

B中截止时间小于等于 k 的任务数少于A中截止时间小于等于 k的任务数

设从时刻 1开始，最后一次出现 $N_t(B) \le N_t(A)$ 的 t 值为 k, 即

$$k = \max\{t \mid N_t(B) \le N_t(A), 1\le t \le n\}，其中n=|S|.$$

由于 $N_n(B)=|B|, N_n(A)=|A|$，而$|B| > |A|$，即 $N_n(B)>N_n(A)$.

因此必有 $k<n$，对于满足$k+1\le j \le n$的 j 有

$$N_j(B) > N_j(A).$$

取 $x\in B-A$，且 x的截止时间为$k+1$．令 $A'=A\cup\{x\}$.

下面证明 A'是独立的.

定理3 设S是带有截止时间的单位时间任务集，I 是 S的所有独立任务子集构成的集合．则有序对(S, I)是拟阵.

证明：下面证明 A'是独立的.

首先，由于A是独立的，故对于$1 \le t \le k$有：

$$N_t(A') = N_t(A) \le t.$$

又由于B是独立的，故对$k+1 \le t \le n$有

$$N_t(A') = N_t(A)+1 \le N_t(B) \le t.$$

由引理4 的条件(2)可知：A'是独立的.

因此，I 满足交换性质.

综上所述，可得 (S, I) 是一个拟阵.

定理3 设S是带有截止时间的单位时间任务集，I 是 S的所有独立任务子集构成的集合．则有序对(S, I)是拟阵.

独立任务集A：如果存在关于A中任务的一个调度，使得没有一个任务是迟的.

问题：如何找到S 的最大独立任务子集？

早任务优先形式

$$a_{i1} \quad a_{i2} \quad \cdots \quad a_{ik} \quad a_{i,\,k+1} \quad \cdots \quad a_{ip} \quad \cdots \quad a_{in}$$

- 最小化迟任务的惩罚之和 等价于
  最大化早任务的惩罚之和

■ 最小化迟任务的惩罚之和等价于最大化早任务的惩罚之和

加权拟阵最优子集的贪心算法框架如下：

输入：具有正权函数 w 的加权拟阵 $M = (S, I)$

输出：M的最优子集A.

```
GREEDY(M, w)
A = Ø
sort M.S into monotonically decreasing order by weight w
for each x ∈ M.S, taken in monotonically decreasing order by weight w(x)
    if A ∪ {x} ∈ M.I
        A = A ∪ {x}
return A
```

$O(n\log n)$

w：误时惩罚

$N_t(A\cup\{x\})\le t$

$O(n)$

时间复杂度：
$O(n^2)$

- 计算时间复杂性是$O(n\log n+nf(n))$，其中$f(n)$是用于检测任务子集A的独立性所需的时间
- 确定A后，按照截止时间单调递增顺序列出A中所有元素，然后按任意顺序列出迟的任务（即S-A）

例：

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

(1) $A=\{1\}$：

$N_i(A)=0\le i$, $i=1, 2, 3$;

$N_j(A)=1\le i$, $j=4, 5, 6, 7$.

(2) $A=\{1, 2\}$：

$N_1(A)=0\le 1$;

$N_i(A)=1\le i$, $i=2, 3$;

$N_j(A)=2\le j$, $j=4, 5, 6, 7$.

(3) $A=\{1,2,3\}$：

$N_1(A)=0\le 1$;

$N_i(A)=1\le i$, $i=2, 3$;

$N_j(\mathrm{A})=3\le j$, $j=4, 5, 6, 7$.

例：

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

(4) $A=\{1, 2, 3, 4\}$：

$N_1(A)=0\leq 1$

$N_2(A)=1\leq 2$

$N_3(\mathrm{A})=2\leq 3$

$N_j(A)=4\leq j, j=4,5,6,7.$

(5) $A=\{1,2,3,4,5\}$：

$N_1(A)=1\leq 1$

$N_2(A)=2\leq 2$

$N_3(A)=3\leq 3$

$N_4(A)= 5 >4.$

(6) $A=\{1,2,3,4,6\}$：

$N_1(A)=0\leq 1$

$N_2(A)=1\leq 2$

$N_3(A)=2\leq 3$

$N_4(A)=5>4.$

(7) $A=\{1,2,3,4,7\}$：

$N_1(A)=0\leq 1$

$N_2(A)=1\leq 2$

$N_3(A)=2\leq 3$

$N_i(A)=4\leq j, j=4,5$

$N_j(A)=5, j=6, 7.$

得$A=\{1, 2, 3, 4, 7\}$，
最优调度为：
2, 4, 1, 3, 7, 5, 6
（早任务优先形式）

总结

- 贪心算法举例
  - 区间调度问题（活动选择问题）
  - 区间划分问题
  - 最小延迟调度问题
- 贪心算法的理论基础——拟阵
  - 什么是拟阵（**Matroids**）？
  - 拟阵的通用贪心解法
  - 拟阵通用贪心解法的正确性
  - 基于拟阵的贪心算法举例