算法分析与设计

Analysis and Design of Algorithm

Lesson 10

要点回顾

- 动态规划算法的设计要点
  - 建模：目标函数、约束条件
  - 分段：确定子问题的边界
  - 分析：子问题之间的依赖关系
  - 判断：最优子结构性质
  - 求解：先定最小子问题（初值），自底向上求解
- 组合问题的动态规划算法
  - 最长公共子序列
  - 背包问题

背包问题(Knapsack Problem)

- 一个旅行者随身携带一个背包。可以放入背包的物品有n种，每种物品的重量和价值分别为w_i，v_i。如果背包的最大承重限制是b，每种物品可以放多个。怎么样选择放入背包的物品使得背包所装物品**价值最大**?
- 假设上述 w_i v_i b 都是正整数。

建模

- 解是$\langle x_1, x_2, \ldots, x_n \rangle$，其中$x_i$是装入背包的第$i$种物品个数

目标函数 $\max \sum_{i=1}^{n} v_i x_i$

约束条件 $\sum_{i=1}^{n} w_i x_i \le b, \quad x_i \in N$

子问题界定和计算顺序

- 子问题界定：由参数k和y界定
  - k：考虑物品1, 2, ..., k的选择
  - y：背包总重量不超过y
- 原始输入：$k=n$, $y=b$
- 子问题计算顺序：
  - $k=1, 2, ..., n$
  - 对于给定的k，$y=1, 2, ..., b$

优化函数的递推方程

$F_k(y)$: 装前 k 种物品，总重不超过 y
背包达到的**最大价值**

$$F_k(y)=\max\{F_{k-1}(y),\ F_k(y-w_k)+v_k\}$$

$$F_0(y)=0,\ 0\leqslant y\leqslant b,\ F_k(0)=0,\ 0\leqslant k\leqslant n$$

$$F_1(y)=\left\lfloor \frac{y}{w_1} \right\rfloor v_1, \qquad F_k(y)=-\infty \quad y<0$$

标记函数

$i_k(y)$: 装前k种物品，总重不超过y，背包达到最大价值时装入物品的**最大标号**

$$
i_k(y) = \begin{cases} i_{k-1}(y) & F_{k-1}(y) > F_k(y - w_k) + v_k \\ k & F_{k-1}(y) \le F_k(y - w_k) + v_k \end{cases}
$$

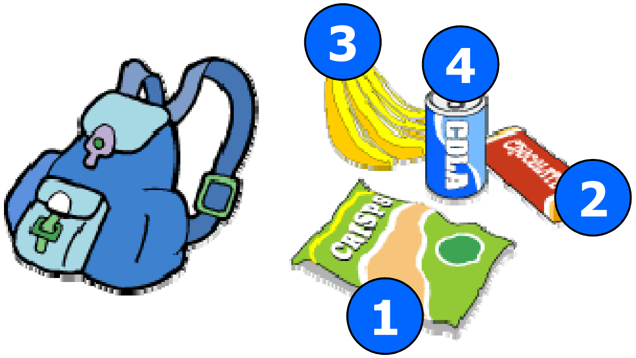
$$
i_1(y) = \begin{cases} 0 & y < w_1 \\ 1 & y \ge w_1 \end{cases}
$$

动态规划求解背包问题

输入： $n=4, b=10$

$v_1=1, v_2=3, v_3=5, v_4=9$

$w_1=2, w_2=3, w_3=4, w_4=7$



$F_k(y)$的计算表如下：

k \ y	1	2	3	4	5	6	7	8	9	10
1	0	1	1	2	2	3	3	4	4	5
2	0	1	3	3	4	6	6	7	9	9
3	0	1	3	5	5	6	8	10	10	11
4	0	1	3	5	5	6	9	10	10	12

输入： $n=4,\ b=10$

$v_1=1,\ v_2=3,\ v_3=5,\ v_4=9$

$w_1=2,\ w_2=3,\ w_3=4,\ w_4=7$

追踪解

$i_k(y)=$

k \ y	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	1	1
2	0	1	2	2	2	2	2	2	2	2
3	0	1	2	3	3	3	3	3	3	3
4	0	1	2	3	3	3	4	3	4	4

$i_4(10)=4 \rightarrow x_4 \geqslant 1$

$i_4(10-w_4)=i_4(3)=2 \rightarrow x_2 \geqslant 1,\quad x_4=1,\quad x_3=0$

$i_2(3-w_2)=i_2(0)=0 \rightarrow x_2=1,\quad x_1=0$

解： $x_1=0,\ x_2=1,\ x_3=0,\ x_4=1$， 价值12

追踪算法

- **算法 TrackSolution**
- **输入：** $i_k(y)$表，k=1, 2, ..., n，y=1, 2, ..., b
- **输出：** $x_1,x_2,...,x_n$，n种物品的装入量

1. **for** $j \leftarrow 1$ **to** n **do** $x_j \leftarrow 0$
2. $y \leftarrow b,\ j \leftarrow n$
3. $j \leftarrow i_j(y)$
4. $x_j \leftarrow 1$
5. $y \leftarrow y - w_j$
6. **while** $i_j(y)=j$ **do**
7. $\quad y \leftarrow y - w_j$
8. $\quad x_j \leftarrow x_j+1$
9. **if** $i_j(y) \neq 0$ **then goto** 3.

初始追踪位置

继续放j种物品

继续追踪下一种

时间复杂度

根据公式

$$F_k(y)=\max\{F_{k-1}(y), F_k(y-w_k)+v_k\}$$

备忘录需计算nb项，每项常数时间，计算时间为$O(nb)$

伪多项式时间算法：时间为参数b和n的多项式，不是输入规模的多项式。参数b是整数，表达b需要$\log b$位，输入规模以n和$\log b$为参数。

背包问题的推广

- 物品数受限背包：第i种物品最多用n_i个
- 0-1背包问题：x_i=0,1，i=1, 2, ..., n
- 多背包问题：m个背包，背包j装入最大重量B_j, j=1, 2, ..., m。在满足所有背包重量约束条件下使装入物品价值最大。
- 二维背包问题：每件物品有重量w_i和体积t_i，i=1, 2, ..., n，背包总重不超过b，体积不超过V，如何选择物品以得到最大价值。

第三章小结

- 理解动态规划算法的概念
- 掌握动态规划算法的基本要素
  - 最优子结构性质
  - 重叠子问题性质
- 掌握设计动态规划算法的步骤
  - 建模→分段→分析→判断→求解
- 动态规划法应用实例
  - 最短路径、矩阵连乘
  - 最长公共子序列、背包问题

第四章 贪心算法

学习要点

- 理解贪心算法的概念，掌握贪心算法的基本要素
  - 最优子结构性质
  - 贪心选择性质
- 理解贪心算法一般理论，与动态规划算法的差异
- 通过应用范例学习贪心设计策略
  1. 背包问题
  2. 活动安排问题
  3. 最优装载问题
  4. 哈夫曼编码
  5. 单源最短路径
  6. 最小生成树

贪心算法概述

贪心算法(Greedy Algorithm)

- 顾名思义，贪心算法总是作出在当前阶段看来最好的选择。也就是说贪心算法并不从整体最优考虑，它所作出的选择只是在某种意义上的局部最优选择，是一种“短视”行为。
  - 希望贪心算法得到的最终结果也是整体最优的。
- 不能对所有问题都得到整体最优解，但对许多问题能产生整体最优解。
  - 如单源最短路径问题，最小生成树问题等。
  - 在一些情况下，即使不能得到整体最优解，其最终结果却是最优解的很好近似。

贪心算法思想

- 从一个实例开始：**找零钱问题**

  设有5、2、1元和5、2、1角货币，需找给顾客4元6角现金，为使付出的货币数目最少。

1. 首先选出1张最大面值不超过4元6角→2元；
2. 再选2元6角→ 2元；
3. 再选不超过6角的最大面值→ 5角；
4. 再选一张不超过1角的→ 1角；

共付出4张货币。

找零钱问题的策略

- 在每一步贪心选择中，在不超过应找零钱的条件下，只选择面值最大的货币，不考虑选择的合理性，且不会改变决定：一旦选出了一张货币，就永远选定。
  - 策略：尽可能使付出的货币最快地满足支付要求，目的是使付出的货币张数最慢增加。
  - 上述实例用贪心法可以得到最优解。
  - 但是，将面值改为3元、1元、8角、5角、1角
  - 贪心法：结果是3元、1元、5角和1角各一张，共4张。
  - 最优解：却为1个3元+2个8角，共3张。

贪心法求解的问题的特征

- 最优子结构性质
  - 当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质，也称此问题满足最优性原理/优化原则。
- 贪心选择性质
  - 所谓贪心选择性质是指问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来得到。

动态规划实质：

  - 以自底向上的方式求解各个子问题

贪心算法实质：

  - 以自顶向下的方式做出一系列的贪心选择

组合问题中的贪心算法

背包问题

- 一般背包问题的解是$\langle x_1,x_2,\ldots,x_n\rangle$，其中$x_i$是装入背包的第$i$种物品一部分，不一定要全部装入背包

  目标函数 $$\max\sum_{i=1}^{n} v_i x_i$$

  约束条件 $$\sum_{i=1}^{n} w_i x_i \le b, \quad x_i \in [0,1]$$

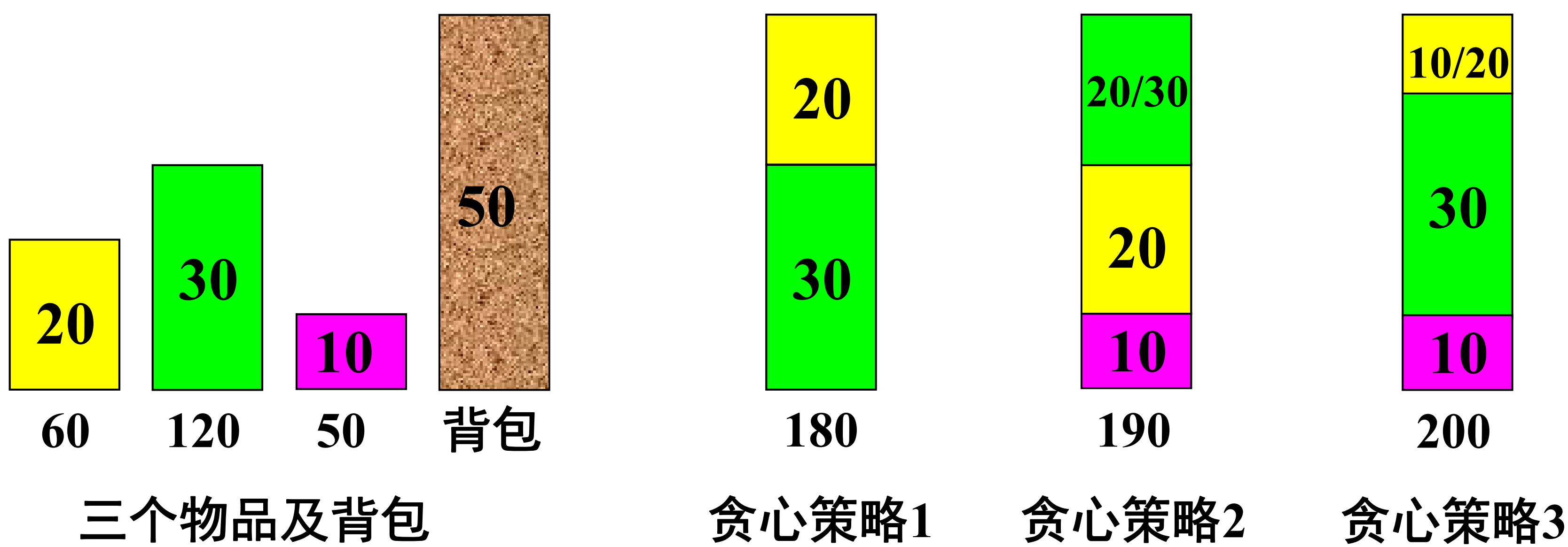
- 当 $x_i \in \{0,1\}$ 时，变为0-1背包问题的解
- 背包问题具有最优子结构性质!

背包问题的贪心法

- 至少有三种看似合理的贪心策略：
  1. 选择价值最大的物品，因为这可以尽可能快地增加背包的总价值。然而，虽然每一步选择获得了背包价值的极大增长，但背包容量却可能消耗得太快，使得装入背包的物品个数减少，从而不能保证目标函数达到最大。
  2. 选择重量最轻的物品，因为这可以装入尽可能多的物品，从而增加背包的总价值。但是，虽然每一步选择使背包的容量消耗得慢了，但背包的价值却没能保证迅速增长，从而不能保证目标函数达到最大。
  3. 选择单位重量价值最大的物品，在背包价值增长和背包容量消耗两者之间寻找平衡。

背包问题的贪心法

- 一般背包问题实例：
  - 有3个物品，其重量分别是{20, 30, 10}，价值分别为{60, 120, 50}，背包的容量为50。应用三种贪心策略装入背包的物品和获得的价值如图所示。

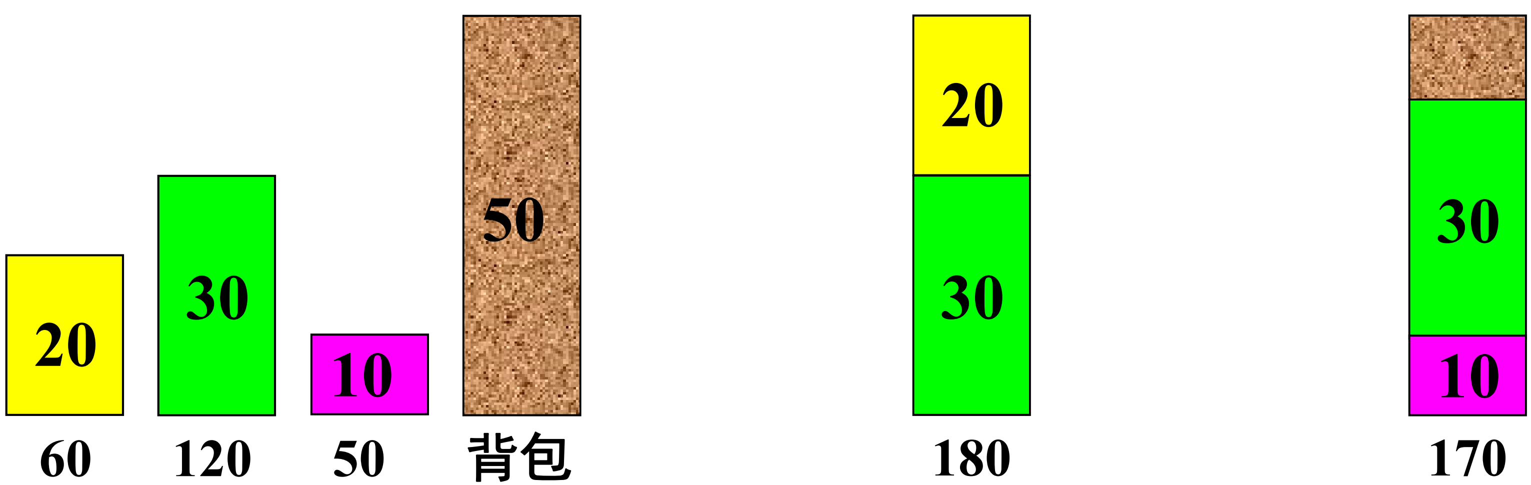


背包问题的贪心法

- 应用第3种贪心策略，每次从物品集合中选择性价比最高的物品，如果其重量小于背包容量，就可以把它装入，并将背包容量减去该物品的重量。
- 然后就面临一个最优子问题——同样是背包问题，只不过背包容量减少了，物品集合减少了。
- 因此背包问题具有最优子结构性质。

背包问题的贪心法

- **0-1背包问题实例：**
  - 有3个物品，其重量分别是{20, 30, 10}，价值分别为{60，120，50}，背包的容量为50。采用第3种贪心策略装入背包的物品和获得的价值如图所示。



三个物品及背包

最优解

贪心策略3

背包问题的贪心法

- **结论：**
  - 对于0-1背包问题，贪心选择之所以不能得到最优解，是因为在这种情况下，无法保证最终能将背包装满，部分闲置的背包空间使每千克背包空间的价值降低了。
  - 事实上，在考虑0-1背包问题时，应比较选择该物品和不选择该物品所导致的最终方案，然后再做出最好选择。
  - 由此可导出许多互相重叠的子问题。这正是该问题可以用动态规划算法求解的另一重要特征。动态规划算法可以有效解决0-1背包问题。

活动选择问题

活动选择问题

问题实例：有11个活动，每个活动的时间表如下

极坐标话剧团（10:00-22:00）
唐仲英爱心社（16:00-21:00）
机器人俱乐部（16:00-20:00）
创行（14:00-18:00）
电子竞技协会（20:00-23:00）
魔方协会（11:00-17:00）
东南大学电视台（08:00-15:00）
辩论协会（09:00-13:00）
大学生心理健康协会（11:00-14:00）
跆拳道协会（14:00-19:00）
华风汉韵文化社(13:00-16:00）

活动选择问题

- **输入：** S={1,2,...,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 ⟺ $s_i \geqslant f_j$或$s_j \geqslant f_i$

- **输出：** 最大的两两相容的活动集A
- **实例：**

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

解： A={1,4,8}

贪心算法

挑选过程是多步判断过程，每步依据某种“短视”的策略进行活动的选择，选择时候需要满足相容性条件。

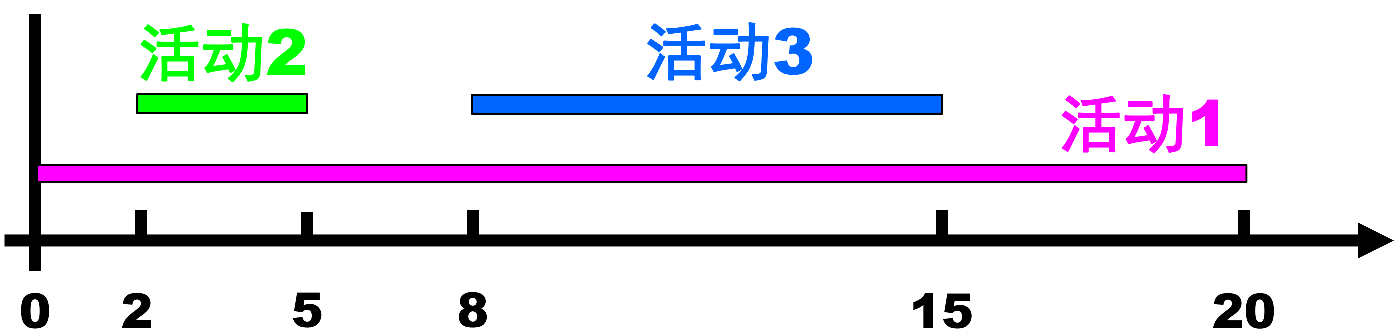
- 策略1：开始时间早的优先
  - 排序使 $s_1 \leqslant s_2 \leqslant ... \leqslant s_n$，从前往后挑选
- 策略2：占用时间少的优先
  - 排序使 $f_1 - s_1 \leqslant f_2 - s_2 \leqslant ... \leqslant f_n - s_n$，从前往后挑选
- 策略3：结束时间早的优先
  - 排序使 $f_1 \leqslant f_2 \leqslant ... \leqslant f_n$，从前往后挑选

策略1的反例

- 策略1：开始早的优先

- 反例：S={1,2,3}

  $s_1=0, f_1=20, s_2=2, f_2=5, s_3=8, f_3=15$

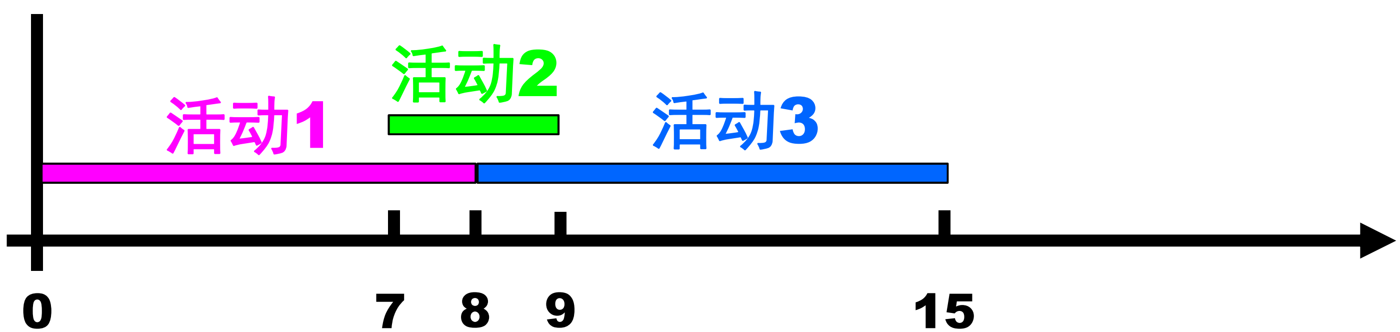


策略2的反例

- 策略2：占时少的优先

- 反例：S={1,2,3}

  $s_1=0, f_1=8, s_2=7, f_2=9, s_3=8, f_3=15$



策略3伪码

- 算法GreedySelect
- 输入：活动集S, $s_i, f_i, i=1,2,\dots,n$, $f_1 \leqslant f_2 \leqslant \dots \leqslant f_n$
- 输出：A⊆S，选中的活动子集

```
n ← length[S]
A ←{1}
j ← 1
for i ← 2 to n do
    if s_i ≥ f_j
    then A ← A∪{ i }
         j ← i
return A
```

已选入的
最后标号

判断
相容性

完成时间 $t = \max \{ f_k : k \in A \}$

运行实例

- 输入：S={1,2, ..., 10}

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

解：A={1,4,8}，完成时间 $t = 11$

时间复杂度：

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

问题：如何证明该算法对所有实例都得到正确的解？

贪心算法的特点

- 设计要素：
  - 贪心法适用于组合优化问题
  - 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
  - 依据某种“短视”的贪心选择性质判断，性质好坏决定算法的成败
  - 贪心法必须进行正确性证明
  - 证明贪心法不正确的技巧：反证法

贪心优势：算法简单，时空复杂度低

贪心法正确性证明

- 数学归纳法
  - **例：** 证明对于任何自然数n，
    $1+2+...+n=n(n+1)/2$
  - **证：** $n=1$，左边$=1$，右边$=1\times(1+1)/2=1$
    假设对于任意自然数n等式成立，则

$$
\begin{aligned}
&1+2+...+(n+1)\\
&= (1+2+...+n)+(n+1)\\
&= n(n+1)/2+(n+1)\\
&= (n+1)(n/2+1)\\
&= (n+1)(n+2)/2\\
&= (n+1)((n+1)+1)/2
\end{aligned}
$$

代入
归纳假设

第一数学归纳法

适合证明涉及自然数的**命题**P(n)

- **归纳基础：** 证明P(1)为真（或P(0)为真）
- **归纳步骤：** 若对所有n有P(n)为真，证明P(n+1)为真

$$\forall n,\ P(n) \rightarrow P(n+1)$$

$$P(1)$$

$$n=1,\ P(1) \Rightarrow P(2)$$

$$n=2,\ P(2) \Rightarrow P(3)$$

$$n=3,\ P(3) \Rightarrow P(4)$$

……

第二数学归纳法

适合证明涉及自然数的**命题**P(n)

- **归纳基础：** 证明P(1)为真（或P(0)为真）
- **归纳步骤：** 若对所有小于n的k有P(k)为真，证明P(n)为真

$$\forall k\ (k<n,\ P(n)) \rightarrow P(n)$$

$$P(1)$$

$n=2$, $P(1) \Rightarrow P(2)$

$n=3$, $P(1) \land P(2) \Rightarrow P(3)$

$n=4$, $P(1) \land P(2) \land P(3) \Rightarrow P(4)$

……

两种归纳法的区别

- 归纳基础一样　　都是P(1/0)为真
- 归纳步骤不同

证明逻辑

- 归纳法1：$P(1) \Rightarrow P(2) \Rightarrow P(3) \Rightarrow \ldots$
- 归纳法2：

$$\left.\begin{matrix} \left.\begin{matrix} P(1) \\ P(1) \Rightarrow P(2) \end{matrix}\right\} \Rightarrow P(3) \end{matrix}\right.$$

$$\left.\begin{matrix} P(1) \\ P(2) \\ P(3) \end{matrix}\right\} \Rightarrow P(4) \Rightarrow \ldots$$

贪心算法正确性归纳证明

- 证明步骤
  1. 叙述一个有关自然数n的命题，该命题断定该贪心策略的执行最终将导致最优解。其中自然数n可以代表步数或问题规模
  2. 证明命题对所有的自然数为真
     - 2.1 归纳基础（从最小实例规模开始）
     - 2.2 归纳步骤（第一或第二数学归纳法）

活动选择贪心策略3伪码

- 算法GreedySelect
- 输入：活动集S, s_i, f_i, i =1,2,...,n, $f_1 \leqslant f_2 \leqslant ... \leqslant f_n$
- 输出：A⊆S，选中的活动子集

```
n ← length[S]
A ←{1}
j ← 1
for i ← 2 to n do
    if s_i ≥ f_j
    then A ← A∪{ i }
        j ← i
return A
```

已选入的最后标号

判断相容性

完成时间 $t = \max\{f_k : k \in A\}$

活动选择算法的命题

1. 命题：

算法GreedySelect执行到第 k 步，选择 k 项活动

$$i_1=1, i_2, \ldots, i_k$$

则存在最优解 A 包含活动 $i_1=1, i_2, \ldots, i_k$。

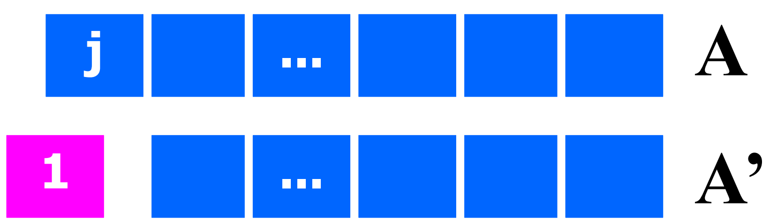
根据上述命题：对于任何 k，算法前 k 步的选择都将导致最优解，至多到第 n 步将得到问题实例的最优解。

归纳法证明

令S={1,2,...,n}是活动集，且$f_1 \leqslant f_2 \leqslant ... \leqslant f_n$

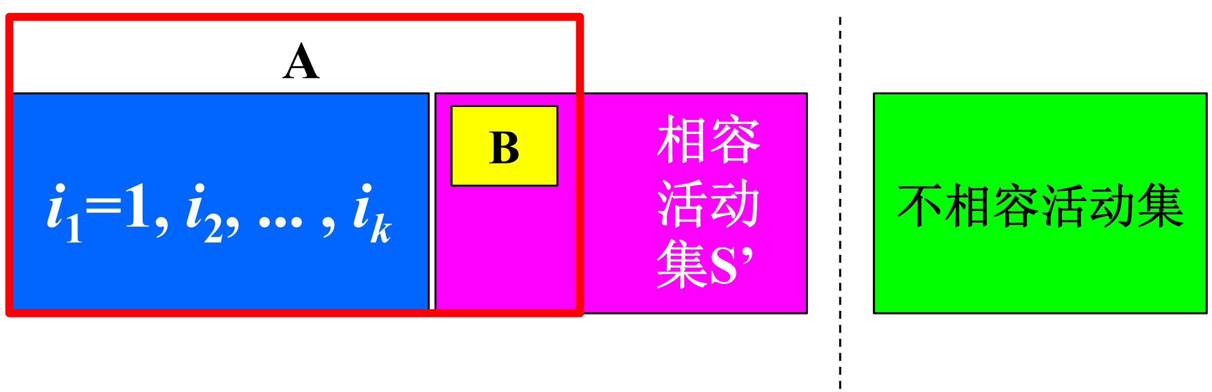
- 2.1 归纳基础：k=1, 证明存在最优解包含活动1
- 证明：
  - 任取最优解A，A中活动按截止时间递增排序。如果A的第一个活动为 j，且 $j \neq 1$，用1替换A的活动 j 得到解A’，即A’=(A-{ j })∪{1}
  - 由于$f_1 \leqslant f_j$，A’也是最优解，且含有1。

$f_1 \leqslant f_j$



归纳法证明

- **2.2 归纳步骤：假设命题对*k*为真，证明对*k*+1也为真**
- **证明：**
  - **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 $i_1=1, i_2, \ldots, i_k$，且A中剩下活动B选自集合S'**
  - **S'={ $i \mid i \in S, s_i \geqslant f_k$}，A={$i_1=1, i_2, \ldots, i_k$}∪B**

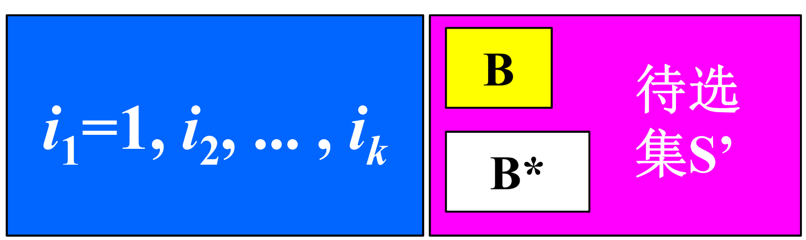


归纳步骤(cont.)

- **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

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

是S的最优解，且比A的活动多，与A的最优性矛盾！

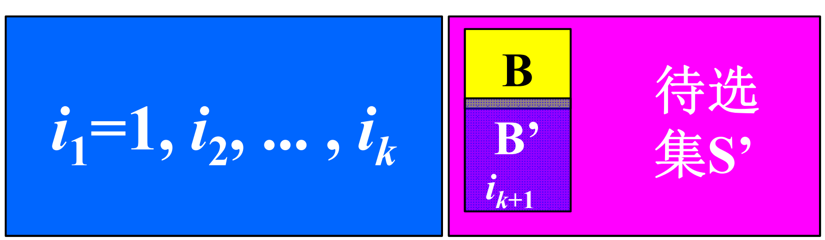


归纳步骤(cont.)

- 将S’看成子问题，根据归纳基础，**存在S’的最优解B’有S’中的第一个活动 i_{k+1}**，且|B’|=|B|，于是：

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

也是原问题的最优解。



证明方法小结

- 贪心法正确性证明方法：数学归纳法
  - 第一数学归纳法
  - 第二数学归纳法
- 活动选择问题的贪心法证明：
  - 叙述一个涉及步数的算法正确性命题
  - 证明归纳基础
  - 证明归纳步骤

最优装载问题

- 问题：有n个集装箱$1,2,\ldots,n$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，使得船上的集装箱的数目最多？不妨设每个箱子的重量$w_i \leqslant C$

最优装载问题

- 该问题是**0-1背包问题的子问题**。集装箱相当于物品，物品的重量是w_i，价值v_i都等于1，轮船载重限制C相当于背包的重量限制b。

问题建模

- 设 $\langle x_1,x_2,\ldots,x_n\rangle$ 是解向量，其中 $x_i=\{0,1\}$，$x_i=1$ 当且仅当第 i 个集装箱上船

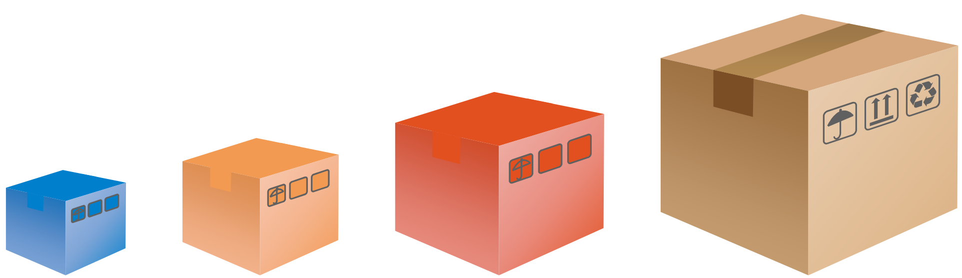
目标函数 $\max \sum_{i=1}^{n} x_i$

约束条件 $\sum_{i=1}^{n} w_i x_i \le C$

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

算法设计

- 贪心策略G：轻者优先
- 算法设计：
  - 将集装箱排序，使得 $w_1 \leqslant w_2 \leqslant ... \leqslant w_n$
  - 按照标号从小到大装箱，直到装入下一个箱子将使得集装箱总重超过轮船的装载重量限制，则停止。



正确性证明思路

- 命题：对于装载问题任何规模为n的输入实例，贪心算法G得到最优解。
- 设集装箱从轻到重记为1,2,..., n
- 归纳基础　证明对任何只含1个箱子的输入实例，贪心法得到最优解。显然正确！
- 归纳步骤　证明：假设对于任何n个箱子的输入实例贪心法都能得到最优解，那么对于任何n+1个箱子的输入实例贪心法也得到最优解。

归纳步骤证明思路

01 N={ 1, 2 , ... , n , n+1 },
$w_1 \leqslant w_2 \leqslant . . . \leqslant w_{n+1}$

02 去掉箱子1，令C’=C- w_1
得到规模为n的输入N’={2,3,...,n,n+1}

03 关于输入N’和C’的最优解I’

04 在I’加入箱子1，得到I

05 证明I是关于输入N的最优解

正确性证明

- 假设对于n个集装箱的输入，贪心法都可以得到最优解，考虑输入

$$N=\{ 1, 2 , \ldots , n , n+1 \}$$

其中$w_1 \leqslant w_2 \leqslant \ldots \leqslant w_{n+1}$

- 由归纳假设，对于

$$N'=\{2,3,\ldots,n,n+1\}, \quad C'=C- w_1$$

- 贪心法得到最优解I'。令

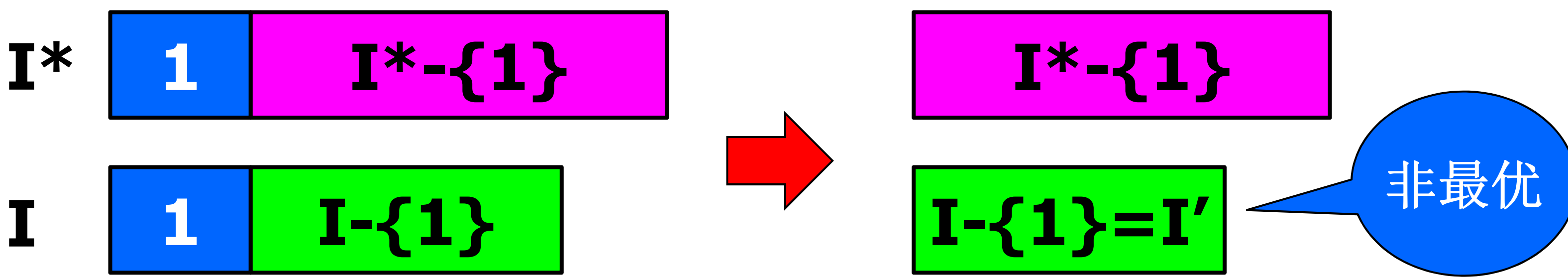
$$I = I' \cup \{1\}$$

正确性证明

- I(算法解)是关于N的最优解
  - 若不然，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’和C’的解且

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

    与I’是关于N’和C’的最优解矛盾。



小结

- 装载问题是0-1背包问题的子问题（每件物品重量为1），NP难的问题存在多项式时间可解的子问题。
- 贪心法证明：对规模进行归纳