算法设计与分析

主讲人：权丽君

Email: *ljquan@suda.edu.cn*

苏州大学 计算机学院







第9讲 贪心算法

内容提要：

- 动态规划
- 贪心算法
- 摊还分析

对于一些类型的最优化问题，使用动态规划算法略显复杂。

有没有更简单一点的算法?

贪心算法是一种选择。

例 找零钱

- 一个小孩买了价值52美分的糖，并将1美元的钱放入取款机。取款机要用数目最少的硬币将零钱找给小孩。假设取款机内有任意多的面值为25美分、10美分、5美分、及1美分的硬币。
- 贪心准则为：每次给出最大面值的硬币，同时给出的硬币面值总数不得超过要找的零钱数。

48，23，13，3，2，1

25，10，10，1，1，1

贪心算法

顾名思义，贪心算法总是作出在当前看来最好的选择。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。当然，希望贪心算法得到的最终结果也是整体最优的。虽然贪心算法不能对所有问题都得到整体最优解，但对许多问题它能产生整体最优解。在一些情况下，即使贪心算法不能得到整体最优解，其最终结果却是最优解的很好近似。

什么是贪心算法?

贪心算法是这样一种方法：分步骤实施，它在每一步仅作出当时看起来最佳的选择，即局部最优的选择，并寄希望这样的选择最终能导致全局最优解。

- 经典问题：最小生成树问题的Prim算法、Kruskal算法，单源最短路径Dijkstra算法等，以及一些近似算法。

- 贪心方法是一种有用的算法设计方法，可以很好地解决很多问题。但贪心算法不总能对所有问题能求解，只是对一些问题确实有效，可以求出最优解或近似最优解。

16.1 活动选择问题

1）问题描述

假定有一个活动的集合S含有n个活动 $\{a_1, a_2, \ldots, a_n\}$，每个活动 a_i 都有一个开始时间 s_i 和结束时间 f_i，$0 \le s_i < f_i < \infty$。同时，这些活动都要使用同一资源(如演讲会场)，而这个资源在任何时刻只能供一个活动使用。

活动的兼容性：如果选择了活动 a_i，则它在半开时间区间 $[s_i, f_i)$ 内占用资源。若两个活动 a_i 和 a_j 满足 $[s_i, f_i)$ 与区间 $[s_j, f_j)$ 不重叠，则称它们是兼容的。

- 也就是说，当 $s_i \ge f_j$ 或 $s_j \ge f_i$ 时，活动 a_i 与活动 a_j 兼容。

活动选择问题：就是对给定的包含n个活动的集合S，在已知每个活动开始时间和结束时间的条件下，从中选出最多可兼容活动的子集合，称为**最大兼容活动集合**。

不失一般性，设活动已经按照**结束时间单调递增**排序：

$$f_1 \le f_2 \le f_3 \le \cdots \le f_{n-1} \le f_n .$$

例：设有11个待安排的活动，它们的开始时间和结束时间如下，并设活动按结束时间的非减次序排列：

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	9	9	10	11	12	14	16

按结束时间的非减序排列

则 $\{a_3, a_9, a_{11}\}$ 、$\{a_1, a_4, a_8, a_{11}\}$ 、$\{a_2, a_4, a_9, a_{11}\}$ 都是兼容活动集合。

其中 $\{a_1, a_4, a_8, a_{11}\}$ 、$\{a_2, a_4, a_9, a_{11}\}$ 是最大兼容活动集合。显然最大兼容活动集合不一定是唯一的。

分析：

- 可以用动态规划方法求解。
- 贪心算法更简单一些。贪心算法将解决活动选择问题转化成一个迭代算法，可以更快地求解。

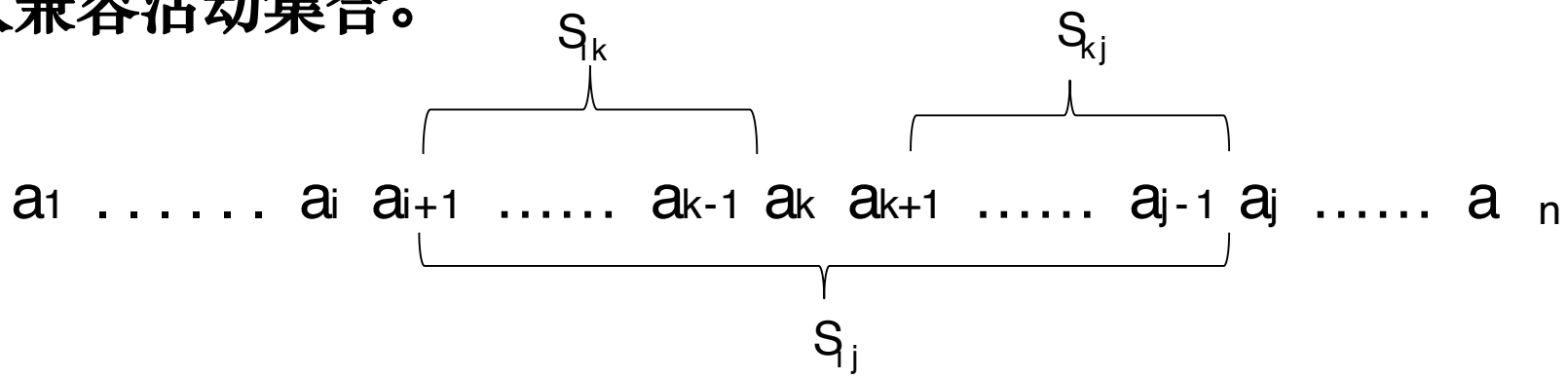
（1）活动选择问题的最优子结构

令S_{ij}表示在a_i结束之后开始且在a_j开始之前结束的那些活动的集合。

问题和子问题的形式定义如下：

设A_{ij}是S_{ij}的一个最大兼容活动集，并设A_{ij}包含活动a_k，则有：A_{ik}表示A_{ij}中a_k开始之前的活动子集， A_{kj}表示A_{ij}中a_k结束之后的活动子集。

并得到两个子问题：寻找S_{ik}的最大兼容活动集合和寻找S_{kj}的最大兼容活动集合。



活动选择问题具有最优子结构性，即：

必有：A_{ik}是S_{ik}一个最大兼容活动子集，A_{kj}是S_{kj}一个最大兼容活动子集。而$A_{ij}= A_{ik} \cup \{a_k\} \cup A_{kj}$。——最优子结构性成立。

证明：

- **用剪切-粘贴法证明最优解A_{ij}必然包含两个子问题S_{ik}和S_{kj}的最优解。**

  设S_{kj}存在一个最大兼容活动集A_{kj}'，满足$|A_{kj}'|>|A_{kj}|$，则可以将A_{kj}'作为S_{ij}最优解的一部分。

这样就构造出一个兼容活动集合，其大小

$$|A_{ik}|+|A_{kj}'|+1 > |A_{ik}|+|A_{kj}|+1 = A_{ij}$$

与A_{ij}是最优解相矛盾。 **得证。**

（1）动态规划方法

活动选择问题具有最优子结构性，所以可以用动态规划方法求解：

令c[i,j]表示集合S_{ij}的最优解大小，可得递归式如下

$$c[i,j] = c[i,k] + c[k,j] + 1 .$$

：为了选择k，有：

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset , \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset . \end{cases}$$

可以设计带备忘机制的递归算法或自底向上的填表算法求解（自学）。

2）活动选择问题的贪心算法

贪心选择：在贪心算法的每一步所做的**当前最优选择（局部最优选择）**就叫做贪心选择。

活动选择问题的贪心选择：每次总选择具有**最早结束时间**的兼容活动加入到集合A中。

为什么？

直观上，按这种方法选择兼容活动可以为未安排的活动留下尽可能多的时间。也就是说，**该算法的贪心选择的意义是使剩余的可安排时间段最大化，以便安排尽可能多的兼容活动**。

例：

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	9	9	10	11	12	14	16

结束时间递增

由于输入的活动已经按照结束时间的递增顺序排列好了，所以，首次选择的活动是a_1，其后选择的是结束时间最早且开始时间不早于前面已选择的最后一个活动的结束时间的活动（活动要兼容）。

- 当输入的活动已按结束时间的递增顺序排列，贪心算法只需O(n)的时间即可选择出来n个活动的最大兼容活动集合。
- 如果所给出的活动未按非减序排列，可以用O(nlogn)的时间重排。

再论活动选择问题的贪心选择

- f_1是最早结束时间，所以不会有活动的结束时间早于a_1，因此所有与a_1兼容的活动都是在a_1结束之后开始。

- 令$S_k=\{a_i \in S: s_i \geq f_k\}$，即在$a_k$结束之后开始的任务集合。 则在首次选择$a_1$后，$S_1$是接下来要求解的（唯一）子问题。
- 由最优子结构性得：如果a_1在最优解中（确实在），那么原问题的最优解由活动a_1及子问题S_1的最优子解构成。
- 对S_i可以继续按照相同的方式求解。

算法正确吗? 即按照上述的贪心选择方法选择的活动集是问题的最优解吗？

定理16.1 考虑任意非空子问题S_k，令a_m是S_k中结束时间最早的活动，则a_m必在S_k的某个最大兼容活动子集中。

证明：

令A_k是S_k的一个最大兼容活动子集，且a_j是A_k中结束最早的活动。若$a_j = a_m$，则得证。否则，令 $A_k' = A_k - \{a_j\} \cup \{a_m\}$。

因为A_k中的活动都是不相交的，a_j是A_k中结束时间最早的活动，而a_m是S_k中结束时间最早的活动，所以$f_m \le f_j$。即A_k'中的活动也是不相交的。

由于$|A_k'| = |A_k|$，所以A_k'也就是S_k的一个最大兼容活动子集，且包含a_m。 得证。

- 定理16.1告诉我们，选a_m不会错！
  - 从S_0开始，反复选择**结束时间最早**的活动，重复这一过程，直至不再有剩余的兼容活动。所得的子集就是最大兼容活动集合。

由于结束时间严格递增，故只需按照结束时间的单调递增顺序处理所有活动，每个活动考查且仅考查一次。

活动选择问题的贪心算法

采用自顶向下的设计：首先做出一个选择，然后求解剩下的子问题。每次选择将问题转化成一个规模更小的问题。

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
3      m = m + 1
4  if m ≤ n
5      return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

这里，数组s、f 分别表示n个活动的开始时间和结束时间。并假定n个活动已经按照结束时间单调递增排列好。对当前的k，算法返回S_k的一个最大兼容活动集。

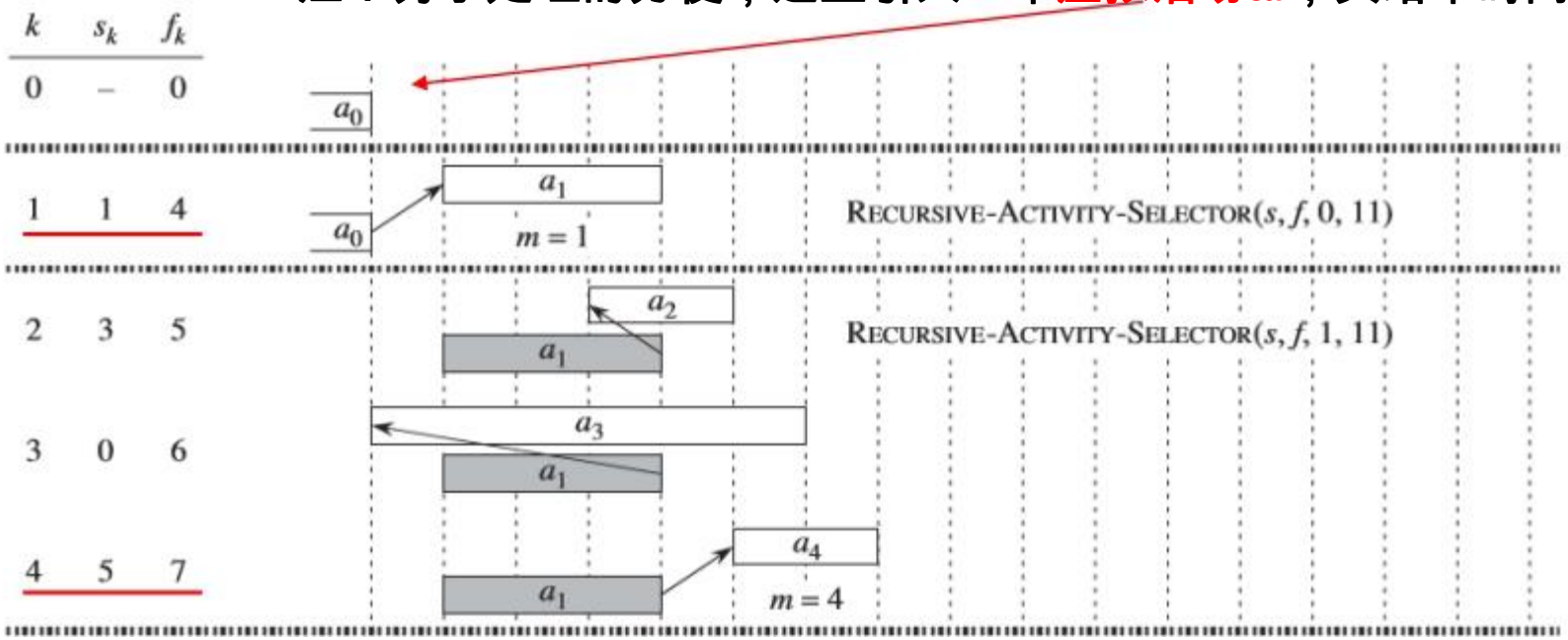
初次调用：RECURSIVE-ACTIVITY-SELECTOR(s,f,0,n)。

例：

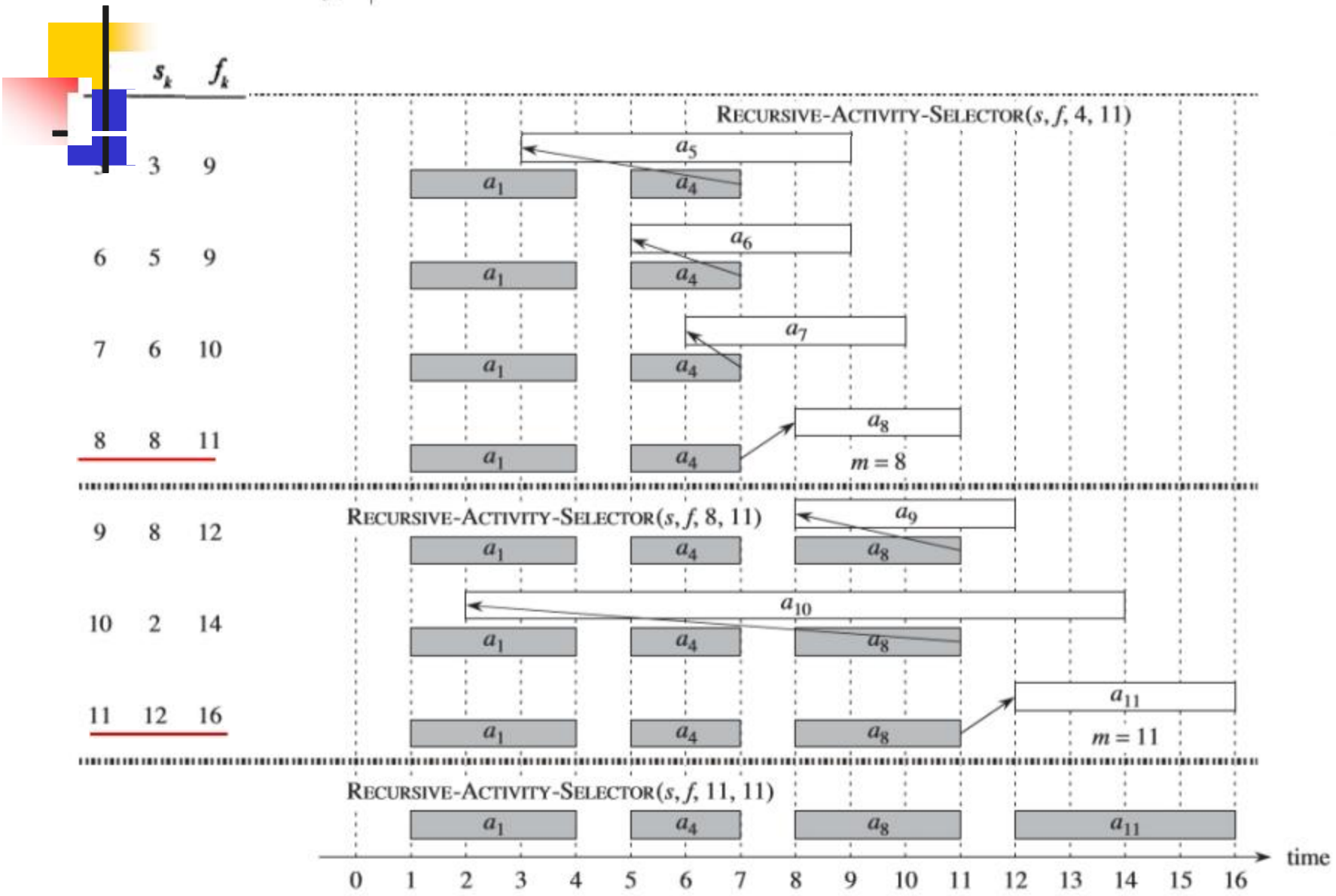
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

执行过程如图所示：

注：为了处理的方便，这里引入一个虚拟活动a_0，其结束时间$f_0=0$



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	9	9	10	11	12	14	16



迭代实现的贪心算法

注：上述的RECURSIVE-ACTIVITY-SELECTOR是一个“尾递归”过程，可以很容易地转换成迭代形式。

假定活动已经按照结束时间单调递增的顺序排列好

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```

集合A用于收集选出的活动

k对应最后一个加入A的活动，f_k是A中活动的最大结束时间，若m的开始时间大于f_k，则m就是下一个被选中的活动。

算法的运行时间是O(n)。

16.2 贪心算法原理

- 贪心算法通过做出一系列选择来求问题的最优解 —— 即贪心选择：在每个决策点，它做出在当时看来是最佳的选择。
  - 这种启发式策略并不保证总能找到最优解，但对有些问题确实有效，相比动态规划算法，贪心算法简单和直接得多。
- 贪心算法通常采用自顶向下的设计，做出一个选择，然后求解剩下的子问题。每次选择将问题转化成一个更小规模的问题。

贪心求解的一般步骤：

1）确定问题的最优子结构；

2）将最优化问题转化为这样的形式：每次对其作出选择后，只剩下一个子问题需要求解；

3）证明作出贪心选择后，剩余的子问题满足：其最优子解与前面的贪心选择组合即可得到原问题的最优解(具有最优子结构)。

注：对应每个贪心算法，都有一个动态规划算法，但动态规划算法要繁琐的多。

如何证明一个最优化问题适合用贪心算法求解?

- 贪心选择性质和最优子结构性是两个关键要素。
  - 如果能够证明问题具有这两个性质，则基本上就可以实施贪心策略。

1）贪心选择性质

贪心选择性质：可以通过做出局部最优（贪心）选择来构造全局最优解的性质。

贪心选择性使得我们进行选择时，只需做出当前看起来最优的选择，而不用考虑子问题的解。

对比动态规划方法：

- 在动态规划方法中，每个步骤也都要进行一次选择，但这种选择通常依赖于子问题的解，这导致我们要先求解较小的子问题，然后才能计算较大的子问题。

- 在贪心方法中，我们总是做出当前看来最佳的选择，然后求解剩下的唯一一个子问题。尽管贪心算法进行选择时可能依赖之前做出的选择，但不依赖任何将来的选择或子问题的解。

- 动态规划要先求解子问题才能进行第一次选择，贪心算法在进行第一次选择之前不需要求解任何子问题。

- 动态规划算法通常采用自底向上的方式完成计算，而贪心算法通常是自顶向下的，每一次选择，将给定问题实例转换成更小的问题。

- 如何证明每次贪心选择能生成全局最优解?

思路：类似动态规划的剪切-粘贴法。

通常先考查某个子问题的最优解，然后用贪心选择替换某个其它选择来修改此解，从而得到一个相似但更小的子问题，从而导出新解或矛盾。

参考定理16.1的证明。

2）最优子结构性

最优子结构性质是能否应用动态规划和贪心方法的关键要素。

- 贪心算法更为直接地使用最优子结构：

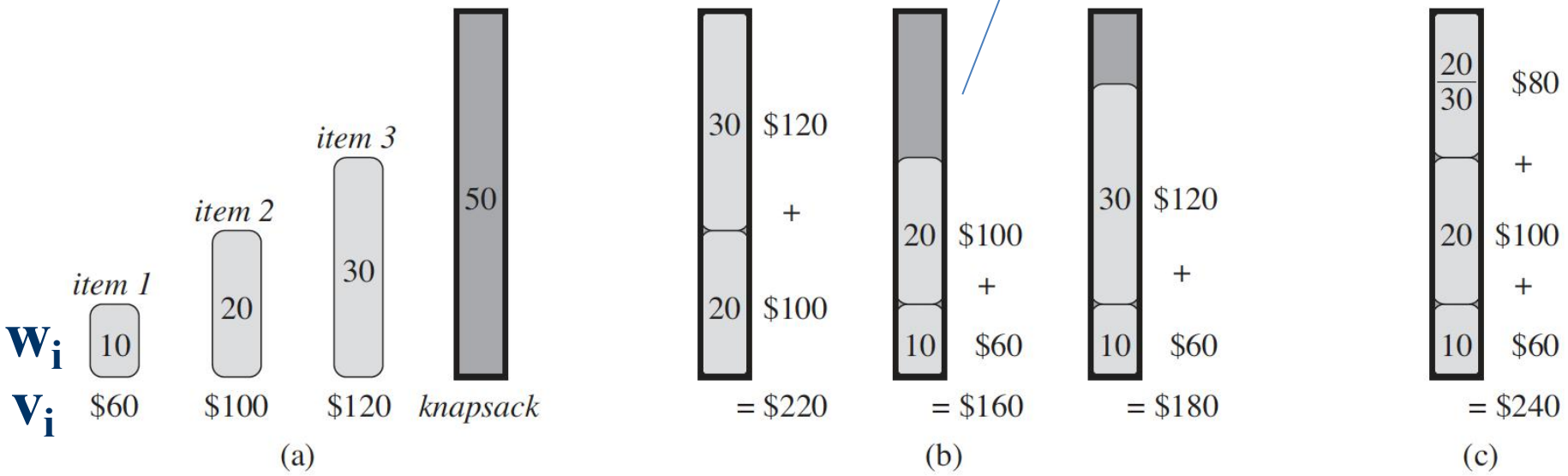
  每次贪心选择后都得到一个子问题，而原问题的最优解就是贪心选择和子问题的最优解的组合。能否成功，最优子结构性质是根本保证。

- 对比动态规划算法和贪心算法：
  - 0-1背包问题和分数背包问题：都具有最优子结构性质。
    - 0-1背包问题：动态规划算法
    - 分数背包问题：贪心算法，按p_i/w_i的降序考虑问题

两个背包问题的不同解法

- 求解0-1**背包**最优解只能用动态规划求解
  - 按每磅价值(v_i/w_i)排序
  - 贪心选择策略：取单位价值最大者装包，若装不下，考虑下一单位价值最大的物品，直至包装满或所有物品都考虑过为止
  - 实际上，装入当前每磅价值最大者只能保证当前最优(局部最优)，然而放弃它可能使得后续选择更优。所以在装包前，**应将某物品装包的子问题的解和放弃它的子问题的解进行比较**，这将导致许多重叠子问题，这正是动态规划的特点。
- **分数背包**：可用贪心算法求出最优解

贪心算法



详细讨论见P244

16.3 Huffman编码

Huffman编码问题是一个典型的贪心算法问题。

Huffman编码：最佳编码方案，通常可以节省20%~90%的空间。

实例说明：

设要压缩一个有10万个字符的数据文件，文件中出现的所有字符和它们的出现频率如下：

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

只有六个字符

分析：

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

采用二进制字符编码（简称编码），每个字符用唯一的二进制串表示，称为码字。

1）定长编码：每个字符的编码长度一样。

- 如上例，考虑到有六个字符，可以用3位码字对每个字符编码，如表中的定长编码方案。
- 10万个字符需要用30万个二进制位来对文件编码。

2）变长编码：每个字符赋予不同长度的码字。

- 思路：赋予高频字符短码字，低频字符长码字，字符的码字互不为前缀，这样才能唯一解码。
- 如表中变长编码方案：a用1位的串0表示，b用3位的串101表示，f 用4位的串1100表示等。
- 10万个字符仅需22.4万个二进制位，节约了25%的空间。

$(45+(13+12+16)\times 3+(9+5)\times 4)\times 1000=224{,}000$位

最优编码方案的设计

前缀码(Prefix code)：**任何码字都不是其它码字的前缀。**

文件编码过程：将文件中的每个字符的码字连接起来即可完成文件的编码过程。

如，设文件中包含3个字符：abc

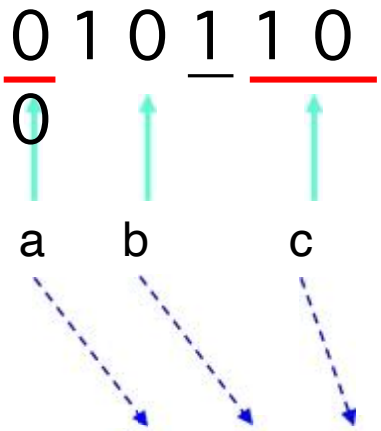
字符编码（前缀码）：

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Variable-length codeword	0	101	100	111	1101	1100

文件编码：0101100

文件解码过程：

前缀码可以简化解码过程：由于没有码字是其它码字的前缀，所以编码文件的开始部分是没有歧义的，可以唯一地转换回原字符，然后对编码文件剩余部分重复解码过程，即可“解读”出原来的文件。



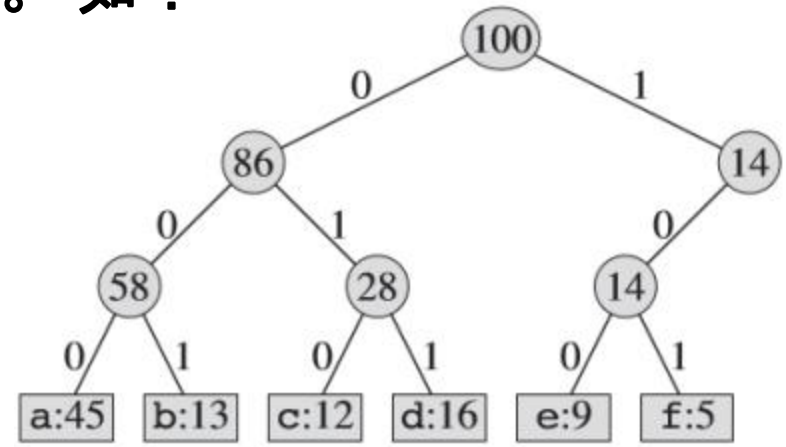
对每一个二进制子位串，在码字表里都只有一个字符唯一地与之对应

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Variable-length codeword	0	101	100	111	1101	1100

编码树：一种用于表示字符二进制编码构造的二叉树。

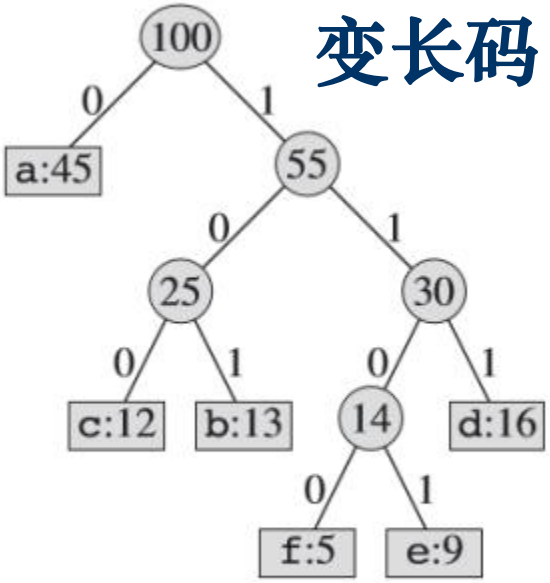
叶子结点：对应给定的字符，每个字符对应一个叶子结点。

编码构造：字符的二进制码字由根结点到该字符叶子结点的简单路径表示：0代表转向左孩子，1代表转向右孩子。如：



等长码

(a)



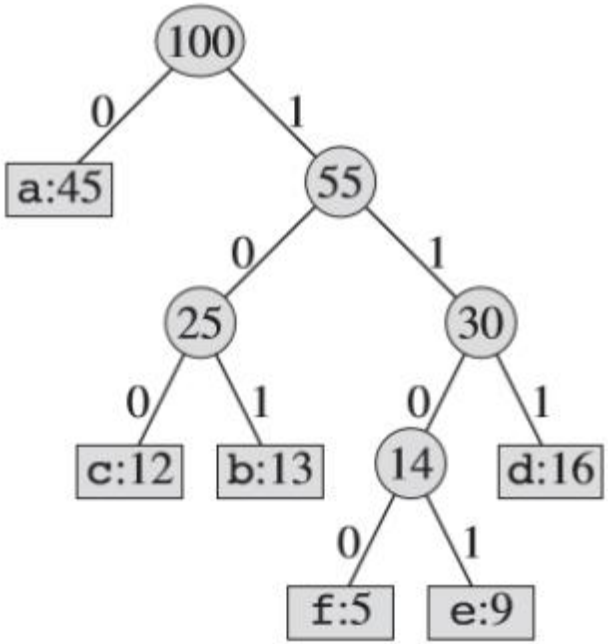
变长码

(b)

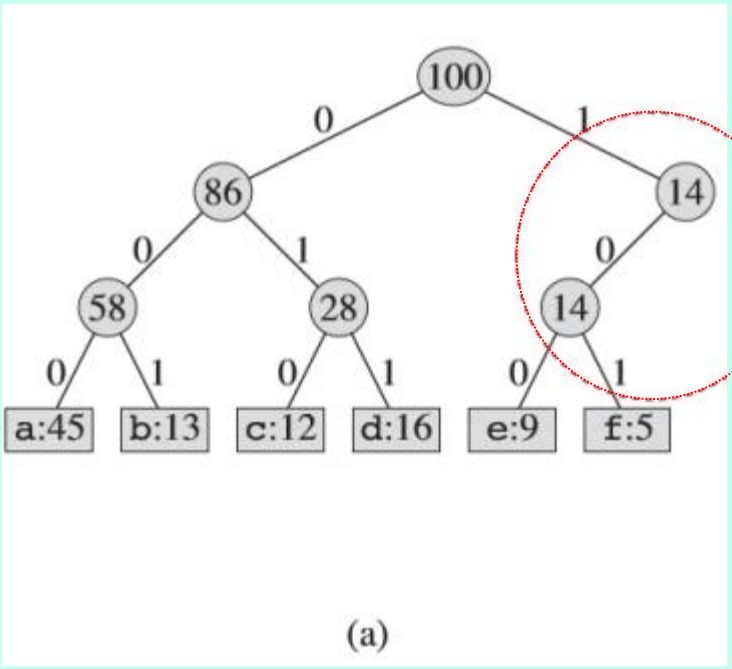
图 16-3 中编码方案的二叉树表示。每个叶结点标记了一个字符及其出现频率。每个内部结点标记了其子树中叶结点的频率之和。(a)对应定长编码 a=000，…，f=101 的二叉树。(b)对应最优前缀码 a=0，b=101，…，f=1100 的二叉树

一个文件的最优字符编码方案总对应一棵满(full)二叉树，即每个非叶子结点都有两个孩子结点。

如图(b)：



(b)



(a)

图(a)的定长编码实例不是最优的，因为它的二叉树不是满二叉树，包含以10开头的码字，但不包含以11开头的码字。

最优编码方案

文件的最优编码方案对应一棵满二叉树（full binary tree）：

- 设C为字母表
  - 对字母表C中的任意字符c，令属性c.freq表示字符c在文件中出现的频率（设所有字符的出现频率均为正数）。
  - 最优前缀码对应的树中恰好有|C|个叶子结点，每个叶子结点对应字母表中的一个字符，且恰有|C|-1个内部结点。
- 令T表示一棵前缀编码树；
- 令$d_T(c)$表示c的叶子结点在树中的深度（根到叶子结点的路径长度）。

$d_T(c)$也是字符c的码字的长度。

令B(T)表示采用编码方案T，文件的编码长度，则：

$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c) ,$$

- 即文件要用B(T)个二进制位表示。

- 称B(T)为T的**代价**。
- **最优编码**：使得B(T)最小的编码称为最优编码。
  - 对给定的字符集和文件，Huffman编码是一种最优编码。

Huffman编码的贪心算法

算法HUFFMAN从|C|个叶子结点开始，每次选择**频率最低**的两个结点**合并**，将得到的新结点加入集合继续合并，这样执行 |C|-1次“合并”后即可构造出一棵编码树——Huffman树。

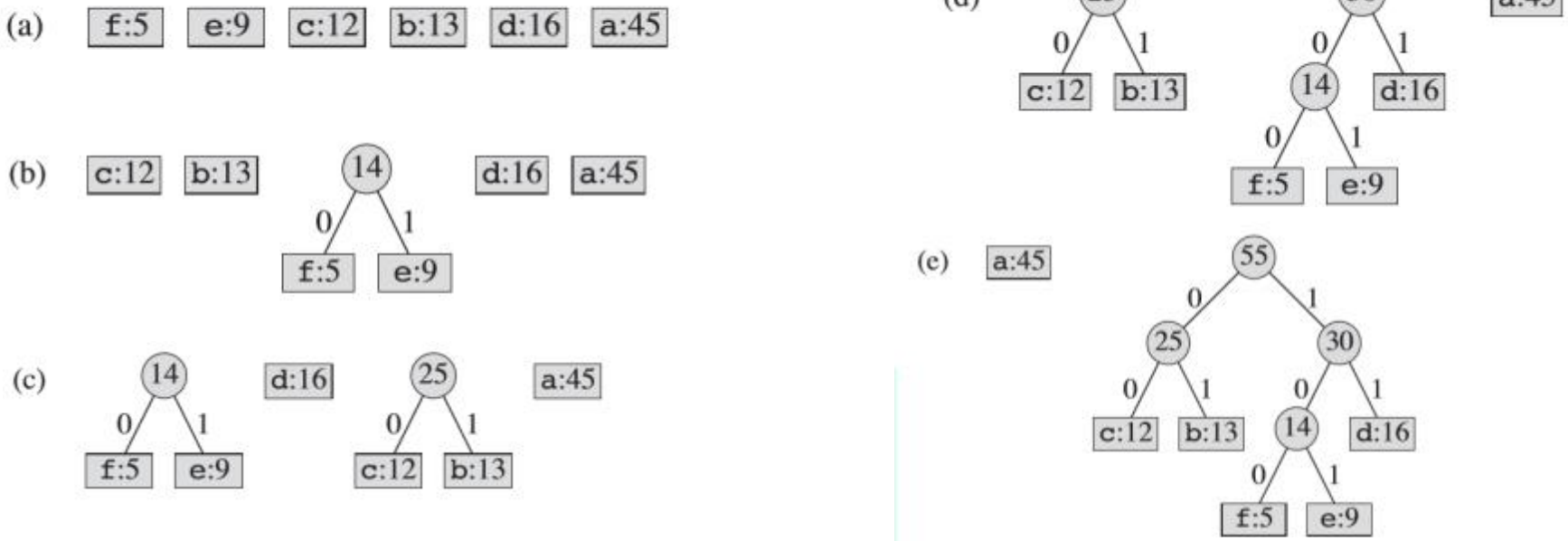
```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

采用以freq为关键字的最小优先队列Q。提取两个最低频率的对象将之合并。

合并后，新对象的频率等于原来两个对象的频率之和。

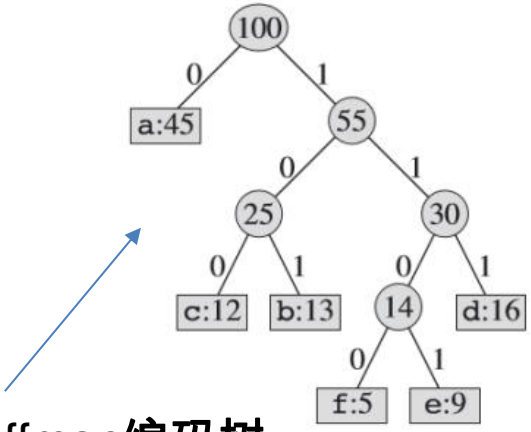
例：构造前面实例的Huffman编码

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5



- 每一步选择频率最低的两棵树进行合并。
- 叶子结点用矩形表示，每个叶子结点包含一个字符及其频率。
- 内结点用圆形结点表示，频率等于其孩子结点的频率之和。
- 内结点指向左孩子的边标记为0，指向右孩子的边标记为1。
- 一个字母的码字对应从根到其叶子结点的路径上的边的标签序列。

如 f 的码字是：1000，e的码字是1001



最后得到的前面实例的Huffman编码树

时间分析

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

假设Q使用最小二叉堆实现，则

- 首先，Q的初始化花费O(n)的时间。
- 其次，循环的总代价是O(nlgn)。
  - for循环共执行了n-1次，每次从堆中找出当前频率最小的两个结点及把合并得到的新结点插入到堆中均花费O(lgn)，所以循环的总代价是O(nlgn).

所以，HUFFMAN的总运行时间O($n$lg$n$)。

注：如果将最小二叉堆换为van Emde Boas树（ Chp 20 ），可以将运行时间减少到O(nlglgn)

HUFFMAN算法的正确性

为了证明贪心算法HUFFMAN是正确的，需要证明确定最优前缀码的问题具有贪心选择和最优子结构性质。

引理 16.2 令C为一个字母表，其中每个字符$c \in C$都有一个频率c.freq。令x和y是C中频率最低的两个字符。那么存在C的一个最优前缀码，x和y的码字长度相同，且只有最后一个二进制位不同。

证明：

令T是一个最优前缀码所对应的编码树——满二叉树。

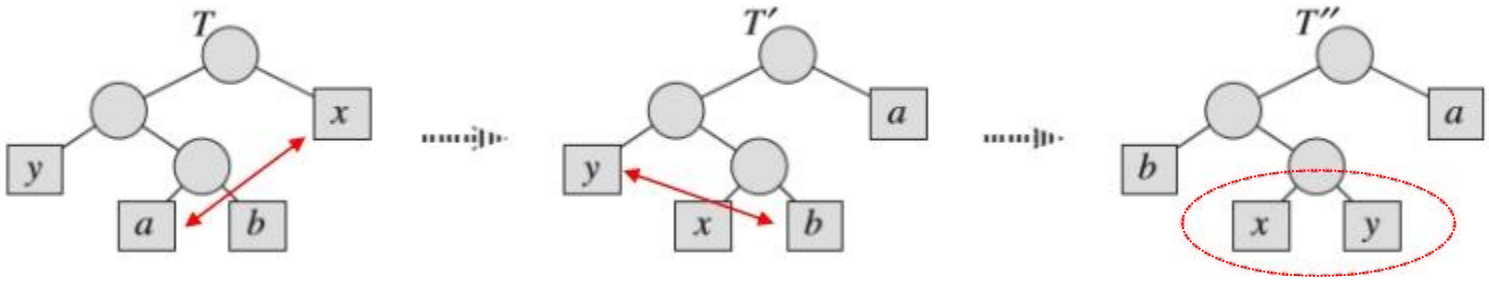
令a和b是T中深度最大的兄弟叶结点。

- 不是一般性，假设$a.freq \le b.freq$且$x.freq \le y.freq$。
- 由于x和y是叶结点中频率最低的两个结点，所以应有$x.freq \le a.freq$且$y.freq \le b.freq$。

注：有可能x.freq=a.freq 或 y.freq=b.freq

- 若x.freq=b.freq，则有a.freq=b.freq=x.freq=y.freq，此时引理显然成立。

- 假定x. freq≠b .freq，即x≠b。则在T中交换x和a，生成一棵新树T’；然后再在T’中交换b和y，生成另一棵新树T”，那 么在T”中x和y是深度最深的两个兄弟结点。如图所示：



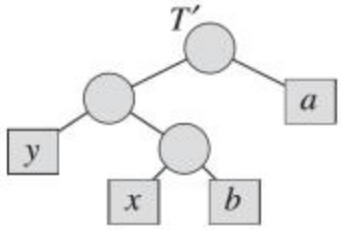
在最优树T中，叶子结点a和b是最深的叶子结点中的两个，并且是兄弟结点。叶子结点x和y为算法首 先合并的两个叶子结点，它们可出现在T中的任意位置上。假设x≠b ，叶子结点a和x交换得到树T’ ,然后交换叶子结点b和y得到树T” 。

根据文件编码的计算公式，T和T'的代价差为：

$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c),$$

$$\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} c.freq \cdot d_T(c) - \sum_{c \in C} c.freq \cdot d_{T'}(c) \\
&= x.freq \cdot d_T(x) + a.freq \cdot d_T(a) - x.freq \cdot d_{T'}(x) - a.freq \cdot d_{T'}(a) \\
&= x.freq \cdot d_T(x) + a.freq \cdot d_T(a) - x.freq \cdot d_T(a) - a.freq \cdot d_T(x) \\
&= (a.freq - x.freq)(d_T(a) - d_T(x)) \\
&\geq 0,
\end{aligned}$$

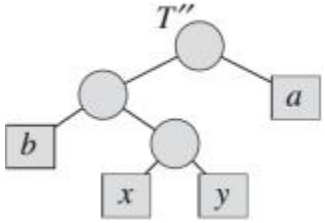
注，其中a.freq-x.freq和dT(a)-dT(x)均为非负值。



- B(T)-B(T')≥0 表示从T到T'并没有增加代价。
- 类似地，从T'到T"，交换y和b也不会增加代价，即

B(T')-B(T")≥0。

因此， B(T") ≤ B(T)。



根据假设，T是最优的，因此B(T”) = B(T)，即得证：T”也是最优解，且x和y是其中深度最大的两个兄弟结点。

得证。

引理 16.2 令C为一个字母表，其中每个字符$c \in C$都有一个频率 c.freq。令x和y是C中频率最低的两个字符。那么存在C的一个最优前缀码，x和y的码字长度相同，且只有最后一个二进制位不同。

下面不失一般性，通过合并来构造最优树。

- 贪心选择：每次选择出现频率最低的两个字符。
  - 将一次合并操作的代价视为被合并的两项的频率之和，而编码树构造的总代价等于所有合并操作的代价之和。
  - 引理16.3表明：在所有的合并操作中，HUFFMAN选择是代价最小的方案：

引理 16.3 令C为一个给定的字母表，其中每个字符$c \in C$都有一个频率c.freq。

- 令x和y是C中频率最低的两个字符。
- 令C'为C去掉字符x和y，并加入一个新字符z后得到的字母表，即$C' = C - \{x, y\} \cup \{z\}$。
  - 类似C，也为C'定义freq，且z.freq= x.freq + y.freq。
- 令T'为字母表C'的任意一个最优前缀码对应的编码树。

则有：可以将T'中叶子结点z替换为一个以x和y为孩子的内部结点，得到树T，而T表示字母表C的一个最优前缀码。

证明：

对C中不是x和y的字符c，即c∈C-{x,y}，有

$$d_T(c)=d_{T'}(c)\ ,$$

亦有：$c.freq\cdot d_T(c)=c.freq\cdot d_{T'}(c)$**。**

由于 $d_T(x)=d_T(y)=d_{T'}(z)+1$

故有：

$$\begin{aligned} & x.freq\cdot d_T(x)+y.freq\cdot d_T(y) \\ = \ & (x.freq+y.freq)(d_{T'}(z)+1) \\ = \ & z.freq\cdot d_{T'}(z)+(x.freq+y.freq) \end{aligned}$$

从而可得：$B(T)=B(T')+x.freq+y.freq$

或等价地：$B(T')=B(T)-x.freq-y.freq$

下面用反证法证明T对应的前缀码是C的最优前缀码：

假定T对应的前缀码不是C的最优前缀码。则会存在最优前缀码树T″满足：B(T″)<B(T)。

不失一般性，由引理16.2有，T″包含兄弟结点x和y。

令T‴为将T″中x、y及它们的父结点替换为叶结点z得到的树，其中z.freq=x.freq+y.freq。于是

$$\begin{aligned} B(T''') &= B(T'') - x.freq - y.freq \\ &< B(T) - x.freq - y.freq \\ &= B(T'), \end{aligned}$$

这与T′对应C′的一个最优前缀码的假设矛盾。 因此，T必然表示字母表C的一个最优前缀码。 证毕。

定理 16.4 过程HUFFMAN会生成一个最优前缀码。

证明：由引理16.2和引理16.3即可得。

贪心选择性：

- **引理16.2说明首次选择频率最低的两个字符和选择其它可能的字符一样，都可以构造相应的最优编码树。**
- **引理16.3说明首次贪心选择，选择出频率最低的两个字符x和y，合并后将z加入元素集合，可以构造包含z的最优编码 树，而还原x和y，一样还是最优编码树。**
- **所以贪心选择性成立。**

作业

- 16.1-4
- 16.2-7
- 16.3-3
- 16-1
- **求以下背包问题的最优解**：$n=7$，$M=15$，$(p_1, \ldots, p_7)=(10,5,15,7,6,18,3)$，$(w_1, \ldots, w_7)=(2,3,5,7,1,4,1)$。