



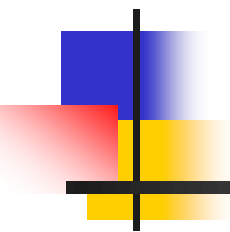
算法设计与分析

Computer Algorithm Design & Analysis
2018.10

何琨

brooklet60@hust.edu.cn

群 号 : 638562638

A decorative graphic on the left side of the slide, consisting of a black crosshair with blue, red, and yellow squares at the intersections.

Chapter 16

Greedy Algorithms

贪心算法

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 $\{a_1, a_2, \dots, a_n\}$ ，每个活动 a_i 都有一个开始时间 s_i 和结束时间 f_i ， $0 \leq s_i < f_i < \infty$ 。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 $s_i \geq f_j$ 或 $s_j \geq f_i$ 时，活动 a_i 与活动 a_j 兼容。

A decorative horizontal bar with a yellow-to-white gradient and a vertical black line is positioned above the text.

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：

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

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为一个**迭代算法**，可以更快地求解。

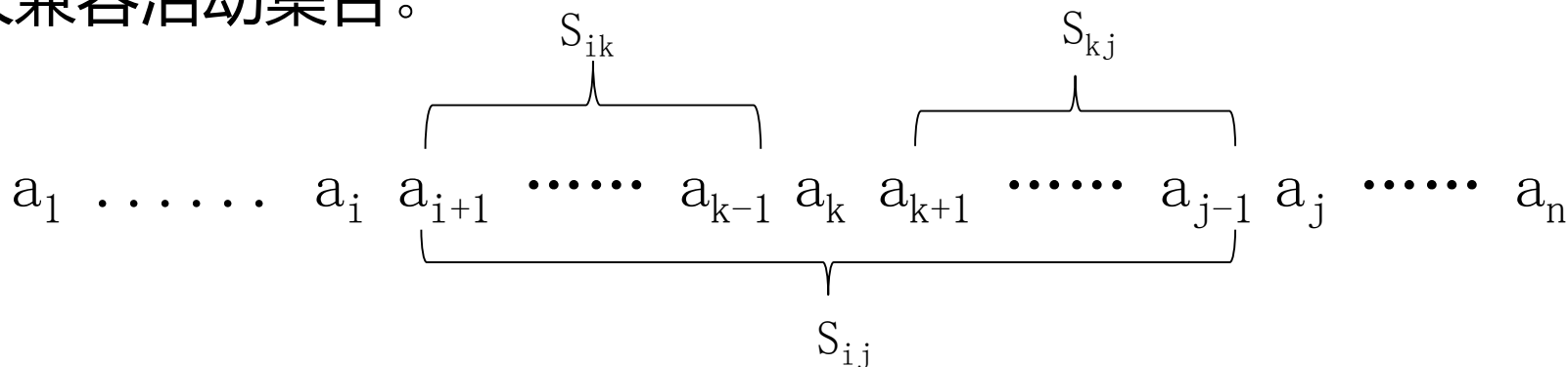
(1) 活动选择问题的最优子结构

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：

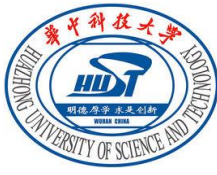
令 $c[i,j]$ 表示集合 S_{ij} 的最优解大小，可得递归式如下：

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

为了选择 k ，有：

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

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 $O(n \log n)$ 的时间重排。

算法正确吗？ 即按照上述的贪心选择方法选择的子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 $f_m \leq f_j$ 。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。

■ 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为一个规模更小的问题。

RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)

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

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。

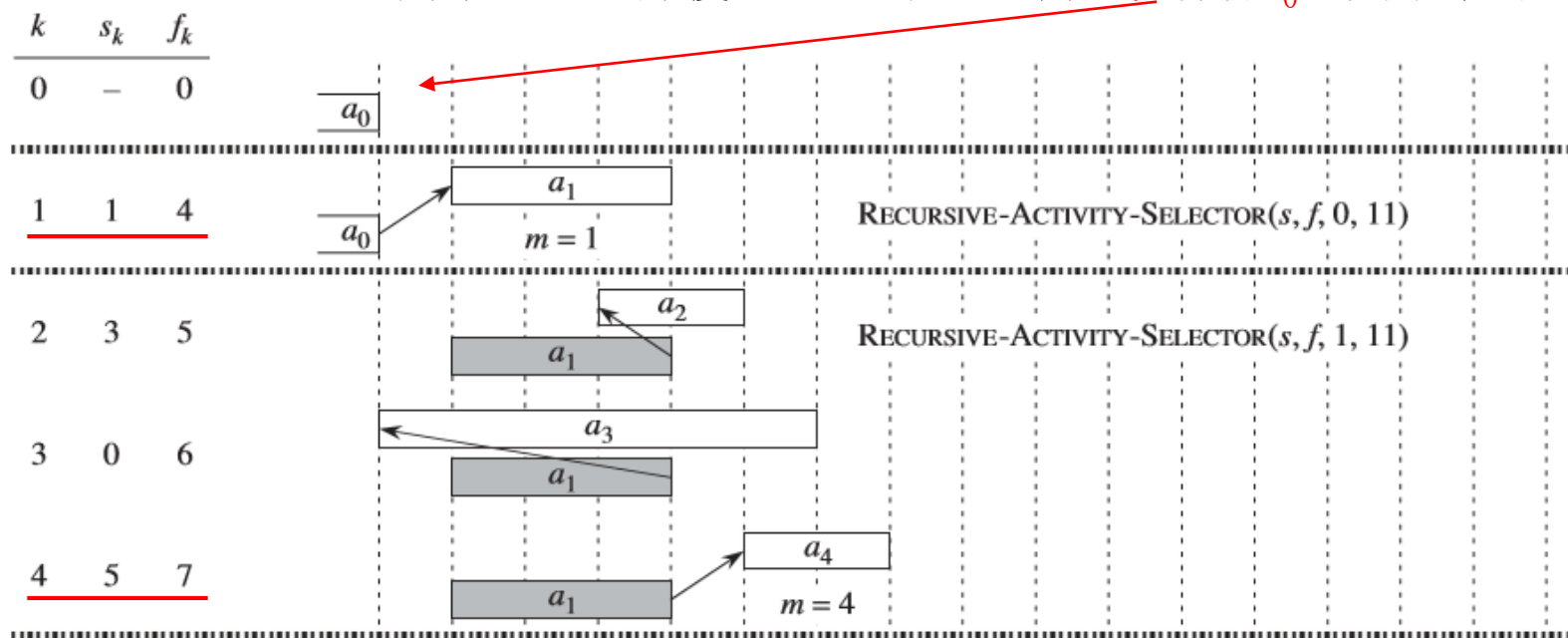
初次调用：RECURSIVE-ACTIVITY-SELECTOR($s, f, 0, n$)。

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

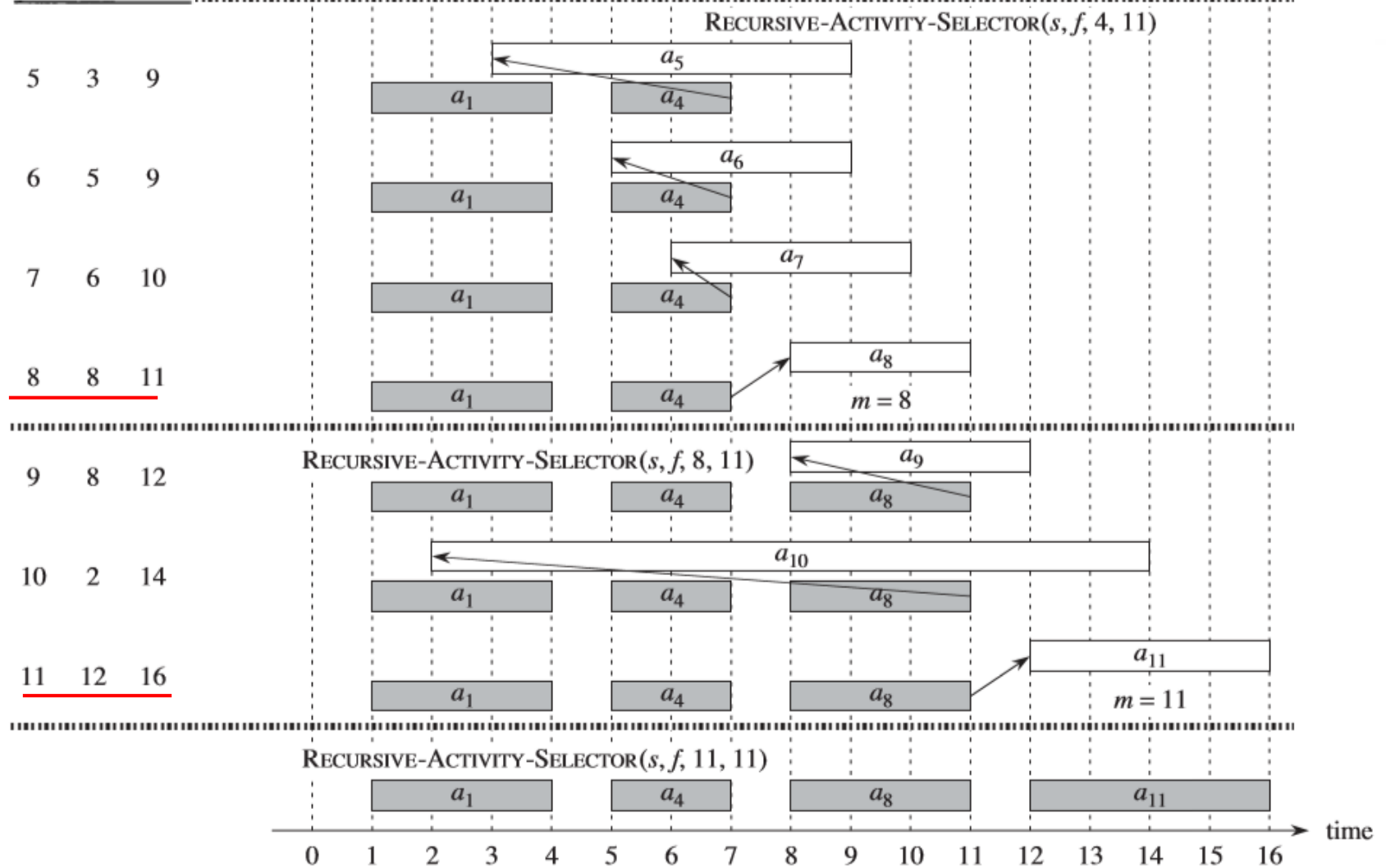
执行过程如图所示：

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

k	s_k	f_k
-----	-------	-------



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  $s[m] \geq f[k]$ 
6           $A = A \cup \{a_m\}$ 
7           $k = m$ 
8  return  $A$ 
```

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 $O(n)$ 。

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。

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。

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。

0 10 1100



a



b



c



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

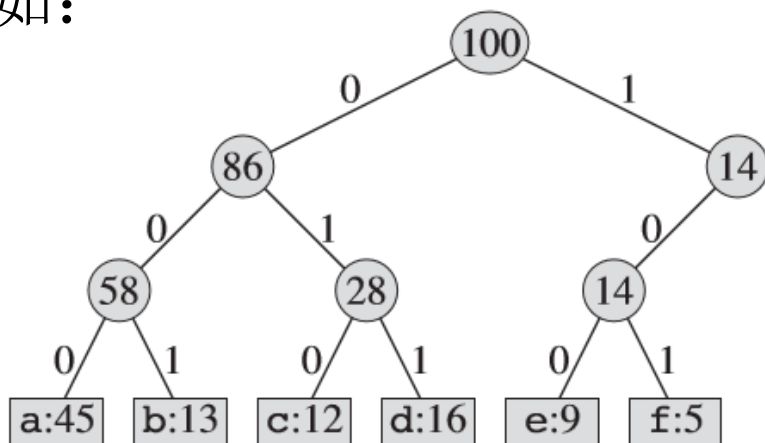
	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Variable-length codeword	0	101	100	111	1101	1100

编码树：一种为表示字符二进制编码而构造的二叉树。

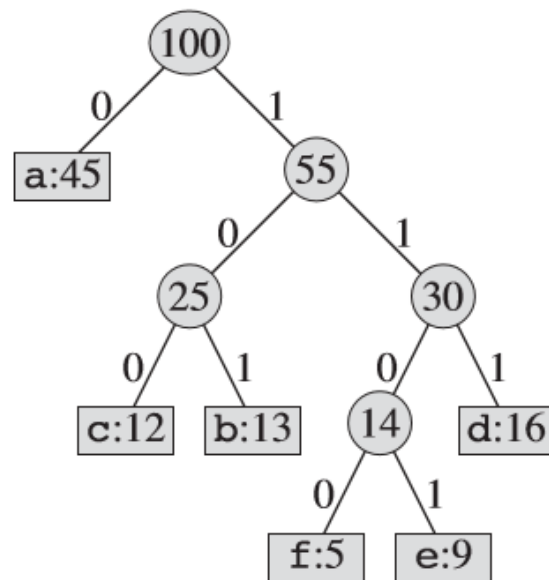
叶子结点：对应给定的字符，每个字符对应一个叶子结点。

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：



(a)

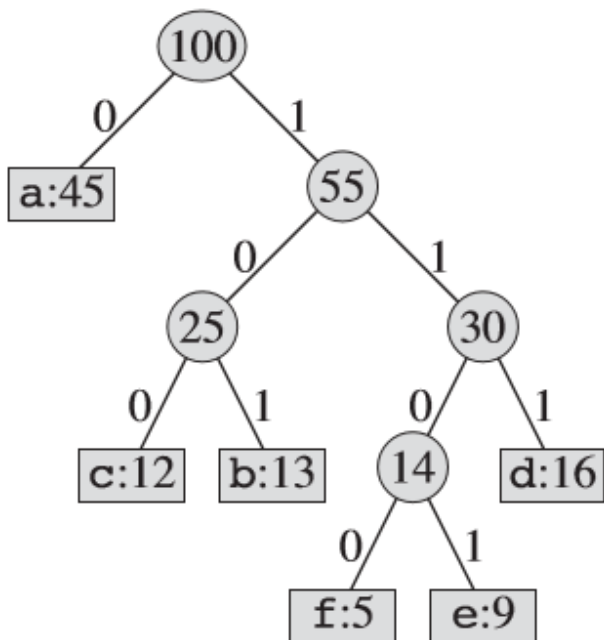


(b)

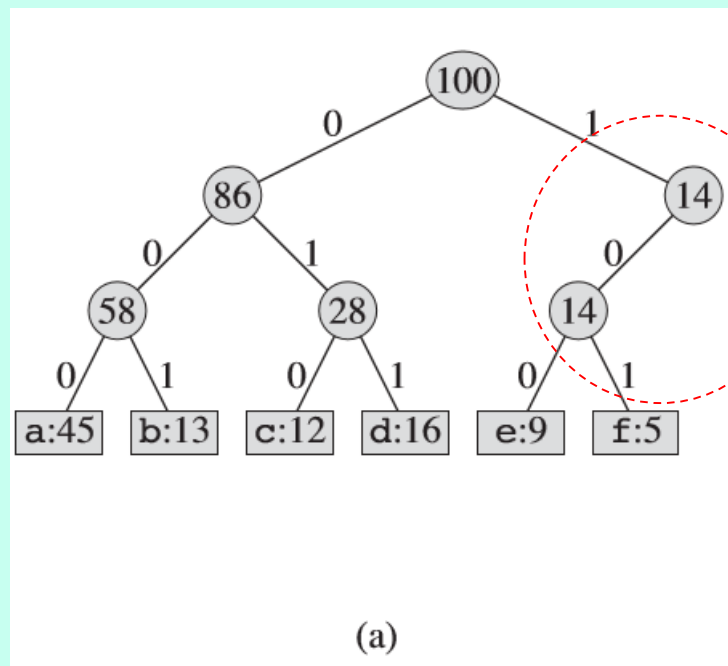
图 16-3 中编码方案的二叉树表示。每个叶结点标记了一个字符及其出现频率。每个内部结点标记了其子树中叶结点的频率之和。(a)对应定长编码 $a=000, \dots, f=101$ 的二叉树。(b)对应最优前缀码 $a=0, b=101, \dots, f=1100$ 的二叉树

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T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 时文件的编码长度，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)

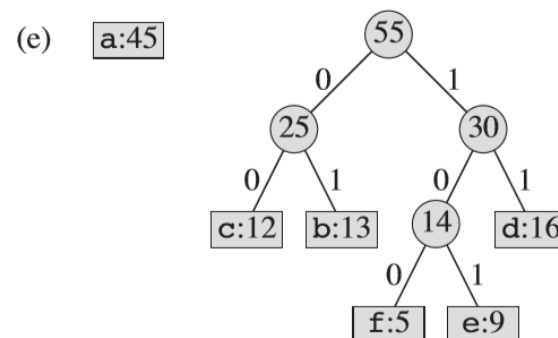
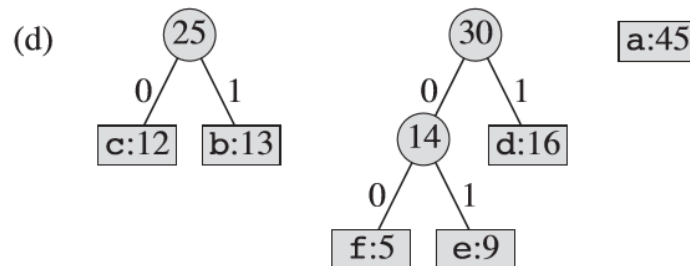
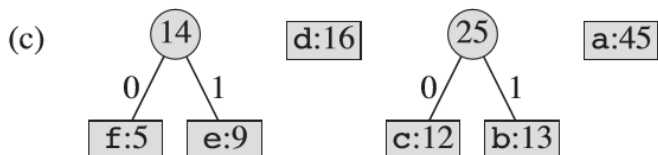
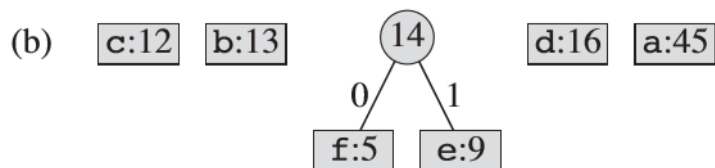
```
1   $n = |C|$ 
2   $Q = C$ 
3  for  $i = 1$  to  $n - 1$ 
4      allocate a new node  $z$ 
5       $z.left = x = \text{EXTRACT-MIN}(Q)$ 
6       $z.right = y = \text{EXTRACT-MIN}(Q)$ 
7       $z.freq = x.freq + y.freq$ 
8       $\text{INSERT}(Q, z)$ 
9  return  $\text{EXTRACT-MIN}(Q)$     // return 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。

例：构造前面实例的Huffman编码

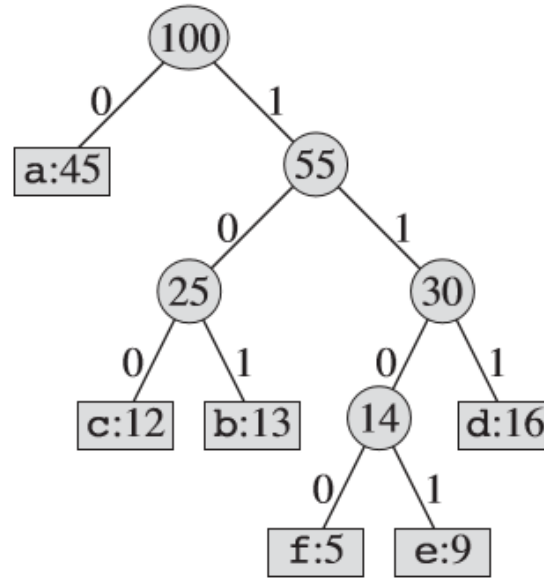
	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5

(a) f:5 e:9 c:12 b:13 d:16 a:45



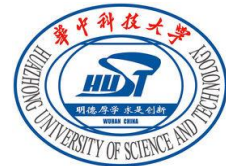
- 每一步选择频率最低的两棵树进行合并。
- 叶子结点用矩形表示，每个叶子结点包含一个字符及其频率。
- 内结点用圆形结点表示，频率等于其孩子结点的频率之和。
- 内结点指向左孩子的边标记为0，指向右孩子的边标记为1。
- **一个字母的码字对应从根到其叶子结点的路径上的边的标签序列。**

如 f 的码字是：1000，e 的码字是1001



(b)

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       $z.left = x = \text{EXTRACT-MIN}(Q)$ 
6       $z.right = y = \text{EXTRACT-MIN}(Q)$ 
7       $z.freq = x.freq + y.freq$ 
8       $\text{INSERT}(Q, z)$ 
9  return  $\text{EXTRACT-MIN}(Q)$  // 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 $O(n \lg n)$ 。
 - 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 $O(\lg n)$ ，所以循环的总代价是 $O(n \lg n)$ 。

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 $O(n \lg \lg n)$

HUFFMAN算法的正确性

有下面的引理16.2、16.3和定理16.4给出。

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 $c.\text{freq}$ 。

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：

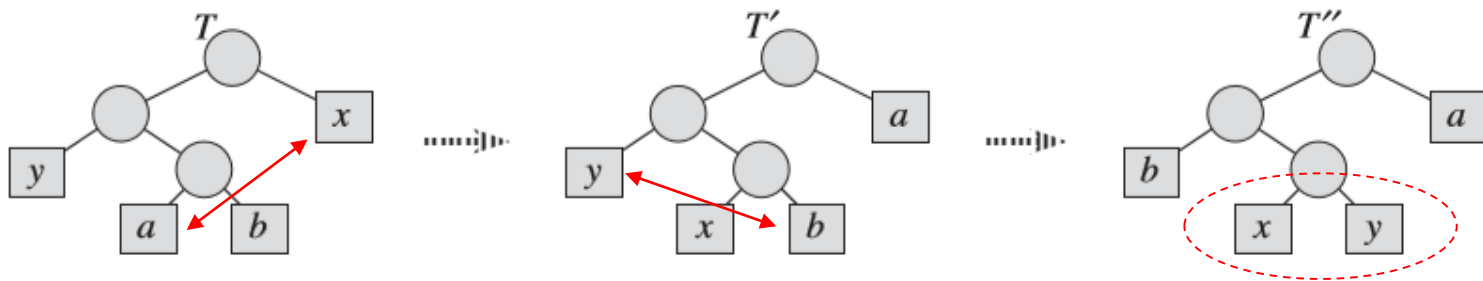
令 T 是一个最优前缀码所对应的编码树——**满二叉树**。

令 a 和 b 是 T 中深度最大的兄弟叶结点。

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

注：有可能 $x.\text{freq} = a.\text{freq}$ 或 $y.\text{freq} = b.\text{freq}$ 。

- 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 $x.freq \neq b.freq$ ，即 $x \neq b$ 。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：



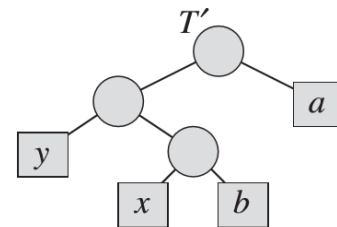
在最优树 T 中，叶子结点 a 和 b 是最深的叶子结点中的两个，并且是兄弟结点。叶子结点 x 和 y 为算法首先合并的两个叶子结点，它们可出现在 T 中的任意位置上。假设 $x \neq b$ ，叶子结点 a 和 x 交换得到树 T' ，然后交换叶子结点 b 和 y 得到树 T'' 。

根据文件编码的计算公式， T 和 T' 的**代价差**为：

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

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

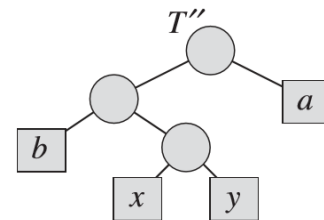
注，其中 $a.freq - x.freq$ 和 $d_T(a) - d_T(x)$ 均为非负值。



- $B(T) - B(T') \geq 0$ 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

$$B(T') - B(T'') \geq 0.$$

因此， $B(T'') \leq B(T)$ 。





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，x和y的
码字长度相同，且只有最后一个二进制位不同。

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 $c.\text{freq}$ 。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 $c.\text{freq}$ 。

- 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 $z.\text{freq} = x.\text{freq} + y.\text{freq}$ 。
- 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 $c \in C - \{x, y\}$ ，有

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

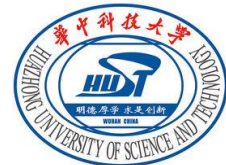
亦有： $c.freq \cdot d_T(c) = c.freq \cdot d_{T'}(c)$ 。

由于 $d_T(x) = d_T(y) = d_{T'}(z) + 1$

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

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, \cdots, p_7)
 $= (10, 5, 15, 7, 6, 18, 3)$, $(w_1, \cdots, w_7) = (2, 3, 5, 7, 1, 4, 1)$ 。