算法设计与分析

Computer Algorithm Design & Analysis

赵峰
zhaof@hust.edu.cn

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，希望这样的选择能导致全局最优解。

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

16.1 活动选择问题

1）问题描述

假定有一个n个活动的集合$S=\{a_1,a_2,\ldots,a_n\}$，这些活动都要求使用同一资源(如演讲会场)，而这个资源在某个时刻只能供一个活动使用。每个活动a_i都有一个开始时间s_i和一个结束时间f_i，且$0 \le s_i < f_i < \infty$。

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。

活动选择问题就是从活动集合中选出最大兼容活动的集合。

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：设有以下待安排的11个活动的开始时间和结束时间，并按结束时间的非减序排列如下：

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

$\{a_3, a_9, a_{11}\}$ $\{a_1, a_4, a_8, a_{11}\}$、$\{a_2, a_4, a_9, a_{11}\}$ 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。最大兼容活动集合不一定是唯一的。

（1）活动选择问题的最优子结构

活动选择问题具有最优子结构：

- 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。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，A_{ik}表示A_{ij}中a_k之前的活动子集， A_{kj}表示A_{ij}中a_k之后的活动子集。同时得到两个子问题：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。
- 则必有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$A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$ 。

贪心算法

在贪心算法的每一步所做的局部最优选择就叫做贪心选择。

活动选择问题的贪心选择：每次选择最早结束时间的活动加入集合A。

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

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 。

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。

- 在首次选择a_1后，下面寻找a_1结束后开始的活动。

- 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

。

- 最优子结构性：如果a_1在最优解中，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。

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，$A_k' = A_k-\{a_j\}\cup\{a_m\}$，且$A_k'$中的活动不相交。

所以A_k'也是S_k的一个最大兼容活动子集，且包含a_m。定理得证。

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。

活动选择问题的贪心算法

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

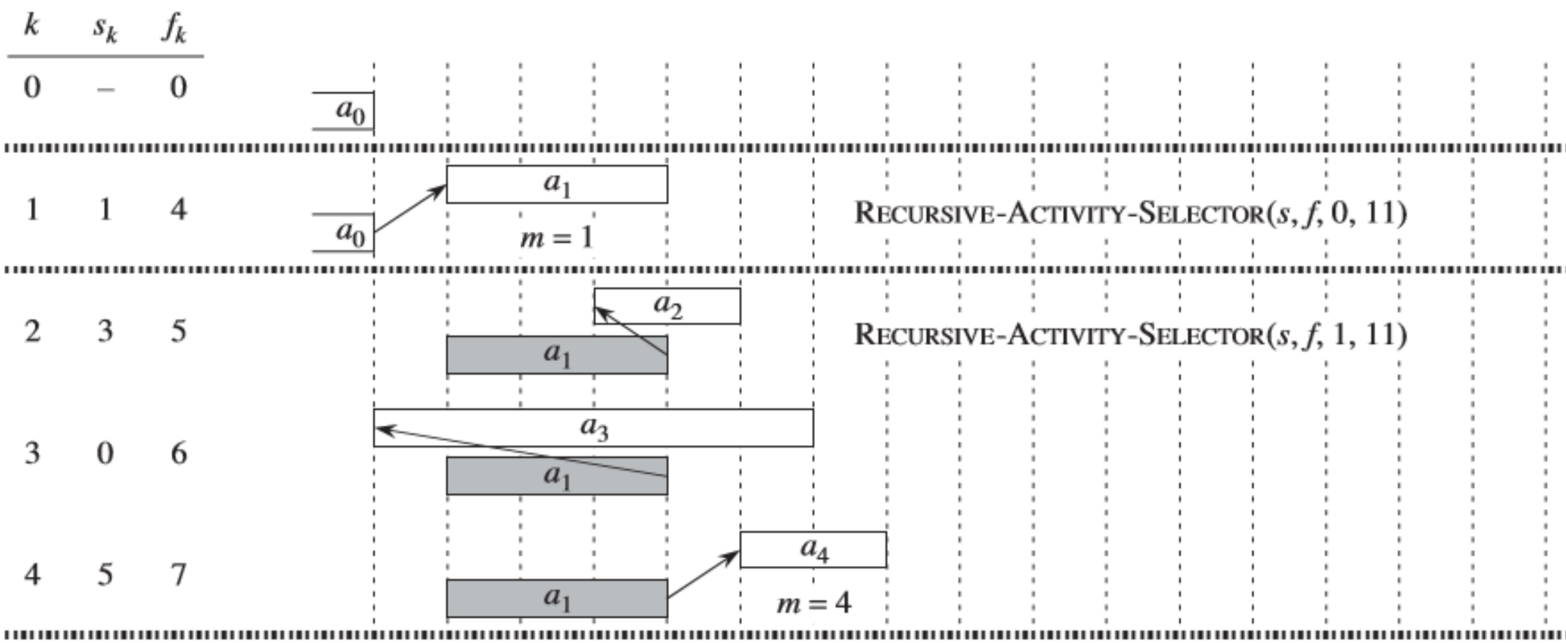
注：为处理方便，引入一个虚拟活动a_0，其结束时间f_0=0。

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

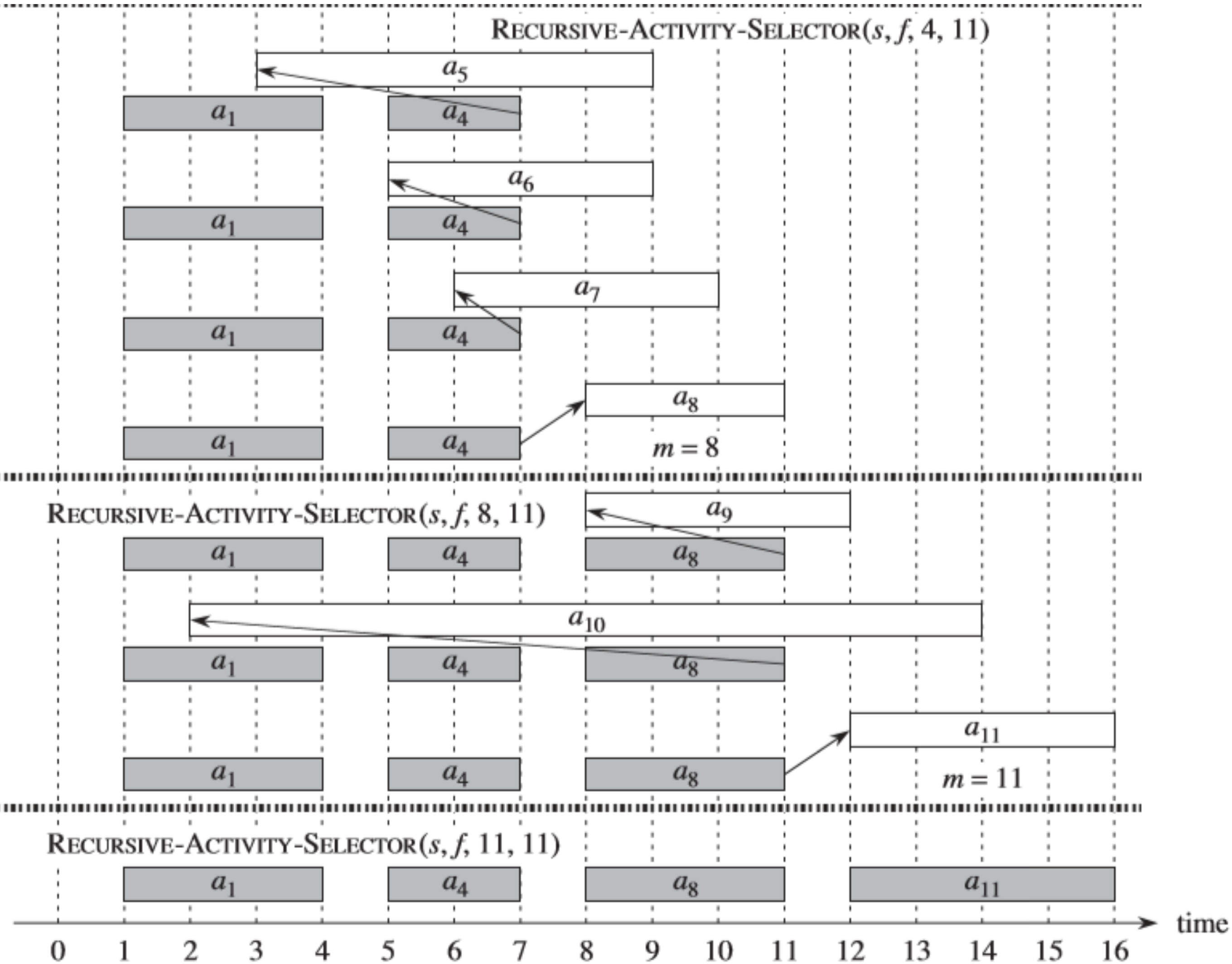
例：

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：



5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



迭代算法

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

- 假定活动已经按照结束时间单调递增的顺序排列好
- 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。
- 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。

贪心求解的一般步骤：

1）确定问题的最优子结构；

2）每次对其作出一次选择；

3）证明作出贪心选择后，原问题总是存在最优解，即安全；

4）证明作出贪心选择后，剩余的子问题满足：其最优解与贪心选择组合即可得到原问题的最优解。

- 贪心算法中贪心选择性质和最优子结构性是两个关键要素。

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。

如何证明每个步骤贪心选择能生成全局最优解？

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。

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。
- 对比动态规划算法：
  - 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

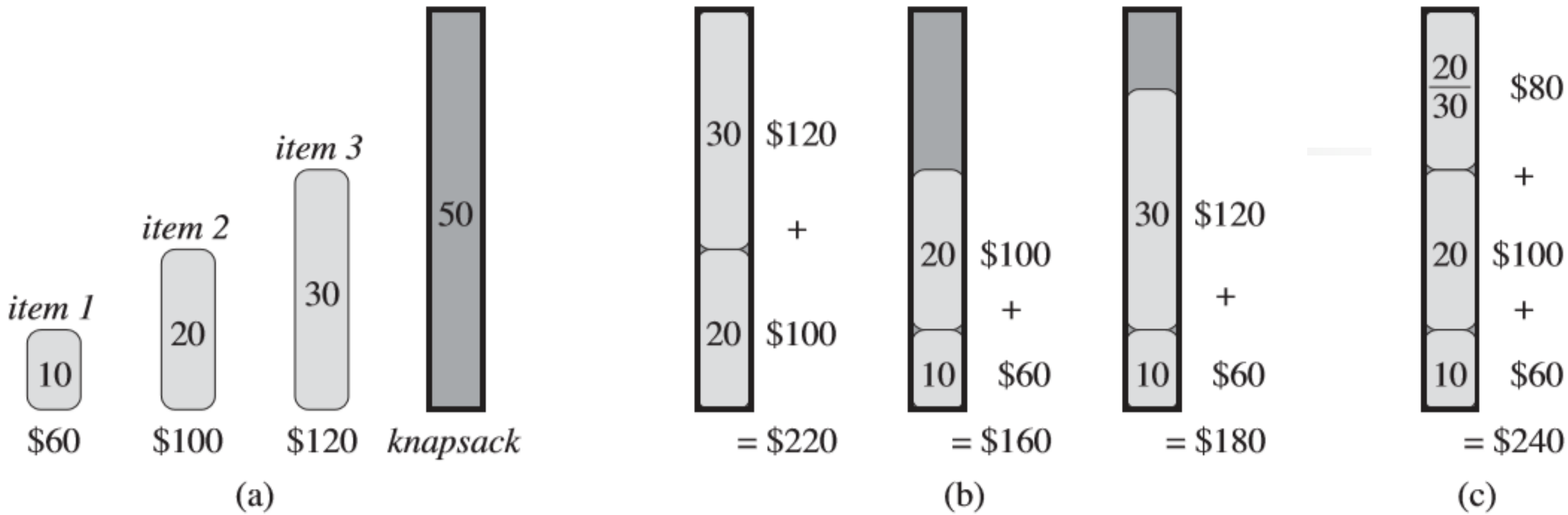


Figure 16.2 An example showing that the greedy strategy does not work for the 0-1 knapsack problem. **(a)** The thief must select a subset of the three items shown whose weight must not exceed 50 pounds. **(b)** The optimal subset includes items 2 and 3. Any solution with item 1 is suboptimal, even though item 1 has the greatest value per pound. **(c)** For the fractional knapsack problem, taking the items in order of greatest value per pound yields an optimal solution.

- 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

实例说明：

设一个有10万个字符的数据文件：

	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

分析：

采用二进制字符编码，每个字符用唯一的二进制串表示，称为码字。

1）定长编码：如3位码字

2）变长编码：每个字符赋予不同长度的码字。

如表中的变长编码方案，10万个字符仅需22.4万个二进制位，节约了25%的空间。

	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

最优编码方案

前缀码（Prefix code）：没有任何码字是其它码字的前缀。

前缀码的作用是简化解码过程。

- 由于没有码字是其它码字的前缀，编码文件的开始部分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。

0 1 0 1 1 0 0

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
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

编码树：用于表示字符二进制编码的二叉树。

叶子结点：对应给定的字符。

编码构造：由从根到字符叶子结点的简单路径：0代表“转向左孩子”，1代表“转向右孩子”。

如

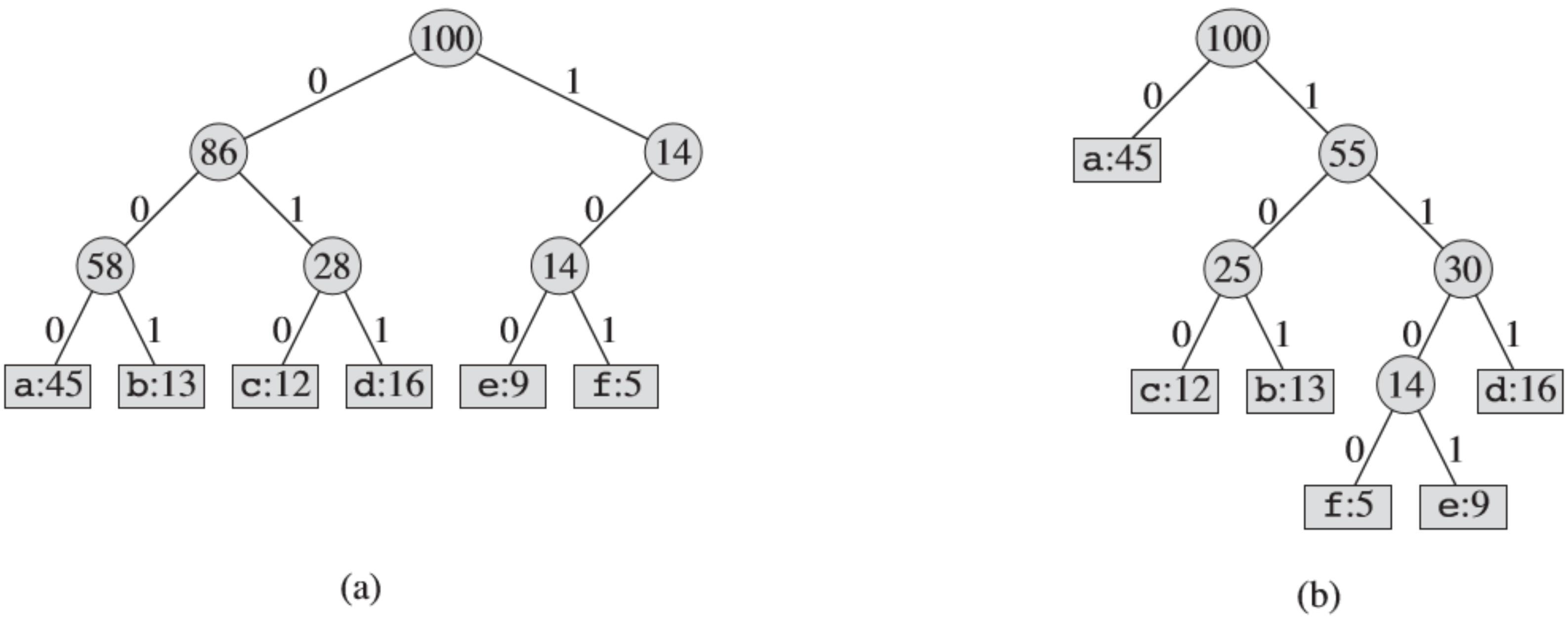
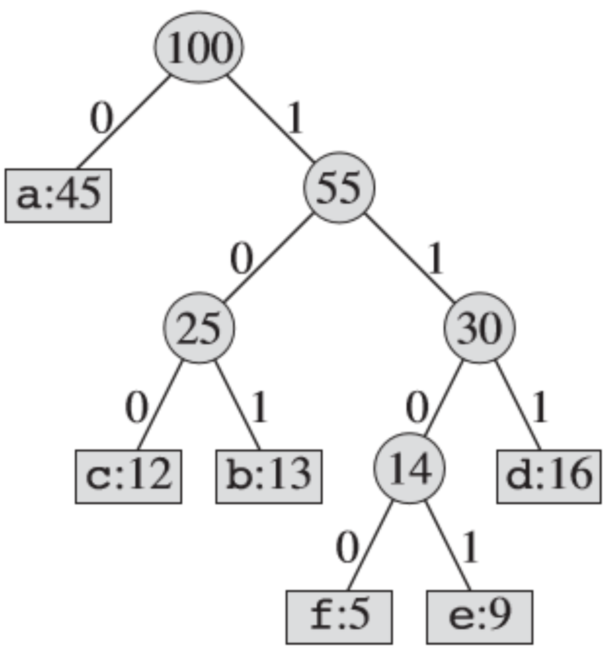


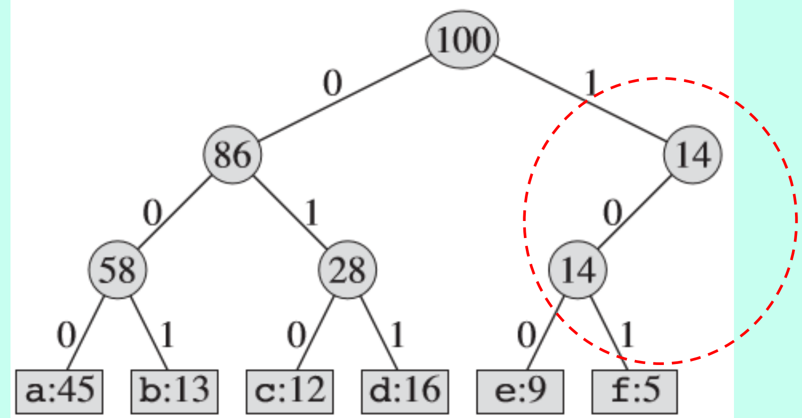
图 16-3 中编码方案的二叉树表示。每个叶结点标记了一个字符及其出现频率。每个内部结点标记了其子树中叶结点的频率之和。(a)对应定长编码 a=000，…，f=101 的二叉树。(b)对应最优前缀码 a=0，b=101，…，f=1100 的二叉树

一个文件的最优字符编码方案总对应一棵满(full)二叉树。

如图(b)：



(b)



(a)

图(a)的定长编码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

- 设C为字母表
  - 任意字符c∈C，令属性c.freq表示字符c在文件中出现的频率。
  - 最优前缀码树中恰好有|C|个叶子结点，每个叶子结点对应一个字符
  - 有|C|-1个内部结点。
- T表示一棵前缀编码树；
- $d_T(c)$表示c的叶子结点在树中的深度。
- B(T)表示采用编码方案T文件的编码长度：

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

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。

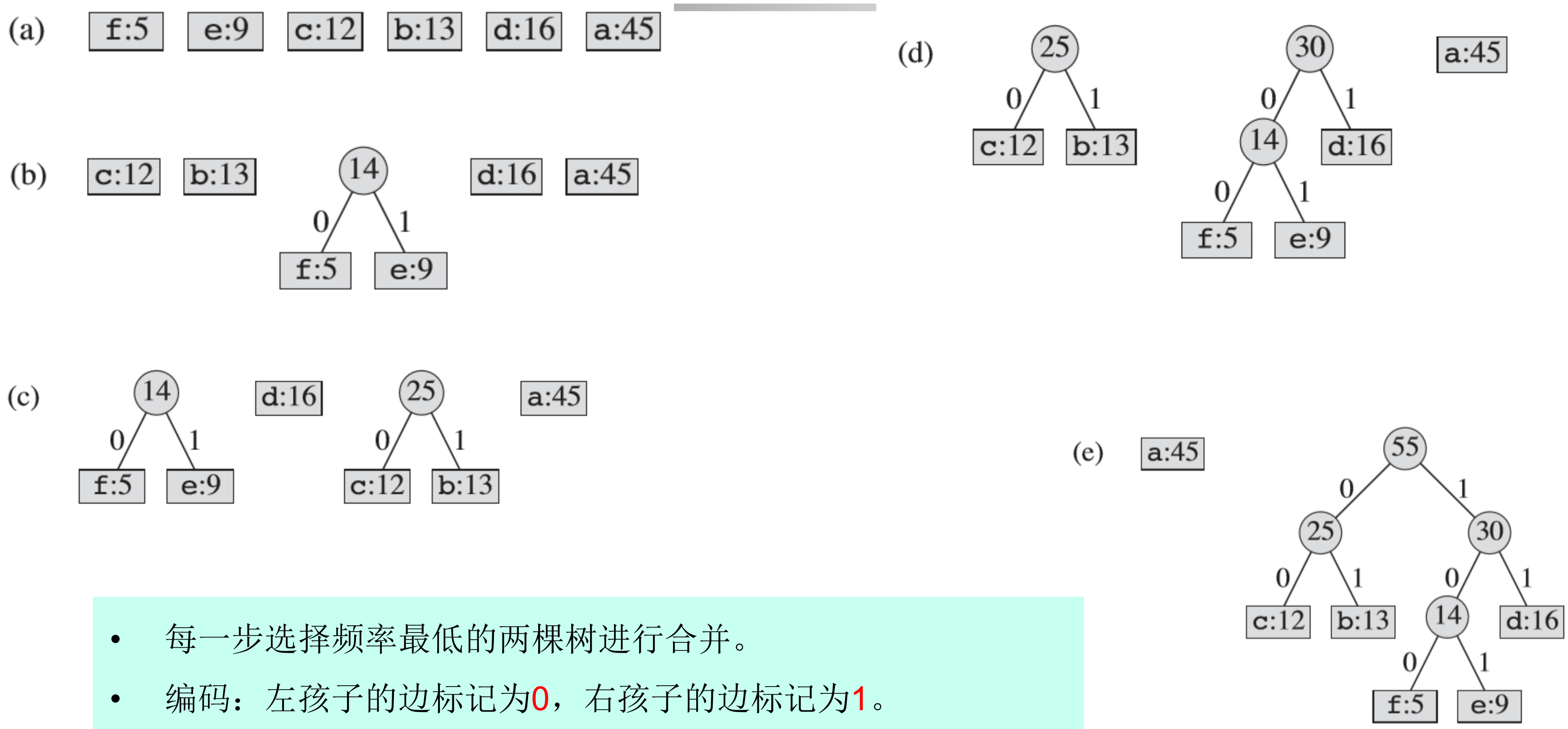
Huffman编码的贪心算法

算法每次选择频率最低的两个结点合并，执行|C|-1次“，构造出一棵编码树。

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

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。
- 编码：左孩子的边标记为0，右孩子的边标记为1。

时间分析

Q使用最小二叉堆实现

- Q的初始化：O(n)。
- 循环的总代价：O($n$lg$n$)。

HUFFMAN的总运行时间O($n$lg$n$)。

- 最小二叉堆换为van Emde Boas树，可以将运行时间减少到O(nlglgn)

HUFFMAN算法的正确性

引理 16.2 令C为一个字母表，其中每个字符c∈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。

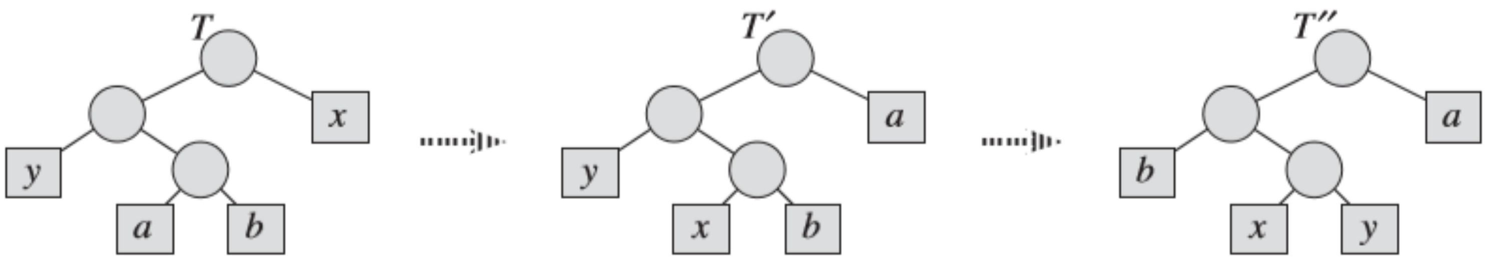
证明：

令T是最优前缀码编码树，a和b是T中深度最大的兄弟叶结点。

- x.freq≤a.freq且y.freq≤b.freq。

- 若x.freq=b.freq，引理成立。

- 若x.freq≠b.freq，在T中交换x和a，生成一棵新树T′；在T′中交换b和y，生成T″，如图所示：



在最优树T中，叶子结点x和y为算法首先合并的两个叶子结点。

有：

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

- 类似地，B(T’)-B(T’’)≥0。

因此， B(T“)≤ B(T)。

根据假设，T是最优的，T“也是最优解。

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。
  - 编码树总代价等于所有合并操作的代价之和。
  - HUFFMAN选择是一个代价最小的方案：

引理 16.3

令C为一个给定的字母表，其中每个字符c∈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。

- C' = C - {x, y}∪{z}。
  - z.freq= x.freq + y.freq。
- 令**T'**为字母表C'的最优前缀码的编码树。

则有：将T'中叶子结点z替换为以x和y为孩子的内部结点，则T表示字母表C的一个最优前缀码。

证明：

对c∈C-{x,y}：$d_T(c)=d_{T'}(c)$，

对x、y有：$d_T(x)=d_T(y)=d_{T'}(z)+1$

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：

即

$$B(T) = B(T') + x.freq + y.freq$$

$$B(T') = B(T) - x.freq - y.freq$$

假定T不是C的最优前缀码。令最优前缀码树T″。

- 不失一般性，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′代表一个最优前缀码相矛盾。

因此，T表示C的一个最优前缀码。

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。

作业

- 16.1-4
- 16.2-6
- 16.2-7
- 16-3.3
- 16-1