计算机问题求解 – 论题3-2 - 贪心算法

2016年09月07日

问题1:

你还记得什么是“Optimal Substructure”吗？该结构特性对求解最优解问题有什么启发？

Activity Selection Problem

Suppose we have a set $S = \{a_1, a_2, \ldots, a_n\}$ of n proposed ***activities*** that wish to use a resource, such as a lecture hall, which can serve only one activity at a time. Each activity a_i has a ***start time*** s_i and a ***finish time*** f_i, where $0 \le s_i < f_i < \infty$. If selected, activity a_i takes place during the half-open time interval $[s_i, f_i)$. Activities a_i and a_j are ***compatible*** if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap. That is, a_i and a_j are compatible if $s_i \ge f_j$ or $s_j \ge f_i$. In the ***activity-selection problem***, we wish to select a maximum-size subset of mutually compatible activities.

一个样本输入：

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

问题2：

Activity Selection问题是否具有“最优子结构”，为什么？

S_{ij}表示开始时间不早于活动a_i的结束时间，而结束时间早于a_j的结束时间的所有活动的集合。

If we denote the size of an optimal solution for the set S_{ij} by $c[i, j]$, then we would have the recurrence

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

假设我们知道其中包含活动a_k。

S_{ij}中最多相互兼容的活动数

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

动态规划解法

在上述递归关系中，a_k可以是S_{ij}中任一活动，每选定一个特定的a_k, 则确定特定的子问题。动态规划方法按照合适的次序解所有的子问题。

问题3：

是否有可能不必解所有的子问题？

问题4:

所谓“GREEDY”是指什么？

Activity Selection: the Idea

- 要解的问题用 $S_k = \{a_i \in S : s_i \geq f_k\}$表示，$S$是原始问题所给的所有活动的集合，则原始问题为$S_0$;
- Greedy方法：
  - 选择完成时间最早的活动，假设是a_1;
  - 解子问题S_1。

Greedy可以指不同的“最”，但有的“最”可以得到正确的解，有的“最”却未必！

如何去“编程表达”这样的递归式？

RECURSIVE-ACTIVITY-SELECTOR ( )

GREEDY-ACTIVITY-SELECTOR(s, f)

$n = s.length$
$A = \{a_1\}$
$k = 1$
for $m = 2$ **to** n
    if $s[m] \geq f[k]$
        $A = A \cup \{a_m\}$
        $k = m$
return A

问题5：

为什么不需要递归？

问题6：

为什么代价是线性的？

问题7：

仅仅具有“最优子结构”
能保证贪心算法的正确吗？
还需要什么条件？

The first key ingredient is the ***greedy-choice property***: we can assemble a globally optimal solution by making locally optimal (greedy) choices. In other words, when we are considering which choice to make, we make the choice that looks best in the current problem, without considering results from subproblems.

问题8：

这里有递归的“影子”，你能解释一下吗？你能设想一个证明这个特性的基本方法吗？

贪心算法解活动选择问题的正确性

Theorem 16.1

Consider any nonempty subproblem S_k, and let a_m be an activity in S_k with the earliest finish time. Then a_m is included in some maximum-size subset of mutually compatible activities of S_k.

Proof Let A_k be a maximum-size subset of mutually compatible activities in S_k, and let a_j be the activity in A_k with the earliest finish time. If $a_j = a_m$, we are done, since we have shown that a_m is in some maximum-size subset of mutually compatible activities of S_k. If $a_j \neq a_m$, let the set $A'_k = A_k - \{a_j\} \cup \{a_m\}$ be A_k but substituting a_m for a_j. The activities in A'_k are disjoint, which follows because the activities in A_k are disjoint, a_j is the first activity in A_k to finish, and $f_m \leq f_j$. Since $|A'_k| = |A_k|$, we conclude that A'_k is a maximum-size subset of mutually compatible activities of S_k, and it includes a_m. ■

How to prove your greedy choice property?

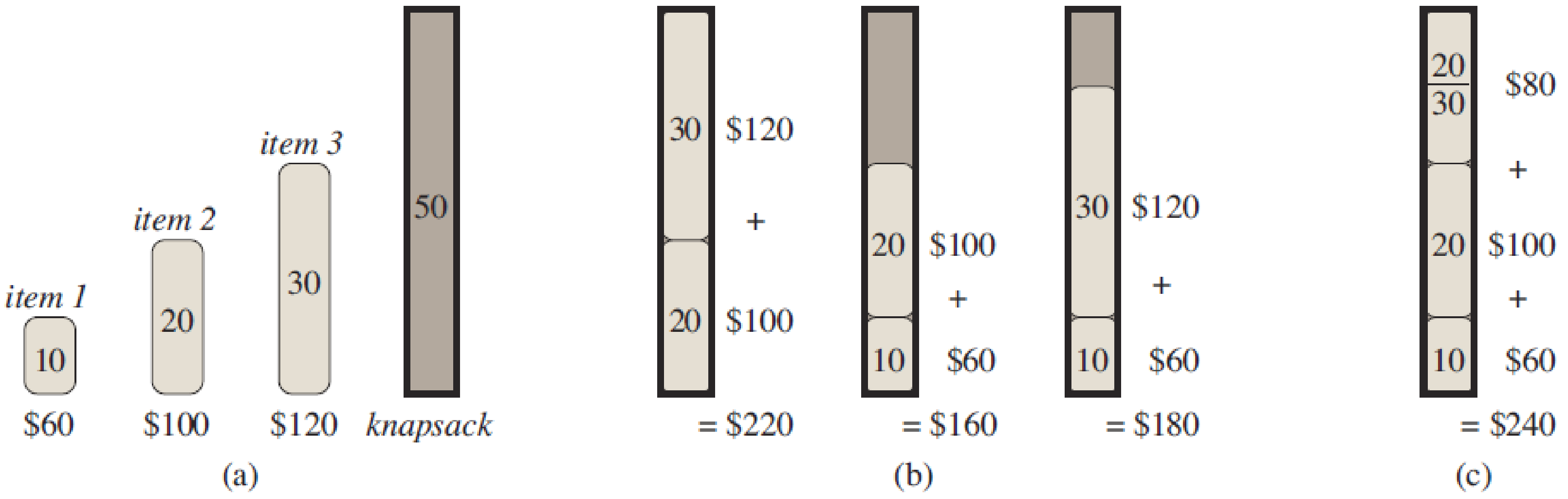
Of course, we must prove that a greedy choice at each step yields a globally optimal solution. Typically, as in the case of Theorem 16.1, the proof examines a globally optimal solution to some subproblem. It then shows how to modify the solution to substitute the greedy choice for some other choice, resulting in one similar, but smaller, subproblem.

问题9：为什么这个定理保证我们需要的正确性？

Optimal substructure tells us that if a_1 is in the optimal solution, then an optimal solution to the original problem consists of activity a_1 and all the activities in an optimal solution to the subproblem S_1.

Greedy vs. Dynamic Programming

- 能用贪心算法你不用，你就“亏”了；
- 不能用贪心算法你用了，你就错了！



问题10：

你觉得为什么fractional knapsack 行，0-1就不行？

Review the idea of Activity Selection

- 要解的问题用 $S_k = \{a_i \in S : s_i \geq f_k\}$表示，$S$是原始问题所给的所有活动的集合，则原始问题为$S_0$;
- Greedy方法：
  - 选择完成时间最早的活动，假设是a_1;
  - 解子问题S_1。

Greedy可以指不同的“最”，但有的“最”可以得到正确的解，有的“最”却未必！

问题：**一个问题能否采用贪心方法，依赖于“如何贪心”吗？**

问题11：

字母编码，用固定长度与可变长度，各有什么利弊？

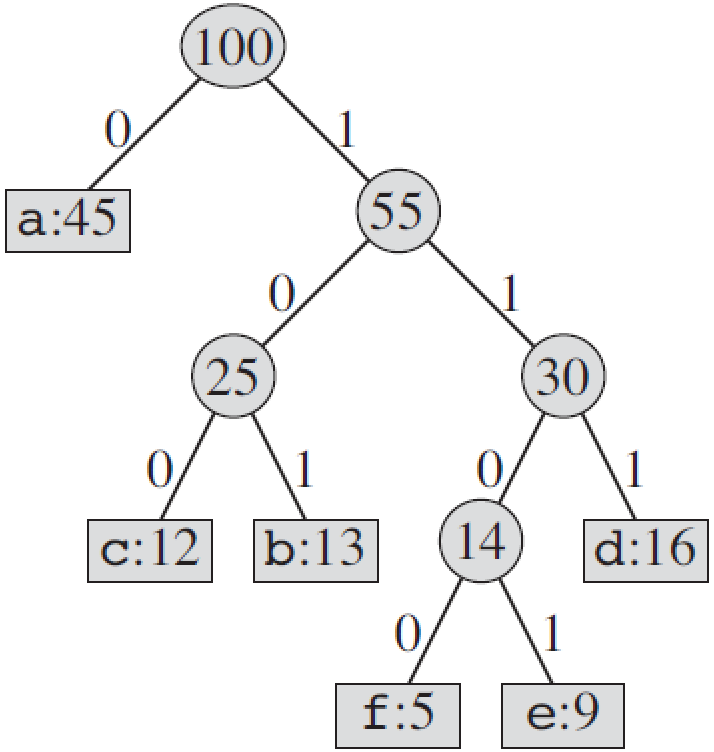
前缀码可以发挥可变长编码的
优点，又可以避免使用界限符。

	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

界限符：　0.0.101.1101

example, the string 001011101 parses uniquely as $0 \cdot 0 \cdot 101 \cdot 1101$, which decodes to aabe.

哈夫曼编码：

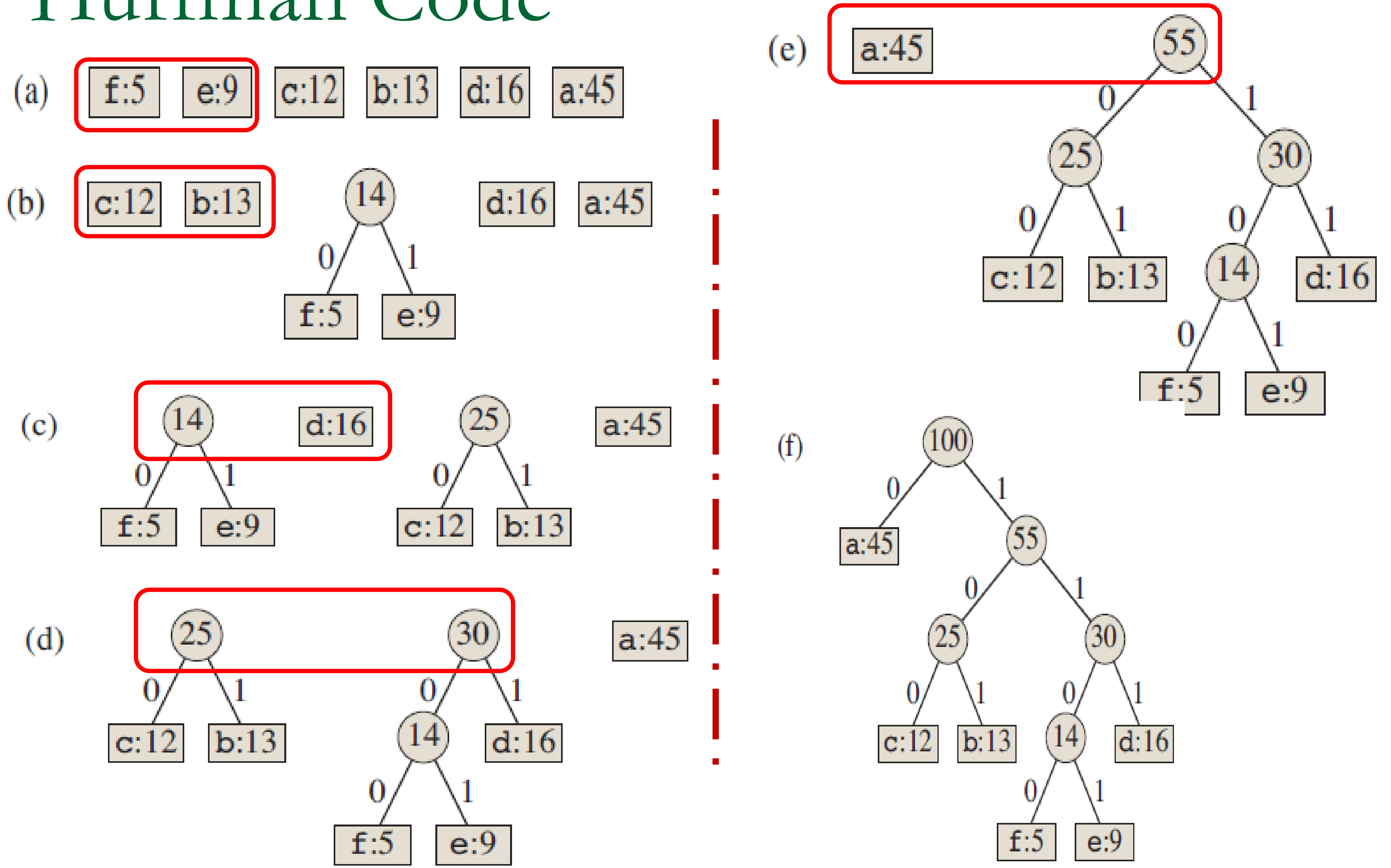


$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c)$$

使得B(T)最小的$d_T(c)$如何构造？

给定一段位串001011101，如何解码？

Huffman Code



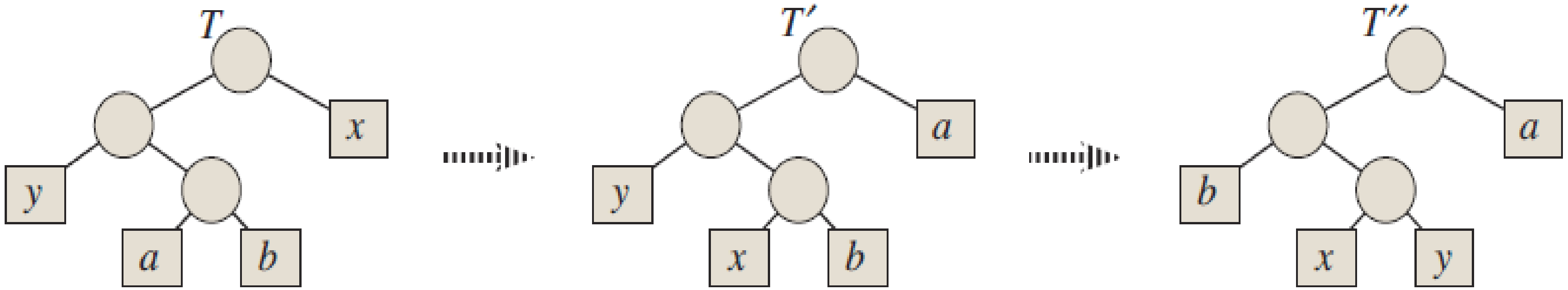
```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n - 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT(Q, z)
9  return EXTRACT-MIN(Q)        // return the root of the tree
```

Q 是一个最小优先队列。

To analyze the running time of Huffman's algorithm, we assume that Q is implemented as a binary min-heap (see Chapter 6). For a set C of n characters, we can initialize Q in line 2 in $O(n)$ time using the BUILD-MIN-HEAP procedure discussed in Section 6.3. The **for** loop in lines 3–8 executes exactly $n-1$ times, and since each heap operation requires time $O(\lg n)$, the loop contributes $O(n \lg n)$ to the running time. Thus, the total running time of HUFFMAN on a set of n characters is $O(n \lg n)$.

问题12：

最优前缀码问题满足greedy-choice property，这一点该如何表述？



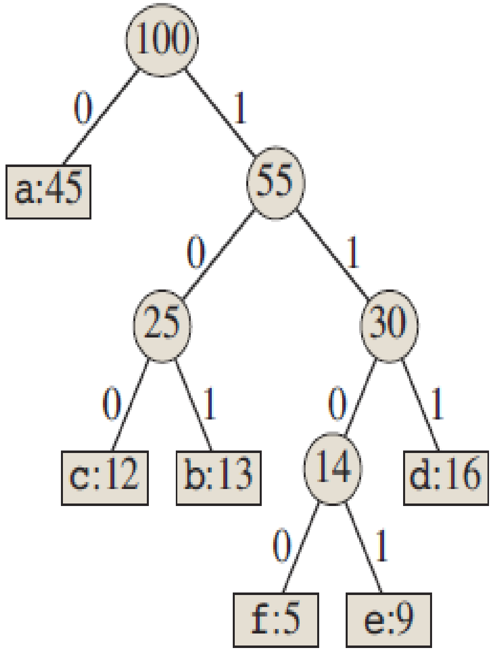
$$
\begin{aligned}
B(T) - B(T') &\\
&= \sum_{c \in C} c.\mathit{freq} \cdot d_T(c) - \sum_{c \in C} c.\mathit{freq} \cdot d_{T'}(c) \\
&= x.\mathit{freq} \cdot d_T(x) + a.\mathit{freq} \cdot d_T(a) - x.\mathit{freq} \cdot d_{T'}(x) - a.\mathit{freq} \cdot d_{T'}(a) \\
&= x.\mathit{freq} \cdot d_T(x)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

Exchanging y and b does not increase the cost, and so $B(T') - B(T'')$ is nonnegative. Therefore, $B(T'') \leq B(T)$, and since T is optimal, we have $B(T) \leq B(T'')$, which implies $B(T'') = B(T)$. Thus, T'' is an optimal tree .

问题13:

最优前缀码问题满足optimal-substructure property，这点该如何表述？

(f)



Let C be a given alphabet with frequency $c.freq$ defined for each character $c \in C$. Let x and y be two characters in C with minimum frequency. Let C' be the alphabet C with the characters x and y removed and a new character z added, so that $C' = C - \{x, y\} \cup \{z\}$. Define f for C' as for C, except that $z.freq = x.freq + y.freq$. Let T' be any tree representing an optimal prefix code for the alphabet C'. Then the tree T, obtained from T' by replacing the leaf node for z with an internal node having x and y as children, represents an optimal prefix code for the alphabet C.

注意： For each character $c \in C - \{x, y\}$, we have that $d_T(c) = d_{T'}(c)$, and hence $c.freq \cdot d_T(c) = c.freq \cdot d_{T'}(c)$. Since $d_T(x) = d_T(y) = d_{T'}(z) + 1$, we have

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

即： $B(T) = B(T') + x.freq + y.freq$

Suppose that T does not represent an optimal prefix code for C. Then there exists an optimal tree T'' such that $B(T'') < B(T)$. Without loss of generality (by Lemma 16.2), T'' has x and y as siblings. Let T''' be the tree T'' with the common parent of x and y replaced by a leaf z with frequency $z.freq = x.freq + y.freq$. Then

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

yielding a contradiction to the assumption that T' represents an optimal prefix code for C'. Thus, T must represent an optimal prefix code for the alphabet C. ∎

Open Topics：

1，证明哈夫曼编码一定是前缀码。

2，设有n个正整数，将他们连接成一排，组成一个最大的多位整数。请写出算法，给出贪心选择特性和最优子结构特性证明。

课外作业

- TC pp.422-: ex.16.1-2, 16.1-3
- TC pp.427-: ex.16.2-1, 16.2-2
- TC pp.436-: ex.16.3-2, 16.3-5, 16.3-8
- TC pp.446-: prob.16-1