计算机问题求解 – 论题2-13 - 贪心算法

2019年05月20日

Part I
Greedy Strategy

问题1:

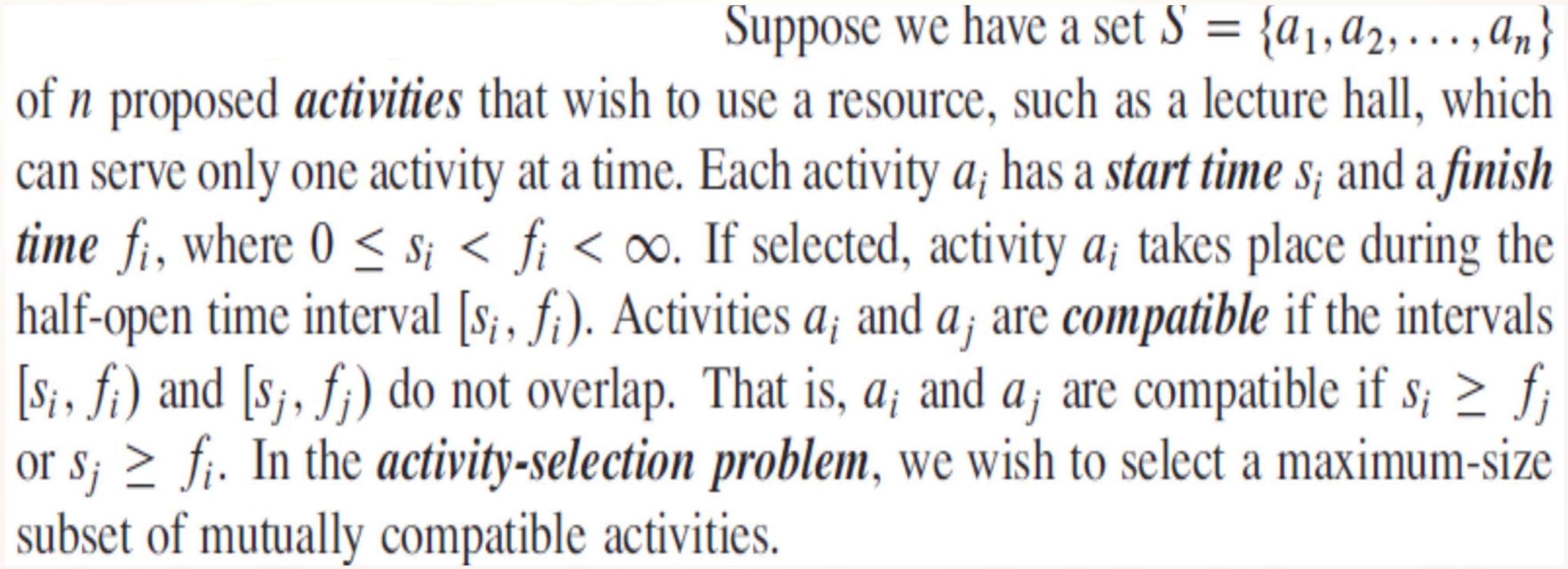
你还记得什么是“Optimal Substructure”吗？你能否用集合论的语言解释这个概念？

optimal substructure:
optimal solutions to a problem incorporate optimal solutions to related subproblems, which we may solve independently.

一个“问题”可以看成其所有instance的集合，任一子问题即一子集。

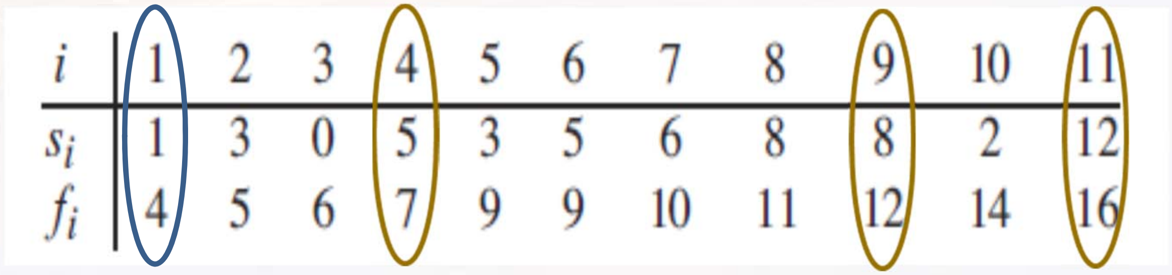
你能否解释：为什么最短路问题具有“最优子结构”，
而“最长路问题”则没有次性质？有多大差别？

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.

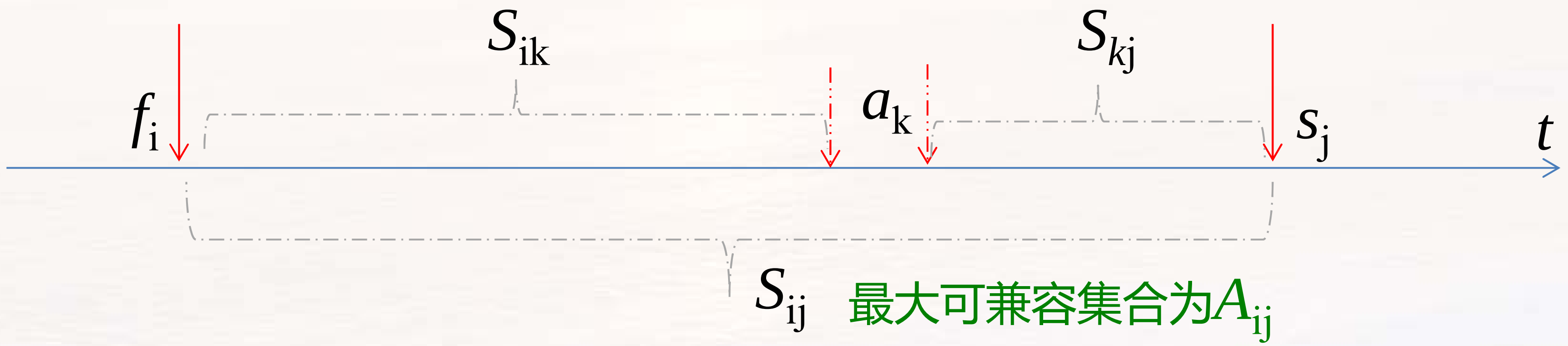
一个样本输入：



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？



最大可兼容集合为A_{ij}

$A_{ik} = A_{ij} \cap S_{ik}$ $A_{kj} = A_{ij} \cap S_{kj}$ ⇨ $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$

显然：若A_{ij}是最优解，A_{ik}与A_{kj}也必为相应子问题的最优解。

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？你能否用集合论的语言描述这一方法?

Greedy by intuition:

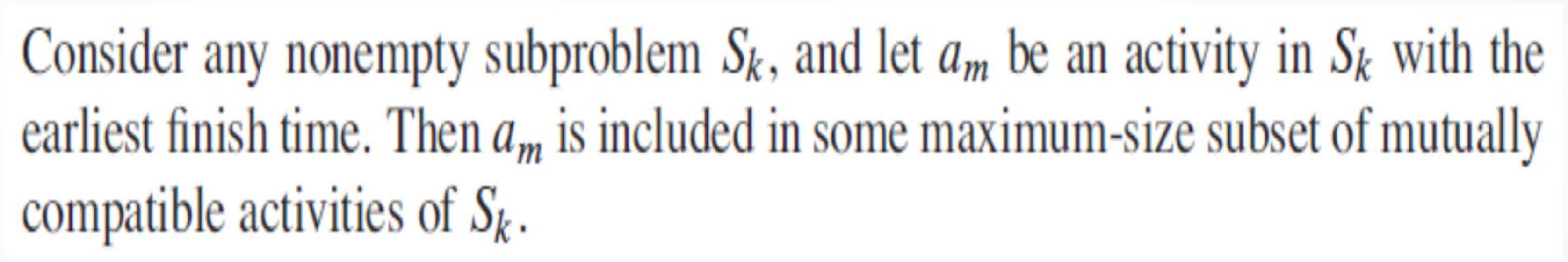
Intuition suggests that we should choose an activity that leaves the resource available for as many other activities as possible. Now, of the activities we end up choosing, one of them must be the first one to finish. Our intuition tells us, therefore, to choose the activity in *S* with the earliest finish time, since that would leave the resource available for as many of the activities that follow it as possible.

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。

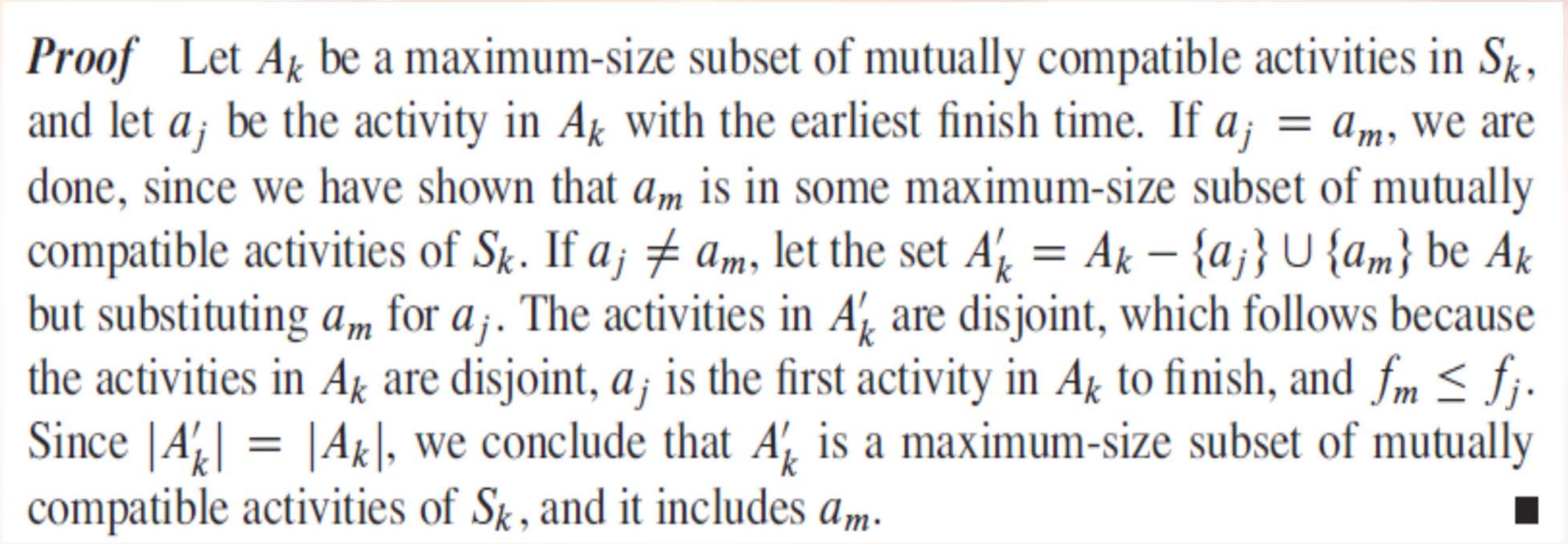
Greedy可以指不同的“最”，但有的“最”可以得到正确的解，有的“最”却未必！

上述思想的合理性


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.

问题5：

这段话是什么意思，与我们的解题思路的合理性有什么关系？



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. ■

问题6：

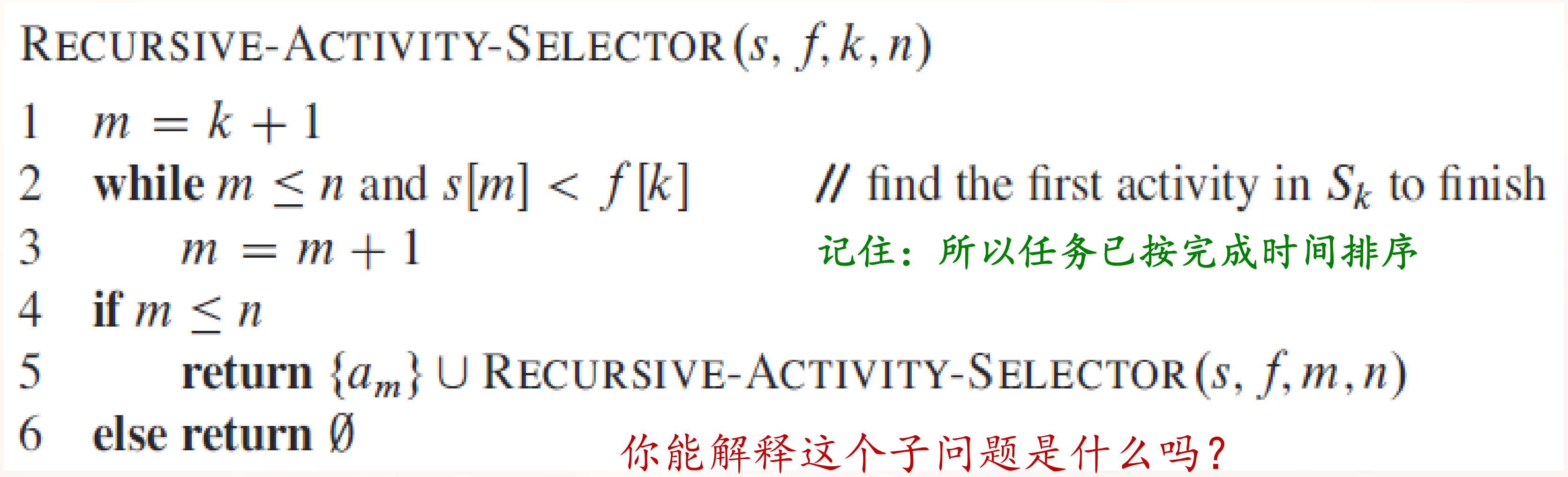
你能用通俗的语言说说这个证明的要点吗？

就算一个最优解中原来不含a_m, 我们也可以让它包含a_m:

(1) 将原来最优解中结束时间最早的任务换成a_m, 不会导致不兼容，因为a_m是S_k中结束时间最早的；

(2) 新的解集合没变小。

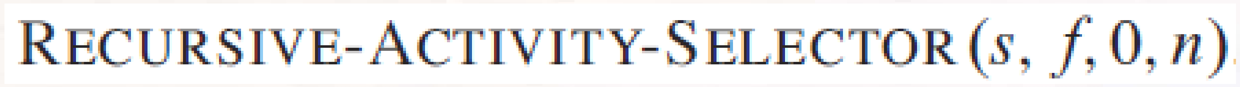
用递归方法计算（子）问题S_k



```
RECURSIVE-ACTIVITY-SELECTOR$(s, f, k, n)$
$m = k + 1$
while $m \leq n$ and $s[m] < f[k]$        // find the first activity in $S_k$ to finish
    $m = m + 1$                            记住：所以任务已按完成时间排序
if $m \leq n$
    return $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR$(s, f, m, n)$
else return $\emptyset$
```

你能解释这个子问题是什么吗？

则解初始问题调用

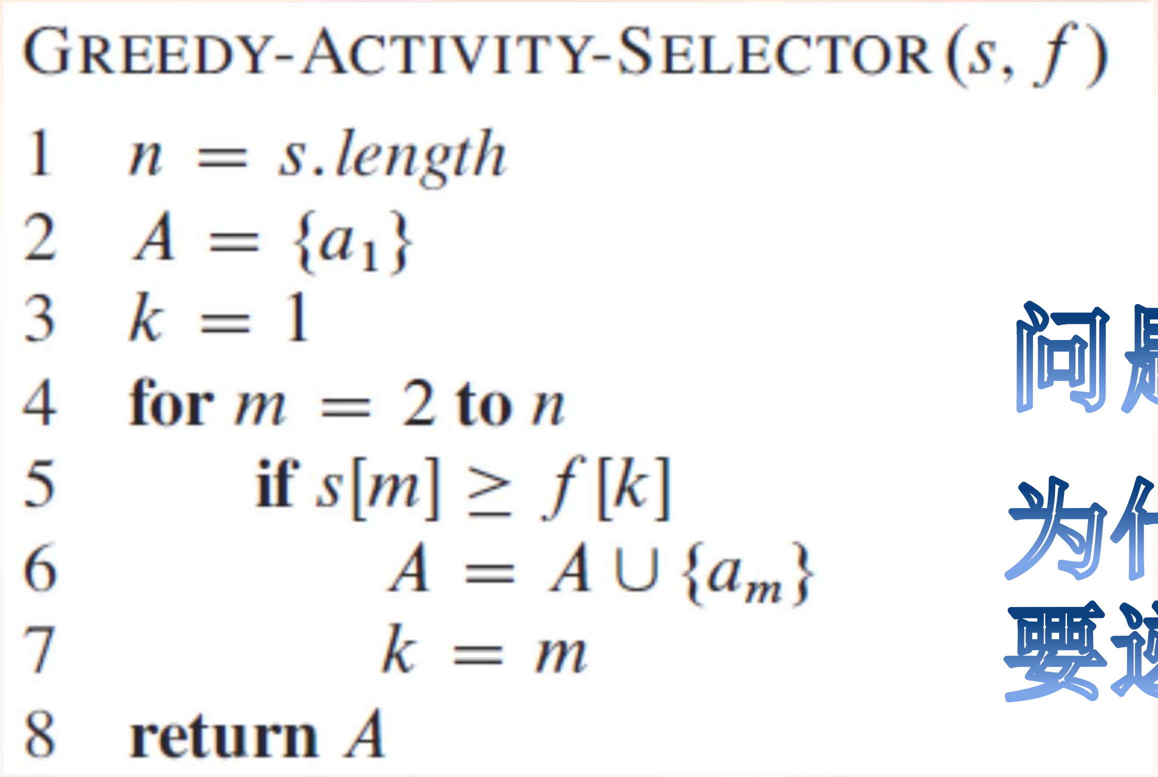


RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0, n)$

The procedure RECURSIVE-ACTIVITY-SELECTOR is almost "**tail recursive**" : it ends with a recursive call to itself followed by a union operation. It is usually a straightforward task to transform a tail-recursive procedure to an iterative form; in fact, some compilers for certain programming languages perform this task automatically.

问题7：

书上这一段关于"尾递归"的话什么意思？



```
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8：

为什么不需要递归？

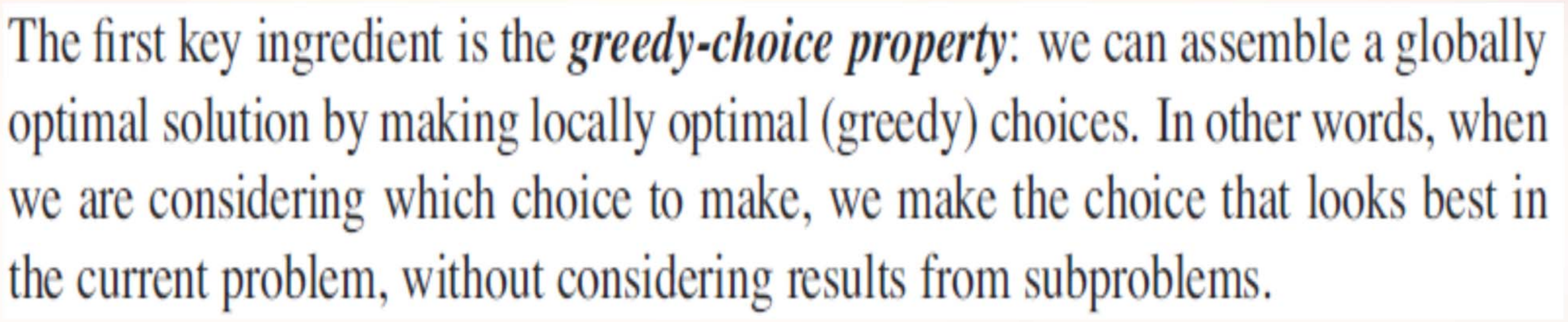
问题9：

为什么代价是线性的？

问题10：

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？

How can we tell whether a greedy algorithm will solve a particular optimization problem? No way works all the time, but **the greedy-choice property and optimal substructure are the two key ingredients**. If we can demonstrate that the problem has these properties, then we are well on the way to developing a greedy algorithm for it.

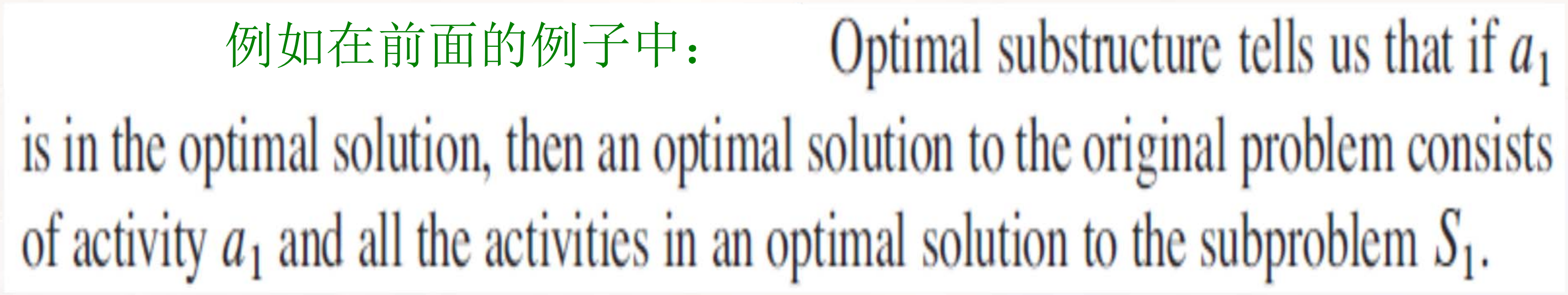


The first key ingredient is the ***greedy-choice property***: we can assemble a globally optimal solution by making locally optimal (greedy) choices. In other words, when we are considering which choice to make, we make the choice that looks best in the current problem, without considering results from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

问题11：

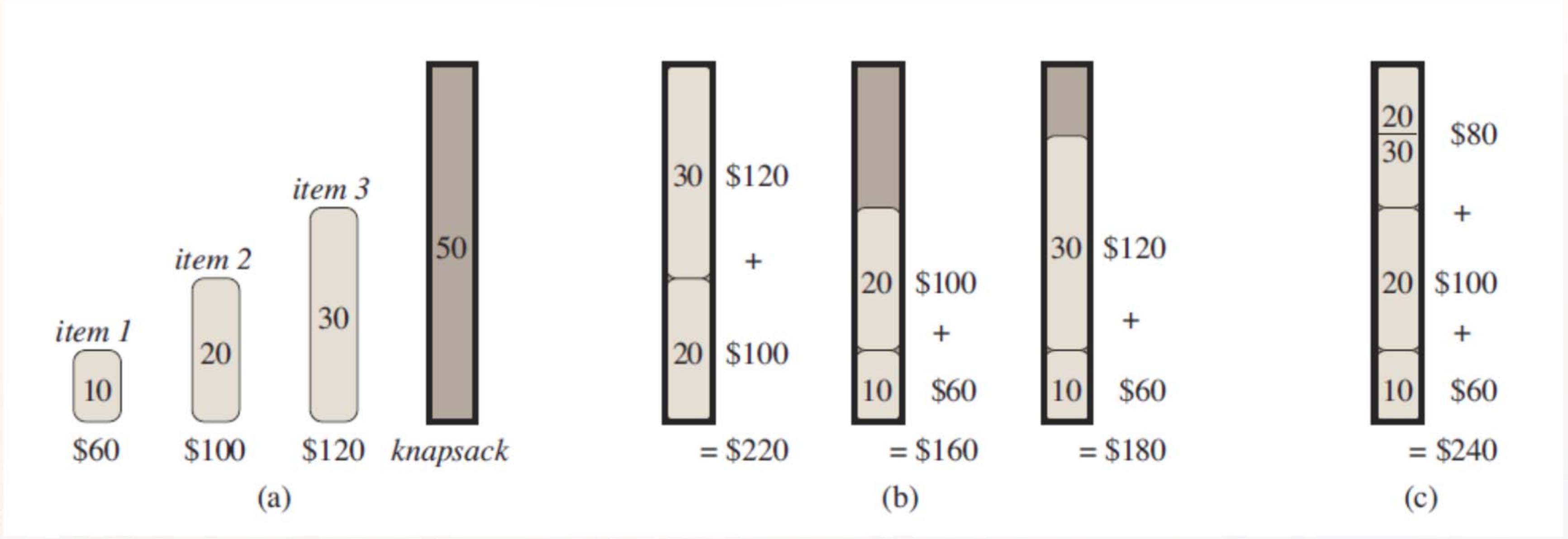
你能用集合论的语言表述这个性质的含义吗？

问题12：为什么具有最优子结构对于采用GREEDY方法解题也很重要？


例如在前面的例子中： 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！



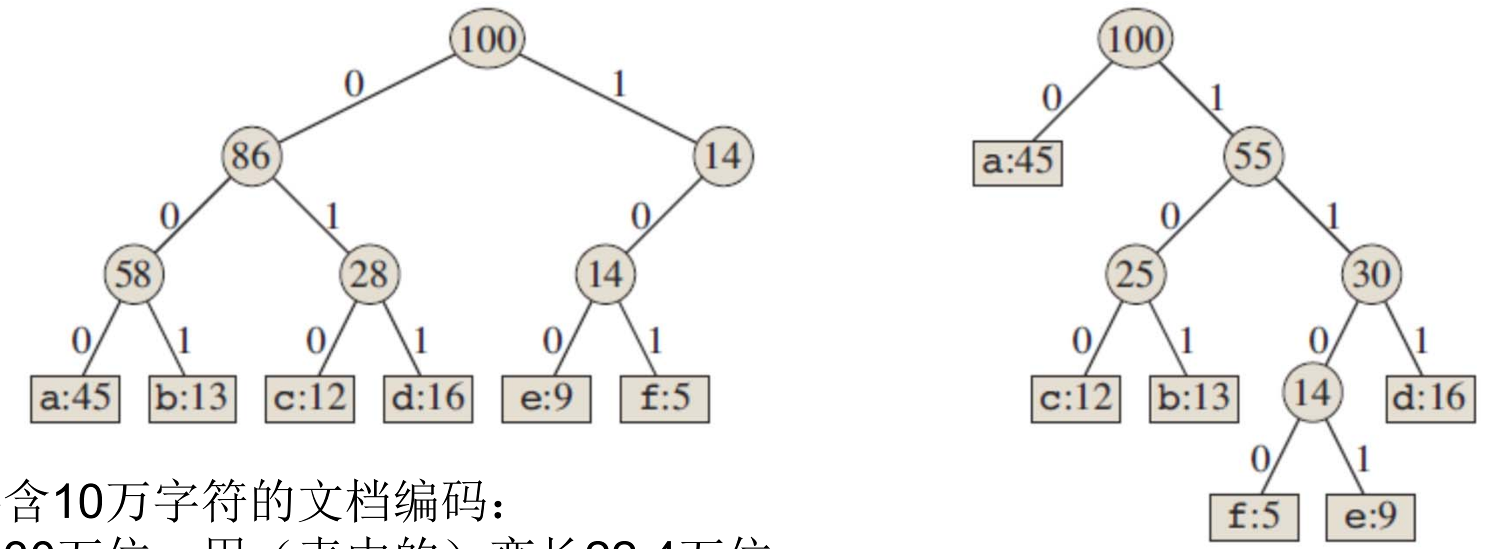
问题13：

你觉得为什么**fractional knapsack** 行，**0-1**就不行？

Part II
Huffman编码

二进编码问题与编码树

	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



如果将含10万字符的文档编码：
用定长30万位；用（表中的）变长22.4万位

问题14：

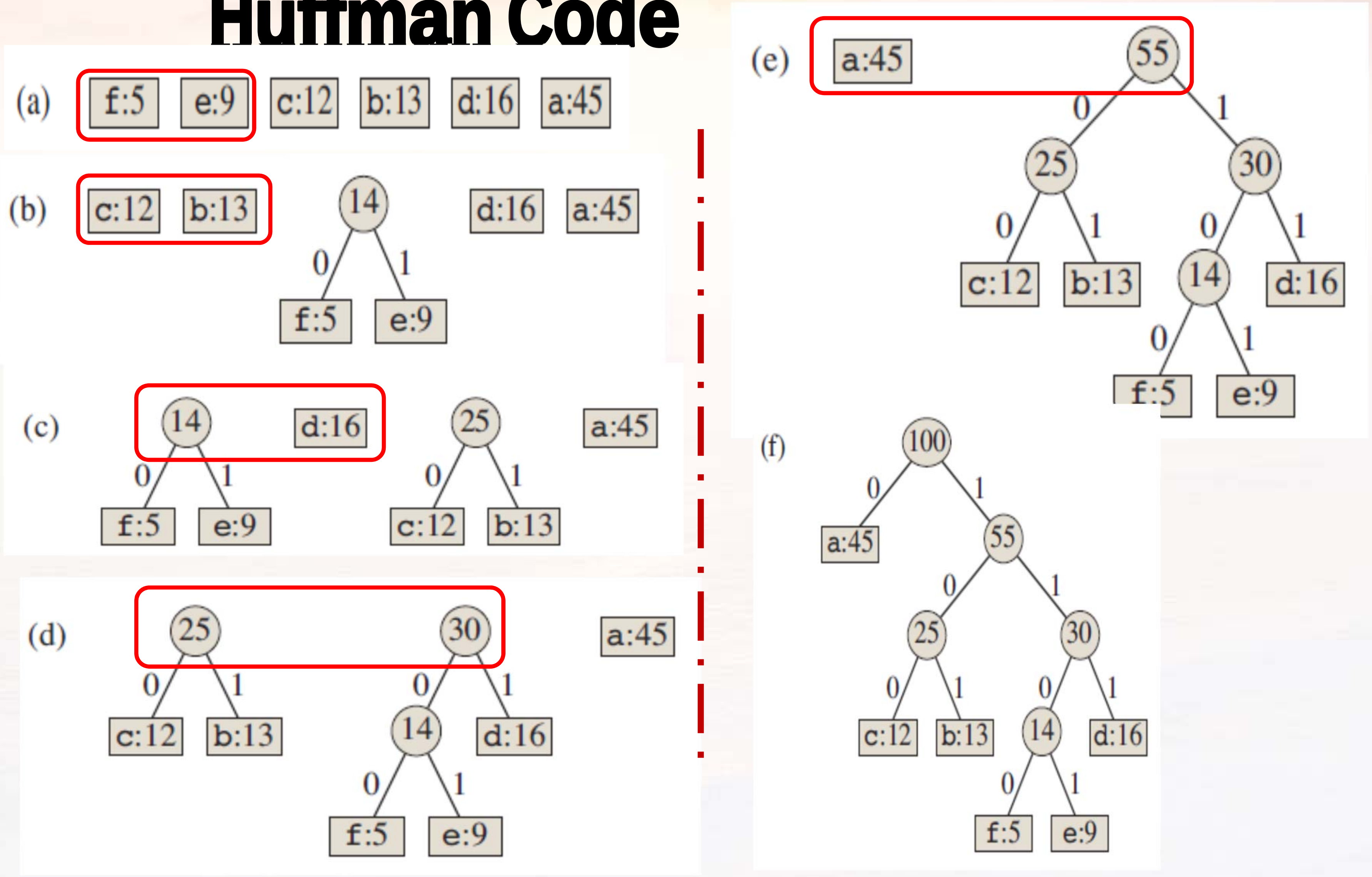
什么是前缀码？它有什么优点？

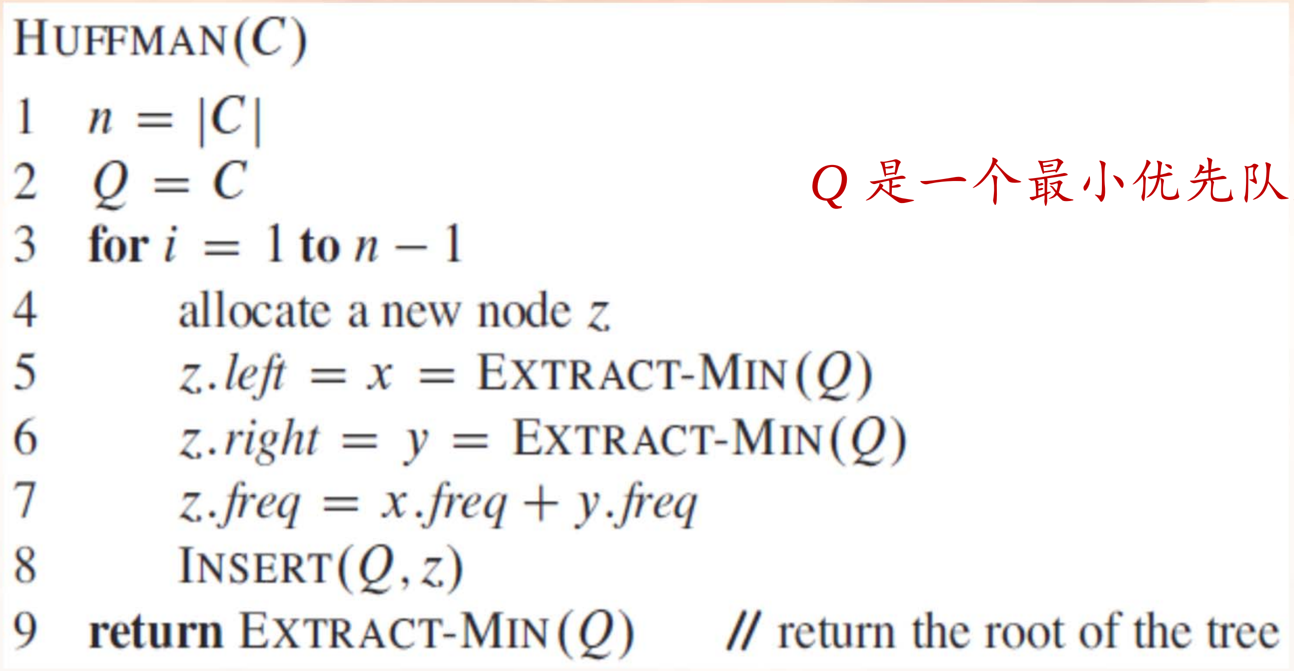
前缀码可以发挥可变长编码的优点，又可以避免使用界限符。

问题15:
“最优”是什么意思?

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

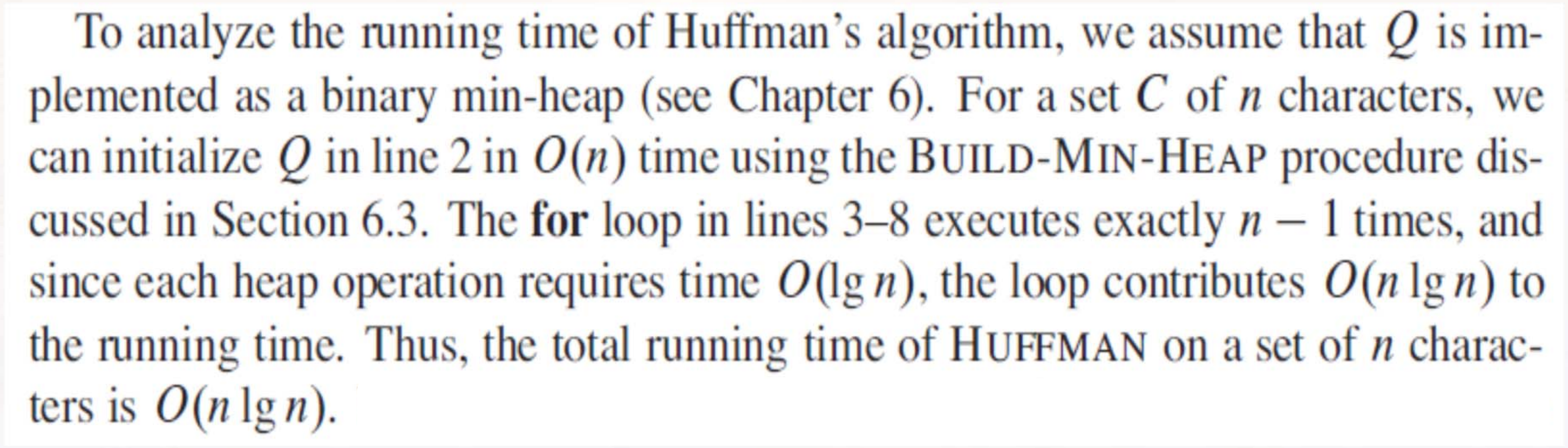
Huffman Code





```
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
```

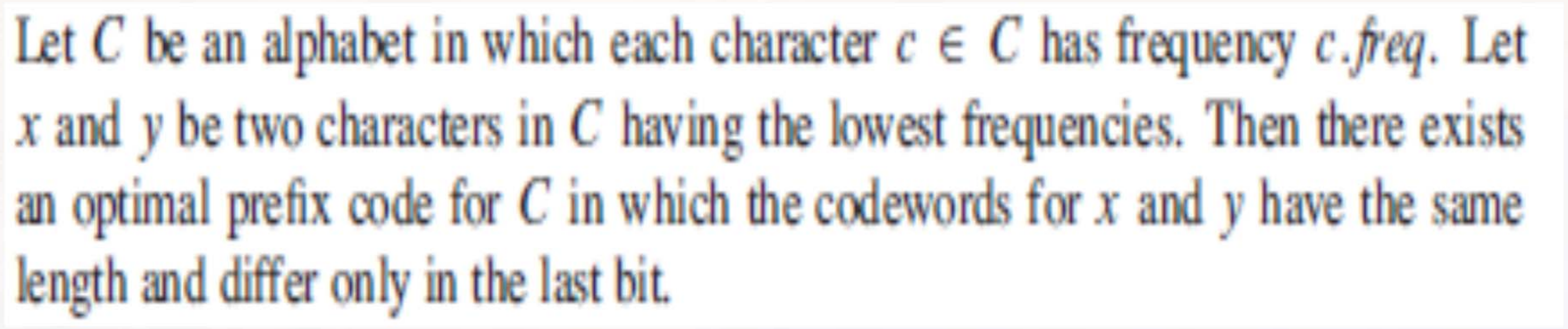
Q 是一个最小优先队列。

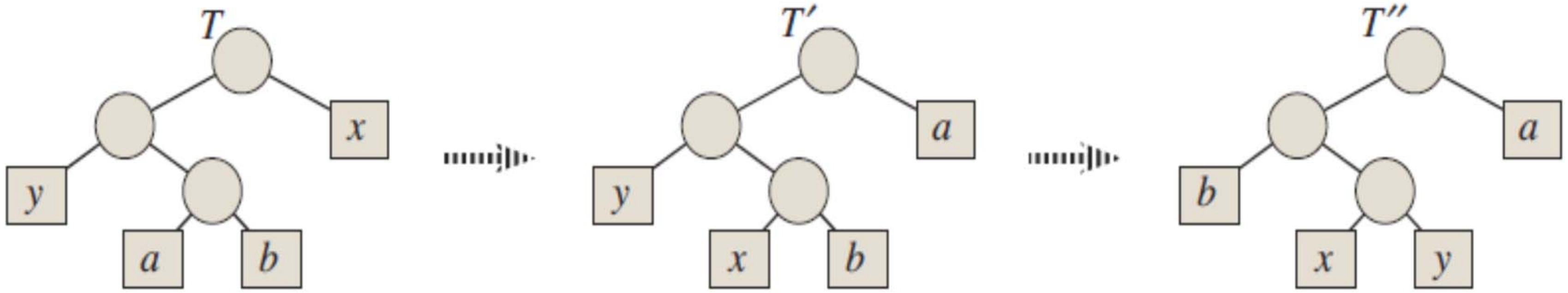


To analyze the running time of Huffman's algorithm, we assume that Q is implemented as a binary min-heap (see Chapter 6). For a set C of n characters, we can initialize Q in line 2 in $O(n)$ time using the BUILD-MIN-HEAP procedure discussed in Section 6.3. The **for** loop in lines 3–8 executes exactly $n - 1$ 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16：

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

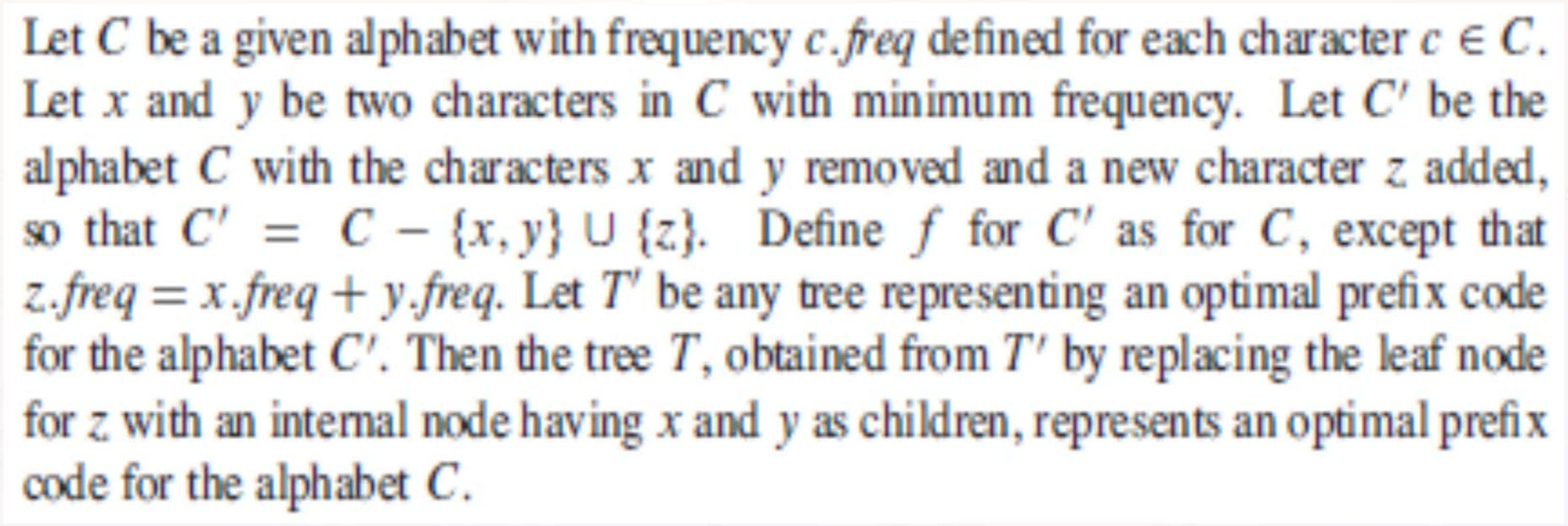

Let C be an alphabet in which each character $c \in C$ has frequency $c.freq$. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.



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

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17:
最优前缀码问题满足optimal-substructure property，这点该如何表述？


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： $B(T) = B(T') + x.freq + y.freq$

We now prove the lemma by contradiction. 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. ■

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