Chapter 6

贪婪策略

本章介绍贪婪策略，又叫做贪心策略，用于解决一种特殊的最优化问题。这种最优化问题与动态规划一样，符合最优子结构的性质，但是更加特殊的是，它还满足贪心选择性的性质。我们在动态规划这一章中学习了，如果问题满足最优子结构的性质，那么对于一个问题来说，它的全局最优解是包含了子问题的局部最优解。这是一种自顶向下的思考方式，但是我们并不知道全局最优解是从哪个局部最优解构造而来的，因此我们要将解空间划分为若干个子空间，并求出所有子空间的最小值或者最大值，从而得到全局最优解。然而一旦问题还满足贪心选择性的性质，那么我们就可以不用去枚举每个子空间了，我们可以直接知道全局最优解是从哪一个具体的子空间构造而来，因此省去了枚举每一个子空间的麻烦，从而得到时间复杂度的提升。接下来我们将通过几个典型的问题来学习贪婪策略。

6.1 活动选择问题

我们接下来考虑这样一个问题：你作为会议室的管理人员，每天都会拿到一张活动申请的汇总表，这张汇总表上列举了所有申请举行活动的开始时间和结束时间，假设活动 A_i 的开始时间是 s_i，结束时间是 f_i，一旦活动被选中，那么该活动将在 $[S_i, f_i)$ 的时间段内在会议室举行。你希望会议室在一天内安排的活动越多越好，可是会议室在任意一个时间点最多只能安排一个活动。于是现在希望你设计一个算法，从众多申请的活动当中选择出最多的可以安排下的活动。

例如，活动申请的汇总表如表6.1所示，对于该汇总表，$\{A_3, A_9, A_{11}\}$ 是一种活动安

Table 6.1: 活动申请汇总表

活动	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}
开始时间	1	3	0	5	3	5	6	8	8	2	12
结束时间	4	5	6	7	9	9	10	11	12	14	16

排方案，因为它们互相之间不冲突，但是这并不是最多的活动安排方案。而安排方案 $\{A_1, A_4, A_8, A_{11}\}$ 是一种最优的活动安排方案，$\{A_2, A_4, A_9, A_{11}\}$ 是另一种最优的活动安排方案，因为按照这个申请表，最多只能安排 4 个活动。除此之外，没有其它方案能够安排 4 个活动了。

6.1.1 最优子结构

该问题满足最优子结构的性质: 假定存在一个最优的活动安排方案，那么中间的任意的一个连续的子方案都是对应连续的时间段的最优方案。否则我们就有更多的活动可以安排在对应的连续时间内，然后再拼接上最优方案当中这段连续时间以外的活动安排方案，就可以安排下更多的活动，这与当前的方案是最优的活动安排方案相矛盾，因此该问题满足最优子结构的性质。

6.1.2 将解空间划分为子空间以及动态规划策略

由于该问题存在最优子结构的方案，我们便可以考虑使用动态规划的策略来解决该问题。跟以往一样，我们首先将解空间划分为若干的子空间。假设我们有备选活动集合 $\{A_1, A_2, A_3, \cdots, A_n\}$，所有的解决方案无非可以分为以下几种: (1) 第一个活动安排为 A_1、(2) 第一个活动安排为 A_2、(3) 第一个活动安排为 A_3、$\cdots$、(n) 第一个活动安排为 A_n。那么最优的活动安排方案就是这些子空间当中安排活动数量最多的那个。

接下来我们便能够写出如 BEST-ACTIVITY-CHOICE 所示的自顶向下的带备忘录的递归求解算法，其中假设 $f[i]$ 代表从时间 i 开始到今天结束能够安排的最多的活动个数，初始条件下全部初始化为-1，然后调用 BEST-ACTIVITY-CHOICE$(A, 0)$ 即可求解出答案。

```
BEST-ACTIVITY-CHOICE(A, time)
 1: if f[time] ≠ −1 then
 2:     return f[time]
 3: f[time] = 0
 4: for i = 1 to n do
 5:     if A[i].start ≥ time then
 6:         count = BEST-ACTIVITY(A, A[i].final) + 1
 7:         if count > f[i] then
 8:             f[time] = count
 9:             S[time] = i
10: return f[time]
```

PRINT-BEST-ACTIVITY-CHOICE(S, time)

```
1: if S[time] ≠ 0 then
2:     Print(S[time])
3:     PRINT-BEST-ACTIVITY-CHOICE(S, A[S[time]].final)
```

f 数组的每个元素最多填一次，而每次确定 $f[time]$ 的值时，都要遍历一遍 A 数组，因此该算法的时间复杂度是 $\Theta(n^2)$。

6.1.3 贪心选择性

在动态规划策略中，我们将解空间划分为不同的子空间，然后求取每个子空间的最优解，最后再取最优解的最优解作为整个问题的最优解。而接下来我们将论证该问题存在的另一个性质，即贪心选择性。该问题满足的贪心选择性是：我们在求解整个解空间的最优解时，不需要求解每个子空间的最优解，而是可以只选择结束时间最早的活动作为第一个活动的那个子空间，这个子空间的最优解就是全局的最优解。

我们接下来证明贪心选择性的正确性。假设我们有一个活动安排的最优解，其中第一个活动不是结束时间最早的那个，那么我们将该活动替换为结束时间最早的活动，首先这个活动与剩余的活动均不冲突，其次得到的新的安排方案的活动个数与之前的最优解的活动安排方案中的活动个数相同，因此我们组合的新的安排方案也是一个最优的活动安排方案，因此贪心选择性是正确的。

接下来由最优子结构的性质，我们就可以递归的在子问题上求解最优的安排方案，而子问题上的最优安排方案也可以由选择可以选择的最早结束时间的活动加上剩余的时间的最优方案组合而成，这样一来我们就可以得到一个贪婪策略的解决方案：我们首先将活动按照结束时间由小到大排列，然后不断选择不冲突的活动即可，该算法如 BEST-ACTIVITY-CHOICE-GREEDY 所示。

BEST-ACTIVITY-CHOICE-GREEDY(A)

```
1: Sort(A by A[i].final)
2: last = 0
3: for i = 1 to n do
4:     if A[i].start ≥ last then
5:         Print(i)
6:         last = A[i].final
```

接下来我们证明算法的正确性，我们将通过数学归纳法来证明。第一步，确定谓词，

- $P(n)$：算法的选出 n 个活动时，一定存在一个最优解决方案包含算法选出的这 n 个活动。

第二步，证明基本情况 $P(0)$，这个是成立的，因为任何最优方案都是空方案的超集。第

三步，证明一般情况 $P(n) \Rightarrow P(n+1)$，我们选取与前 n 个活动兼容的活动集合 S'，由最优子结构的性质，总的最优方案一定包含在这个集合上选择最优的子方案，此时 S' 上选择的第一个活动如果不是结束时间最早的那个，我们一定可以将其替换为结束时间最早的那个，而不产生冲突，同时保证方案仍然是最优的，所以 $P(n+1)$ 成立，因此算法是正确的。该算法的时间复杂度是 $\Theta(n \lg n)$，因为它首先需要排序 A 数组，然后只对数组做一次遍历。

6.1.4 应用贪婪策略的基本步骤

接下来我们来总结应用贪婪策略的基本步骤。首先我们要先观察一个最优化问题是不是满足最优子结构的性质，如果这个最优化问题满足最优子结构的性质，那么我们便可以考虑构造一个动态规划策略来解决它。在应用动态规划策略过程中，通常需要枚举每个子空间的最优解，此时我们可以试图寻找问题是否具有贪心选择性的性质，如果问题满足贪心选择性的性质，那么我们就不需要枚举每个子空间的最优解了，而是直接找到问题的最优解是来自于哪个子空间，进一步构造贪婪策略的解决方案。

6.2 排队接水问题

接下来我们考虑排队接水问题: 有 n 个人在一个水龙头前排队接水，假如每个人接水的时间为 T_i，请你找出这 n 个人排队的一种顺序，使得 n 个人的平均等待时间最小。

例如: 我们有 10 个人，$P_1 \sim P_{10}$，每个人的接水的时间 T_i 如表6.2所示。如果我们按照接水顺序 $P_3, P_2, P_7, P_8, P_1, P_4, P_9, P_6, P_{10}, P_5$ 来接水，那么这 10 个人的平均等待时间为 291.90，是最小的。除此之外，平均等待时间都要更大。

Table 6.2: 接水时间对应关系

人	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
T_i	56	12	1	99	1000	234	33	55	99	812

6.2.1 最优子结构性质

首先这个问题满足最优子结构的性质。对于最优方案而言，当中的任意一个连续的子方案都得是最优的，否则我们可以把这一部分重新排列得到一个更优的方案。例如: 我们的最优方案是 $P_{i_1}, P_{i_2}, P_{i_3}, P_{i_4}, \cdots, P_{i_n}$，其中，对于第 i_k 个人而言，他贡献的平均等待时间为

$$\frac{1}{n}\sum_{j=1}^{k} T_{i_j}$$

也就是说，如果我们重新排列 $P_{i_s}, P_{i_{s+1}}, P_{i_{s+2}}, \cdots, P_{i_t}$，对于 $j < s$ 的人和 $j > t$ 的人而言，贡献的平均等待时间均不变，于是如果这部分人的平均等待时间的总和变小了，整体的平均等待时间就更小了，这与当前方案是最优方案是矛盾的。因此该问题满足最优子结构的性质。

6.2.2 贪心选择性

我们可以根据最优子结构的性质写出一个集合上的动态规划策略，这种策略的复杂度非常高，这里我们就不展开说了。接下来我们论证该问题还满足的贪心选择性的性质。假设对于一个方案，其中有两个相邻的人 P_{i_k} 和 $P_{i_{k+1}}$，且 $T_{i_k} > T_{i_{k+1}}$。那么对于 P_{i_k}，他贡献的平均等待时间为:

$$Q_1 = \frac{1}{n}(\sum_{j=1}^{k-1} T_{i_j} + T_{i_k})$$

对于 $P_{i_{k+1}}$，他贡献的平均等待时间为:

$$Q_1' = \frac{1}{n}(\sum_{j=1}^{k-1} T_{i_j} + T_{i_k} + T_{i_{k+1}})$$

此时如果我们将 P_{i_k} 和 $P_{i_{k+1}}$ 交换，那么对于 $P_{i_{k+1}}$ 而言，他贡献的平均等待时间为:

$$Q_2 = \frac{1}{n}(\sum_{j=1}^{k-1} T_{i_j} + T_{i_{k+1}})$$

此时对于 P_{i_k}，他贡献的平均等待时间为:

$$Q_2' = \frac{1}{n}(\sum_{j=1}^{k-1} T_{i_j} + T_{i_{k+1}} + T_{i_k})$$

于是我们有:

$$Q_1 + Q_1' - Q_2 - Q_2' = T_{i_k} - T_{i_{k+1}} > 0$$

即 $Q_1 + Q_1' > Q_2 + Q_2'$，也就是说，交换后，总的平均等待时间变小了。那么我们由此类推，只要对于一个方案中相邻的两个人，前面的人比后面的人接水的时间要长，我们就将他们交换，就可以缩小总的平均等待时间，因此最终我们可以得到，平均等待时间最小的方案就是先选择接水时间最小的人，然后这个人接完水后，再从剩余的人里选择接水时间最小的人，以此类推。

6.2.3 应用贪婪策略的算法

由上面的分析，我们可以将接水的人按照接水所需的时间从小到大排序，然后按照这个顺序接水即可。代码这里就不写了，时间复杂度为 $\Theta(n \lg n)$。

6.3 霍夫曼编码

接下来我们研究霍夫曼编码问题。霍夫曼编码是一种最短前缀码的编码方案，所谓前缀码是指没有任何码字是其它码字的前缀。使用霍夫曼编码可以有效压缩信息。

假定我们希望压缩一个 10 万个字符的数据文件，表6.3给出了文件中所出现的字符和它们出现的次数，也就是说，文件中只出现了 6 个不同字符，其中 a 字符出现了 45000 次。

Table 6.3: 字符编码

字符	a	b	c	d	e	f
出现次数（千次）	45	13	12	16	9	5
定长编码	000	001	010	011	100	101
变长编码	0	101	100	111	1101	1100

我们可以看见, 如果我们使用定长编码, 那么上面的文件需要 $100,000 \times 3 = 300,000$ 位。然而如果我们使用变长编码, 我们只需要用 $1 \times 45,000 + (13,000 + 12,000 + 16,000) \times 3 + (9,000 + 5,000) \times 4 = 224,000$ 位即可。霍夫曼编码就是希望能够找到一种变长编码方案，使得编码后的文件总的位数最少。

6.3.1 霍夫曼编码方案

我们首先先介绍霍夫曼编码的方案，然后再去证明它的正确性。对于霍夫曼编码, 则要构建一个霍夫曼编码树，霍夫曼编码树的构建方式是: 不断选择出现次数最少的两个字母，然后将它们合并在一起作为一棵子树，两个字母分别是这棵子树的叶子结点, 这两个字母的码字长度是相同的, 因此这棵子树的根节点可以视为两个叶子结点综合在一起，即根节点的出现次数等于两棵叶子节点之和。然后我们去掉这两棵叶子结点的字母，加入根节点，持续不断的递归选取两个最少的，直到最终只剩下一个节点。此时我们便得到了一棵霍夫曼编码树，然后我们再从根节点开始走到每一个字母，每一个字母都是这棵霍夫曼编码树的叶子结点，我们从根节点走到一个叶子节点的时候，往左走我们就给这个字母的码字添加一个 0，往右走我们就给这个字母的码字添加一个 1。由此我们便得到了霍夫曼编码表。该编码过程由图6.1所示。

上面的编码过程由算法 HUFFMAN 所示，该算法接收每个字母和它的出现次数的对应表 C，Q 是一个最小堆，它的键是字母的出现次数。

HUFFMAN(C)

1: $n = |C|$
2: $Q = C$
3: for $i = 1$ to $n - 1$ do

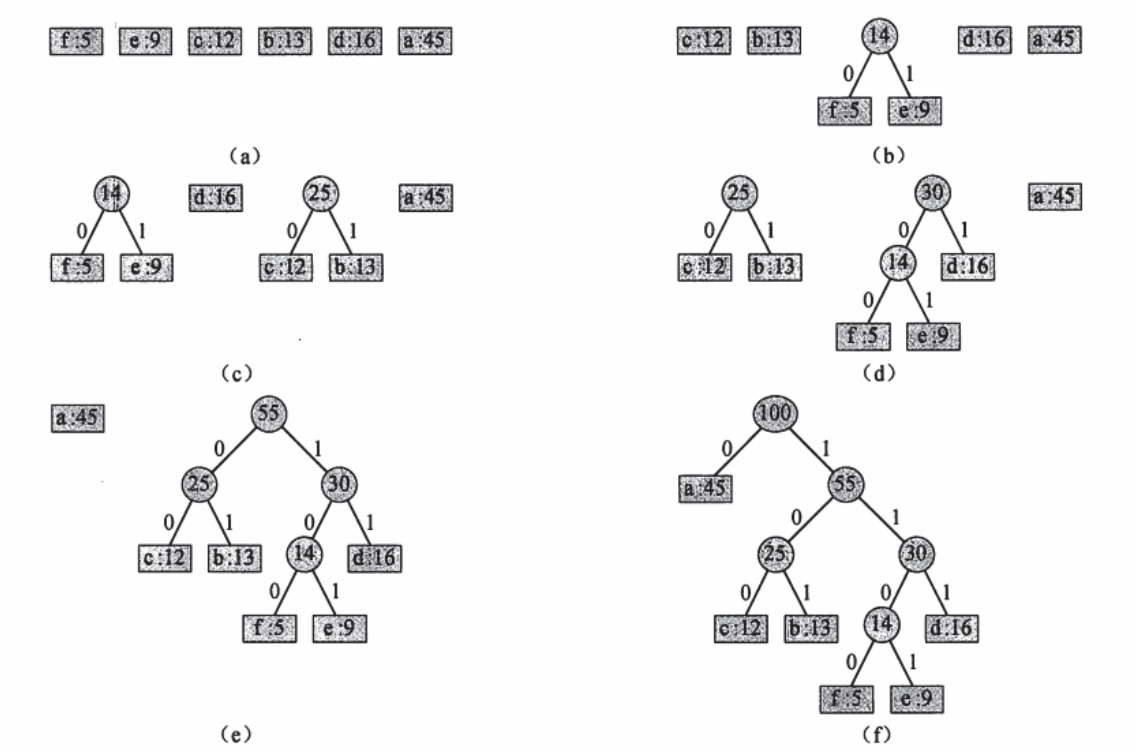

Figure 6.1: 霍夫曼编码过程

4: allocate a new node z
5: $z.left = x$ = EXTRACT-MIN(Q)
6: $z.right = y$ = EXTRACT-MIN(Q)
7: $z.count = x.count + y.count$
8: INSERT(Q, z)
9: return EXTRACT-MIN(Q)

该算法的时间复杂度是 $\Theta(n \lg n)$，接下来我们证明该问题满足的最优子结构性质和贪心的选择性质。与以往不同的是，我们先证明贪心选择性质，再证明贪心选择性质下的最优子结构的性质，这样更加简单一点。

6.3.2 贪心选择性质

该问题满足贪心选择性质。假设有一棵编码树 T，我们用 $B(T)$ 表示用这棵编码树编码文件的总的位数，$d_T(x)$ 表示字母 x 在这棵编码树中的深度，于是我们有:

$$B(T) = \sum_{c \in C} d_T(c) \times c.count$$

该问题的贪心选择性质为: 出现次数最少的两个字母需要在编码树的最深的位置，它们两个的码字长度相同，只有最后一位是不同。

假设我们有另一棵编码树 T'，这一棵编码树的最深的两个节点是 a 和 b，它们不是出现次数最少的两个字母，设字母表中出现最少的次数的两个字母是 x 和 y，那么我们将 T' 中的 a 替换为 x，得到一棵新的编码树 T''，那么我们有:

$$\begin{aligned} B(T') - B(T'') &= d_{T'}(x) \cdot x.count + d_{T'}(a) \cdot a.count - d_{T''}(x) \cdot x.count - d_{T''}(a) \cdot a.count \\ &= d_{T'}(x) \cdot x.count + d_{T'}(a) \cdot a.count - d_{T'}(a) \cdot x.count - d_{T'}(x) \cdot a.count \\ &= (d_{T'}(x) - d_{T'}(a))(x.count - a.count) \\ &\geq 0 \end{aligned}$$

即 $B(T'') \leq B(T')$，类似地，如果我们再将 T'' 中的 b 替换为 y，再得到一棵新的编码树 T'''，那么我们还有 $B(T''') \leq B(T'')$。于是，最优的编码树一定是最深的两个叶子结点是出现次数最少的两个字母。

6.3.3 最优子结构性质

接下来我们来证明该问题符合最优子结构的性质，令 C 为给定的一个字母表，其中每个字符 $c \in C$ 的出现的次数为 $c.count$，令 x 和 y 是 C 中出现次数最低的两个字符，令 C' 为 C 去掉 x 和 y，加入一个新字符 z 后得到的一个新的字母表，即 $C' = C - \{x, y\} \bigcup \{z\}$。同样我们为 C' 中的字母也定义 $c.count$，它与 C 的唯一区别是 $z.count = x.count + y.count$。那么若 T' 为 C' 的最优编码树，那么将 T' 中的 z 节点替换为以 x 和 y 为孩子节点的内部节点，构成一棵新的编码树 T，那么编码树 T 是 C 的最优编码树。

首先我们先用 $B(T')$ 来刻画 $B(T)$，我们有:

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

于是我们有: $B(T) = B(T') + x.count + y.count$，或者 $B(T') = B(T) - x.count - y.count$。

接下来我们就可以用反证法来证明最优子结构的性质了，如果 $B(T)$ 不是 C 的最优编码树，那么就有一棵更优的编码树 T''，有 $B(T'') < B(T)$，由贪心选择性的性质，这里面最深的两个叶子节点一定是 x 和 y，我们将 x 和 y 替换为 z，得到 T'''，它是 C' 上的一个编码方案，于是有:

$$B(T''') = B(T'') - x.count - y.count < B(T) - x.count - y.count = B(T')$$

这与 T' 是 C' 的最优编码树相矛盾，因此 T 一定是 C 的最优编码树。

最后，我们综合最优子结构性质和贪心选择性质，便能够证明出 HUFFMAN 算法的正确性。