第 3-2 讲: 贪心

姓名：林凡琪　　**学号：**211240042

评分：_____　　**评阅：**_____

2022 年 9 月 14 日

请独立完成作业，不得抄袭。
若得到他人帮助, 请致谢。
若参考了其它资料，请给出引用。
鼓励讨论，但需独立书写解题过程。

1　作业 (必做部分)

题目 1 (TC 16.1-2)

解答：
如果我们想象反向运行，这将变得与原始问题完全相同，因此它出于基本相同的原因产生最佳解决方案。这是贪婪的，我们在每一步都做出了最好的选择。

题目 2 (TC 16.1-3)

解答：
作为贪婪地选择最短的最优性的反例，假设我们的活动时间为 $1\,9$　$8\,11$　$10\,20$，那么，首先选择最短的，我们必须消除其他两个，如果我们选择其他两个，我们将有两个任务而不是一个。

作为贪婪地选择与剩余最少活动冲突的任务的最优性的反例，假设活动时间为 $-1\,1$　$2\,5$　$0\,3$　$0\,3$　$0\,3$　$4\,7$　$6\,9$　$8\,11$　$8\,11$　$8\,11$　$10\,12$。然后，通过这种贪婪的策略，我们首先选择 $(4,\ 7)$ $(4,\ 7)$，因为它只有两个冲突。但是，这样做意味着我们将无法选择 $-1\,1$，$2\,5$，$6\,9$，$10\,12$ 的唯一最优解

作为贪婪地选择最早开始时间的最优性的反例,假设我们的活动时间为 $1\,10$　$2\,3$　$4\,5$。如果我们选择最早的开始时间，我们将只有一个活动 $1\,10$，而最佳解决方案是选择其他两个活动。

题目 3 (TC 16.2-1)

解答：
为背包问题设 I ：设 n 为项目数，设 v_i 为第 i 项的值，设 w_i 为第 i 项的权重，设 W 为容量。假设这些项目已按 v_i/w_i 按递增顺序排序，并且 $Wgew_n$。

设 $s = s_1\ s_2\ ldots\ s_n$ 作为解决方案。贪婪算法的工作原理是分配到 $s_n = min\ w_n\ W$ ，然后继续解决子问题

$$I' = (n-1, \{v_1, v_2, \ldots, v_{n-1}\}, \{w_1, w_2, \ldots, w_{n-1}\}, W - w_n)$$

直到它达到状态 $W = 0$ 或 $n = 0$。

我们需要证明，这种策略总是给出一个最优的解决方案。我们用反证法来证明这一点。假设 I 的最优解是 $s_1, s_2, \ldots, s_n$，其中 $s_n < \min(w_n, W)$. 通过将 s_i 减小到 $max(0, W - w_n)$ 并将 s_n 增加相同的数量，我们得到了更好的解决方案。既然这是一个矛盾，那么假设一定是错误的。因此，问题具有贪婪选择属性。

题目 4 (TC 16.2-2)

解答:

假设我们知道一个特定的重量项 w 在解决方案中。然后，我们必须解决 $n-1$ 项目的子问题，最大权重 $W-w$。因此，要采用自下而上的方法，我们必须解决所有项目和可能小于 W 的重量的 0-1 背包问题。我们将构建一个 $n+1$ by $W+1$ 的值表，其中行按项目索引，列按总权重索引。(表的第一行和一列将是一个虚拟行)。

对于行 i 列 j，我们通过比较背包的总价值（包括项目 1 到 $i-1$ 和最大重量 j）和包括项目 1 到 $i\ -1$（最大重量 $j-i.weight$ 和项目 i）来决定是否有利地将项目 i 包含在背包中。为了解决这个问题，我们只需检查表的 n，WW 条目，以确定我们可以达到的最大值。要读出我们包含的项目，请从条目 n，W 开始。通常，请按以下步骤操作：如果条目 i，j 等于条目 i - 1，j，则不包括项目 i，然后检查条目 i - 1，j 接下来。如果条目 i，j 不等于条目 i − 1，j，包括项目 i 并检查条目 i − 1，j − i.weight 接下来。有关表的构造，请参阅下面的算法：

```
0-1-KNAPSACK(n, W)

    Initialize an (n + 1) by (W + 1) table K
        for i = 1 to n
            K[i, 0] = 0
        for j = 1 to W
            K[0, j] = 0
        for i = 1 to n
            for j = 1 to W
                if j < i.weight
                    K[i, j] = K[i - 1, j]
                else
                    K[i, j] = max(K[i - 1, j], K[i - 1, j - i.weight] + i.value)
```

题目 5 (TC 16.3-2)

解答:

让 T 成为未满的二叉树。T 表示由字母 C 中的字符组成的文件的二进制前缀代码，其中 $cinC$，$f\ c$ 是文件中 c 的出现次数。树 T 的成本或编码中的位数为 $sum_{cinC} d_T\ c\ cdotf\ c$，其中 $d_T\ c$ 是树 T 中字符 c 的深度。

设 N 是只有一个子节点的最大深度节点。如果 N 是 T 的根，则可以删除 N，并将每个节点的深度减少一，从而产生一棵代表相同字母表的树，并且成本更低。这意味着原始代码不是最佳的。

否则，设 M 为 N 的父级，设 T_1 为 N 的（可能不存在的）同级，并让 T_2 成为 N 的子节点的子树。如果 T_1 为空，请重复该过程。我们有一个成本较低的新前缀代码，因此原始代码不是最佳的。

题目 6 (TC 16.3-5)

解答:

Little formal-mathematical note here: We are required to prove existence of an optimal code with some property. Therefore we are required also to show, that some optimal code exists. It is trivial in this case, since we know that the code produced by a run of Huffman's algorithm produce one such code for us. However, it is good to be aware of this. Proving just the implication "if a code is optimal then it has the desired property" doesn't suffice.

OK, now we are ready to prove the already mentioned implication "if a code is optimal then it has the desired property". Main idea of our proof is that if the code violates desired property, then we find two symbols which violate the property and 'fix the code'. For the formal proof we go as follows.

Suppose that we have an alphabet $C = a_1, \ldots, a_n$ where the characters are written in monotonically decreasing order, i.e. $a_1.freq \geq a_2.freq \geq \ldots \geq a_n$. Let us consider an optimal code B for C. Let us denote the codeword for the character $c \in C$ in the code B by $cw_B(c)$. W.l.o.g. we can assume that for any ii such that $a_i.freq = a_{i+1}$. This assumption can be made since for any $a_i.freq = a_{i+1}.freq$ for which $|cw(a_i)| > |cw(a_{i+1})|$ we can simply swap codewords for a_i and a_{i+1} and obtain a code with desired property and the same cost as is the cost of B. We prove that B has the desired property, i.e., its codeword lengths are monotonically increasing.

We proceed by contradiction. If lengths of the codewords are not monotonically increasing, then there exist an index i such that $|cw_B(a_i)| > |cw_B(a_{i+1})|$. Using our assumptions on C and BB we get that $a_i.freq > a_{i+1}$.Define new code B' for C such that for a_j such that $j \neq i$ and $j \neq i+1$ we keep $cw_{B'}(a_j) = cw_B(a_j)$ and we swap codewords for a_i and a_{j+1}, i.e. we set $cw_{B'}(a_i) = cw_B(a_{i+1})$ and $cw_{B'}(a_{i+1}) = cw_B(a_i)$. Now compare costs of the codes B and B'. It holds that

$$\begin{aligned} cost(B') &= cost(B) - (|cw_B(a_i)|(a_i.freq) + |cw_B(a_{i+1})|(a_{i+1}.freq)) \\ &\quad + (|cw_B(a_i)|(a_{i+1}.freq) + |cw_B(a_{i+1})|(a_i.freq)) \\ &= cost(B) + |cw_B(a_i)|(a_{i+1}.freq - a_i.freq) + |cw_B(a_{i+1})|(a_i.freq - a_{i+1}.freq) \end{aligned}$$

For better readability now denote $a_i.freq - a_{i+1}.freq = \phi$. Since $a_i.freq > a_{i+1}.freq$, we get $\phi > 0$ and we can write

$$cost(B') = cost(B) - \phi|cw_B(a_i)| + \phi|cw_B(a_{i+1})| = cost(B) - \phi(|cw_B(a_i)| - |cw_B(a_{i+1})|)$$

Since $|cw_B(a_i)| > |cw_B(a_{i+1})|$, we get $|cw_B(a_i)| - |cw_B(a_{i+1})| > 0$. Thus $\phi(|cw_B(a_i)| - |cw_B(a_{i+1})|) > 0$ which imply $cost(B') < cost(B)$. Therefore the code B is not optimal, a contradiction.

Therefore, we conclude that codeword lengths of BB are monotonically increasing and the proof is complete.

Note: For those not familiar with mathematical parlance, w.l.o.g means without loss of generality.

题目 7 (TC 16.3-8)

解答:
对于任何 2 个字符，它们的频率之和超过任何其他字符的频率，因此最初霍夫曼编码会产生 128 棵树，每棵树有 2 个叶节点。在下一阶段，没有一个内部节点的标签是其他节点的两倍以上，所以我们的设置与以前相同。继续以这种方式，霍夫曼编码构建了一个高度为 $lg256 = 8$ 的完整二叉树，这并不比普通的 8 位长度代码更有效。

---

2 作业 (选做部分)

题目 1 (TC Problem 16-1 (Coin Changing))

解答:

---

3 Open Topics

Open Topics 1 (Ternary Disk)
Trimedia Disks Inc. has developed "ternary" hard disks. Each cell on a disk can now store values 0, 1, or 2 (instead of just 0 or 1).

To take advantage of this new technology, provide a modified Huffman algorithm for constructing an optimal variable-length prefix-free code for characters from an alphabet of size n, where the characters occur with known frequencies $f_1, f_2, \cdots, f_n$.

Prove that your algorithm is correct.

Open Topics 2 (Intervals)
Let X be a set of n intervals on the real line. A subset of intervals $Y \subseteq X$ is called a ***full path*** if the intervals in Y cover the intervals in X, that is, any real value that is contained in some interval in X is also contained in some interval in Y . The *size* of the full path is the number of intervals it contains.

Describe and analyze a greedy algorithm to compute the smallest full path of X as quickly as possible. Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$, representing the left and right endpoints of the intervals in X. Don't forget to prove your greedy algorithm is correct!

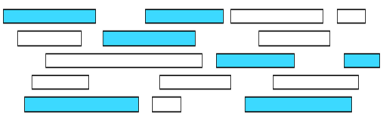

图 1: 蓝色的 7 个区间组成一个完整路径 (full path)

4 反馈