算法分析与设计基础 第九周作业

徐浩博 软件02 2020010108

Problem 1

首先我们分析Huffman编码的过程，对于一开始的256个字符，记它们的频率为$freq_1, \cdots, freq_{256}$. 由最高频率低于最低频率2倍得，$freq_i + freq_j \geq 2MIN\{freq\} > MAX\{freq\} \geq freq_k, \forall i, j, k$，因此初始这256个节点进行Huffman算法时，不存在任何已被合并的节点和未被合并的节点进行合并；对应到Huffman树上，这256个节点均是Huffman树T的叶子节点. 下面考虑它们合成的128个节点$freq'_1, \cdots, freq'_{128}$. 我们依然有$freq'_i + freq'_j \geq 4MIN\{freq\} > 2MAX\{freq\} \geq freq'_k, \forall i', j', k'$，因此这128个节点也不会和二次合并的节点进行合并，对应到Huffman树上，表现为256个叶子节点的父节点两个两个构成兄弟. 依次类推，最终合并剩1个节点时，整个Huffman编码树是一个满二叉树，树高为$log_2 256 = 8$，因此每个字符对应的前缀码的码长仍是8，这与固定编码的8位固定长度是一样的，所以对于这个数据文件来说，Huffman编码并不比8位固定长度编码更高效.

Problem 2

a. 贪心算法如下：

```
GREEDY_ALGORITHM(amount):
    let array A be the result
    count = 0
    while(amount > 25):
        A[count] = 25
        count += 1
        amount -= 25
    while(amount > 10):
        A[count] = 10
        count += 1
        amount -= 10
    while(amount > 5):
        A[count] = 5
        count += 1
        amount -= 5
    while(amount > 1):
        A[count] = 1
        count += 1
        amount -= 1
    return A
```

采用**归纳法**证明贪心算法能够取到最优策略.

首先，n=1时，取贪心算法要求我们取1张1元硬币，则显然是最优策略.

假设$n < k$时贪心算法也正确，n=k时，我们分情况讨论：

i) $k \leq 4$，显然贪心地取k张1元硬币是最优策略.

ii) $5 \leq k \leq 9$，只有两种策略："k张1元"或者"1张5元+(k-5)张1元"，后者明显更优，而贪心算法给出的也是后一种策略，因此贪心能取到最优策略.

iii) $10 \leq k < 25$，在最优策略中，1元的数量不应超过4，否则用1个五元替换5个一元能够获得更优解. 类似地，5元数量不应该超过1. 在这种情况下，1元和5元能够构成的最大面额为1元×4+5元×1=9元，要构成k元必须至少要一张10元，因此最优策略中肯定有10元. 我们先贪心地取10元，根据归纳假设，剩下$k-10$元也可以贪心地取得到最优策略，因此这k元都可以贪心地取到最优策略.

iv)$k \geq 25$. 在最优策略中，1元的数量不应超过4，否则用1个五元替换5个一元能够获得更优解. 类似地，5元数量不应该超过1，10元数量不应超过4（2个25元可以替代5个10元）. 最优策略中没有25元，则1元和5元组合成的最大面额为9元（5元最多1张，1元最多4张），需要组成$k \geq 25$的情况仍需要至少2张10元；而至少2张10元意味着最优策略中没有5元（否则2张10元和1张5元可以用1张25元来替代）；没有5元，而1元最多4张，则意味着10元至少需要3张才能使总额超过25元，而3张10元可以用1张25元和1张5元代替，则没有25元的策略不是最优策略，产生矛盾. 所以，$k > 25$时最优策略必须有一张25元，先取这张25元，由归纳假设，剩下$k-25$元也可以贪心地取，因此，这k元用贪心算法可以得到最优策略.

由归纳法知，贪心算法求出的是最优策略.

b. 对于每个需要找零的面值n，假设k是满足$c^k \leq n$的最大整数.

首先，在最优方案中，$c^0, \cdots, c^{k-1}$每个面额的硬币最多有$c-1$个. 这一点可以通过反证法证明：假设最优方案中c^i超过c-1个，那么c个c^i面额硬币可以用1个c^{i+1}面额硬币代替，而且这样替代能够使得方案中用的硬币数更少，与最优方案矛盾. 因此，最优方案中每个面额的硬币最多$c-1$个.

其次，我们要说明，最优方案中$c^0, \cdots, c^{k-1}$组合获得的总额不超过c^k. 我们假设$c^0, \cdots, c^{k-1}$都取上限$c-1$个，总和为$(c-1)(c^0 + \cdots + c^{k-1}) = (c-1)\frac{c^k-1}{c-1} = c^k - 1 < c^k$，因此我们要找零n元且方案最优，必须得取$c^k$面额的硬币.

下面我们用归纳法，归纳需要找零的面额n证明贪心策略是正确的. n=1时显然取能取的最大面额$c^0 = 1$是最优策略. 假设$n \leq t-1$时贪心均能取到最优策略，那么n=t时，由上述讨论，最优策略必须含有2^k. 由于$t - 2^k \leq t-1$，由归纳假设，$t-2^k$用贪心法可以取到最优策略，而t的最优策略即为$t-2^k$的最优策略加上2^k（如果有比$t-2^k$的最优策略加上2^k更优的取t策略，那么t策略中除去2^k，得到的策略比$t-2^k$的贪心策略更优，与归纳假设矛盾）；因此最优策略可以视为先贪心地取到2^k，再贪心地取剩下的$t-2^k$，也是贪心策略. 因此n=t时，贪心也能取到最优解.

综上，贪心是正确的.

c. 设一组硬币面额为1,4,6，则8元按照贪心算法的取法为“6元×1+1元×2”，共3个硬币. 而实际最优策略为“4元×2”，共2个硬币. 这说明了贪心并不能保证总能取到最优解.

d. 这是一道完全背包问题，可以采用dp方法来做，递推式为a[i] = min(a[i], a[i-c[j]] + 1)，其中a[i]表示i元最少可以用几个硬币凑出，c[j]表示给定的第j种硬币的面额. 为了使得每个金额可以多次取到，因此i从

小到大进行循环. 伪代码如下，seq[i]是一个可重复集合，表示凑够i元的硬币取法：

```
DP_ALGORITHM(n):
    let elements in a[i] be INF
    a[0] = 0
    seq[0] = {}
    for j from 0 to k - 1:
        for i from c[j] to n:
            if(a[i] > a[i - c[j]] + 1):
                a[i] = a[i - c[j]] + 1
                seq[i] = seq[i - c[j]]
                insert c[j] into seq[i]
    return seq[n]
```

很明显，两重循环，算法的时间复杂度为O(nk).