Homework8

刘喆骐 2020013163 探微化01

16.3-8

假设所有字符出现的频率从小到大为$f_1, f_2, ..., f_{256}, 2f1 > f_{256}$.那么对于任意的i,j,k属于[1,256]，有$f_i + f_j \geq 2f_1 > f_{256} > f_k$，也就是说，任意两元素之和大于第三个元素。那么，按照霍夫曼编码的要求，必然会得到如下的子树:$(f_1, f_2), (f_3, f_4), ...(f_{255}, f_{256})$。记这些子树的代价为$f'_1, f'_2, f'_3 ... f'_{128}$，则$2f'_1 = 2f_1 + 2f_2 > 2f_1 + 2f_1 = 4f_1 > 2f_{256} > f_{255} + f_{256} = f'_{128}$。故和上述过程相同，对于任意的i,j,k属于[1,128]，有$f'_i + f'_j \geq 2f'_1 > f'_{128} > f'_k$，得到如下子树：$(f'_1, f'_2), (f'_3, f'_4), ...(f'_{127}, f'_{128})$。由此类推，最终必然会得到一个满二叉树，其叶子节点的字节长度均为8，和正常编码相同。

16.1

a.
算法伪代码如下

```
value=[25,10,5,1]
Give_change(n)
    plan=[]
    while(n>0)
        for i in value
            if i<=n
                plan.insert[i]
                n-=i
                break
    return plan
```

显然，由于有1美分，故贪心算法必然能够找到问题的一个可行解。对于贪心得到的可行解，尝试进行如下变换至无法操作：
1.若面值1的数量不小于5，则将5枚面值1替换为1枚面值5；2.若面值10的数量不少于3，则将3枚面值10替换为1枚面值25和1枚面值5；3.若面值5的数量不少于2，则将2枚面值5替换为1枚面值10；4.若面值10的数量恰为2且面值5的数量恰为1，则将这3枚替换为1枚面值25。

上述操作中的任何一种都使硬币的数目严格递减，而仍为可行解。最后得到的情况中，必有面值10数量小于3，面值5数量小于2，面值1数量小于4，且面值10有2枚与面值5有1枚不能同时成立。

假设另有最优解所需的硬币数量小于等于贪心算法的结果，则将两者按硬币面值从大到小排列，

考察两序列第一个不同点（必定存在，否则若最优序列结束了，而贪心算法还剩余一些硬币，则两者的总和不同，矛盾），设从该不同点到末尾，贪心算法和最优解的序列长度分别为m和n(m>n)，面值和分别为s_1和s_2。由于贪心算法每次取的是剩下面值中最大可能的，故在该不同点必有贪心算法的面值大于等于最优解的面值。

若此处贪心算法取值为25，则$s_1 \geq 25+1\times(m-1)=24+m\geq n+24>10\times 2+(n-1)\times 1\geq s_2$。

若贪心算法取值为10，则$s_1\geq 10+1\times(m-1)=9+m\geq 9+n>5+(n-1)\times 1\geq s_2$

若贪心算法取值为5，则$s_1\geq 5+(m-1)\times 1=4+m\geq 4+n>n=s_2$.

无论如何取值，都有$s_1>s_2$，和s_1,s_2都是可行解矛盾。故贪心得到的是最优解。

b. 选硬币的方法和a中相同。

显然，除最大面值外，其他硬币的数量都小于c，不然就可以使用面值更大的硬币代替。两者按硬币面值从大到小排列，考察两序列第一个不同点，此处硬币面值为c_l设从该不同点到末尾，贪心算法和最优解的序列长度分别为m和n(m>n)，面值和分别为s_1和s_2。$s_2\leq(c-1)\sum_{i=0}^{l-1}c_i^l=c^l-1<s_1$，矛盾。故贪心得到最优解。

c. 例如硬币面值为5，4，1，需要8美分。按照贪心算法，会选择5+1+1+1，而最好为选择4+4.

d. 构造如下的递推式：

$d[i]=1+min\ \ d[i-p_k]$,d是总找零个数，p[]是硬币价值的集合。这样，算法由双重循环构成，外层循环n次，遍历0-目标找零值；内层循环k次，遍历所有硬币的面值。时间复杂度为$O(nk)$。

伪代码如下:

```
give_change(value,res)
    min_num[0]=0;
    min_plan[0]=[[]]
    for i =1:res:
        best=1+min_num[i-value[0]]
        k=0
        for j in value
            if i>=j and best>1+min_num[i-j]
                best=1+min_num[i-j]
                k=j
        for plan in min_plan[i-k]
            min_plan[i-k].append(plan+j)
    return min_plan[res]
```