Problem Solving Homework(Week 13)

161180162 Xu Zhiming

May 24, 2017

TC

16.1-2 In this algorithm, every step we make a choice and get a subproblem. So it's a greedy algorithm. Write the pseudocode of it.

```
1 GREEDY–ACTIVITY–SELECTIOR–REVISED(s, f)
2 //Prelimenary: the activities are sorted using start time
3 n=s.length
4 k=n
5 for i from n−1 down to 1
6         if f[i]<=f[k]
7                 A.push(a[i])
8                 k=i
9 return A
```

The proof of the correctness:

Proof. Consider any nonempty subproblem S^k, and let a_m be an activity in S_k with the last start time. Next I will prove that a_m is included in some maximum-size of subset of mutually compatible activities of S_k. 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 last start time. If $a_i = a_j$, it's done. If $a_i \neq a_j$, 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_r^k are disjoint, which follows because the activities in $A - k$ are disjoint, a_j is the last activity in A_k to start and $s_m >= s_j$. 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. □

16.1-3

Example for case 1:
s 1 4 7 11
f 6 8 10 15
The set S in textbook can be an example for case 2 and 3

16.2-1

We need to prove local greedy choices will eventually lead to a global optimal solution.

Proof. In this problem, since the material we take can be fractional. That's to say, we can take as much as we want, partial or whole. Then we can assign a unit price which is price divided by weight to it. It's obvious that we shall take the material of highest unit price as much as possible. If not, we choose another material of a lower unit price, for that weight it takes, it's not optimal(Since we could have obtained a larger total price). So the local optimal choice will lead to a global optimal choice, the greedy choice property holds. □

16.2-2

```
KNAPSACK(w[1..n],v[1..n],limit)
let DP[0..limit] be a new array
for i from 0 to limit
        DP[i]=0
for i from 1 to i
        for j from limit down to w[i]
                DP[j]=max(DP[j],DP[j-w[i]]+v[j])
return DP[limit]
```

16.3-2

Proof. Obviously, if a node has only one child, then it can't be a keyword but an ordinary node. Then we can delete it and move all its decedents up 1 level, shortening the code length of them by 1. Therefore, a tree that not full can't correspond to an optimal prefix tree. □

16.3-5

Proof. Assum we have k elements, their frequencies are $f_1 \leq f_2 \leq \cdots \leq f_k$, and one kind of optimal prefix code makes them stay in depth $d_1, d_2, \ldots, d_k$. Then we have $W(k) = \sum_{i=1}^{k} f_i \times d_i$ is a minimum. Suppose the depths are monotonically increasing, $W^{'}(k) = \sum_{i=1}^{k} f_i \times d_i$, where $d_1 \geq d_2 \geq \cdots \geq d_k$. According to sequence inequality, $W^{'}(k) \leq W(k)$. Thus we are done with the proof. □

16.3-8

Proof. Alike the proof of Lemma 16.2,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} c.freq \cdot d_T(c) - \sum_{c \in C} c.freq \cdot d_{T'}(c) \\ &= (c.freq - c.freq)(d_T(c) - d_{T'}(c)) \\ &= 0 \end{aligned}$$

Therefore, Huffman code isn't more efficient than ordinary code. □

16-1

a.

```
COIN-CHANGE(n)
let q,d,n,p all be 0     //Initializing,numbers of every kind of coins
remain=n
while(remain>0)
        if(remain>15)
                q++
                remain-=15
        else if(remain>10)
                d++
                remain-=10
        else if(remain>5)
                n++
                remain-=5
        else
                p+=remain
```

```
16                      remain=0
17  return q,d,n,p
```

Proof. Optimal Substructure: In an optimal solution, for a certain amount of money, suppose we need at least n coins. If there is another way to change the money and requires less coins, then we can substitute this way for the former ways and yield a better solution, which contradicts our assumption. Therefore, the optimal substructure is satisfied.
Greedy Choice: If we keep choosing coins with the highest price since the remaining amount permits, it's obvious that we can minimize the total number of coins used. □

b.

Proof. We can view the available coin denominations $c^0, c^1, \ldots, c^k$ as the radixes of c-radix numbers. It's trivial that any number can be denoted by the radix. So we can use these coins to change any amount of money. Alike the proof in problem a, this problem also has the two properties. So it can be solved by a greedy algorithm. □

c.
1 cent, 8 cents and 10 cents. 17 cents' case.
d.

```
1  COIN-CHANGE(d[1..k])      //d is the denominations of coins
2  for(i = 1; i <= n; i++)
3          for(j = 0; d[j] <= i&&j<= k; j++)
4                  if DP[i] == 0 or DP[i-d[j]]+1 < DP[i]
5                          DP[i] = DP[i-d[j]]+1
```