CSCE-629 Homework 4

Wang, Han-Yi
wanghy917@email.tamu.edu

September 11, 2020

Exercise 16.2-2

Algorithm 1 KNAPSACK(v, w, n, W)

```
let dp[0...n, 0...W] be a new array
for i = 0 to n
    dp[i, 0] = 0
for j = 0 to W
    dp[0, j] = 0
for i = 1 to n
    for j = 1 to W
        if w[i] ≤ j
            dp[i, j] = max(dp[i − 1, j], dp[i − 1, j − w[i]] + v[i])
        else
            dp[i, j] = dp[i − 1, j]
return dp[n, W]
```

The 0-1 knapsack problem is described on page 425 of the textbook and summarized here. There are n items, v_i and w_i represent the value and weight of the i_{th} item, and $i = 1 \dots n$. The thief can carry at most W pounds, and each item can either take zero or one. The goal is to take as valuable a load as possible.

The problem can be solved by the above KNAPSACK algorithm. Let $dp[i, j]$ represent the maximum value when $(0 \dots i)_{th}$ item is included, where $i = 0$ means no item, and j represent the weight limit. Then, we could write the below recursive function, and use dynamic programming to solve the problem. The value of $dp[n, W]$ is the final output.

$$dp[i,j] = \begin{cases} \max\left\{dp[i-1,j], dp\left[i-1, j-w_i\right] + v_i\right\} & \text{if } w_i \le j \\ dp[i-1,j] & \text{if } w_i > j \end{cases} \tag{1}$$

Exercise 16.2-5

Algorithm 2 SMALLEST-INTERVAL(X)

let n be the length of X
if $n == 0$
  return 0
sort(X) $\triangleright$ $x[1] \leq x[2] ... \leq x[n]$
let ans be an empty array
for $i = 1$ **to** n
  if $ans.length == 0$ **or** $x[i] > ans[ans.length][1]$
    append $\{[x[i], x[i]+1]\}$ onto ans
return ans

The above SMALLEST-INTERVAL algorithm can find the smallest set of a unit-length closed interval. The input is the given set $\{x_1, x_2, \ldots, x_n\}$, and we let all the index of the arrays in the algorithm start from 1.

Proof:

1. When $X.length = 0$ the algorithm returns 0
2. When $X.length = 1$ the algorithm returns $[x_1, x_1 + 1]$, which covers the only point, and it is an optimal choice
3. Suppose $X.length = n - 1$ is correct, we add $[x_{n-1}, x_{n-1} + 1]$ to ans, then we consider an additional right point x_n
   - If $x_n > x_{n-1} + 1$, the algorithm will add a new interval $[x_n, x_n + 1]$ which is an optimal choice, because the interval can cover x_n, and it can cover most points on the rights of x_n
   - If $x_n \leq x_{n-1}$, x_n will be covered by $[x_{n-1}, x_{n-1} + 1]$
4. Since the algorithm performs the optimal choice at each step, the algorithm is correct

Exercise 16.3-2

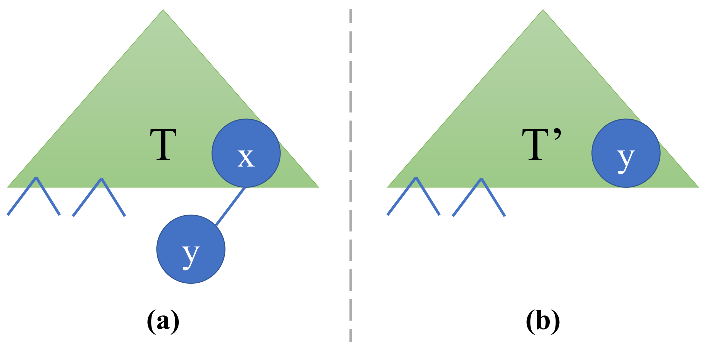

Figure 1: **(a)** T is a non-full binary tree, and its node x, only have one child node y. **(b)** T' is generated by replacing node x with its child node y.

Assuming T is a tree, and it has optimal prefix code in a non-full binary tree. There is at least one node in T, which has only one child. Let the node x has only one child node y as in Figure 1 (a).

By replacing x with y, we can get the tree T' as in Figure 1 (b). If y is a leaf node, the code-word of y is decreased by one. If y is not a leaf node, all leaf nodes of y's descendant decrease by one, and the difference in cost between T and T' is

$$\begin{aligned} & B(T) - B\left(T'\right) \\ = & \sum_{c \in C} c.freq \cdot d_T(c) - \sum_{c \in C} c.freq \cdot d_{T'}(c) \\ = & \sum_{c \in S} c.freq \cdot d_T(c) - \sum_{c \in S} c.freq \cdot d_{T'}(c) \geq 0 \end{aligned} \tag{2}$$

, where S represent the leaf node set of $y's$ descendant. The $\geq$ holds because the depth of all nodes in S decrease by one in T'. The cost could be reduced which is contradict to the assumption. Therefore, T cannot be a non-full binary tree if it is an optimal solution.

Exercise 16.3-3

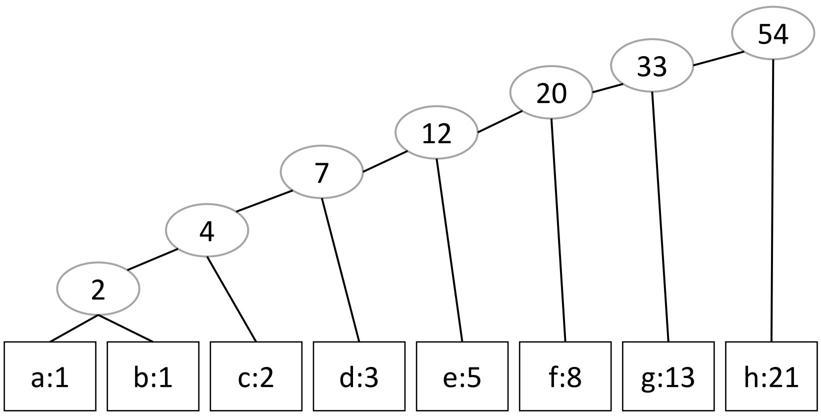

Figure 2: Final tree for the optimal code or the first 8 Fibonacci numbers.

We can use Huffman's algorithm to get the optimal Huffman code, and the final tree is shown in Figure 2. The code is as below:
$a = 0000000, b = 0000001, c = 000001, d = 00001, e = 0001, f = 001, g = 01, h = 1.$

In the general case, the optimal code of the first n Fibonacci numbers is:
$f_1 = \overbrace{000 \cdots 000}^{n-1}, f_2 = \overbrace{000 \cdots 00}^{n-2}1, f_3 = \overbrace{000 \cdots 00}^{n-3}1, \ldots f_{n-2} = 001, f_{n-1} = 01, f_n = 1$