Assignment 6

Jixuan Ruan PB20000188

November 5, 2022

1. We can simply change the algorithm to the below one:

```
function BottomUpCutRod(p, n)
    r[0..n] = newarray
    for all j in range(n) do
        q = −∞
        for all i in range(j) do
            q = max(q, p_i − c + r[j − i])
        end for
        r[j] = q
    end for
end function
```

2. If we assume that c(m,n) represents the total number of solutions to put the m apples into n baskets, then we can see:

$$c(m,n) = c(m,n-1) + c(m-n,n)$$

$c(m,n-1)$ represents the situation when at least one basket is empty and $c(m-n,n)$ represents the situation when all the baskets are full. So we can design the dynamic programming solution as below:

```
function CountApples(m, n)
    for all i in range(m) do
        c[i][1] = 1
    end for
    for all i in range(n) do
        c[1][i] = 1
    end for
    for all i in rang(2, m) do
        for all j in range(2, n) do
            if i > j then
                c(i, j) = c(i, j − 1) + c(i − j, j)
            end if
            if i <= j then
                c(i, j) = c(i, i)
            end if
        end for
    end for
end function
```

3. 1) We can choose one unit-length interval with the start point of x_1, and delete the concluded points.
   2) Refresh the order number of the left points.
   3) Go back to step 1 and repeat until the point set is empty.
   Proof:
   We can prove that the unit-length interval starting with the first point must be in one of the optimum solution.
   If this interval doesn't exist in the optimum solution, then we can substitute it for the first interval in this optimum solution. And it is easy to see that the solution is still the optimum.

4. It is like to find the last m nodes to form a Huffman tree.
   1)Perform the process to produce a Huffman tree until the number of the left nodes is equal to m.
   2)Distribute the leaves of these m nodes to the m machines. And we can see that the optimum time should equal to the largest weight in these m nodes.

5.
```
function MoneyBack(money)
    if money > 25 then
        give the customer 25 cents
        MoneyBack(money - 25)
    end if
    if 10 < money < 25 then
        give the customer 10 cents
        MoneyBack(money - 10)
    end if
    if 5 < money < 10 then
        give the customer 5 cents
        MoneyBack(money - 5)
    end if
    if 0 < money < 5 then
        give the customer 1 cents
        MoneyBack(money - 1)
    end if
end function
```

   Proof:
   When $money > 25$, if you don't give the customer 25 cents then you have to give him much more cash like the 10 cents, 5 cents or 1 cent. And the situation is the quite the same in other three condition.

6. 1) Sort the objects according to its weight/volume.
   2)
```
function Knapsack(int p[],int w,int n)
    q = -∞
    r[0] = 0
    for all i in range(w) do
        for all j in range(n) do
            q = max(q, p[j] + r[i - p[j]])
        end for
        r[i] = q
```

```
    end for
end function
```