Hw 6

Page210：15.1-3

Page215：15.2-3，15.2-6

15.1

Page210：15.1-3

15.1-3 我们对钢条切割问题进行一点修改，除了切割下的钢条段具有不同价格 p_i 外，每次切割还要付出固定的成本 c。这样，切割方案的收益就等于钢条段的价格之和减去切割的成本。设计一个动态规划算法解决修改后的钢条切割问题。

$$r_n = \begin{cases} 0 & \text{n=0} \\ \max\begin{pmatrix} p_i + r_{n-i} - c & i = 1..n-1 \\ p_n & i = n \end{pmatrix} & n \neq 0' \end{cases}$$ 注意i=n时，不切割，不减c。

原始DP算法：

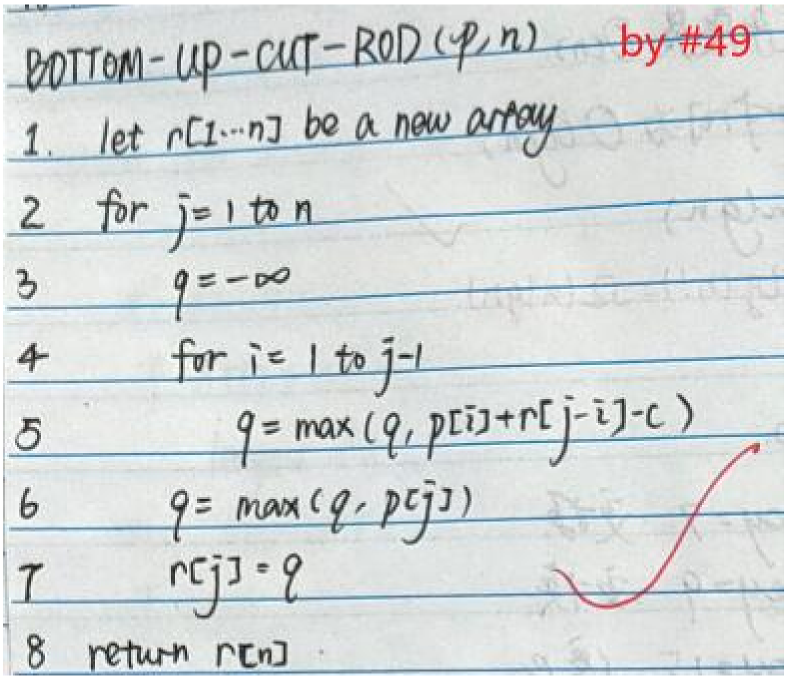
```
BOTTOM-UP-CUT-ROD(p,n)
1  let r[0..n]be a new array
2  r[0]=0
3  for j = 1 to n
4       q=-∞
5       for i = 1 to j
6            q=max(q,p[i]+r[j-i])
7       r[j]=q
8  return r[n]
```

方法一：(效率高)

4-6行改为：

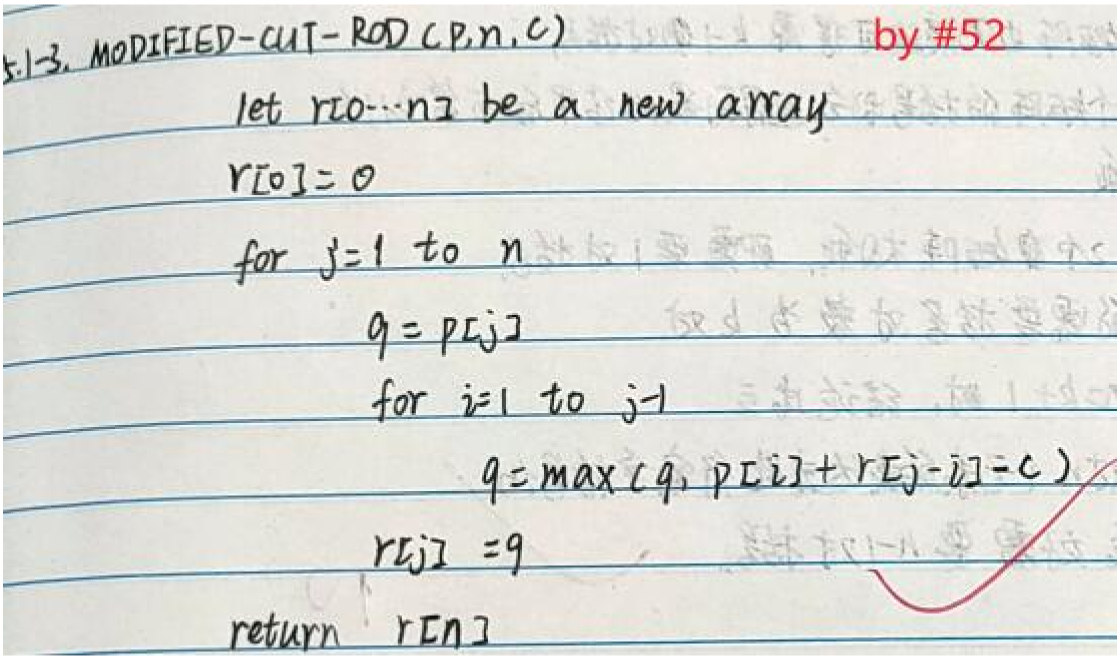
```
# 先忽略成本，求最大收益q
for i=1 to j-1:
    q = max(q, p[i]+r[j-i])
# q-c即为算上成本的最大收益，与p[j]（即不切割收益）进行比较
if p[j]>q-c:
    r[j]=p[j]
else:
    r[j]=q-c
```

方法二：



```
BOTTOM-UP-CUT-ROD(p, n)
let r[1…n] be a new array
for j=1 to n
    q=-∞
    for i=1 to j-1
        q=max(q, p[i]+r[j-i]-c)
    q=max(q, p[j])
    r[j]=q
return r[n]
```


方法三：



```
15.1-3. MODIFIED-CUT-ROD(p,n,c)
    let r[0…n] be a new array
    r[0]=0
    for j=1 to n
        q=p[j]
        for i=1 to j-1
            q=max(q, p[i]+r[j-i]-c)
        r[j]=q
    return r[n]
```


方法一效率是最高的，方法二和方法三是根据上面递推式来的，但是直接根据递推式减c肯定是不对的，未考虑j=i时不切割的情况。

15.2

Page215：15.2-3，15.2-6

15.2-3 用代入法证明递归公式(15.6)的结果为 $\Omega(2^n)$。

$$P(n)=\begin{cases}1 & \text{如果 } n=1\\ \sum_{k=1}^{n-1}P(k)P(n-k) & \text{如果 } n\geqslant 2\end{cases} \tag{15.6}$$

$$
\begin{aligned}
&Suppose P(n) \geq c2^n \\
P(n) &\geq \sum_{k=1}^{n-1} c2^k * c2^{n-k} \\
&= \sum_{k=1}^{n-1} c^2 2^n \\
&= c^2(n-1)2^n \\
&\geq c^2 2^n \qquad (n>1) \\
&\geq c2^n \qquad (c>1)
\end{aligned}
$$

得证。

15.2-6 证明：对 n 个元素的表达式进行完全括号化，恰好需要 $n-1$ 对括号。

We proceed by induction on the number of matrices. A single matrix has no pairs of parentheses. Assume that a full parenthesization of an n-element expression has exactly n − 1 pairs of parentheses. Given a full parenthesization of an (n+1)-element expression, there must exist some k such that we first multiply B=A1⋯Ak in some way, then multiply $C = A_{k+1} \cdots A_{n+1}$ in some way, then multiply B and C. By our induction hypothesis, we have k − 1 pairs of parentheses for the full parenthesization of B and n+1−k−1 pairs of parentheses for the full parenthesization of C. Adding these together, plus the pair of outer parentheses for the entire expression, yields k−1+n+1−k−1+1=(n+1)−1 parentheses, as desired.