Chuong Vu
HW1

15.1-3
Consider a modification of the rod-cutting problem in which, in addition to a price pi for each rod, each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a dynamic-programming algorithm to solve this modified problem.

```
MEMOIZED-CUT-ROD(P, n, c)
Allocate r[0..n]
For i = 0 to n
        r[i] = -infinite
return MEMOIZED-CUT-ROD-AUX(P, n, R, c)

MEMOIZED-CUT-ROD-AUX(P, n, R, c)
if r[n] >= 0
        return r[n]
if n == 0
        q = 0
else q = -infinite
        for i = 1 to n
                if i == n
                        q = max(q, p[n])
                else
                        q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(P,  n, R, c) – c)
r[n] = q
return q
```

To add the costs of making cut in to the code. I need to find where it actually the real cost for each piece and see that q is the price of each piece so before I store q to r[n], I need to modified the actual price after minus the cost and get the maximum out of two values. So right before store value to q, I need to subtract the cost cutting a new piece, q is already included the cost of making a cuts so I don't need to minus cost c from q in max function. Also if i equal n which mean it will be less than 1 cut (1 cut for make a rob into two piece).

```
BOTTOM-UP-CUT-ROD(p[1..n], n, c)
Allocate r[0..n]
r[0] = 0
for j  = 1 to n
        v = 0
        for i = 1 to j - 1
                v = max(v, p[i] + r[j – 1] – c)
        r[j] = v
return r
```

So I used to bottom-up-cut-rod and cost for each cut is c. So the table will update based on v = max{v, p[i] + r[j – 1] – c}. v is already included the cost of making cut so only the other part that have not include the cost so I need to subtract that to cost c to get the correct value after cut. And the loop will run from 1 to j – 1 which is it use to take care case of no cut or at least only 1 cut in the case of two pieces.

Problem 2 (20 points). Consider a variation of the longest subsequence (LCS) problem, which we call a String Similarity Score (SSS). For two strings s1 and s2, each matching character in a subsequence alignment gives a score of 3, while each character in s1 or s2 that is not matched gives a penalty score of -1. For example, ACGTC and CATGCTC have an SSS of $4 \times 3 - 4 = 8$, since an LCS has length 4 and there are one and three unmatched characters in s1 and s2, respectively. Modify the original LCS dynamic programming algorithm to compute the SSS of two strings directly. That is, your algorithm must not compute the LCS first and then calculate the SSS from the LCS.

Since we don't compute the LCS first, based on the question we will get a penalty -1 so for the first row and first column, there will be $x_i = x_{i-1} - 1$ (gap = -1). The table will be look like

	Y_j	A	C	G	T	C
X_i	0	-1	-2	-3	-4	-5
C	-1	-2	+2	+1	+0	-1
A	-2	+2	+1	+0	-1	-2
T	-3	+1	+0	-1	+3	+2
G	-4	+0	-1	+3	+2	+1
C	-5	-1	+3	+2	+1	+5
T	-6	-2	+2	+1	+5	+4
C	-7	-3	+1	+0	+4	+8

And the answer will be at c[m,n]

```
LCS-LENGTH(X,Y)
m = X.length
n = Y.length
let and c[0..m, 0..n] be new tables
c[0,0] = 0
for i = 1 to m
        c[i,0] = c[i-1,0] - 1
for j = 0 to n
        c[0, j] = c[0,j-1] -1
for i = 1 to m
        for j = 1 to n
        if Xi == Yj
                c[i,j] = c[i – 1, j – 1] + 3
        else if c[i – 1, j] >= c[i, j – 1]
                c[i,j] = c[i – 1, j] – 1
        else
                c[i,j] = c[i, j – 1] – 1
return c[m,n]
```

Problem 3 (20 points). Exercise 15.3-3 (page 389)
Consider a variant of the matrix-chain multiplication problem in which the goal is to parenthesize the sequence of matrices so as to maximize, rather than minimize, the number of scalar multiplications. Does this problem exhibit optimal substructure?

Yes, it does exhibit optimal substructure. To optimal parenthesize, we split the product into two subchain and in each subchain we can compute the maximum of multiplication so in the end we can get a total multiplication of all chain.

Problem 4 (40 points). Problem 15-2 (page 405)
Longest palindrome subsequence

A *palindrome* is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, civic, racecar, and aibohphobia (fear of palindromes).

Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. For example, given the input character, your algorithm should return carac. What is the running time of your algorithm?

Give a recursive formula, pseudo code for computing the length of the longest palindrome, correctness justification, and the running time of your algorithm. Also give the pseudo code on how to construct the longest palindrome.

We first need to optimal the substructure of the LPS before we can go any further.
Let S[1..n] be a input string of length n and L[1..n] be a LPS output.

- If S[1] == S[n] then L[1, n] = L[2, n-1] + 2
- Else L[1,n] = MAX(L[2, n], L[1, n – 1])

Recursive formula: For string S, we say L[i,j] is the sequence of LPS of S. There are few cases to handle:

1) Every single character in palindrom length is 1, mean if i == j ➔ L[i,j] = 1
2) If first and last character are not the same
   if S[i] != S[j] ➔ L[i,j] = MAX(L[i, j-1], L[i+1,j])
3) If the first and last character are the same
   if S[i] == S[j] ➔ L[i,j] = L[i+1, j-1] + 2
4) If there are only two characters and both are same, this to help fill the table when L[i+1, j-1] is NULL
   if j = i+1 and s[i] == s[j] then L[i,j] = 2, else L[i,j] = 1

Pseudocode for LPS(S)

```
LPS(S)
n = S.length
Allocate L[1..n,1..n]

// for single length = 1
for i = 1 to n
        L[i,i] = 1

for i = 2 to i < n
        for j = 1 to n - i
                y = j + i
                if S[j] == S[y] && i == 2       //only two characters and both are same
                        L[j,y] = 2
                elseif S[j] == S[y]      // first and last character are the same
                        L[j,y] = L[j+1,y-1] + 2
                else     // If first and last character are not the same
                        L[j,y] = MAX(L[j,y-1], L[j+1,y])
return L[0..n]
```

T(n) = O(n^2) because using two for loop for fill out the table

Pseudocode for construct the longest palindrome