1.

T1 和 T2 是两棵有序树，其中每个结点都有一个标签，考虑树上的三种操作，删除一个子树、插入一个子树和更改一个结点的标签，请设计一个算法，求得从T1 变化到 T2 所需要的最少操作数，要求写出递推方程，程序伪代码并分析时间复杂性。

题目补充: 删除子树和插入子数的代价等于子树节点的个数

解： 全遍历的lisp伪代码如下

```
(defun diffForeast (F1 F2)
    (cond ((isNull? F1) (count F2)) ;如果F1是空森林，需要插入整个F2
          ((isNull? F2) (count F1)) ;如果F2是空森林，需要删除整个F1
          (else (min
                (+ (count (car F2)) (diffForeast F1 (cdr F2))) ;直接插入F2的最左树
                (+ (count (car F1)) (diffForeast (cdr F1) F2)) ;删除F1的最左树
                (+ (diffForeast (cdr F1) (cdr F2)) ;两个森林除去最左子树剩下的森林
                   (diffForeast (car F1) (car F2)) ;两个森林的最左子树的子森林
                   (if (eq? (car F1) (car F2)) ;如果最左子树标签不同，需要一次修改标签操作
                       1
                       0))))))
```

可以看到这种递归算法存在大量的重复计算，递推方程为:

```
C(null, F2) = count(F2)
C(F1, null) = count(F1)
C(F1, F2) = min{count(F2[0])+C(F1,F2[1..]),
               count(F1[0])+C(F1[1..],F2),
               C(F1[0],F2[0])+C(F1[1..],F2[1..])+(F1[0]==F2?1:0)}
```

为了消除重复计算，可以采取记忆的方法，即如下代码:

```
memo = new HashTable<*Foreast, *Foreast>

diffForeastMemorized(F1, F2):
    if not memo[F1, F2]
        memo[F1, F2] = diffForeast(F1, F2)
    return memo[F1, F2]
```

其中diffForeast方法里的递归调用改为调用diffForeastMemorized，这样可以通过储存每次的计算结果来避免重复子问题的多次计算。

假设两棵树的高度为n，平均每层的子树个数为m，每个子森林的计算时都会计算m^2种组合，由于下层的计算会被记忆无需重复求值，因此总共的计算量为$\sum_{i=1}^{n} m^2 * m^{i-1} = O(m^{n+1})$

2.

给定三个字符串 A, B 和 C，设计一个多项式时间动态规划算法，求出它们的最长公共子序列，要求写出递归方程，算法伪代码并分析算法复杂性

递推方程：

```
C[i,j,k] = 0 if i*j*k = 0
C[i,j,k] = C[i-1,j-1,k-1] + 1 if i=j=k>0
C[i,j,k] = max(C[i-1,j,k]
               C[i,j-1,k]
               C[i,j,k-1])
```

伪代码：

```
LCS-Length(X,Y,Z):
for i in X
    C[i,0,0] = 0
for j in Y
    C[0,j,0] = 0
for k in Z
    C[0,0,k] = 0
for i in X
    for j in Y
        for k in Z
            if X[i] == Y[j] == Z[k]
                C[i,j,k] = C[i-1,j-1,k-1] + 1
                B[i,j,k] = '0'
            else
                m = max(C[i-1,j,k],C[i,j-1,k],C[i,j,k-1])
                C[i,j,k] = m
                if C[i-1,j,k] == m
                    B[i-1,j,k] = '1'
                else if C[i,j-1,k] == m
                    B[i,j-1,k] = '2'
                else
                    B[i,j,k-1] = '3'
return (C,B)
```

```
print-LCS(X,B,i,j,k)
if i*j*k = 0
    return
switch B[i,j,k]
    case '0':
        print-LCS(X,B,i-1,j-1,k-1)
        print(X[i])
        break
    case '1':
        print-LCS(X,B,i-1,j,k)
        break
    case '2':
        print-LCS(X,B,i,j-1,k)
        break
    case '3':
        print-LCS(C,B,i,j,k-1)
        break
```

复杂度分析：

计算代价包括三重循环，因此复杂度为O(XYZ)，其中X,Y,Z分别表示三个字符串的长度
构造最优解为O(X+Y+Z)
总时间复杂度为O(XYZ)
使用了三维数组B和C，空间复杂度为O(XYZ)

3.

考虑字符串变换操作，增加一个字符，删除一个字符以及修改一个字符，设增加字符操作的代价为 i, 删除字符操作代价为 d, 修改字符的代价为 m，给定两个字符串 S1 和 S2，设计一个动态规划算法，求得从 S1 变换到 S2 代价最小的变换序列，要求写出递推方程，程序伪代码并分析算法复杂性。

设X[0..i]变换到Y[0..j]的最优变换序列为C[i,j]

递推方程：

```
C[i,j] = C[i-1,j-1] if X[i]=Y[j]
C[i,j] = min(C[i-1,j-1]+m,
             C[i,j-1] + i,
             C[i-1,j] + d)
```

伪代码：

```
calc(X,Y):
C[0,0] = 0
for i in X
    C[i,0] = i * d
for j in Y
    C[0,j] = j * i
for i in X
    for j in Y
        if X[i] == Y[j]
            C[i,j] = C[i-1,j-1]
            B[i-1,j-1] = '↖'
        else
            M = min(C[i-1,j-1]+m,C[i,j-1]+i,C[i-1,j]+d)
            C[i,j] = M
            if C[i-1,j-1]+m == M
                B[i,j] = '↖'
            else if C[i,j-1]+i == m
                B[i,j] = '←'
            else C[i-1,j]+d == m
                B[i,j] = '↑'

print(B,X,Y,i,j):
if i==0 and j==0
    return
switch B[i,j]
    case '↖':
        print(B,X,Y,i-1,j-1)
        if X[i] != Y[j]
            printf("modify %d %c", i, X[i])
        break
    case '↑':
        print(B,X,Y,i-1,j)
        printf("delete %d", i)
        break
    case '←':
        print(B,X,Y,i,j-1)
        printf("insert %d %c", i, X[i])
        break
```

复杂度分析：

计算代价为两重循环，复杂度为O(XY)，其中XY是两个数组的长度
打印最优解为O(X+Y)
因此总时间复杂度为O(XY)
使用了C,B两个二维数组，空间复杂度为O(XY)

4.

将一根木棒折成若干份，每折一次的代价是当前这段木棒的长度，总代价是折这根木棒直到满足要求所需要的所有操作的代价。例如，将一根长度为10的木棒折成四段，长度分别为2, 2, 3, 3，如果先折成长度为2和8的两段，再将长度为8的折成长度为2和6的两段，最后将长度为6的折成长度为3的两段，这些操作的代价是10+8+6=24；如果先折成长度为4和6的两段，在分别将长度为4的折成长度为2的两段、长度为6的折成长度为3的两段，则这些操作的代价是10+4+6=20，比上一种方案更好一些。该问题的输入是木棒的长度 L 和一些整数 c1,…,cn, 要求将木棒折成长度为c1, …, cn 的 n 段且操作代价最小，请设计动态规划算法解决该问题。

题目补充：折断方法需要按照顺序，即最终折断的木棒序列不能交换

设$C_{i,j}$是$c_i...c_j$段拼接成整个木棒的最小代价

递推方程：

$$C_{i,j} = 0 \quad if\ i = j$$

$$C_{i,j} = min_{i \le k < j}\{C_{i,k} + C_{k+1,j} + \sum_{x=i}^{k} c_x + \sum_{x=k+1}^{j} c_x\}$$

其中规定i=j时$\sum_{x=i}^{j} c_x = 0$(因为只有折断的木棒才计算长度)

伪代码：

```
cost(X,i,j):
    if j - i == 1
        return 0
    else
        sum(X[i..j])

calc(X):
for i in 1:X.length
    C[i,i] = 0
for i in 2:X.length
    for j in i:X.length-i+1
        l = j+i-1
        C[j,l] = ∞
        for k in j:l-1
            q = C[j,k] + C[k+1,l] + cost(X,j,l)
            if q < C[j,l]
                C[j,l] = q
                S[j,l] = k
return C

print(S,X,i,j):
k = S[i,j]
if i == j
    return
printf("%d, %d\n", sum(X[i..k]), sum(X[k+1..j]))
print(S,X,i,k)
print(S,X,k+1,j)
```

5.

满足递归式 F(n)=F(n-1)+F(n-2)和初始值 F(0)=F(1)=1 的数列称为斐波那契数列。考虑如何计算该数列的第 n 项 F(n)。（1）说明根据递归式直接完成计算，将有子问题重复求解；（2）说明该问题具有优化子结构；（3）写出求解 F(n)的动态规划算法，并分析算法的时间复杂性。

(1):

F(n) = F(n-1) + F(n-2) = F(n-2) + F(n-3) + F(n-2)，可见F(n-2)被重复求值了

(2):

由于F(n)的计算代价为F(n-1)和F(n-2)的代价之和+1，因此F(n-1)和F(n-2)最优时F(n)最优，即具有优化子结构

(3):

```
calc(n):
fib = [1,1]
for i in 2:n
    fib[i] = fib[i-1]+fib[i-2]
return fib[n]
```

时间复杂度为O(n) 空间复杂度为O(n)

6.

输入是具有 n 个数的向量 x，输出时输入向量的任何连续子向量的最大和，要求写出递归方程、伪代码并分析时间和空间复杂度。

递归方程：

$$C_{i,j} = X_i \quad if\ i = j$$

$$C_{i,j} = \max_{i \le k < j-1} \{C_{i,k}, C_{k+1,j}, \sum_{k=i}^{j} X_k\}$$

伪代码：

求值顺序按照区间长度l=1 to X.length-1

```
for i in 1:X.length
    C[i,i] = X[i]
for l in 2:X.length
    for i in 1:X.length-l
        j = i+l-1
        for k in i:j-1
            q = max(C[i,k],C[k+1,j])
            if q > C[i,j]
                C[i,j] = K
                B = C[i,k] == q ? [i,k] : [k+1,j]
        if sum(X[i..j]) > q
            C[i,j] = sum(X[i..j])
            B = [i,j]
print(B)
```

复杂度分析：

时间复杂度为三重循环$O(n^3)$
使用二维数组C，时间复杂度为$O(n^2)$

7.

令 I1, …, In 是 n 个区间，其中任一区间 Ii=(ai,bi)，假设这些区间按照 bi 从小到大排序，每一个区间有一个权重 vi.找一个互不相交区间的集合，使得这些区间的权重之和最大，例如 I1 = (1,2), v1=0.9; I2 = (2,3), v2=0.5; I3 = (1,4), v3=4; I4= (4,5), v4=2，解是{I3, I4}。给出解决问题 P2 的动态规划算法，要求写出递归方程和伪代码，并分析算法时间空间复杂性。

设C(i,j)为区间右界小于j，可选区间为i的最优解的代价

递推方程：

```
C(i,j) = C(i-1,j) if j > A[i]
C(i,j) = max(C(i-1,j), C(i-1,j+B[i])+V[i]) if j <= A[i]
```

伪代码：

这段代码假设区间端点都是大于0的整数，实际实现的时候可以通过修改区间的判断方法来避免多余的循环和整数的限制

```
Input: A,B,V分别表示I[n]的左界，右界和价值
Output: 不交叉区间的最高价值

C[0,:] = 0
for i in 1:n
    for j in 0:∞
        if j > A[i]
            C[i,j] = C[i-1,j]
        eles
            C[i,j] = max(C[i-1,j], C(i-1,j+B[i])+V[i])
print(C[n,0])
```

复杂度分析：
使用了两重循环，但是由于j的扫描区间是固定的，所以是常数时间，因此时间复杂度为O(n)
使用了二维数组C，但是由于列数固定，因此空间复杂度为O(n)

8.

在一个圆形操场的四周摆放着 n 堆石子，现要将石子有次序地合并成一堆。规定每次只能选择相邻的两堆石子合并成新的一堆，并将新一堆石子数记为该次合并的得分。试设计一个动态规划算法，计算出将 n 堆石子合并成一堆的最小得分和最大得分，要求列出递归方程，写出算法的伪代码并分析算法的时间空间复杂性。

感觉和第4题没什么本质区别，就是变成了圆形而已，最内层的循环变量由一个改为两个就行了。

9.

给定两个字符串 s1, s2，其上的操作包括增加一个字符、删除一个字符、修改一个字符和交换两个相邻的字符，其中增加和删除一个字符和交换相邻字符的代价均为 1,将字符 a 修改为字符 b 的代价记作 Ca,b，写出一个动态规划算法求出从s1 变化为 s2 代价最小的变化序列，要求写出递推方程和伪代码并分析时间复杂性。

10.

设有 n 种不同面值的硬币，面值分别为 c1, c2, …., cn 分钱, 求用最少个数硬币来找 K 分钱的策略。要求写出递归方程、伪代码并分析时间和空间复杂度。

C(i)为找i分钱所用硬币的最小数量

递推方程：

$$C(i) = 0 \quad if\ i = 0$$
$$C(i) = \infty \quad if\ i < 0$$
$$C(i) = min_i\{C(i - c_i) + 1\}$$

代码略