1)

优化子结构:

对于长度为[1 .. n – 1]的字符串，现在要考虑把第 N 个字符加入当前字符串，那么它将有如下的若干可能性：存在 i，1 <= i <= n – 1,使得第 i 到第 N 个字符串是回文串（条件 1）。那么长度为 N 的字符串最好分割情况将从所有满足条件 1 的情况中选择分割次数最少的情况。每种情况对应的分割次数是：[1..i-1]的最少分割次数 + 1。重叠的部分在于每次比较都需要计算[1..i-1]的部分。

状态转移方程：

f(n) = min{f(i – 1) + 1|1<=i<=n-1 && [i..n]可以构成回文串}

f(0) = 0

伪代码：

函数 panding（i，j）表示判定从 i 到 j 的字符串时候构成回文串，是返回 true，否则返回 false;

```
for(int i = 1; I <= n;  i++){
    int min = 9999;
    for(int j = I – 1; j > 0; j--){
    if(panding(j,i)){
        if(f[j – 1] + 1 < min)min = f[j – 1] + 1;
    }
    f[i] = min;
}
```

复杂度分析：o(n^2)

2)

优化子结构：上第 N 层只能由 N-1 层 上一阶，或者由 N-2 层上两阶。第 i 层被重复算了若干次。

状态转移方程：

f(n) = f(n – 1) + f(n – 2);f(0) = 1,f(1) = 1

伪代码：

```
for(int I = 2; I <= n; i++){
    f[i] = f[i – 1] + f[i – 2];
}
```

复杂度分析：o(n)

3)

证明：因为是求拆法总数，所以我们需要考虑有哪几种情况。对于(i,j)可以分成拆法中有 j 和拆法中没有 j 的两种情况，这两种情况互斥，所以是加法，对于没有 j 的拆法是(i,j-1),对于有 j 的拆法，我们从这种拆法中去掉一个 j，那么总数是 i – j,所以他的拆法数是(i-j,j)。

```
for(i = 1; i<= n; i++){
    for(int j = 1; j <= n; j++){
        f[j][i] = f[j][i – 1] + f[j – i][i];
    }
}
```

4)

(1) x = 1,4,3

y = 1,3,2,4,3

z= 1,4,3,2,4

lcs(x,y,z) = 1,4,3

lcs(x,lcs(y,z)) = lcs(x,1,3,2,4) = 1,4

(2)

```
for I = 1 to n
    for j = 1 to m
        for k = 1 to l
        {
            //<i,j,k>在 x[i] == y[j] == z[k]时等于 1,其他为 0
            f[i][j][k] =    max{f[i – 1][j – 1][k – 1] + <i,j,k>,f[i][j][k-1],f[i][j-1][k-1],
            f[i][j – 1][k],f[i – 1][j][k],f[i -1][j][k – 1],f[i – 1][ j - 1][k]}
        }
```

时间复杂度 o(m*n*l)

5)

优化子结构：假设这个有 i 位，最高位为 j，则可以将第一位，（也是最高位先抛出，所有满足条件的数取决于后面即有 i-1 位，而最高位为 X 的情况）我们要考虑的只是 X 的取值|j – x|>=2。i-1 位，最高位为 x 的情况将被计算多次。

状态转移方程：

d[i][j]=∑d[i-1][k](0≤k≤9 且|k-j|≥2

d[1][j]=1

伪代码：

```
for I = 1 to 9
    d[1][i] =1
for I =2 to B 的位数
    for j = 0 to 9
        for k = 0 to 9
            if |j – k|>= 2 then d[i][j] = d[i – 1][k]
```

时间复杂度 o(log n)