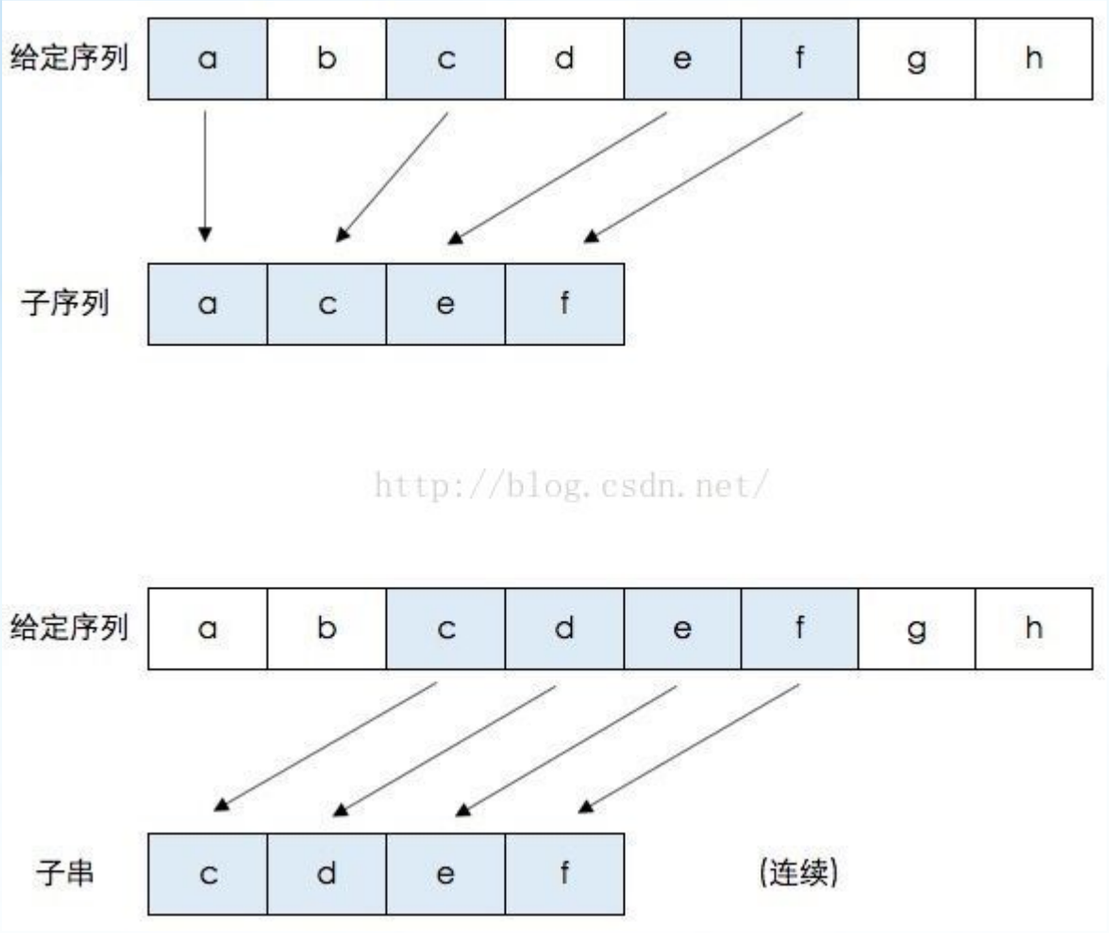
LCS—最长公共子序列



最长公共子序列不需要连续

给定序列

s1={3,5,7,4,8,6,7,8,2}

s2={1,3,4,5,6,7,7,8}

s1和s2的相同子序列，且该子序列的长度最长，即是LCS

s1和s2的其中一个最长公共子序列是 {3,4,6,7,8}

动态规划法求LCS

动态规划算法通常用于求解具有某种最优性质的问题。

在这类问题中，可能会有许多可行解。每一个解都对应于一个值，我们希望找到具有最优值的解

例如LCS中的‘最长’

动态规划法的基本思路

1. 动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。
2. 若用分治法来解这类问题，则分解得到的子问题数目太多，有些子问题被重复计算了很多次。
3. 如果我们能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，这样就可以避免大量的重复计算
4. 将各阶段按照一定的次序排列好之后，对于某个给定的阶段状态，它以前各阶段的状态无法直接影响它未来的决策。换句话说，每个状态都是过去历史的一个完整总结，这就是无后效性

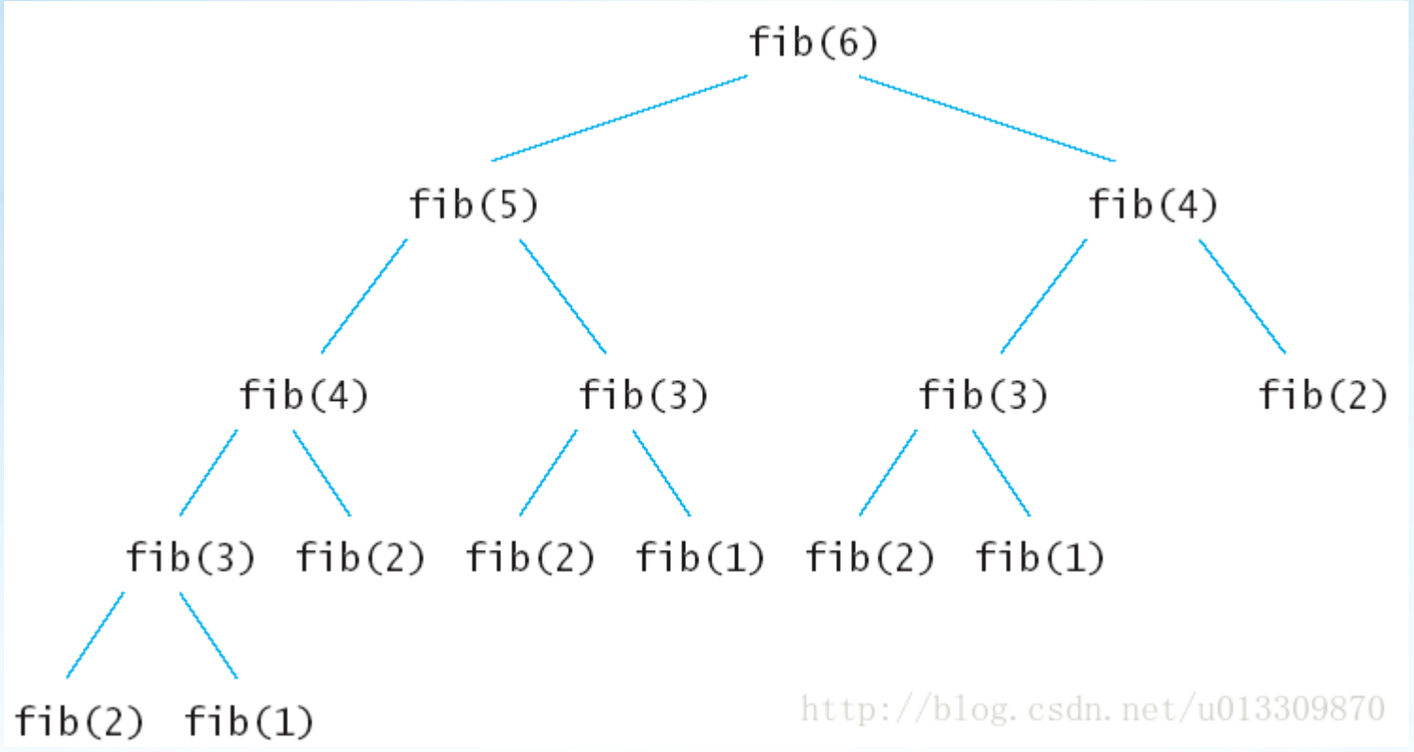
三个特点：最优子结构、重叠子问题、无后效性

递归求斐波那契数列

数列：1、1、2、3、5、8、13、21、34、……

```
int f(int n)
{
    if(n == 0) return 0;
    if(n == 1 ) return 1;
    if(n >= 2)
    {
        return f(n-1)+f(n-2);
    }
}
```

这种算法并不高效,它做了很多重复计算,它的时间复杂度为$O(2^n)$



在斐波拉契数列，可以看到大量的重叠子问题，比如说在求`fib(6)`的时候，`fib(2)`被调用了5次。如果使用递归算法的时候会反复的求解相同的子问题，不停的调用函数，而不是生成新的子问题。

动规求斐波那契数列

使用动态规划来将重复计算的结果具有"记忆性",就可以将时间复杂度降低为O(n)

```
void f()
{
    int f[10];
    f[0] = 0;
    f[1] = 1;
    for(int i = 2; i <= 10; i++)
        f[i] = f[i-1] + f[i-2];
}
```

$$F(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ F(n\text{-}1) + F(n\text{-}2) & \text{if } n > 1 \end{cases}$$

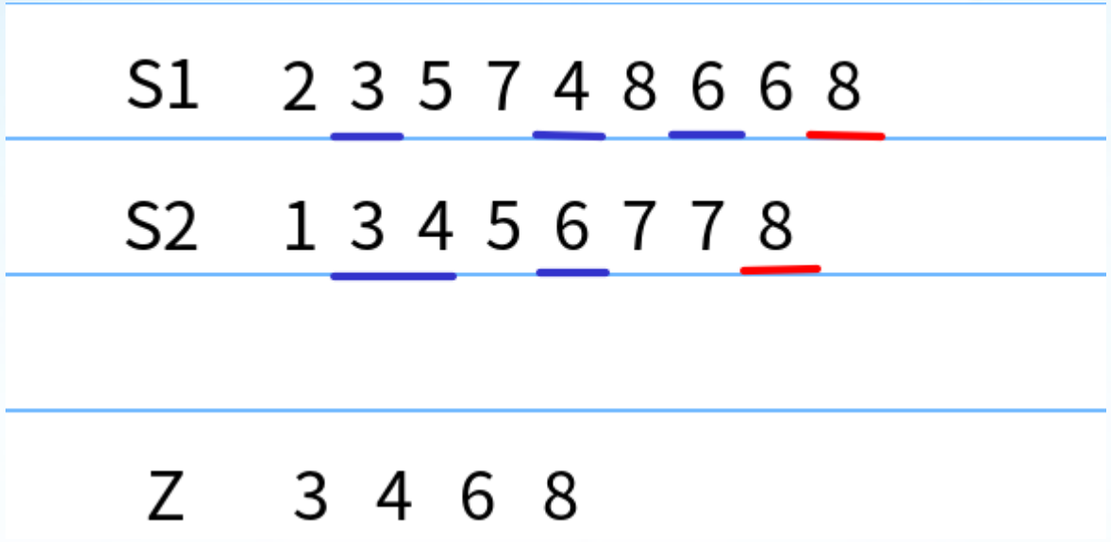
n	0	1	2	3	4	5	6	7	8	9	10
$F(n)$	1	1	2	3	5	8	13	21	34	55	89

回到原题，求解LCS

解决LCS问题，需要把原问题分解成若干个子问题，所以需要刻画LCS的特征

设S1={A0,A1, ... ,Am}，S2={B0,B1, ... Bn}，它们LCS为Z={Z1,Z2, ... ,Zk}

1. 如果Am=Bn，则Zk=Am=Bn，且{Z1, Z2, ..., z(K-1)}是{A0,A1, ... ,A(m-1)}和{B0,B1, ... B(n-1)}的一个最长公共子序列

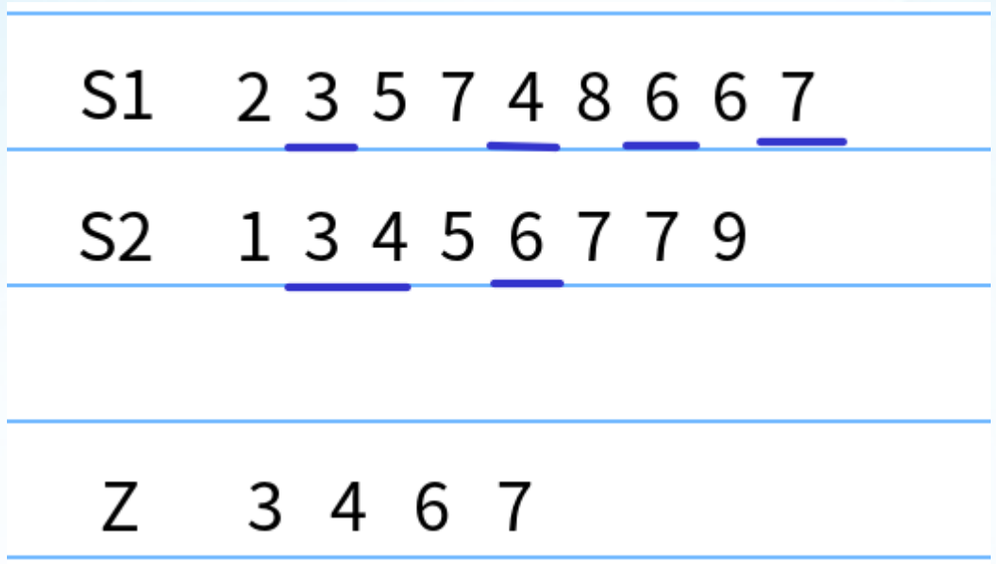


即假如S1的最后一个元素 与S2的最后一个元素相等，那么S1和S2的LCS就等于 {S1减去最后一个元素} 与 {S2减去最后一个元素} 的 LCS 再加上 S1和S2相等的最后一个元素

设S1={A0,A1, … ,Am}，S2={B0,B1, … Bn}，它们LCS为Z={Z1,Z2, … ,Zk}

2. 如果Am=Bn

- 若Zk!=Am，则{Z1,Z2, … ,Zk}是{A0,A1, … ,A(m-1)}和{B0,B1, … Bn}的一个最长公共子序列
- 若Zk!=Bn，则{Z1,Z2, … ,Zk}是{A0,A1, … ,Am}和{B0,B1, … B(n-1)}的一个最长公共子序列



假如S1的最后一个元素 与 S2的最后一个元素不等，那么S1和S2的LCS就等于： MAX({S1减去最后一个元素} 与 S2 的LCS， {S2减去最后一个元素} 与 S1 的LCS)

动态转移方程

假设我们用C[i,j]表示Xi 和 Yj 的LCS的长度（直接保存最长公共子序列的中间结果不现实，需要先借助LCS的长度）。其中X = {x1 ... xm}，Y ={y1...yn}，Xi = {x1 ... xi}，Yj={y1... yj}。

可得动态转移方程如下：

$$C[i,j]=\begin{cases}0 & \text{若 } i=0 \text{或} j=0\\ C[i-1,j-1]+1 & \text{若 } i,j>0, x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{若 } i,j>0, x_i\neq y_j\end{cases}$$

s1={1,3,4,5,6,7,7,8}，s2={3,5,7,4,8,6,7,8,2}

下标i \ 下标j		0	1	2	3	4	5	6	7	8	9
		$S2_j$	3	5	7	4	8	6	7	8	2
0	$S1_i$										
1	1										
2	3										
3	4										
4	5										
5	6										
6	7										
7	7										
8	8										

图中的空白格子需要填上相应的数字（这个数字就是C[i,j]的定义，记录的LCS的长度值）。填的规则依据公式，简单来说：如果横竖（i,j）对应的两个元素相等，该格子的值 = c[i-1,j-1] + 1。如果不等，取c[i-1,j] 和 c[i,j-1]的最大值。

首先初始化该表。

下标j / 下标i		0	1	2	3	4	5	6	7	8	9
	$S2_j$	3	5	7	4	8	6	7	8	2	
0	$S1_i$	0	0	0	0	0	0	0	0	0	0
1	1	0									
2	3	0									
3	4	0									
4	5	0									
5	6	0									
6	7	0									
7	7	0									
8	8	0									

当i=2，j=1时，S1的元素3 与 S2的元素3 相等，所以 C[2,1] = C[1,0] + 1

表格

下标i \ 下标j		0	1	2	3	4	5	6	7	8	9
	$S2_j$		3	5	7	4	8	6	7	8	2
0	$S1_i$	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
2	3	0	1								
3	4	0									
4	5	0									
5	6	0									
6	7	0									
7	7	0									
8	8	0									

动态转移方程

$$C[i,j]=\begin{cases}0 & \text{若 } i=0 \text{或} j=0\\ C[i-1,j-1]+1 & \text{若 } i,j>0, x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{若 } i,j>0, x_i\neq y_j\end{cases}$$

当i=2，j=2时，S1的元素3 与 S2的元素5 不等，C[2,2] =max(C[1,2],C[2,1])

图中C[1,2] 和 C[2,1] 背景色为浅黄色。

表格

下标i	下标j	0	1	2	3	4	5	6	7	8	9
	$S2_j$	3	5	7	4	8	6	7	8	2	
0	$S1_i$	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
2	3	0	1	1	1	1	1	1	1	1	1
3	4	0									
4	5	0									
5	6	0									
6	7	0									
7	7	0									
8	8	0									

动态转移方程

$$C[i,j]=\begin{cases}0 & \text{若 } i=0 \text{或} j=0\\ C[i-1,j-1]+1 & \text{若 } i,j>0, x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{若 } i,j>0, x_i\neq y_j\end{cases}$$

完整表格

下标i	下标j	0	1	2	3	4	5	6	7	8	9
		$S2_j$	3	5	7	4	8	6	7	8	2
0	$S1_i$	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
2	3	0	1	1	1	1	1	1	1	1	1
3	4	0	1	1	1	2	2	2	2	2	2
4	5	0	1	2	2	2	2	2	2	2	2
5	6	0	1	2	2	2	2	3	3	3	3
6	7	0	1	2	3	3	3	3	4	4	4
7	7	0	1	2	3	3	3	3	4	4	4
8	8	0	1	2	3	3	4	4	4	5	5

动态转移方程

$$C[i,j]=\begin{cases}0 & \text{若 } i=0 \text{或} j=0\\ C[i-1,j-1]+1 & \text{若 } i,j>0, x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{若 } i,j>0, x_i\neq y_j\end{cases}$$

根据性质，c[8,9] = S1 和 S2 的 LCS的长度，即为5

模板题--POJ1458(Common Subsequence)

Common Subsequence

Language: Default

Time Limit: 1000MS **Memory Limit:** 10000K
Total Submissions: 66100 **Accepted:** 27607

Description

A subsequence of a given sequence is the given sequence with some elements (possible none) left out. Given a sequence X = < x1, x2, ..., xm > another sequence Z = < z1, z2, ..., zk > is a subsequence of X if there exists a strictly increasing sequence < i1, i2, ..., ik > of indices of X such that for all j = 1,2,...,k, x_{ij} = zj. For example, Z = < a, b, f, c > is a subsequence of X = < a, b, c, f, b, c > with index sequence < 1, 2, 4, 6 >. Given two sequences X and Y the problem is to find the length of the maximum-length common subsequence of X and Y.

Input

The program input is from the std input. Each data set in the input contains two strings representing the given sequences. The sequences are separated by any number of white spaces. The input data are correct.

Output

For each set of data the program prints on the standard output the length of the maximum-length common subsequence from the beginning of a separate line.

Sample Input

```
abcfbc         abfcab
programming    contest
abcd           mnp
```

Sample Output

```
4
2
0
```

题意：输入不定行，每行两个字符串，求每一行两个字符串的最长公共子序列长度

```
const int MAXDP = 1e3;
const int MAXS = 1e7;
int dp[MAXDP][MAXDP];
char s1[MAXS], s2[MAXS];
int LCS(char* s1, char* s2)
{
    int len1 = strlen(s1) - 1;//因为s1,s2是从1开始存的，所以长度要减1
    int len2 = strlen(s2) - 1;
    for(int i = 0; i <= len1; i++)
        for(int j = 0; j <= len2; j++)
        {
            if(i == 0 || j == 0)
                dp[i][j] = 0;
            else if(s1[i] == s2[j])
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
        return dp[len1][len2];
}
int main()
{
    s1[0] = ' ', s2[0] = ' ';
    while(cin >> s1 +1 >> s2 + 1)
        cout << LCS(s1, s2) << endl;
    return 0;
} //时间复杂度：O（len1*len2)
```

LCS进阶

1. 还原最长公共子序列、记录路径：HDU1503(Advanced Fruits)
2. LCS变形：
    - POJ1080(Human Gene Functions)
    - POJ3356(AGTC)

最长公共子串（连续）

和LCS区别是区别就是因为是连续的，如果两个元素不等，那么就要=0了而不能用之前一个状态的最大元素

最长公共子串	最长公共子序列
$c[i,j]=\begin{cases}0 & i=0 \text{ or } j=0\\ c[i-1,j-1]+1 & x_i=y_j\\ 0 & x_i\neq y_j\end{cases}$	$C[i,j]=\begin{cases}0 & \text{若 } i=0 \text{或} j=0\\ C[i-1,j-1]+1 & \text{若 } i,j>0, x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{若 } i,j>0, x_i\neq y_j\end{cases}$

LIS--最长递增子序列

假设有序列A = {5, 2, 8, 6, 3, 6, 9, 7}

其递增子序列有：{5,8,9}, {2,6,9}, {5,6,7}……

其中，最长递增子序列为{2, 3, 6, 9}和{2, 3, 6, 7}

动态规划求LIS

设dp[i]表示以i结尾的子序列中LIS的长度，dp[j] (0<= j < i) 来表示在i之前的LIS的长度

有一序列A={5, 3, 4, 8, 6, 7}

前1个数的LIS长度d(1)=1(序列：5)
前2个数的LIS长度d(2)=1(序列：3；3前面没有比3小的)
前3个数的LIS长度d(3)=2(序列：3，4；4前面有个比它小的3，所以d(3)=d(2)+1)
前4个数的LIS长度d(4)=3(序列：3，4，8；8前面比它小的有3个数，所以 d(4)=max{d(1),d(2),d(3)}+1=3)

OK，分析到这，我觉得状态转移方程已经很明显了，如果我们已经求出了d(1)到d(i-1)，那么d(i)可以用下面的状态转移方程得到：

d(i) = max{1, d(j)+1}, 其中j<i, A[j]<=A[i]

进一步分析

d(i) = max{ 1, d(j)+1} ,且满足A[i] >= A[j]

1. max显然是为了找到最长的满足条件的序列，容易理解
2. 在max里面加入1作为比较的一员，是因为，最坏的情况就是序列是单调递减的，那么每个数都可以算是一个子序列，一个数的长度当然为1
3. d[j]为什么要加1呢，因为比较的数A[i] > A[j]，那么A[i]就是最长子序列的一员，所以直接在d[j]上加1

模板题--HDU1257-最少拦截系统

最少拦截系统

Time Limit: 2000/1000 MS (Java/Others) Memory Limit: 65536/32768 K (Java/Others)
Total Submission(s): 65659 Accepted Submission(s): 25536

Problem Description

某国为了防御敌国的导弹袭击,发展出一种导弹拦截系统.但是这种导弹拦截系统有一个缺陷:虽然它的第一发炮弹能够到达任意的高度,但是以后每一发炮弹都不能超过前一发的高度.某天,雷达捕捉到敌国的导弹来袭.由于该系统还在试用阶段,所以只有一套系统,因此有可能不能拦截所有的导弹.
怎么办呢?多搞几套系统呗!你说说倒蛮容易,成本呢?成本是个大问题啊.所以俺就到这里来求救了,请帮助计算一下最少需要多少套拦截系统.

Input

输入若干组数据.每组数据包括:导弹总个数(正整数),导弹依此飞来的高度(雷达给出的高度数据是不大于30000的正整数,用空格分隔)

Output

对应每组数据输出拦截所有导弹最少要配备多少套这种导弹拦截系统.

Sample Input

```
8 389 207 155 300 299 170 158 65
```

Sample Output

```
2
```

解题

我们所求的拦截系统的数目其实就是一个序列的所有递减子序列，并使其数量尽量减少，然后递减子序列的数目又会等于最长上升子序列中所含元素的个数；
不理解的话可以去看下下面的test

```
Input
8 389 207 155 300 299 170 158 65
1 100
6 300 200 400 200 100 500
8 500 300 400 200 300 100 200 50
8 500 300 400 200 80 200 100 50
8 500 300 400 200 80 500 100 50

Output
2
1
3
2
2
3
```

Code

```
int dp[100000], arr[100000], len;
int LIS(){
    memset(dp, 0, sizeof(dp));
    int MAX = -1e9;
    for(int i = 1; i <= len; i++)
        for(int j = 0; j < i; j++){
            if(arr[i] > arr[j] && dp[i] < dp[j] + 1){
                dp[i] = dp[j] + 1;
                if(dp[i] > MAX)
                        MAX = dp[i];
            }

        }
    return MAX;
}
int main(){
    while(cin >> len){
        for(int i = 1; i <= len; i++)
            cin >> arr[i];
        cout << LIS() << endl;
    }
    return 0;
}//时间复杂度:O(n^2)
```

LIS进阶

1. 算法优化：NlogN时间复杂度--可参考(20:41开始)
   HRBU ACM 01背包 LIS 拓扑 凸包—-bilibil
2. 最大递增子数组和--由LIS O(n2)的办法变化而来的，对应的模板题：
   HDU1087(Super Jumping! Jumping! Jumping!)
3. LIS变形：
   - HDU5256-序列变换
   - 洛谷P1091--合唱队形