算法

动态规划

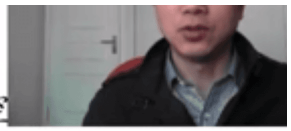
动态规划就是去掉重复计算

动规解题的一般思路

1. 将原问题分解为子问题

- 把原问题分解为若干个子问题，子问题和原问题形式相同或类似，只不过规模变小了。子问题都解决，原问题即解决(数字三角形例）。
- 子问题的解一旦求出就会被保存，所以每个子问题只需求解一次。

■



2. 确定状态

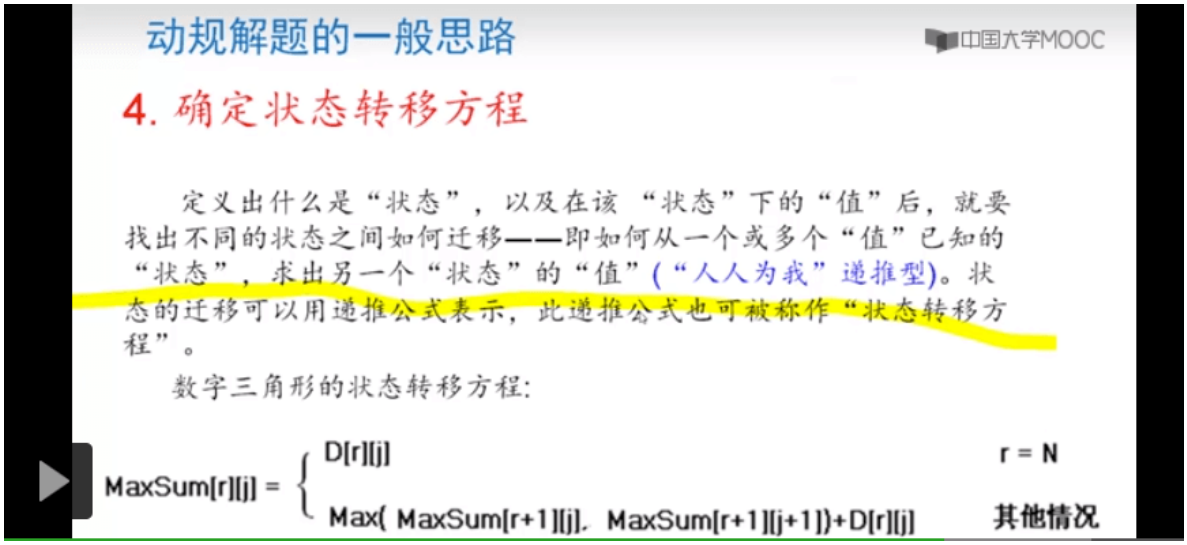
所有“状态”的集合，构成问题的“状态空间”的大小，与用动态规划解决问题的时间复杂度直接相关。在数字三角形的例子里，一共有N×(N+1)/2个数字，所以这个问题的状态空间里一共就有N×(N+1)/2个状态。

整个问题的时间复杂度是状态数目乘以计算每个状态所需时间。

在数字三角形里每个“状态”只需要经过一次，且在每个状态上作计算所花的时间都是和N无关的常数。

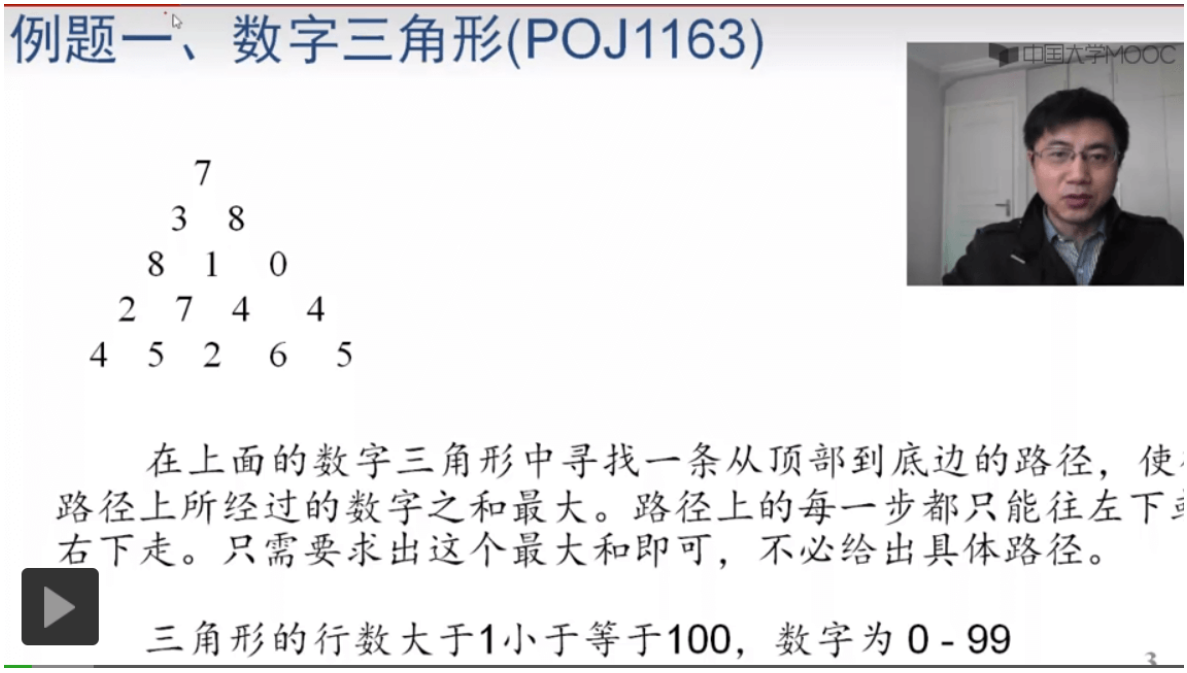
3. 确定一些初始状态（边界状态）的值

以“数字三角形”为例，初始状态就是底边数字，值就是底边数字值。


动规解题的一般思路
中国大学MOOC
4. 确定状态转移方程
定义出什么是“状态”，以及在该“状态”下的“值”后，就要找出不同的状态之间如何迁移——即如何从一个或多个“值”已知的“状态”，求出另一个“状态”的“值”(“人人为我”递推型)。状态的迁移可以用递推公式表示，此递推公式也可被称作“状态转移方程”。
数字三角形的状态转移方程:
MaxSum[r][j] = { D[r][j] r = N
Max(MaxSum[r+1][j], MaxSum[r+1][j+1])+D[r][j] 其他情况

数字三角形

题解


例题一、数字三角形(POJ1163)
中国大学MOOC
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
在上面的数字三角形中寻找一条从顶部到底边的路径，使
路径上所经过的数字之和最大。路径上的每一步都只能往左下
右下走。只需要求出这个最大和即可，不必给出具体路径。
三角形的行数大于1小于等于100，数字为 0 - 99

解题思路：

用二维数组存放数字三角形。

D(r, j) ：第r行第 j 个数字(r,j从1开始算)
MaxSum(r, j)： 从D(r,j)到底边的各条路径中，
最佳路径的数字之和。
问题：求 MaxSum(1,1)

典型的递归问题。
D(r, j)出发，下一步只能走D(r+1, j)或者D(r+1, j+1)。故对于N行的三角形：

```
if ( r == N)
    MaxSum(r,j) = D(r,j)
else
    MaxSum( r, j) = Max{ MaxSum(r+1,j), MaxSum(r+1,j+1) }
```

例题一、数字三角形(POJ1163)

```
        7
      3   8
    8   1   0
  2   7   4   4
4   5   2   6   5
```

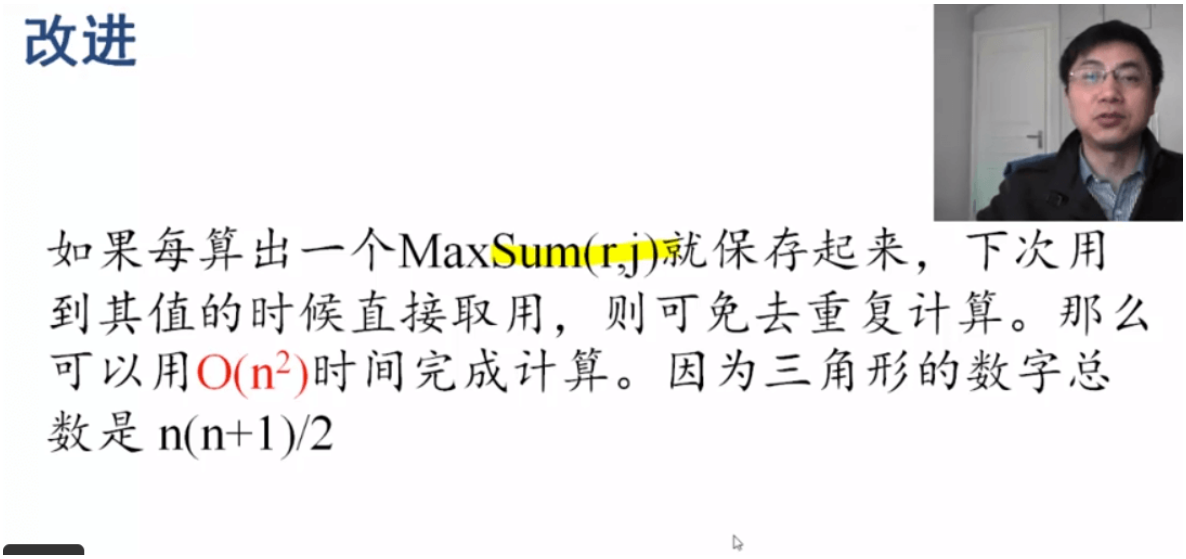
在上面的数字三角形中寻找一条从顶部到底边的路径，使
路径上所经过的数字之和最大。路径上的每一步都只能往左下
右下走。只需要求出这个最大和即可，不必给出具体路径。

三角形的行数大于1小于等于100，数字为 0 - 99

为什么超时？

- 回答：重复计算

如果采用递规的方法，深度遍历每条路径，存在大量重复计算。则时间复杂度为2^n，对于 n = 100行，肯定超时。



code c++

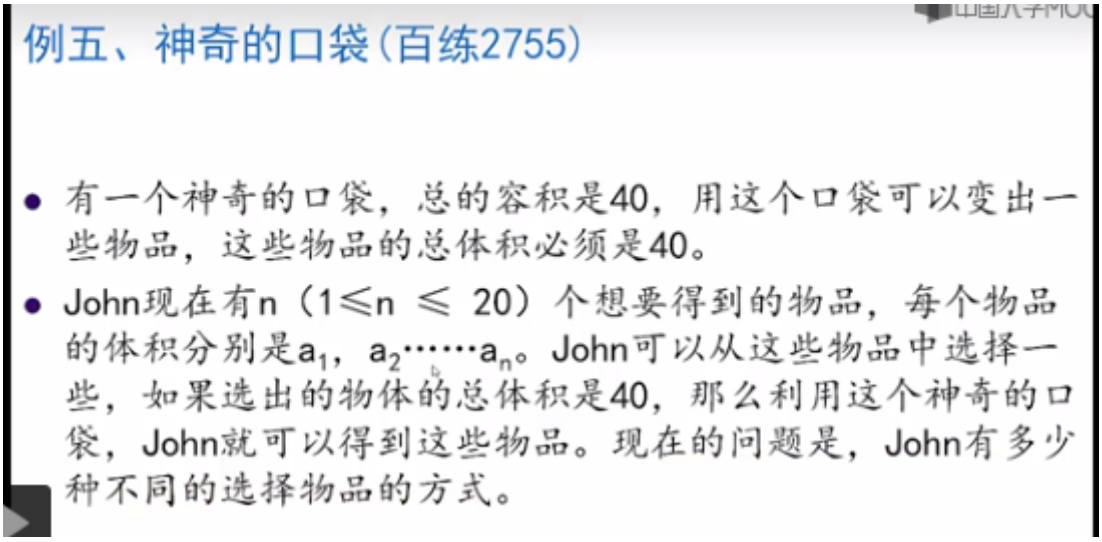
```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
//动态规划 递归 利用一个数组 保存递归产生的重复计算 数字三角形 找下面最大
int Maxsum(vector<vector<int>>a, vector<vector<int>>maxsum, int i, int j){
    int n = a.size() - 1;
    if (maxsum[i][j] != -1)return maxsum[i][j];
    if (i == n)maxsum[i][j] = a[i][j];
    else
    {
        int x = Maxsum(a, maxsum, i + 1, j);
        int y = Maxsum(a, maxsum, i+1, j + 1);
        maxsum[i][j] = max(x, y) + a[i][j];
    }
    return maxsum[i][j];
}
// 递归改为递推
int Maxsum1(vector<vector<int>>a, vector<vector<int>>maxsum, int i, int j){
    int n = a.size() - 1;
    for (int j = 0; i <= n; ++i)maxsum[n][i]=a[n][i];
    for (int i = n-1; i >=0; --i){
        for (int t = 0; t <= i; ++t){
            maxsum[i][t] = max(maxsum[i + 1][t], maxsum[i + 1][t + 1]) + a[i]
[t];
        }
    }
    return maxsum[0][0];
}
// 空间优化 用一个一维数组
int Maxsum2(vector<vector<int>>a, vector<int>  maxsum, int i, int j){

    int n = a.size() - 1;
    maxsum = a[n];
    /*for (int j = 0; i <= n; ++i)maxsum[n][i] = a[n][i];*/
    for (int i = n - 1; i >= 0; --i){
        for (int t = 0; t <= i; ++t){
            maxsum[t] = max(maxsum[t], maxsum[t + 1]) + a[i][t];
        }
```

```
    }
    return maxsum[0];
}
void main(){
    vector<vector<int>>a = { { 7 }, { 3, 8 }, { 8, 1, 0 }, { 2, 7, 4, 4 }, { 4,
5, 2, 6, 5 } };
    int n = a.size();
    vector<vector<int>>b;
    for (int i = 1; i <=n; ++i){
        vector<int>c(i, -1);
        b.push_back(c);
    }
    vector<vector<int>>t=b;
    cout<<Maxsum(a,b, 0, 0)<<endl;
    cout << Maxsum1(a, t, 0, 0) << endl;
    vector<int>  maxsum;
    cout << Maxsum2(a, maxsum, 0, 0) << endl;
    system("pause");
}
```

神奇的口袋

题解



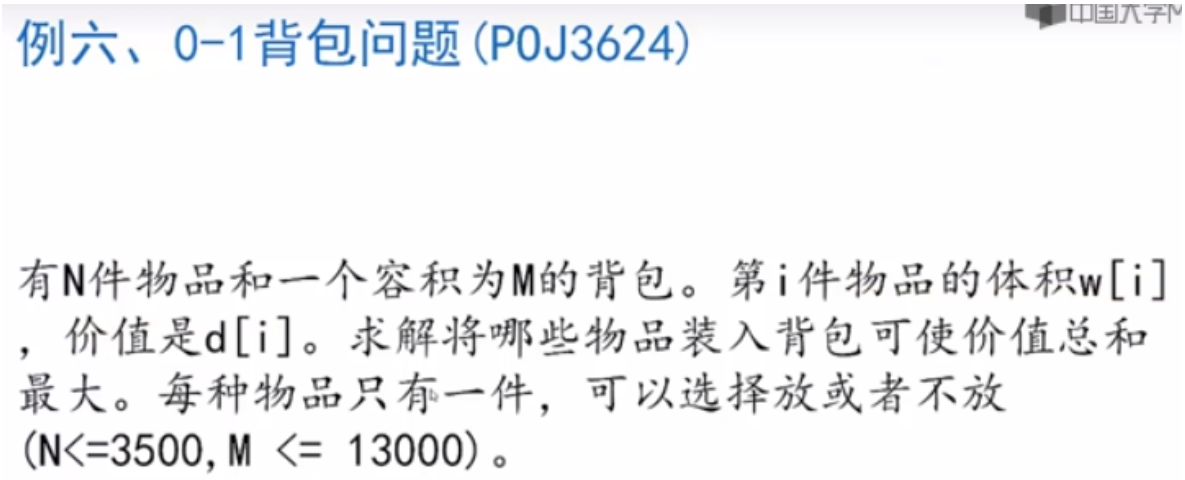
code c++

```
#include<iostream>
#include<vector>
using namespace std;
//递归
int Ways(int s, vector<int>a,int k){//从前k个商品里 找一些商品凑数
    if (s == 0)return 1;
    if (k <= 0)return 0;
    if (s - a[k - 1]>=0)return Ways(s, a, k - 1) + Ways(s - a[k-1], a, k - 1);
    else return Ways(s, a, k - 1);
}
//动态规划
int Ways1(vector<vector<int>>c, vector<int>a, int k){//从前k个商品里 找一些商品凑数
    vector<int> b(k+1, 1);
```

```
    //c[w][k] 代表从前k个物品 凑出w体积的方法数目
    c[0] = b;
    for (int w = 1; w <=40; ++w){
        for (int i = 1; i <= k; ++i){
            if (w - a[i-1] >= 0)c[w][i] = c[w - a[i-1]][i - 1] + c[w][i - 1];
            else c[w][i] = c[w][i - 1];
        }
    }
    return c[40][k];
}
int main(){
    vector<int>a = { 10, 20, 30, 40, 20, 5, 5, 5, 5, 15, 25, 35 };
    int k = a.size();
    //cout << Ways(40, a,k) << endl;
    vector<int>d(k+1);
    vector<vector<int>>c(41,d);
    cout << Ways1(c, a, k) << endl;
    system("pause");
}
```

背包问题

题解



code c++

```
#include<iostream>
using namespace std;
#include<algorithm>
int dp[5][9] = { { 0 } };//动态规划记录值
int item[5];//记录是不是最优解
int w[5] = { 0, 2, 3, 4, 5 };            //商品的体积2、3、4、5
int v[5] = { 0, 3, 4, 5, 6 };            //商品的价值3、4、5、6
int dpless[9] = { 0 };
int itemless[5];//记录是不是最优解

int bagv = 8;                            //背包大小
//递推动态规划 空间复杂度大 需要n*m记录背包总价
int beibao(){
    for (int i = 1; i <= 4; ++i){
```

```
        for (int j = 1; j <= bagv; j++){
            if (j < w[i])dp[i][j] = dp[i - 1][j];
            else dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]]+v[i]);
        }
    }
    return dp[4][8];
}
//背包问题最优解回溯，输出最优解的方案
void findwhat(int i, int j){
    if (i >= 0){
        if (dp[i][j] == dp[i - 1][j]){
            item[i] = 0;
            findwhat(i - 1, j);
        }
        else if (j >=w[i] && dp[i][j] == dp[i - 1][j - w[i]] + v[i]){
            item[i] = 1;
            findwhat(i - 1, j - w[i]);
        }
    }
}
//优化空间 一维数组 从第一行 最后一列往前走 回溯不了 前面的覆盖了
int beibaoless(){
    for (int i = 1; i <= 4; ++i){
        for (int j = bagv; j >=1; --j){
            if (j < w[i])dpless[j] = dpless[j];
            else dpless[j] = max(dpless[j - w[i]] + v[i], dpless[j]);
        }
    }
    return dpless[8];
}

void main(){
    cout<<beibao()<<endl;
    findwhat(4,8);
    for (int i = 0; i <= 4; ++i){
        if (item[i])cout << w[i] << " " << v[i]<<endl;
    }
    cout << beibaoless() << endl;
    system("pause");

}
```

最长子序列

时间复杂度 n*n

题解

例题二：最长上升子序列(百练2757)

问题描述

一个数的序列ai，当$a_1 < a_2 < \ldots < a_S$的时候，我们称这个序列是上升的。对于给定的一个序列(a_1, a_2, ..., a_N)，我们可以得到一些上升的子序列(a_{i1}, a_{i2}, ..., a_{iK})，这里1 <= i1 < i2 < ... < iK <= N。比如，对于序列(1, 7, 3, 5, 9, 4, 8)，有它的一些上升子序列，如(1, 7)，(3, 4, 8)等等。这些子序列中最长的长度是4，比如子序列(1, 3, 5, 8).

你的任务，就是对于给定的序列，求出最长上升子序列的长度。

解题思路

1. 找子问题

“求序列的前n个元素的最长上升子序列的长度”是个子问题，但这样分解子问题，不具有“无后效性”

假设F(n) = x，但可能有多个序列满足F(n) = x。有的序列的最后一个元素比 a_{n+1}小，则加上a_{n+1}就能形成更长上升子序列；有的序列最后一个元素不比a_{n+1}小……以后的事情受如何达到状态n的影响，不符合“无后效性”

解题思路

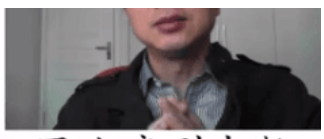
1. 找子问题

“求以a_k (k=1, 2, 3…N) 为终点的最长上升子序列的长度”

一个上升子序列中最右边的那个数，称为该子序列的“终点”。

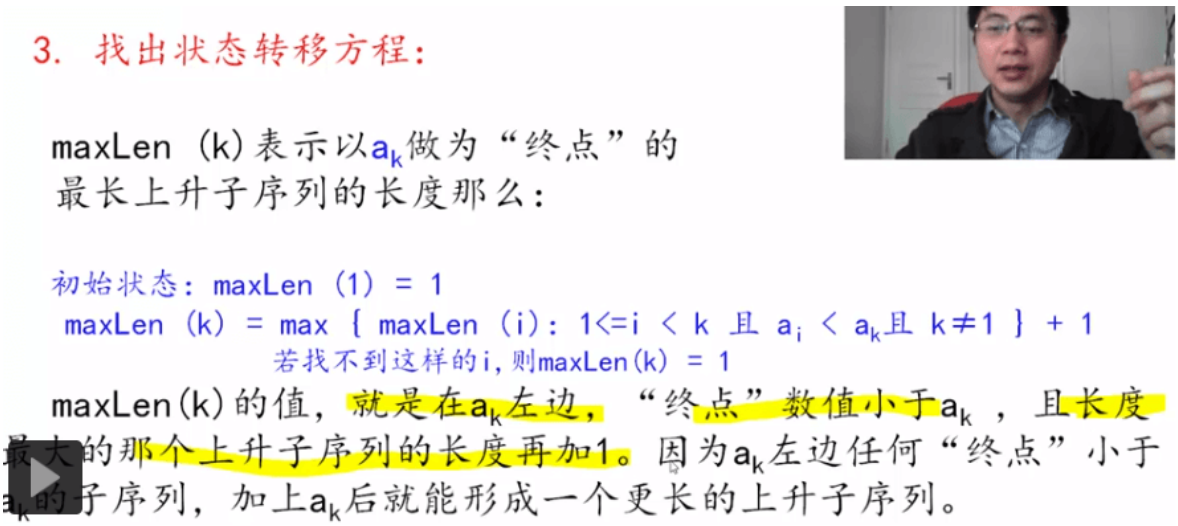
虽然这个子问题和原问题形式上并不完全一样，但是只要这N个子问题都解决了，那么这N个子问题的解中，最大的那个就是整个问题的解。

2. 确定状态：



子问题只和一个变量-- 数字的位置相关。因此序列中数的位置k 就是“状态”，而状态 k 对应的“值”，就是以a_k做为“终点”的最长上升子序列的长度。

状态一共有N个。



code c++

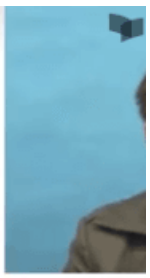
```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
//const int MAXN = 1000;
vector<int> a = { 1, 2, 5, 6, 8, 4, 2, 1, 3, 6, 9, 8, 44, 556, 1156, 4458, 1,
25, 5 };
void main(){
    int len = a.size();
    vector<int> b(len, 1);//存 max ak
    for (int i = 1; i < len; ++i){
        //从 0 到i 依次比较 取最大的序列长度 存起来
        for (int j = 0; j < i; ++j){
            if (a[i]>a[j])b[i] = max(b[j] + 1, b[i]);
        }
    }
    //b数组里存了以ak为终点的最长子序列长度
    cout<<*max_element(b.begin(), b.end());
    //max_element 求区间中的最大值 返回最大元素迭代器
    system("pause");

}
```

两字符串最长公共序列

题解

例三、最长公共子序列（POJ1458)



给出两个字符串，求出这样的一个最长的公共子序列的长度：子序列中的每个字符都能在两个原串中找到，而且每个字符的先后顺序和原串中的先后顺序一致。

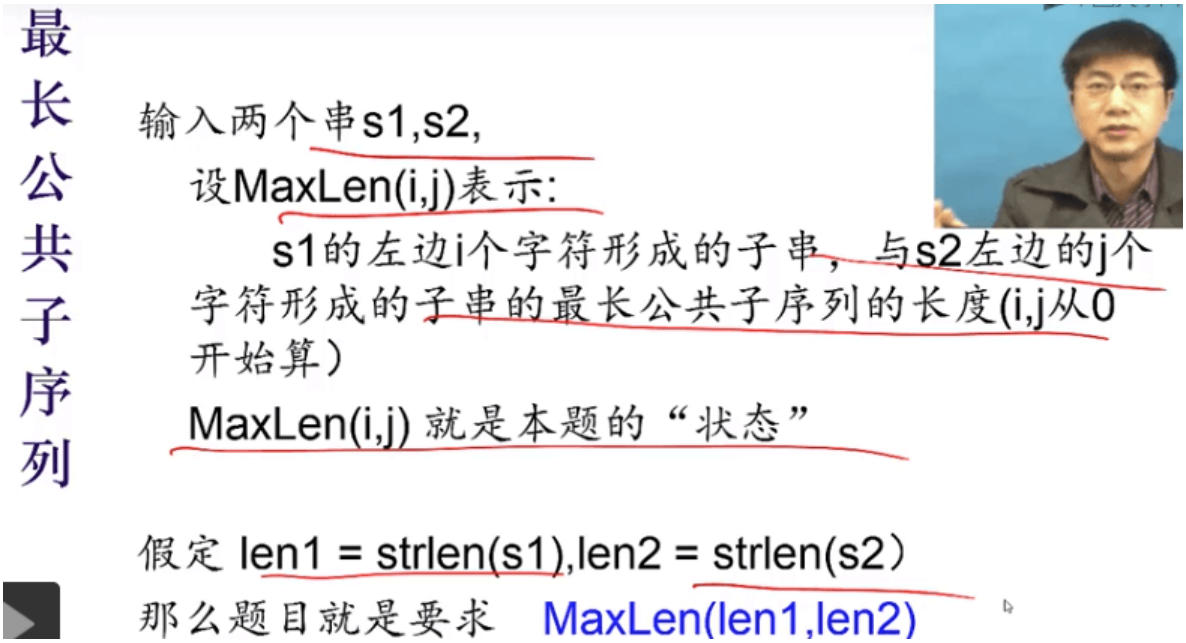
最长公共子序列

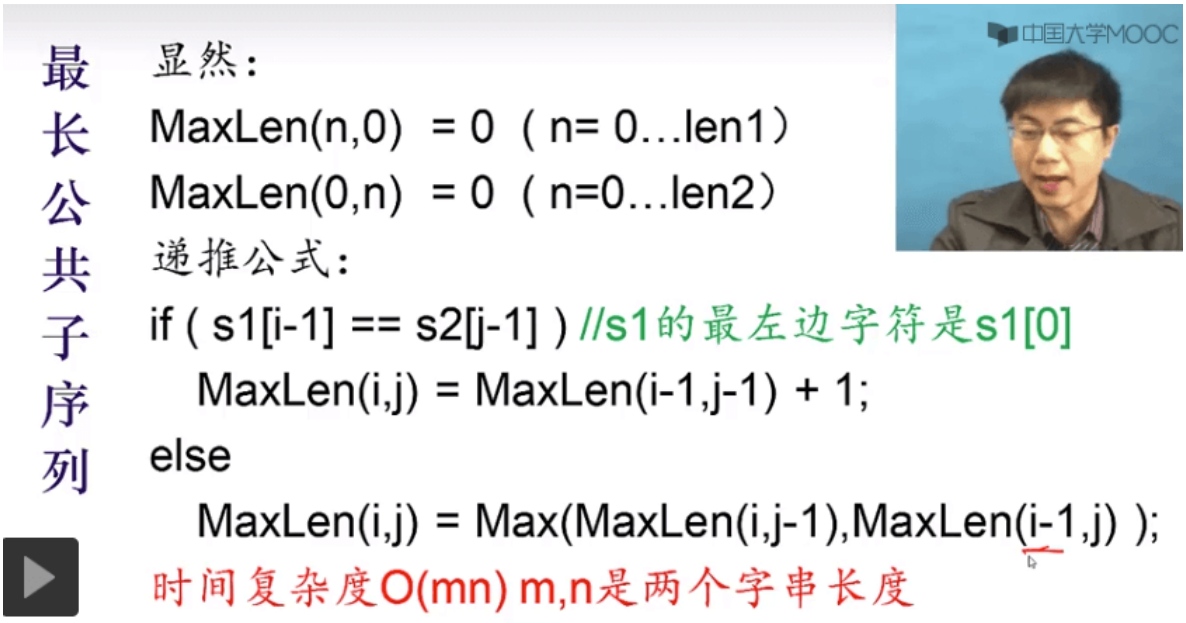
Sample Input

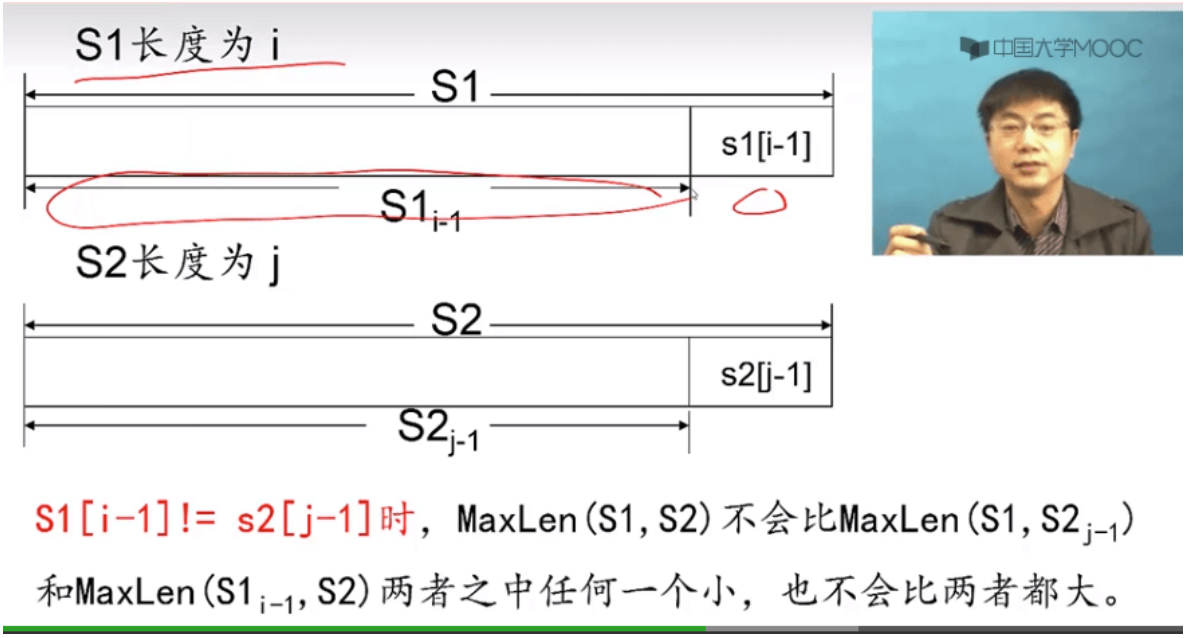
abcfbc abfcab
programming contest
abcd mnp

Sample Output

4
2
0







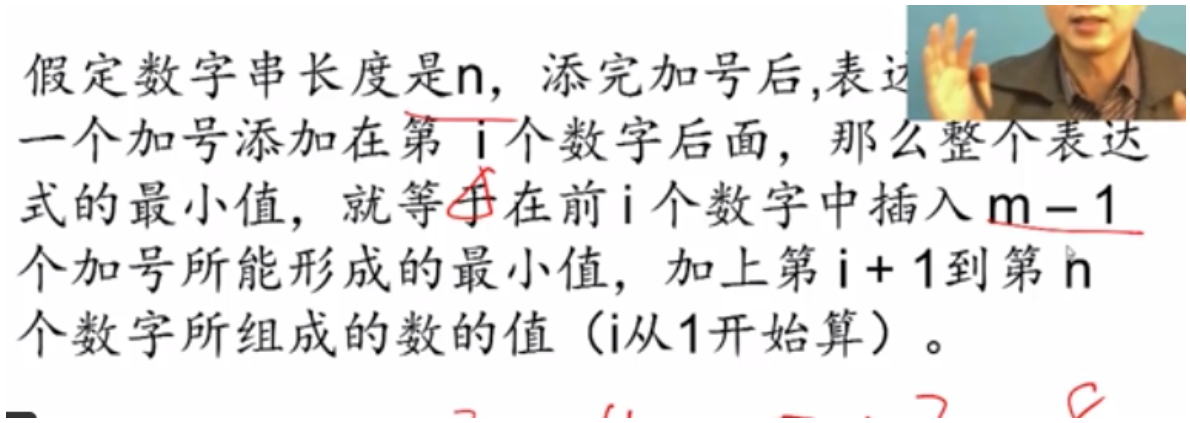
code c++

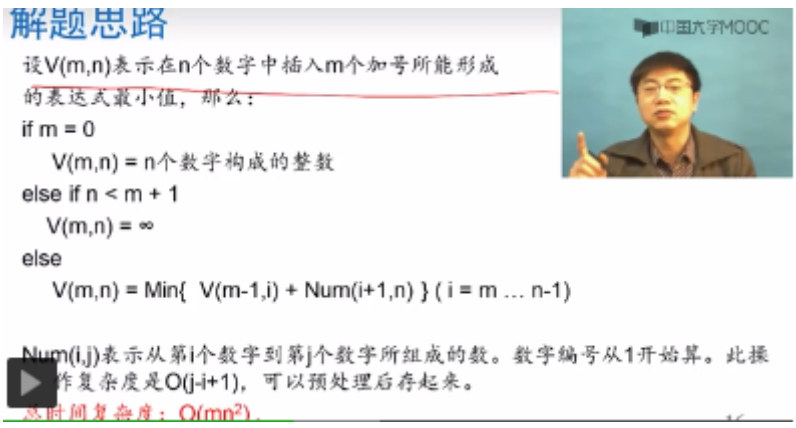
```
#include<iostream>
```

```
#include<String>
#include<vector>
#include<algorithm>
using namespace std;
void main(){
    string s1, s2;
    s1 = "abcdfgtd";
    s2 = "abcfdduti";
    int len1 = s1.size();
    int len2 = s2.size();
    vector<int>a(len1+1,0);
    vector<vector<int>>b(len2+1, a);
    cout << s1[len1 - 1];
    //已经将b[i][0]和b[0][j] 赋值为0
    for (int i = 1; i <=len2; ++i){
        // b[i - 1][j - 1] 不能越界 所以 i j从1开始
        // s2[i-1] == s1[j-1] 从0开始 -1
        for (int j = 1; j <len1; ++j){
            if (s2[i-1] == s1[j-1])b[i][j] = b[i - 1][j - 1] + 1;
            else b[i][j] = max(b[i - 1][j], b[i][j - 1]);
        }
    }
    cout << b[len2-1][len1-1];
    system("pause");
}
```

最佳加法表达式

题解





code c++

```cpp
#include<iostream>
#include<String>
#include<vector>
#include<limits>//最大值
#include<algorithm>
using namespace std;
void main(){
    int INF = INT_MAX;
    vector<int> a = { 1, 22, 3};
    int m = 1;//加号的个数是3
    int len = a.size();
    vector<int>d(len+1);
    vector<vector<int>>b(len + 1, d);
    for (int i = 1; i <= len; ++i){
        //i 位和j位组成的数字
        b[i][i] = a[i-1];
        for (int j = i + 1; j <= len; ++j){
            int t = 1, c = a[j - 1];
            while (c / 10 >= 1){ ++t; c = c / 10; }//判断后一位的位数
                b[i][j] = b[i][j - 1] * pow(10,t) + a[j - 1];
        }
    }
    vector<int> res1(len+1,0);
    vector<vector<int>>res(m+1, res1);
    for (int i = 1; i <= len; ++i)
        res[0][i] = b[1][i];//没有加号的情况 res[i][j]代表i个加号，j个数字组成的最小数
    for (int i = 1; i <= m; ++i){
        for (int j = 0; j <= len; ++j){
            if (j < i + 1)res[i][j] = INF;//数要比加号多1
            else{
                int tmp = INF;
                for (int k = i; k < j; ++k){
                    //res[i][j]= 在前k个数里面 插入i-1个加号的最小值加上 第k+1到j组成的
数字大小 最小值
                    tmp = min(tmp, res[i - 1][k] + b[k + 1][j]);
                }
                res[i][j] = tmp;
            }

        }
    }
    cout <<"zuixiao"<< res[m][len];
    system("pause");

}
```