

# 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 $N \times (N+1)/2$ 个数字，所以这个问题的状态空间里一共就有 $N \times (N+1)/2$ 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。

## 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：

$$\text{MaxSum}[r][j] = \begin{cases} D[r][j] & r = N \\ \text{Max}(\text{MaxSum}[r+1][j], \text{MaxSum}[r+1][j+1]) + D[r][j] & \text{其他情况} \end{cases}$$

## 数字三角形

### 题解

### 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或右下走。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或右下走。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

如果采用递归的方法, 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。

## 改进



如果每算出一个MaxSum(r,j)就保存起来，下次用到其值的时候直接取用，则可免去重复计算。那么可以用 $O(n^2)$ 时间完成计算。因为三角形的数字总数是  $n(n+1)/2$

### 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; j <= n; ++j)maxsum[n][j]=a[n][j];
    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; j <= n; ++j)maxsum[n][j] = a[n][j];*/
    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

#### 例五、神奇的口袋(百练2755)

- 有一个神奇的口袋，总的容积是40，用这个口袋可以变出一些物品，这些物品的总体积必须是40。
- John现在有 $n$  ( $1 \leq n \leq 20$ ) 个想要得到的物品，每个物品的体积分别是 $a_1, a_2, \dots, a_n$ 。John可以从这些物品中选择一些，如果选出的物体的总体积是40，那么利用这个神奇的口袋，John就可以得到这些物品。现在的问题是，John有多少种不同的选择物品的方式。

### 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

#### 例六、0-1背包问题 (POJ3624)

中国大学MOOC

有N件物品和一个容积为M的背包。第i件物品的体积 $w[i]$ ，价值是 $d[i]$ 。求解将哪些物品装入背包可使价值总和最大。每种物品只有一件，可以选择放或者不放 ( $N \leq 3500, M \leq 13000$ )。

### 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 $a_i$ ，当 $a_1 < a_2 < \dots < a_s$ 的时候，我们称这个序列是上升的。对于给定的一个序列 $(a_1, a_2, \dots, a_N)$ ，我们可以得到一些上升的子序列 $(a_{i_1}, a_{i_2}, \dots, a_{i_K})$ ，这里 $1 \leq i_1 < i_2 < \dots < i_K \leq N$ 。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

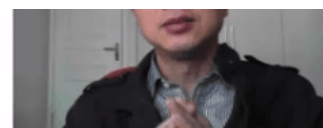


“求以 $a_k$  ( $k=1, 2, 3 \dots N$ ) 为终点的最长上升子序列的长度”

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。



### 3. 找出状态转移方程:



$\text{maxLen}(k)$  表示以  $a_k$  做为“终点”的最长上升子序列的长度那么:

初始状态:  $\text{maxLen}(1) = 1$

$\text{maxLen}(k) = \max \{ \text{maxLen}(i) : 1 \leq i < k \text{ 且 } a_i < a_k \text{ 且 } k \neq 1 \} + 1$

若找不到这样的  $i$ , 则  $\text{maxLen}(k) = 1$

$\text{maxLen}(k)$  的值, 就是在  $a_k$  左边, “终点”数值小于  $a_k$ , 且长度最大的那个上升子序列的长度再加1。因为  $a_k$  左边任何“终点”小于  $a_k$  的子序列, 加上  $a_k$  后就能形成一个更长的上升子序列。

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

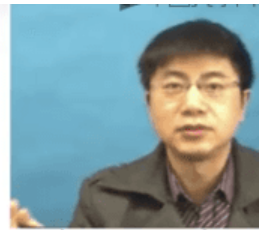
# 最长公共子序列

输入两个串  $s1, s2$ ,

设  $\text{MaxLen}(i, j)$  表示:

$s1$  的左边  $i$  个字符形成的子串, 与  $s2$  左边的  $j$  个字符形成的子串的最长公共子序列的长度 ( $i, j$  从 0 开始算)

$\text{MaxLen}(i, j)$  就是本题的“状态”



假定  $\text{len1} = \text{strlen}(s1), \text{len2} = \text{strlen}(s2)$

那么题目就是要求  $\text{MaxLen}(\text{len1}, \text{len2})$



## 最长公共子序列

显然:

$\text{MaxLen}(n, 0) = 0 \quad (n = 0 \dots \text{len1})$

$\text{MaxLen}(0, n) = 0 \quad (n = 0 \dots \text{len2})$

递推公式:

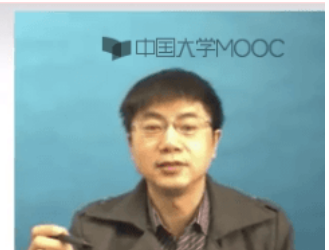
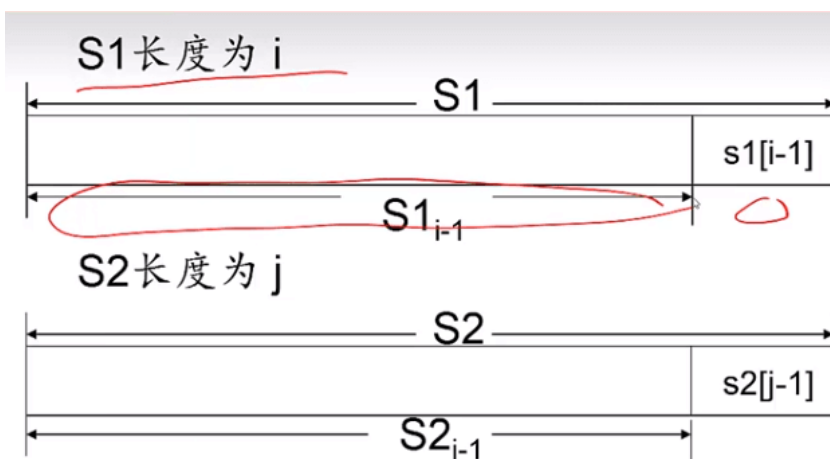
if (  $s1[i-1] == s2[j-1]$  ) //  $s1$  的最左边字符是  $s1[0]$

$\text{MaxLen}(i, j) = \text{MaxLen}(i-1, j-1) + 1;$

else

$\text{MaxLen}(i, j) = \text{Max}(\text{MaxLen}(i, j-1), \text{MaxLen}(i-1, j));$

时间复杂度  $O(mn)$   $m, n$  是两个字符串长度



$s1[i-1] \neq s2[j-1]$  时,  $\text{MaxLen}(S1, S2)$  不会比  $\text{MaxLen}(S1, S2_{j-1})$  和  $\text{MaxLen}(S1_{i-1}, S2)$  两者之中任何一个小, 也不会比两者都大。

code c++

```
#include<iostream>
```

```

#include<String>
#include<vector>
#include<algorithm>
using namespace std;
void main(){
    string s1, s2;
    s1 = "abcfgtd";
    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

假定数字串长度是 $n$ ，添完加号后，表达式的值。一个加号添加在第 $i$ 个数字后面，那么整个表达式的最小值，就等在前 $i$ 个数字中插入 $m-1$ 个加号所能形成的最小值，加上第 $i+1$ 到第 $n$ 个数字所组成的数的值（ $i$ 从1开始算）。

#### 解题思路

设 $V(m,n)$ 表示在 $n$ 个数字中插入 $m$ 个加号所能形成的表达式最小值，那么：

if  $m = 0$

$V(m,n) = n$ 个数字构成的整数

else if  $n < m + 1$

$V(m,n) = \infty$

else

$V(m,n) = \min\{ V(m-1,i) + \text{Num}(i+1,n) \} (i = m \dots n-1)$

$\text{Num}(i,j)$ 表示从第 $i$ 个数字到第 $j$ 个数字所组成的数。数字编号从1开始算。此操作复杂度是 $O(j-i+1)$ ，可以预处理后存起来。

总时间复杂度： $O(mn^2)$

### code c++

```

#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");
}

```