# 目录



青少年信息学编程 河南省教育

# 区间动规

## 1. 区间动态规划

所谓区间动规，顾名思义，就是在一段区间上进行的动态规划。通常由一个二维数组 dp[i][j]表示。一般 i，j 的含义有以下几种：

- 表示从 i 个物品到第 j 个物品的最优值
- 表示从 i 开始，数据规模为 j 时的最优值
- 表示前 i 个物品，分成 j 段时的最优值
- …

区间的 dp 的求解方法一般：

- 不能顺推也不能倒推
- 是以区间长度的大小划分阶段
- 按区间从小到大的顺序划分

## 2. 基础题目

### 2.1. 合并石子

**【问题描述】**

在一个操场上一排地摆放着 N 堆石子。现要将石子有次序地合并成一堆。规定每次只能选相邻的 2 堆石子合并成新的一堆，并将新的一堆石子数记为该次合并的得分。

设计一个程序，计算出将 N 堆石子合并成一堆的最小得分。

**【输入】**

第一行为一个正整数 N（2≤N≤100）；

以下 N 行，每行一个正整数，小于 10000，分别表示第 i 堆石子的个数(1≤i≤N)。

**【输出】**

为一个正整数，即最小得分。

**【样例输入】**

```
7
13
7
8
16
21
4
18
```

**【样例输出】**

```
239
```

**【算法分析】**

s[i]表示前 i 堆石头的数量总和，f[i][j]表示把第 i 堆到第 j 堆的石头合并成一堆的最优值。

码谷教育 青少年信息学编程

```
    for (i=n-1;i>=1;i--)
      for (j=i+1;j<=n;j++)
        for (k=i;k<= j-1;k++)
          f[i][j]=min(f[i][j],f[i][k]+f[k+1][j]+s[j]-s[i-1]);
```

输出 f[1][n]

**【参考代码】**

```
#include <iostream>
#include <cstring>
using namespace std;
int f[101][101];
int a[101],s[101];
int n;
int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
        s[i]=s[i-1]+a[i];
    }
    memset(f,127/3,sizeof(f));//赋值很大的数 ， 若无/3 后面的相加可能超出 int 范围
    for(int i=1;i<=n;i++)
    {
        f[i][i]=0;
    }
    for(int i=n-1;i>=1;i--)
    {
        for(int j=i+1;j<=n;j++)
        {
            for(int k=i;k<=j-1;k++)
            {
                f[i][j]=min(f[i][j],f[i][k]+f[k+1][j]+s[j]-s[i-1]);
            }
        }
    }
    cout<<f[1][n]<<endl;
    return 0;
}
```

## 2.2. 最长公共子序列（P338）

**【问题描述】**

一个给定序列的子序列是在该序列中删去若干元素后得到的序列。确切地说，若给定序列 X=<x1, x2, …, xm>，则另一序列 Z=<z1，z2，…，zk>是 X 的子序列是指存在一个严格递增的下标序列<i1, i2, …, ik>, 使得对于所有 j=1, 2, …, k 有：

Xij=Zj

例如，序列 Z=<B, C, D, B>是序列 X=<A, B, C, B, D, A, B>的子序列，相应的递增下标序列为<2, 3, 5, 7>。给定两个序列 X 和 Y，当另一序列 Z 既是 X 的子序列又是 Y 的子序列时，称 Z 是序列 X 和 Y 的公共子序列。例如，若 X=<A, B, C, B, D, A, B>和 Y=<B, D, C, A, B, A>，则序列

码谷教育 青少年信息学编程

<B,C,A>是 X 和 Y 的一个公共子序列,序列 <B,C,B,A>也是 X 和 Y 的一个公共子序列。而且，后者是 X 和 Y 的一个最长公共子序列．因为 X 和 Y 没有长度大于 4 的公共子序列。

给定两个序列 X＝<x1，x2，…，xm>和 Y=<y1,y2….yn>．要求找出 X 和 Y 的一个最长公共子序列。

**【输入】**

输入文件共有两行。每行为一个由大写字母构成的长度不超过 200 的字符串，表示序列 X 和 Y。

**【输出】**

输出文件第一行为一个非负整数。表示所求得的最长公共子序列的长度。若不存在公共子序列．则输出文件仅有一行输出一个整数 0。否则在输出文件的第二行输出所求得的最长公共子序列(也用一个大写字母组成的字符串表示)。若符合条件的最长公共子序列不止一个，只需输出其中任意一个。

**【样例输入】**

```
ABCBDAB
BDCABA
```

**【样例输出】**

```
4
```

**【提示】**

最长公共子串（Longest Common Substirng）和最长公共子序列（Longest Common Subsequence，LCS）的区别为：子串是串的一个连续的部分，子序列则是从不改变序列的顺序，而从序列中去掉任意的元素而获得新的序列；也就是说，子串中字符的位置必须是连续的，子序列则可以不必连续。字符串长度小于等于 1000。

分析：

与最长不下降子序列(LIS)类似的，我们可以以子序列的结尾作为状态，但现在有两个子序列，那么直接以两个子序列当前的结尾作为状态即可。

① 确定状态：设 F[x][y]表示 S[1..x]与 T[1..y]的最长公共子序列的长度。答案为 F[|S|][|T|]。

② 确定状态转移方程和边界条件：

分三种情况来考虑：

S[x]不在公共子序列中：该情况下 F[x][y]=F[x-1][y]；

T[y]不在公共子序列中：该情况下 F[x][y]=F[x][y-1]；

S[x]=T[y],S[x]与 T[y]在公共子序列中：该情况下,F[x][y]=F[x-1][y-1]+1。

F[x][y]取上述三种情况的最大值。综上：

状态转移方程：F[x][y]=max{F[x-1][y],F[x][y-1],F[x-1][y-1]+1},其中第三种情况要满足 S[x]=T[y]；

边界条件：F[0][y]=0,F[x][0]=0。

③ 程序实现：

计算 F[x][y]时用到 F[x-1][y-1]，F[x-1][y]，F[x][y-1]这些状态，它们要么在 F[x][y]的上一行，要么在 F[x][y]的左边。因此预处理出第 0 行，然后按照

行从小到大、同一行按照列从小到大的顺序来计算就可以用迭代法计算出来。时间复杂度为O(|S|*|T|)。

**【参考代码】**

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
const int MAXN = 5005;
string S,T;
int F[MAXN][MAXN];

int main()
{
    cin >> S;
    cin >> T;
    int ls = S.length(),lt = T.length();
    for(int i = 1;i <= ls;i ++)
    {
        for(int j = 1;j <= lt;j ++)
        {
            F[i][j] = max(F[i - 1][j],F[i][j - 1]);
            if (S[i - 1] == T[j - 1])
            {
                F[i][j] = max(F[i][j],F[i - 1][j - 1] + 1);
            }
        }
    }
    cout << F[ls][lt] << endl;
    return 0;
}
```

## 2.3. 乘积最大

**【问题描述】**

今年是国际数学联盟确定的“2000——世界数学年”，又恰逢我国著名数学家华罗庚先生诞辰90周年。在华罗庚先生的家乡江苏金坛，组织了一场别开生面的数学智力竞赛的活动，你的一个好朋友XZ也有幸得以参加。活动中，主持人给所有参加活动的选手出了这样一道题目：

设有一个长度为N的数字串，要求选手使用K个乘号将它分成K+1个部分，找出一种分法，使得这K+1个部分的乘积最大。

同时，为了帮助选手能够正确理解题意，主持人还举了如下的一个例子：

有一个数字串：312， 当N=3，K=1时会有以下两种分法：

1）3*12=36

2）31*2=62


咨询电话：18561566921

这时，符合题目要求的结果是：31*2=62。

现在，请你帮助你的好朋友 XZ 设计一个程序，求得正确的答案。

**【输入】**

第一行共有 2 个自然数 N，K（6≤N≤40，1≤K≤6）

第二行是一个长度为 N 的数字串。

**【输出】**

输出所求得的最大乘积（一个自然数）。

**【算法分析】**

此题满足动态规划法的求解标准，我们把它按插入的乘号数来划分阶段，若插入 K 个乘号，可把问题看做是 K 个阶段的决策问题。设 f[i][k]表示在前 i 位数中插入 K 个乘号所得的最大值，a[j][i]表示从第 j 位到第 i 位所组成的自然数。用 f[i][k]存储阶段 K 的每一个状态，可以得状态转移方程：

f[i][k]=max{f[j][k-1]*a[j+1][i]}（k<=j<=i）

边界值：

f[j][0]=a[1][j]（1<j<=i）

根据状态转移方程，我们就很容易写出动态规划程序：

```
for(k=1;k<=r;k++)          //r 为乘号个数
  for( i=k+1;i<=n;i++)
    for (j=k;j<i;j++)
      if (f[i][k]<f[j][k-1]*a[j+1][i]) f[i][k]=f[j][k-1]*a[j+1][i]
```

**【样例输入】**

```
4 2
1231
```

**【样例输出】**

```
62
```

**【参考代码】**

```
#include <cstdio>
#include <iostream>
#include <iomanip>
using namespace std;
long long a[11][11],f[11][11];
long long s;
int n,i,p,k,j;
int main()
{
    cin>>n>>k;
    cin>>s;
    for(int i=n;i>=1;i--)
    {
        a[i][i]=s%10;
        s=s/10;
    }
    for(int i=2;i<=n;i++)
    {
        for(int j=i-1;j>=1;j--)
        {
            a[j][i]=a[j][i-1]*10+a[i][i];
        }
    }
```


咨询电话：18561566921

```
    for(int i=1;i<=n;i++)
    {
        f[i][0]=a[1][i];
    }
    for(int p=1;p<=k;p++)
    {
        for(int i=p+1;i<=n;i++)
        {
            for(int j=p;j<i;j++)
            {
                f[i][p]=max(f[i][p],f[j][p-1]*a[j+1][i]);
            }
        }
    }
    cout<<f[n][k]<<endl;
    return 0;
}
```

## 2.4. 编辑距离

**【问题描述】**

设 A 和 B 是两个字符串。我们要用最少的字符操作次数，将字符串 A 转换为字符串 B。这里所说的字符操作共有三种：

1、删除一个字符；

2、插入一个字符；

3、将一个字符改为另一个字符。

对任的两个字符串 A 和 B，计算出将字符串 A 变换为字符串 B 所用的最少字符操作次数。

**【输入】**

第一行为字符串 A；第二行为字符串 B；字符串 A 和 B 的长度均小于 200。

**【输出】**

只有一个正整数，为最少字符操作次数。

**【样例输入】**

```
sfdqxbw
gfdgw
```

**【样例输出】**

```
4
```

**【算法分析】**

状态：f[i][j]记录 ai 与 bj 的最优编辑距离

结果：f[m][n]，其中 m、n 分别是 a、b 的串长

初值：b 串空，要删 a 串长个字符；a 串空，要插 b 串长个字符

转移方程：当 a[i]==b[j]时，f[i][j]=f[i-1][j-1]，

否则，f[i][j]=min(f[i-1][j-1]+1,f[i][j-1]+1,f[i-1][j]+1)

说明：f[i-1][j-1]+1：改 a[i]为 b[j]；

f[i][j-1]+1：a[i]后插入 b[j-1]；

f[i-1][j]+1：删 a[i]。

**【参考代码】**

```
#include <iostream>
```


咨询电话：18561566921

```
#include <cstring>
using namespace std;
char s1[210],s2[210];
int f[210][210];
int main()
{
    cin>>s1>>s2;
    int m=strlen(s1);
    int n=strlen(s2);
    for(int i=1;i<=m;i++)
    {
        f[i][0]=i;
    }
    for(int j=1;j<=n;j++)
    {
        f[0][j]=j;
    }
    for(int i=1;i<=m;i++)
    {
        for(int j=1;j<=n;j++)
        {
            if(s1[i-1]==s2[j-1])
            {
                f[i][j]=f[i-1][j-1];
            }
            else
            {
                f[i][j]=min(min(f[i-1][j],f[i][j-1]),f[i-1][j-1])+1;
            }
        }
    }
    cout<<f[m][n]<<endl;
    return 0;
}
```

## 2.5. 方格取数

**【问题描述】**

设有N×N的方格图，我们在其中的某些方格中填入正整数，而其它的方格中则放入数字0。如下图所示：

A

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 13 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 7 | 0 | 0 | 0 |
| 0 | 0 | 0 | 14 | 0 | 0 | 0 | 0 |
| 0 | 21 | 0 | 0 | 0 | 4 | 0 | 0 |
| 0 | 0 | 15 | 0 | 0 | 0 | 0 | 0 |
| 0 | 14 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

B

码谷教育 青少年信息学编程

某人从图中的左上角的 A 出发，可以向下行走，也可以向右行走，直到达右下角的 B 点。在走过的路上，他可以取走方格中的数(取走后的方格中将变为数字 0)。

此人从 A 点到 B 点共走了两次，试找出两条这样的路径，使得取得的数字和为最大。

**【输入】**

输入文件 Pane.in 第一行为一个整数 N（N≤10），表示 N×N 的方格图。

接下来的每行有三个整数，第一个为行号数，第二个为列号数，第三个为在该行、该列上所放的数。一行 0 0 0 表示结束。

**【输出】**

输出文件 Pane.out 包含一个整数，表示两条路径上取得的最大的和。

**【样例输入】**

```
8
2 3 13
2 6 6
3 5 7
4 4 14
5 2 21
5 6 4
6 3 15
7 2 14
0 0 0
```

**【样例输出】**

```
67
```

**【算法分析】**

一个四重循环枚举两条路分别走到的位置。由于每个点均从上或左继承而来，故内部有四个 if，分别表示两个点从上上、上左、左上、左左继承来时，加上当前两个点所取得的最大值。a[i][j]表示(i,j)格上的值，sum[i][j][h][k]表示第一条路走到(i,j)，第二条路走到(h,k)时的最优解。例如，sum[i][j][h][k] = max{sum[i][j][h][k], sum[i-1][j][h-1][k] + a[i][j] + a[h][k]}，表示两点均从上面位置走来。

当(i,j) ◇ (h,k))时

sum[i][j][h][k]:=max{sum[i-1][j][h-1][k],sum[i][j-1][h][k-1],sum[i-1][j][h][k-1],sum[i][j-1][h-1][k]}+a[i][j]+a[h][k];

当(i,j) = (h,k)时

sum[i][j][h][k]:=max{sum[i-1][j][h-1][k],sum[i][j-1][h][k-1],sum[i-1][j][h][k-1],sum[i][j-1][h-1][k]}+a[i][j];

**【参考代码】**

```
#include <iostream>
using namespace std;
int a[11][11];
int s[11][11][11][11];
int n,x,y,z;
int main()
{
    cin>>n;
    while(1)
    {
    cin>>x>>y>>z;
    if(x==0 && y==0 && z==0)
    {
        break;
```



咨询电话：18561566921

```
    }
    a[x][y]=z;
    }
    for(int i=1;i<=n;i++)
    {
    for(int j=1;j<=n;j++)
    {
        for(int h=1;h<=n;h++)
        {
            for(int k=1;k<=n;k++)
            {
                int t1=max(s[i-1][j][h-1][k],s[i][j-1][h][k-1]);
                int t2=max(s[i-1][j][h][k-1],s[i][j-1][h-1][k]);
                s[i][j][h][k]=max(t1,t2)+a[i][j];
                if(i!=h && j!=k)
                {
                    s[i][j][h][k]+=a[h][k];
                }
            }
        }
    }
    }
    cout<<s[n][n][n][n]<<endl;
    return 0;
}
```

**【算法分析 2】**

dp[i][j][k]代表走 i 步第一条在 j 列，第二条在 k 列的最大和。

dp[i][j][k]=max{dp[i-1][j-1][k-1],dp[i-1][j-1][k],dp[i-1][j][k-1],dp[i-1][j][k]}+当前的第一个点和第二个点的值（若相同只加一次）；

可以用滚动数组优化一下，这里N很小，可以不考虑。

**【参考代码 2】**

```
#include <bits/stdc++.h>
using namespace std;
int a[10][10];
int dp[20][10][10];
int f(int a,int b,int c,int d)
{
    return max(max(max(a,b),c),d);
}
int main()
{
    int n,x,y,z;
    cin>>n;
    while(1)
    {
    cin>>x>>y>>z;
    if(x==0 && y==0 && z==0)
    {
        break;
    }
    a[x][y]=z;
    }
    for(int k=1;k<=2*n;k++)//走 2*n 步到达右下角
    {
```


咨询电话：18561566921

```
    for(int i=1;i<=k;i++)
    {
        for(int j=1;j<=k;j++)
          {

dp[k][i][j]=f(dp[k-1][i][j],dp[k-1][i-1][j],dp[k-1][i][j-1],dp[k-1][i-1][j-1]
);
              if(i==j) //相同
                {
                    dp[k][i][j]+=a[k-i][i];
                }
              else
                {
                    dp[k][i][j]+=a[k-i][i]+a[k-j][j];
                }
          }
    }
    }
    cout<<dp[2*n][n][n];
}
```

## 2.6. 复制书稿

**【问题描述】**

现在要把 m 本有顺序的书分给 k 个人复制（抄写），每一个人的抄写速度都一样，一本书不允许给两个（或以上）的人抄写，分给每一个人的书，必须是连续的，比如不能把第一、第三和第四本书给同一个人抄写。

现在请你设计一种方案，使得复制时间最短。复制时间为抄写页数最多的人用去的时间。

**【输入】**

第一行两个整数 m，k；（k≤m≤500）

第二行 m 个整数，第 i 个整数表示第 i 本书的页数。

**【输出】**

共 k 行，每行两个整数，第 i 行表示第 i 个人抄写的书的起始编号和终止编号。k 行的起始编号应该从小到大排列，如果有多解，则尽可能让前面的人少抄写。

**【样例输入】**

```
9 3
1 2 3 4 5 6 7 8 9
```

**【样例输出】**

```
1 5
6 7
8 9
```

**【算法分析】**

本题可以用动态规划解决，设 f(k,m)为前 m 本书交由 k 个人抄写，需要的最短时间，则状态转移方程为

$$f[k][m]=\min\{\max\{f[k-1][i],\ \sum_{j=i+1}^{m}T_j\}\},\ i=1,\ 2,\ \cdots,\ m-1\}$$

动态规划求出的仅仅是最优值，如果要输出具体方案，还需根据动态规划计算得到的最优值，做一个


咨询电话：18561566921

贪心设计。具体来说，设最优值为 T，那么 k 个人，每个人抄写最多 T 页。从最后一本书开始按逆序将书分配给 k 人去抄写，从第 k 个人开始，如果他还能写，就给他；否则第 k 个人工作分配完毕，开始分配第 k-1 个人的工作；以后再是第 k-2 个、第 k-3 个、……直至第 1 个。一遍贪心结束后，具体的分配方案也就出来了。

**【参考代码】**

```
#include <iostream>
using namespace std;
int a[510],f[510][510],s[510];
int m,k;
void prt(int i,int j)//递归输出抄写页数的方案
{
    int t,x;
    if(j==0)
    {
        return;
    }
    if(j==1)
    {
        cout<<1<<" "<<i<<endl;
        return;
    }
    t=i;
    x=a[i];
    while(x+a[t-1]<=f[k][m])
    {
        x+=a[t-1];
        t--;
    }
    prt(t-1,j-1);
    cout<<t<<" "<<i<<endl;
}
int main()
{
    cin>>m>>k;
    for(int i=0;i<=500;i++)
    {
        for(int j=0;j<=500;j++)
        {
            f[i][j]=10000000;
        }
    }
    for(int i=1;i<=m;i++)
    {
        cin>>a[i];
        s[i]=s[i-1]+a[i];
        f[1][i]=s[i];
    }
    for(int i=2;i<=k;i++)
    {
        for(int j=1;j<=m;j++)
        {
            for(int p=1;p<=j-1;p++)
            {
                if(f[i][j]>max(f[i-1][p],s[j]-s[p]))
                {
```

码谷教育 青少年信息学编程

```
                    f[i][j]=max(f[i-1][p],s[j]-s[p]);
                }
            }
        }
    }
    prt(m,k);
    return 0;
}
```

## 2.7. 橱窗布置

**【问题描述】**

假设以最美观的方式布置花店的橱窗，有 F 束花，每束花的品种都不一样，同时，至少有同样数量的花瓶，被按顺序摆成一行，花瓶的位置是固定的，并从左到右，从 1 到 V 顺序编号，V 是花瓶的数目，编号为 1 的花瓶在最左边，编号为 V 的花瓶在最右边，花束可以移动，并且每束花用 1 到 F 的整数惟一标识，标识花束的整数决定了花束在花瓶中列的顺序即如果 I<J，则花束 I 必须放在花束 J 左边的花瓶中。

例如，假设杜鹃花的标识数为 1，秋海棠的标识数为 2，康乃馨的标识数为 3，所有的花束在放人花瓶时必须保持其标识数的顺序，即：杜鹃花必须放在秋海棠左边的花瓶中，秋海棠必须放在康乃馨左边的花瓶中。如果花瓶的数目大于花束的数目，则多余的花瓶必须空，即每个花瓶中只能放一束花。

每一个花瓶的形状和颜色也不相同，因此，当各个花瓶中放人不同的花束时会产生不同的美学效果，并以美学值(一个整数)来表示，空置花瓶的美学值为 0。在上述例子中，花瓶与花束的不同搭配所具有的美学值，可以用如下表格表示。

根据表格，杜鹃花放在花瓶 2 中，会显得非常好看，但若放在花瓶 4 中则显得很难看。

为取得最佳美学效果，必须在保持花束顺序的前提下，使花的摆放取得最大的美学值，如果具有最大美学值的摆放方式不止一种，则输出任何一种方案即可。题中数据满足下面条件：1≤F≤100，F≤V≤100，－50≤AIJ≤50，其中 Aij 是花束 I 摆放在花瓶 J 中的美学值。输入整数 F，V 和矩阵(AIJ)，输出最大美学值和每束花摆放在各个花瓶中的花瓶编号。

| | 花瓶1 | 花瓶2 | 花瓶3 | 花瓶4 | 花瓶5 |
|---|---|---|---|---|---|
| 杜鹃花 | 7 | 23 | -5 | -24 | 16 |
| 秋海棠 | 5 | 21 | -4 | 10 | 23 |
| 康乃馨 | -21 | 5 | -4 | -20 | 20 |

**【数据范围】**

1≤F≤100，其中 F 为花束的数量，花束编号从 1 至 F。

F≤V≤100，其中 V 是花瓶的数量。

-50≤Aij≤50，其中 $A_{ij}$ 是花束 i 在花瓶 j 中的美学值。

**【输入】**

第一行包含两个数：F，V。

随后的 F 行中，每行包含 V 个整数，$A_{ij}$ 即为输入文件中第（i+1 ）行中的第 j 个数。

**【输出】**

输出文件必须是名为 flower.out 的正文文件，文件应包含两行：

第一行是程序所产生摆放方式的美学值。

第二行必须用 F 个数表示摆放方式，即该行的第 K 个数表示花束 K 所在的花瓶的编号。

**【样例输入】**


咨询电话：18561566921

```
3 5
7 23 -5 -24 16
5 21 -4 10 23
-21 5 -4 -20 20
```

**【样例输出】**

```
53
2 4 5
```

**【解法 1】**

**【算法分析】**

问题实际就是给定 F 束花和 V 个花瓶，以及各束花放到不同花瓶中的美学值，要求你找出一种摆放的方案，使得在满足编号小的花放进编号小的花瓶中的条件下，美学值达到最大。

将问题进行转化，找出问题的原型。首先，看一下上述题目的样例数据表格。

将摆放方案的要求用表格表现出来，则摆放方案需要满足：每行选且只选一个数(花瓶)；摆放方案的相邻两行中，下面一行的花瓶编号要大于上面一行的花瓶编号两个条件。这时可将问题转化为：给定一个数字表格，要求编程计算从顶行至底行的一条路径，使得这条路径所经过的数字总和最大(要求每行选且仅选一个数字)。同时，路径中相邻两行的数字，必须保证下一行数字的列数大于上一行数字的列数。

看到经过转化后的问题，发现问题与“数学三角形”问题十分相似，数字三角形问题的题意是：

给定一个数字三角形，要求编程计算从顶至底的一条路径，使得路径所经过的数字总和最大(要求每行选且仅选一个数字)。同时，路径中相邻两行的数字，必须保证下一行数字的列数与上一行数字的列数相等或者等于上一行数字的列数加 1。

上例中已经知道：数字三角形中的经过数字之和最大的最佳路径，路径的每个中间点到最底层的路径必然也是最优的，可以用动态规划方法求解，对于“花店橱窗布置”问题经过转化后，也可采取同样的方法得出本题同样符合最优性原理。因此，可以对此题采用动态规划的方法。

**【参考代码】**

```
#include <iostream>
#include <cstring>
using namespace std;
int a[110][110],b[110][110],c[110][110],d[110];
int f,v,k,mmax=-2100000000;
int main()
{
    cin>>f>>v;
    for(int i=1;i<=f;i++)
    {
        for(int j=1;j<=v;j++)
        {
            cin>>a[i][j];
        }
    }
    memset(b,128,sizeof(b));
    for(int i=1;i<=v-f+1;i++)//初始化第 1 束花放到第 i 个花瓶的情况
    {
        b[1][i]=a[1][i];
    }
    for(int i=2;i<=f;i++)
    {
        for(int j=i;j<=v-f+i;j++)
        {
            for(int k=i-1;k<=j-1;k++)//枚举花束 i-1 的位置
```


咨询电话：18561566921

```
            {
                if(b[i-1][k]+a[i][j]>b[i][j])
                {
                    b[i][j]=b[i-1][k]+a[i][j];
                    c[i][j]=k;//记录前一个花束的位置为 k
                }
            }
        }
    }
    for(int i=f;i<=v;i++)
    {
        if(b[f][i]>mmax)
        {
            mmax=b[f][i];//选择全局最优解
            k=i;//k 记录最后一束花的位置
        }
    }
    cout<<mmax<<endl;
    for(int i=1;i<=f;i++)
    {
        d[i]=k;
        k=c[f-i+1][k];
    }
    for(int i=f;i>=2;i--)
    {
        cout<<d[i]<<" ";
    }
    cout<<d[1]<<endl;
    return 0;
}
```

由此可看出，对于看似复杂的问题，通过转化就可变成简单的经典的动态规划问题。在问题原型的基础上，通过分析新问题与原问题的不同之处，修改状态转移方程，改变问题状态的描述和表示方式，就会降低问题规划和实现的难度，提高算法的效率。由此可见，动态规划问题中具体的规划方法将直接决定解决问题的难易程度和算法的时间与空间效率，而注意在具体的规划过程中的灵活性和技巧性将是动态规划方法提出的更高要求。

## 2.8. 滑雪

**【问题描述】**

小明喜欢滑雪，因为滑雪的确很刺激，可是为了获得速度，滑的区域必须向下倾斜，当小明滑到坡底，不得不再次走上坡或等着直升机来载他，小明想知道在一个区域中最长的滑坡。滑坡的长度由滑过点的个数来计算，区域由一个二维数组给出，数组的每个数字代表点的高度。下面是一个例子：

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 16 | 17 | 18 | 19 | 6 |
| 15 | 24 | 25 | 20 | 7 |
| 14 | 23 | 22 | 21 | 8 |
| 13 | 12 | 11 | 10 | 9 |

一个人可以从某个点滑向上下左右相邻四个点之一，当且仅当高度减小，在上面的例子中，一条可行的滑坡为 25-24-17-16-1（从 25 开始到 1 结束），当然 25-24……2…1 更长，事实上这是最长的一条。


咨询电话：18561566921

**【输入】**

输入的第一行为表示区域的二维数组的行数 R 和列数 C（1≤R、C≤100）下面是 R 行，每行有 C 个数代表高度。

**【输出】**

输出区域中最长的滑坡长度。

| | | |
|---|---|---|
| | [i-1, j]↓ | |
| [i, j-1]→ | [i, j] | ←[i, j+1] |
| | [i+1, j]↑ | |

**【样例输入】**

```
5      5
1      2      3      4      5
16     17     18     19     6
15     24     25     20     7
14     23     22     21     8
13     12     11     10     9
```

**【样例输出】**

```
25
```

**【算法分析 dfs 思路】**

典型的搜索思路，由于一个人可以从某个点滑向上下左右相邻四个点之一，如上图所示。当且仅当高度减小，对于任意一个点[i, j]，当它的高度小于与之相邻的四个点（[i-1, j]，[i, j+1]，[i+1, j]，[i, j-1]）的高度时，这四个点可以滑向[i, j]，用 f[i][j]表示到[i, j]为止的最大长度，则 f[i][j]=max{f[i+a][j+b]}+1，其中坐标增量{(a, b)=[(1, 0), (-1, 0), (0, 1), (0, -1)], 0<i+a<=r, 0<j+b<=c, High[i][j]<High[i+a][j+b]}。为了避免重复，提高效率，可以引入记忆化搜索。

**【参考代码】**

```
#include <iostream>
#include <cstdio>
using namespace std;
int dx[5]={0,-1,0,1,0},dy[5]={0,0,1,0,-1};
long m[110][110],f[110][110];
int r,c,ans=0;
int search(int x,int y)
{
    int i,t,tmp,nx,ny;
    if(f[x][y]>0)
    {
        return f[x][y];
    }
    t=1;
    for(int i=1;i<=4;i++)
    {
        nx=x+dx[i];
        ny=y+dy[i];
        if(nx>=1 && nx<=r && ny>=1 && ny<=c && m[x][y]<m[nx][ny])
        {
            tmp=search(nx,ny)+1;
            if(tmp>t)
            {
                t=tmp;
            }
        }
    }
```

码谷教育 青少年信息学编程

```
    f[x][y]=t;
    return t;
}
int main()
{
    cin>>r>>c;
    ans=0;
    for(int i=1;i<=r;i++)
    {
        for(int j=1;j<=c;j++)
        {
            cin>>m[i][j];
        }
    }
    for(int i=1;i<=r;i++)
    {
        for(int j=1;j<=c;j++)
        {
            int t=search(i,j);
            f[i][j]=t;
            if(t>ans)
            {
                ans=t;
            }
        }
    }
    cout<<ans<<endl;
    return 0;
}
```

**【算法分析 dp 思路】**

动态规划方法，为了保证满足条件的 f[i+a][j+b]在 f[i][j]前算出，需要对高度排一次序，然后从大到小规划（高度）。最后再比较一下所有 f[i][j]{0<i≤r,0<j≤c}，找出其中最长的一条路线。我们还可以用记忆化搜索的方法，它的优点是不需进行排序，按照行的顺序，利用递归逐点求出区域中到达此点的最长路径，每个点的最长路径只求一次。

**【参考代码】**

```
#include <iostream>
#include <algorithm>
#define Max 105
using namespace std;
struct node
{
    int r,c,h;
};
//排序规则，将结构从大到小排序
int cmp(const node &p1,const node &p2)
{
    return p1.h>p2.h;
}
node a[Max*Max];//储存每一个节点，按高度排序后将二维数组化成了一维数组
int R,C,dp[Max*Max];//R 表示行，R 表示列，dp[i]表示第 i 个点对应的最长区域的长度
int pre[Max*Max];//记录每个点的路径
int maxn=0,maxn_i=R*C;//记录最长路径对应的点的值和编号
bool isClose(node a,node b)//判断两个点是否相连
{
    int dir[4][2]={{1,0},{-1,0},{0,-1},{0,1}};
```

码谷教育 青少年信息学编程

```
    for(int i=0;i<4;i++)
     {
        if((a.r+dir[i][0]==b.r)&&(a.c+dir[i][1]==b.c))
         {
            return true;
        }
    }
    return false;
}
void print1(int i)//递归方法打印路径
{
    if(i==0)
     {
        return ;
    }
    else
     {
        print1(pre[i]);
        cout<<a[i].r<<","<<a[i].c<<"-->";
    }
}
void print2(int i)//递推方法打印路径
{
    while(true){
        if(i==0){
            break;
        }
        else{
            cout<<"-->"<<a[i].r<<","<<a[i].c;
            i=pre[i];
        }
    }
}
int main()
{
    cin>>R>>C;
    int num=1;
    for(int i=1;i<=R;i++)
     {
        for(int j=1;j<=C;j++)
         {
            a[num].r=i;
            a[num].c=j;
            cin>>a[num++].h;
        }
    }
    sort(a+1,a+R*C+1,cmp);//对每个点按高度进行排序
    for(int i=1;i<=R*C;i++)
     {
        dp[i]=1;
    }
    //寻找最长连续下降子序列
    for(int i=1;i<=R*C;i++)
     {
        for(int j=1;j<i;j++)
         {
            if(isClose(a[i],a[j]))
              {
```

码谷教育 青少年信息学编程

```
                if(dp[j]+1>=dp[i])
                   {
                    dp[i]=dp[j]+1;
                    pre[i]=j;
                }
            }
        }
    }
    //路径最长的点并不是一定从最后面出来的那个点，所以 cout<<dp[R*C]<<endl;是不对的
    //还要找到 dp 里面那个最大的值，和那个值对应的编号
    for(int i=1;i<=R*C;i++)
    {
        if(dp[i]>maxn)
         {
           maxn=dp[i];
           maxn_i=i;
        }
    }
    cout<<maxn<<endl;
    print1(pre[maxn_i]);
    cout<<a[maxn_i].r<<","<<a[maxn_i].c<<endl;//输出第一个点
    return 0;
}
```

码谷教育 青少年信息学编程