动态规划（一）

一、动态规划

1、什么是动态规划

动态规划是一种用于求解包含重叠子问题的最优化问题的方法。

其基本思想是，将原问题分解为相似的子问题，在求解的过程中通过子问题的解求出原问题的解。

2、动态规划解决问题的一般步骤：

(1)确定状态：

状态的参数一般有

1）描述位置的：前(后)i 单位，第 i 到第 j 单位，坐标为(i,j)等

2）描述数量的：取 i 个，不超过 i 个，至少 i 个等

3）描述对后有影响的：状态压缩的，一些特殊的性质

(2)转移方程

1）检查参数是否足够；

2）分情况：最后一次操作的方式，取不取，怎么样放，前一项是什么

3）初始条件是什么。

4）注意无后效性。比如说，求 A 就要求 B，求 B 就要求 C，而求 C 就要求 A，这就不符合无后效性了。

(3)编程实现方式

1）递推（注意顺序：逆推、顺推）

2）记忆化搜索（一般在状态的拓朴顺序不很明确时使用）

二、例题解析

例 1：数字三角形 P2043

问题：给出一个数字三角形，从第一层走到最后一层，每次向左下或右下走，求路径的最大权值和。

（1） 状态定义：

设 f[i][j]表示从第 i 行第 j 个点到底部的路径权值和的最大值

（2） 状态转移方程

f[i][j]= max(f[i+1][j],f[i+1][j+1]) +a[i][j]

意义：位置（i,j）到底部的最大值为下一行的两个位置的最大值+当前的数字。

（3） 实现

```
for (int i=1; i<=n; i++)// 初始化边界
    f[n][i]=a[n][i];
// 转移顺序很重要
for (int i=n-1; i>0; i--)
    for (int j=1; j<=i; j++)
        dp[i][j] = max(dp[i+1][j+1],dp[i+1][j]) + a[i][j];
```

例 2：数字三角形 II P2044

问题：给出一个数字三角形，从第一层走到最后一层，每次向左下或右下走，求路径的最大权值和，其中运行某一行可以任意跳。

（1）状态定义：

由于本题比上一题多了一个条件，可以任意跳，所以在状态设定时应该包含这一信息，

于是可以增加一个维度，记 f[i][j][k]为以(i,j)为顶点的子问题的解，k=1 表示可以随意跳，k=0 表示不能随意跳。原问题的解即为 f[1][1][1] 。

（2）状态转移方程（写出）

意义：

（3）实现

小结，状态定义时需要考虑后效性，动态规划要求状态转移无后效性，即当前状态的答案是过去的完美总结，当前的决策并不会对过去产生影响。

例 3：数字三角形 III P2045

题意：给出一个数字三角形，从第一层走到最后一层，每次向左下或右下走，求路径的权值和的**个位数**的最大值。

（1）状态定义：

由于计算的是个位最大，那么按前面的状态定义就不合适了，因为我们发现当前的选择不一定是下一行的位置的选择。可以这样定义，记 f[i][j][k]为，以(i,j)为起点走到底部，路径权值和的个位数是否有可能等于 k。

（2）状态转移方程（写出）

对于每一个(i,j)，枚举 k，遍历所有状态即可。

意义：

（3）实现

例 4：滑雪

为了获得速度，滑雪的路径必须向下倾斜，每次可以向上下左右 4 个方向滑行。区域由一个二维数组给出，每个数字代表该点的高度。求滑行的最长距离。

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

【解析】很容易想到状态定义方法：dp[i][j]表示 i 行 j 列为起点的滑行最远距离。

状态转移方程

dp[i][j]=max{dp[i][j],dp[i+dx[k]][j+dy[k]]+1}   0<=k<4

由于，这里的状态与状态之间的转移并不像前面的例题，有一个严格的线性关系，于是没办法采取递推的方式来实现。对于这样非线性的关系，可以采取记忆化搜索来实现。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
const int N=110;
```

```
int f[N][N],a[N][N];
int n,m;
int dx[4]={1,-1,0,0};
int dy[4]={0,0,1,-1};
int maxn=1,x,y;
int dfs(int x,int y){
    if(f[x][y]!=-1) return f[x][y];
    f[x][y]=1;
    for(int i=0;i<4;i++){
        int tx=x+dx[i],ty=y+dy[i];
        if(tx>0&&tx<=n&&ty>0&&ty<=m&&a[x][y]>a[tx][ty])
            f[x][y]=max(f[x][y],dfs(tx,ty)+1);
    }
    return f[x][y];
}
int main(){
    memset(f,-1,sizeof(f));
    cin>>n>>m;
    for(int i=1;i<=n;i++)
       for(int j=1;j<=m;j++) cin>>a[i][j];
    for(int i=1;i<=n;i++)
       for(int j=1;j<=m;j++)
         maxn=max(dfs(i,j),maxn);
    cout<<maxn;
    return 0;
}
```

小结：动态规划求解问题，主要在于状态定义与状态转移方程，列出状态转移方程时，需要同时将初值与边界列出，然后再考虑如何求解。

（1）如何利用转移方程求解

递推

递归（记忆化搜索）

求解通项公式

（2）如何看待记忆化

避免大量重复计算

简洁明了，方便理解

递推比较繁琐，或没有明确的依赖顺序（图）

例 5：最长上升子序列 P2046

给定一个长度为 N 的整数序列 A

找到一组最长的整数序列 x 满足：

$1 \leqslant x1 < x2 < \cdots < xk \leqslant N$

$A[x1] < A[x2] < \cdots < A[xk]$

即寻找 A 的一个最长子序列，满足:该子序列中每个元素递增

【解析】

- 状态定义：
  - F[i] 表示以元素 A[i]结尾的最长上升子序列的最大长度。
- 状态转移方程为：
  - F[i]=Max{F[j]}+1，满足 1≤j<i≤n 且 A[j]<A[i]
  - 边界条件：f[i]=1
  - Answer=max{f[i]},(1≤i≤n)

【实现】

```
for(i=1;i<=n;i++)f[i]=1;
for(i=2;i<=n;i++)
     for(j=1;j<=i-1;j++)
          if(a[j]<a[i]&&f[j]+1>f[i])f[i]=f[j]+1;
for(i=1;i<=n;i++)
      if(f[i]>max)max=f[i];
```

例 6：合唱队形 P2048

N 位同学站成一排，音乐老师要请其中的(N-K)位同学出列，使得剩下的 K 位同学排成合唱队形。

合唱队形是指这样的一种队形：设 K 位同学从左到右依次编号为 1，2…，K，他们的身高分别为 T_1，T_2，…，T_K， 则他们的身高满足存在 i（1<=i<=K）使得 $T_1<T_2<......<T_{i-1}<T_i>T_{i+1}>......>T_K$

你的任务是，已知所有 N 位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。

【解析】

问题简述：在一条直线上的 n 个人，要找出某个最长的队列，其中队列中最高的人，他左边的人的高度呈升序排列，他右边的人的高度呈降序排列

思路点拨：一种比较直观的想法是：在队列中枚举那个最高的人，然后从该人开始对左边求最长上升序列，假设长度为 len1，对右边求最长下降序列，假设长度为 len2，那么答案就是 len=len1+len2。很明显还需要枚举最高的人，因此总的时间复杂度为 O(n^3)

进一步分析：其实在做枚举时，左边求最长上升序列，右边倒着求最长上升序列，也可以这样说，从左至右求一遍最长上升，再从右至左求一遍最长上升，求完后，任取某个元素作为最高者，都会符合合唱队形。因此只需要对所有这样的合唱队形找出一个最大数值即可

例 7：机器分配 P2063

总公司拥有高效设备 M 台，准备分给下属的 N 个分公司。各分公司若获得这些设备，可以为国家提供一定的盈利。问：如何分配这 M 台设备才能使国家得到的盈利最大？求出最大盈利值。其中 M≤15，N≤10。分配原则：每个公司有权获得任意数目的设备，但总台数不超过设备数 M。

【解析】

直接根据问题定义状态 dp[i][j] 表示前 i 个公司分配 j 台设备的最大利益

状态转移方程：

f[i][j]=max {dp[i-1][j-k] + val[i][k] },0<=k<=j

表示第 i 个公司分 k 台，前面 i-1 个公司分 j-k 台

例 8：求最长公共子序列 P2047

给定的字符序列 X =“x_0，x_1，…，x_{m-1}”，序列 Y=“y_0，y_1，…，y_{k-1}”是 X 的子序列，

存在 X 的一个严格递增下标序列< i_0，i_1，…，i_{k-1} >，使得对所有的 j=0，1，…，k-1，有 $x_{ij} = y_j$。

例如，X="ABCBDAB"，Y="BCDB"是 X 的一个子序列。

给出两个字符串 S1 和 S2，长度不超过 5000

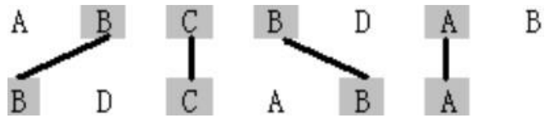
求这两个字符串的最长公共子串长度。

【解析】

样例解析：

S1="ABCBDAB"

S2="BDCABA"



可以看出他们的最长公共子串有 BCBA,BDAB，长度为 4

从样例分析，我们思考的方式为要找出 S1 串与 S2 串的公共子串，假设将 S1 固定，从第 1 个位置开始直到最后一个位置为止，与 S2 的各个部分不断找最长公共子串

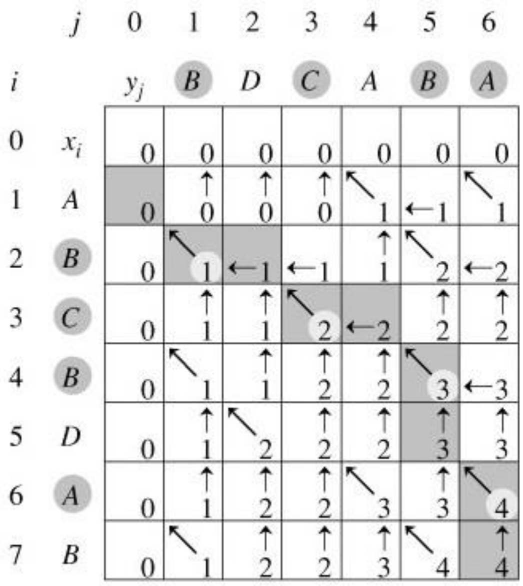
当然 S1 也可以变化，这样我们即得出了思路：

枚举 S1 的位置 i

枚举 S2 的位置 j

找出 S1 的前 i 位与 S2 的前 j 位的最长公共子串，直到两个串的最后一个位置为止。

状态转移表：



状态定义：设 f[i,j]表示 S 的前 i 位与 T 的前 j 位的最长公共子串长度。

转移方程：

$$f[i,j]=\begin{cases}\max\{f[i-1,j],f[i,j-1]\}, & S[i]\neq T[j]\\ f[i-1,j-1]+1 & S[i]=T[j]\end{cases}$$

时间复杂度 O(n*m)

【参考代码】

```
char x[210],y[210];
int dp[210][210]={0};
int dfs(int i,int j){
    if(i==0||j==0)return dp[i][j]=0;
    if(dp[i][j]!=-1)return dp[i][j];
```

```
    if(x[i-1]==y[j-1])
        dp[i][j]=dfs(i-1,j-1)+1;
    else
        dp[i][j]=max(dfs(i-1,j) ,dfs(i,j-1));
    return dp[i][j];
}
void write(int i,int j){
    if(j<=0||i<=0)return ;
    if(dp[i][j]==dp[i-1][j])
            write(i-1,j);
    else if(dp[i][j]==dp[i][j-1])
            write(i,j-1);

    else{
        write(i-1,j-1);
        cout<<x[i-1];
    }
}
int main(){
    memset(dp,-1,sizeof(dp));
    scanf("%s",x);
    scanf("%s",y);
    int a=strlen(x);
    int b=strlen(y);
    cout<<dfs(a,b)<<endl;
    write(a,b);
    return 0;
}
```

例 9：编辑距离 配 P2064

设 A 和 B 是两个字符串。我们用最少的字符操作，将字符串 A 转换为字符串 B。这里所说的字符操作包括：

(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。

将字符串 A 转换为字符串 B 所用的最少字符操作数称为字符串 A 到字符串 B 的编辑距离，记为 δ(A，B)。

对任给的两个字符串 A 和 B，计算出它们的编辑距离 δ(A，B)。

【解析】

1、阶段和状态：

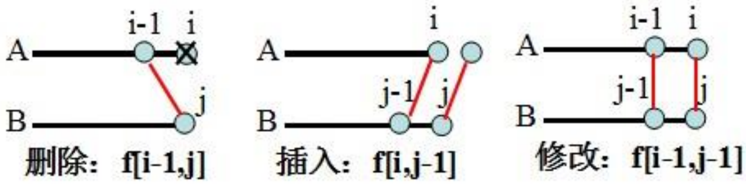
f[i][j]：表示将 A 的前 i 个字符变成 B 的前 j 个字符的最少操作次数

求 f[i][j]要考虑 3 种操作：

(1).在 A 中删除一个字符：f[i-1][j]

(2).在 A 中插入一个字符：f[i][j-1]

(3).在 A 中将一个字符改为另一个字符，如果 a[i]=b[i]为 f[i-1][j-1]，如果 a[i]!=b[i]为 f[i-1][j-1]+1。



2、状态转移方程：

$$f[i][j]=\begin{cases} f[i-1][j-1] & \text{当a[i]= b[i]时} \\ \min\{f[i-1][j],f[i][j-1],f[i-1][j-1]\}+1 & \text{当a[i]<>b[i]时} \end{cases}$$

初始化：f[0][i]=f[i][0]=i;

Answer=f[L1][L2]

【参考代码】

```
int main() {
    scanf("%s",a+1); scanf("%s",b+1);
    len1=strlen(a+1); len2=strlen(b+1);
    for(int i=1;i<=3000;i++) f[i][0]=f[0][i]=i;
    for(int i=1;i<=len1;i++)
        for(int j=1;j<=len2;j++) {
            if(a[i]==b[j]) f[i][j]=f[i-1][j-1];
            else f[i][j]=min(f[i-1][j-1],min(f[i-1][j],f[i][j-1]))+1;
        }
    cout<<f[len1][len2];
    return 0;
}
```