NOIP基础算法综合

重庆巴蜀中学　黄新军

第一部分

枚举策略

一、枚举法的基本思想

- 枚举法的基本思想是根据提出的问题枚举所有可能状态，并用问题给定的条件检验哪些是需要的，哪些是不需要的。能使命题成立，即为其解。
- 枚举结构：循环+判断语句。

二、枚举法的条件：

- 虽然枚举法本质上属于搜索策略，但是它与后面讲的回溯法有所不同。因为适用枚举法求解的问题必须满足两个条件：
- ⑴可预先确定每个状态的元素个数n；
- ⑵状态元素a1，a2，…，an的可能值为一个连续的值域。

三、枚举法的框架结构

■设a_{i1}—状态元素a_i的最小值；a_{ik}—状态元素a_i的最大值($1 \leqslant i \leqslant n$)，即$a_{11} \leqslant a_1 \leqslant a_{1k}$，$a_{21} \leqslant a_2 \leqslant a_{2k}$，$a_{i1} \leqslant a_i \leqslant a_{ik}$，……，$a_{n1} \leqslant a_n \leqslant a_{nk}$

for $a_1 \leftarrow a_{11}$ to a_{1k} do

  for $a_2 \leftarrow a_{21}$ to a_{2k} do

    ……

      for $a_i \leftarrow a_{i1}$ to a_{ik} do

        ……

          for $a_n \leftarrow a_{n1}$ to a_{nk} do

            if状态(a_1，…，a_i，…，a_n)满足检验条件

              then 输出问题的解;

四、枚举法的优缺点

枚举法的优点

■(1)由于枚举算法一般是现实生活中问题的“直译”，因此比较直观，易于理解；

■(2)由于枚举算法建立在考察大量状态、甚至是穷举所有状态的基础上，所以算法的正确性比较容易证明。

枚举法的缺点

■枚举算法的效率取决于枚举状态的数量以及单个状态枚举的代价，因此效率比较低。

- “直译”枚举：直接根据题意设定枚举对象、范围和约束条件。
- 注意认真审题，不要疏漏任何条件

例题1：砝码称重(noip1996)

【问题描述】设有1g、2g、3g、5g、10g、20g的砝码各若干枚（其总重<=1000），求用这些砝码能称出不同的重量个数。

【文件输入】输入1g、2g、3g、5g、10g、20g的砝码个数。

【文件输出】输出能称出不同重量的个数。

【样例输入】1 1 0 0 0 0

【样例输出】3

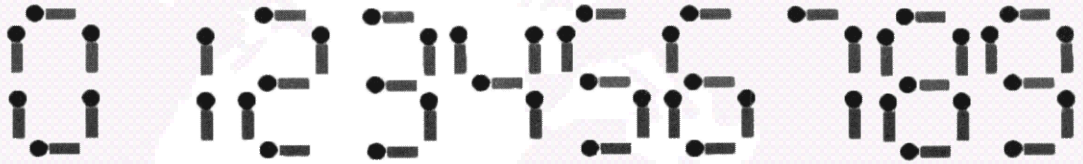
【分析】根据输入的砝码信息，每种砝码可用的最大个数是确定的，而且每种砝码的个数是连续的，能取0到最大个数，所以符合枚举法的两个条件，可以使用枚举法。枚举时，重量可以由1g, 2g, ……, 20g砝码中的任何一个或者多个构成，枚举对象可以确定为6种重量的砝码，范围为每种砝码的个数。判定时，只需判断这次得到的重量是新得到的，还是前一次已经得到的，即判重。由于重量<=1000g，所以，可以开一个a[1001]的数组来判重。

核心参考代码：

```
readln(a,b,c,d,e,f)
for c[1]:=0 to a do   //1g砝码的个数
      for c[2]:=0 to b do   //2g砝码的个数
            for c[3]:=0 to c do   //3g砝码的个数
                  for c[4]:=0 to d do   //5g砝码的个数
                        for c[5]:=0 to e do  //10g砝码的个数
                              for c[6]:=0 to f do   //20g砝码的个数
                                    begin
                                          sum:=0;
                                          for i:=1 to 6 do sum:=sum+c[i]*w[i];
                                          a[sum]:=1; //标记
                                    end;
for i:=1 to 1000 do if a[i]=1 then inc(num); //统计不同重量的个数
Writeln(num);
```

例题2：火柴棒等式（NOIP2008）

【问题描述】给你n根火柴棍，你可以拼出多少个形如“A+B=C”的等式？等式中的A、B、C是用火柴棍拼出的整数（若该数非零，则最高位不能是0）。用火柴棍拼数字0-9的拼法如图所示：



注意：

1. 加号与等号各自需要两根火柴棍

2. 如果A≠B，则A+B=C与B+A=C视为不同的等式（A、B、C≥0）

3. n根火柴棍必须全部用上

【输入】输入文件matches.in共一行，又一个整数n（n≤24）。

【输出】输出文件matches.out共一行，表示能拼成的不同等式的数目。

例题2：火柴棒等式（NOIP2008）

【问题简述】给你n(n<=24)根火柴棒,叫你拼出 “A + B = C”这样的等式,求方案数。

【思路点拨】本题主要考查对枚举法的掌握，可以枚举A和B的取值，考查等式是否刚好用了24根火柴棒。1S的时限对枚举的范围有所要求，必须要仔细分析A和B的取值。

例题2：火柴棒等式（NOIP2008）

- 本题最多24根火柴，等号和加号共用4根火柴，所以A,B,C这3个数字需用20根火柴。我们考查A和B的最大的取值可能：0~9这10个数字所用的火柴数为6,2,5,5,4,5,6,3,7,6，很明显数字1用的火柴棒最少只要2根，不妨让B为1，那么A和C最多可以使用18根火柴，而C>=A，满足条件的A的最大取值为1111。所以枚举A和B的范围是从0~1111。

- **为了加快速度，可以将0到2222的所有整数需要的火柴棒数目提前算好保存在数组中。**

五、枚举算法的优化

枚举算法的时间复杂度：状态总数*单个状态的耗时。

- ⑴提取有效信息；
- ⑵减少重复计算；
- ⑶将原问题化为更小的问题；
- ⑷根据问题的性质进行截枝；
- ⑸引进其他算法

【例题3】给你n(n<=10^5)个整数，然后要有m(m<=10^5)个询问。每个询问两个整数i和j，问第i个数字到第j个数字所有数字之和。

【朴素算法】

```
readln(n);
for i:=1 to n do read(a[i]);
for i:=1 to m do
begin
    read(x,y);
    sum:=0;
    for j:=x to y do sum:=sum+a[j];
    writeln(sum);
end;
```

时间复杂度为：O(nm)

【优化算法】先递推计算出**s[i]=s[i-1]+a[i]**，再回答询问情况；

```
readln(n);
for i:=1 to n do{统计求和}
    begin
        read(a[i]);
        s[i]:=s[i-1]+a[i];
    end;
for i:=1 to m do
  begin
      read(x,y);
      writeln(s[y]-s[x-1]);
  end;
```

【例题4】对于给定的N*M的矩形，在其中找一个R*C的权值最大的区域, 1<=N,M<=1,000。

【算法一】：以每一个格子为左上角枚举**R*C**的区域，并求出它的权值之和。最后取其中的最大值。此算法包含**4**重循环。

时间复杂度：**O(N^4)**

【算法二】：我们设**S[i,j]**表示以**(1,1)**为左上角，**(i,j)**为右下角区域的权值之和，那么我们以**(i,j)**为右下角的**R*C**区域权值之和的计算公式为：

Area[i,j]=S[i,j]+S[i-R,j-C]-S[i-R,j]-S[i,j-C]

其中**S[i,j]**的计算公式为：

S[i,j]=value[i,j]+S[i-1,j]+S[i,j-1]-S[i-1,j-1]

你可以随手画图出来，很容易即可证明上面两个式子。最后取**Area[]**中的最大值即可。

时间复杂度：**O(N^2)**

【例题5】最大连续子序列的和

【问题描述】给你一个有n(n<=100000)个整数的序列，要求你求出其中最大连续子序列的和。

【算法1】枚举起始位置，结束位置，计算中间的和。时间复杂度O(n^3)。

```
maxx:=a[1];
for i:=1  to n  do
    for j:=i to n do
      begin
          sum:=0;
          for k:=i to j do sum:=sum+a[k];
          if  sum>maxx then maxx:=sum;
      end;
```

【算法2】：优化的枚举——O(*n*^2)

```
s[0]:=0;
for i:=1 to n do
        s[i]:=s[i-1]+a[i];//初始化求和
maxx:=a[1];
for i:=1 to n do
      for j:=i to n do
           if s[j]-s[i-1]>maxx then maxx:=s[j]-s[i-1];
```

时间复杂度：预处理+主程序=*O*(*n*+*n*^2)=*O*(*n*^2)，*n*<=5000

【算法3】**分治——$O(n\log n)$**、

最大连续子序列的位置有三种可能：

①完全处于序列的左半；

②完全处于序列的右半；

③跨越序列中间；

【算法4】DP——$O(n)$

1、阶段和状态：

以第i个数结尾的最大连续子序列的和，只存在两种选择：

情形1：只包含a[i]

情形2：包含a[i]和以a[i-1]结尾的最大连续和序列

故设f[i]表示以a[i]结尾的最大连续子序列的和

2、状态转移方程：

转移方程：f[i]=max{a[i],f[i-1]+a[i]}(2<=i<=n)

初始化：f[1]=a[1]

Answer=max{f[i]|1<=i<=n}

【例题6】最大子矩阵问题

【问题描述】给定一个二维的数组（含正数或负数），请从中找出和最大的子矩阵。

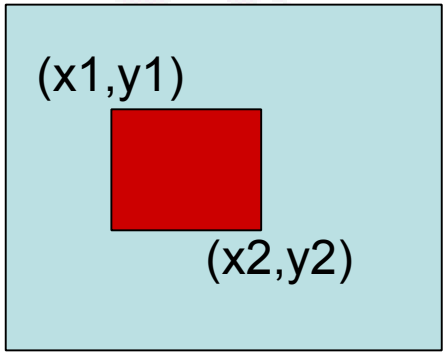
例如：

$$\begin{matrix} 0 & -2 & -7 & 0 \\ 9 & 2 & -6 & 2 \\ -4 & 1 & -4 & 1 \\ -1 & 8 & 0 & -2 \end{matrix} \qquad \begin{matrix} 9 & 2 \\ -4 & 1 \\ -1 & 8 \end{matrix}$$

1、“直译”枚举过程

```
For x1:=1 to n do//枚举矩形左上角(x1,y1)
   for y1:=1 to n do
       for x2:=1 to n do//枚举矩形右下角(x2,y2)
           for y2:=1 to n do
             //考察状态左上角为(x1,y1)右下角为(x2,y2)内矩形的元素之和；
            begin
               sum:=0；
               for x:=x1 to x2 do//计算当前矩形内元素的和
                     for y:=y1 to y2 do sum:=sum+a[x,y]；
               if sum>best then best:=sum；//调整最优解
           end;
```

- 这个算法相当粗糙，枚举状态的费用为$O(n^6)$



2、从减少重复计算入手

■有刚才一维情况可以推广到二维，在统计左上角为**(x1,y1)**右下角为**(x2,y2)**内矩形的元素之和时，我们同样可以先初始化，计算出左上角为**(1,1)**，右下角为**(x,y)**内矩形的元素之和**s[x][y]**。

```
for i:=1 to n do  //枚举矩形右下角，求和
      for j:=1 to n do
        begin
          read(a[i,j]);
          s[i,j]:=s[i-1,j]+s[i,j-1]-s[i-1,j-1]+a[i,j];
        end;
```

■对于状态左上角为**(x1,y1)**,右下角为**(x2,y2)**内矩形的元素之和，可以改为：

```
sum:=s[x2,y2]-s[x1-1,y2]-s[x2,y1-1]+s[x1-1,y1-1];
if sum>best then best:=sum;      //调整最优解
```

■由于先进行了预处理，整个算法的时间复杂度降为$O(n^4)$

3、提取恰当的信息

■容易观察到，最大子矩阵问题是最大连续子序列和问题的提升，即将一条线换成一个面，将一维问题提升到二维问题。所以我们计算最大子矩阵的方法就是将一行行的数进行累加以求得最大值。

■但是还有一个问题，那就是应该如何高效地存储矩阵？

■我们可以想到：在一个一维的数列中，设数组b[i]表示从第1个元素到第i个元素的和，则如果想要求第i个元素到第j个元素的和，只需要计算b[j]-b[i-1]的值就行了。由此推广到二维矩阵，设b[i,j]表示矩阵第j列前i个元素的和，a[i,j]表示元素数据，则压缩存储：

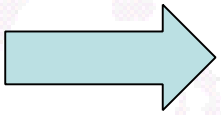
```
for i:=1 to n do
    for j:=1 to n do
      begin  read(a[i,j]);b[i,j]:=b[i-1,j]+a[i,j];}
```

■因此，我们可以使用三重循环求出所有的矩形值，即**枚举起始行i和终止行j，压缩子矩形成为一行，变成一维求最大子段和问题**。

即**t[k]=max(t[k-1],0)+b[j,k]-b[i-1,k];**

■时间复杂度为$O(n^3)$

$$\begin{bmatrix} 0 & -2 & -7 & 0 \\ 9 & 2 & -6 & 2 \\ -4 & 1 & -4 & 1 \\ -1 & 8 & 0 & -2 \end{bmatrix}$$



$$\begin{bmatrix} 0 & -2 & -7 & 0 \\ 9 & 0 & -13 & 2 \\ 5 & 1 & -17 & 3 \\ 4 & 9 & -17 & 1 \end{bmatrix}$$

②核心代码

```
sum:=-99999999;    //置初值
for i:=1 to n do      //阶段:起始行
 begin
     for j:=i to n do   //状态:结束行
     begin
          t[1]:=b[j,1]-b[i-1,1];  //初始化第1列的值
          for k:=2 to n do      //决策:第几列
            begin
               if t[k-1]>0 then  t[k]:=t[k-1]+b[j,k]-b[i-1,k]
                             else  t[k]:=b[j,k]-b[i-1,k];
               if t[k]>sum then sum:=t[k];
            end;
     end;
end;
writeln(sum);
```

六、局部枚举

例题7：求第一、第二、第三最短路问题

例题8：新年好

- 重庆城里有n个车站，m条双向公路连接其中的某些车站。每两个车站最多用一条公路直接相连，从任何一个车站出发都可以经过一条或多条公路到达其他车站，但不同的路径需要花费的时间可能不同。在一条路上花费的时间等于路径上所有公路需要的时间之和。
- 佳佳的家在车站1，他有五个亲戚，分别住在车站a,b,c,d,e。过年了，他需要从自己的家出发，拜访每个亲戚（顺序任意），给他们送去节日的祝福。怎样走，才需要最少的时间？
- 数据范围：n(n<=50,000),m(m<=100,000)

算法框架为：

- （1）用6次计算单源最短路算法，即枚举这6个点，每个点都做为源点求单源最短路。
- （2）枚举（a,b,c,d,e）这5个结点的全排列，找到一个使得总路程长度最短的方案。
- 这一题中的边数远小于n^2，所以复杂度也只有nlogn+m。

第二部分

递推策略

一、引例：Fibonacci数列

- Fibonacci数列的代表问题是由意大利著名数学家Fibonacci于1202年提出的“兔子繁殖问题”(又称“Fibonacci问题”)。
- 问题：

  一个数列的第0项为0，第1项为1，以后每一项都是前两项的和，这个数列就是著名的裴波那契数列，求裴波那契数列的第N项。

解答

- 由问题,可写出递推方程

$$f_n = \begin{cases} 1(n=0) \\ 2(n=1) \\ f_{n-1} + f_{n-2}(n \geq 2) \end{cases}$$

- 算法：

```
f[0]:=1; f[1]:=2;
for i:=2 to n do f[i]:=f[i–1]+f[i–2];
```

总结

- 从这个问题可以看出，在计算裴波那契数列的每一项目时，都可以由前两项推出。这样，相邻两项之间的变化有一定的规律性，我们可以将这种规律归纳成如下简捷的递推关系式：$F_n=g(F_{n-1})$，这就在数的序列中，建立起后项和前项之间的关系。然后从初始条件（或是最终结果）入手，按递推关系式递推，直至求出最终结果（或初始值）。很多问题就是这样逐步求解的。
- 对一个试题，我们要是能找到后一项与前一项的关系并清楚其起始条件（或最终结果），问题就可以递推了，接下来便是让计算机一步步了。让高速的计算机从事这种重复运算，真正起到“物尽其用”的效果。

二、递推概念

- 给定一个数的序列H0,H1,…,Hn,…若存在整数n_0，使当$n>n_0$时,可以用等号(或大于号、小于号)将Hn与其前面的某些项Hn(0<i<n)联系起来，这样的式子就叫做递推关系。
  - 如何建立递推关系
  - 递推关系有何性质
  - 如何求解递推关系

三、解决递推问题的一般步骤

- 建立递推关系式
- 确定边界条件
- 递推求解

四、递推的两种形式

- 顺推法和倒推法

满足求解初始条件 F_1	
Y{顺推}	N{倒推}
由题意（或递推关系）定初始值 F_1（边界条件）求出顺推关系式 $F_i=g(F_{i-1})$；	由题意（或递推关系）确定最终结果 F_n；求出倒推关系式 $F_{i-1}=g'(F_i)$；
I=1；{由边界条件 F_1 出发进行顺推}	I=n；{从最终结果 F_n 出发进行倒推}
While 当前结果 F_i 非最终结果 F_n do 由 $F_i=g(F_{i-1})$ 顺推后项；	While 当前结果 F_i 非初始值 F_1 do 由 $F_{i-1}=g(F_i)$ 倒推前项；
输出顺推结果 F_n 和顺推过程；	输出倒推结果 F_1 和倒推过程；

五、递推的应用分类

- 一般递推问题
- 组合计数类问题
- 一类博弈问题的求解
- 动态规划问题的递推关系

（一）递推的应用（一般递推问题）

例题1：faibonacci数列

【问题描述】已知faibonacci数列的前几个数分别为0，1，1，2，3，5，……编程求出此数列的第n项。（n<=60）

- **思考：当n<=10^9时，如何求解？**

（一）递推的应用（一般递推问题）

例题2：输出杨辉三角的前N行

【问题描述】输出杨辉三角的前N行(N<10)。

【文件输入】输入只有一行，包括1个整数N(N<10)。

【文件输出】输出只有N行。

【样例输入】3

【样例输出】

1

1 1

1 2 1

递推方程：f[i,j]=f[i-1][j-1]+f[i-1,j]

边界条件：f[i,1]=1;f[i,i]=1

（一）递推的应用（一般递推问题）

- **例题3*：Hanoi塔问题*.**

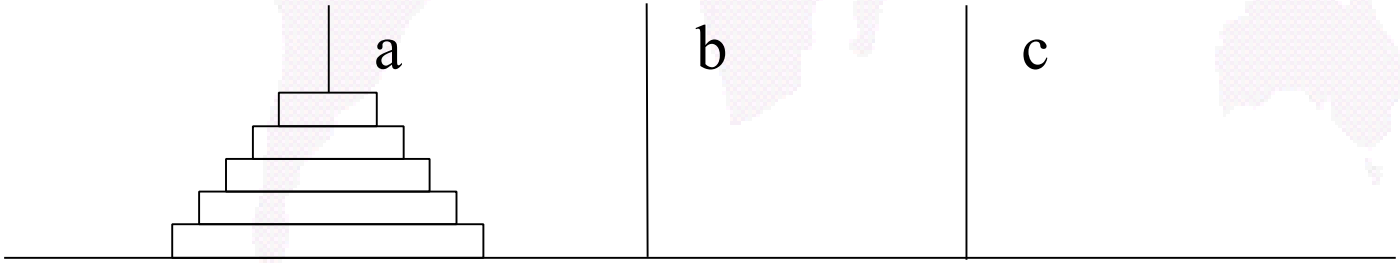
Hanoi塔由n个大小不同的圆盘和三根木柱a,b,c组成。开始时，这n个圆盘由大到小依次套在a柱上，如图1所示。要求把a柱上n个圆盘按下述规则移到c柱上：

(1)一次只能移一个圆盘；

(2)圆盘只能在三个柱上存放；

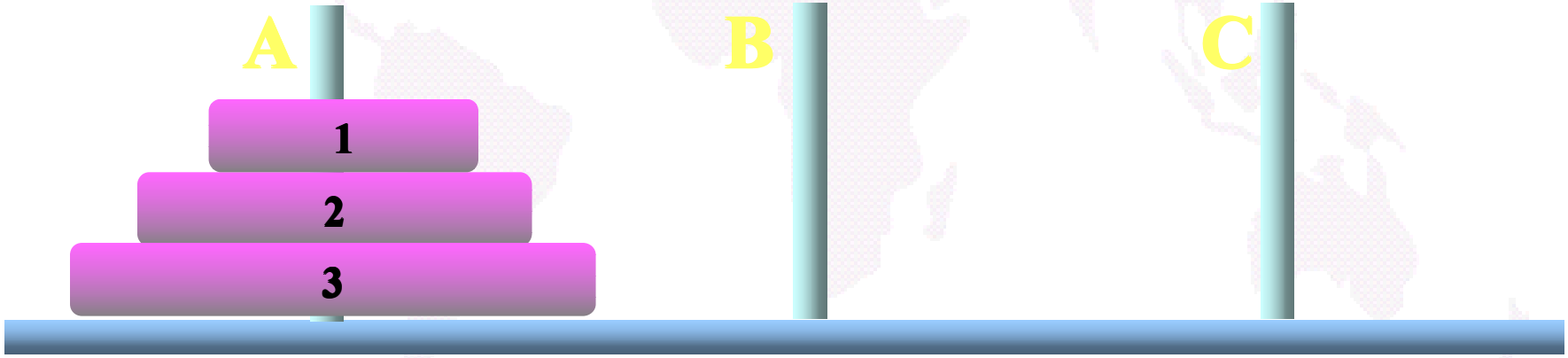
(3)在移动过程中，不允许大盘压小盘。

问将这n个盘子从a柱移动到c柱上，总计需要移动多少个盘次？



分析

当n=1时：A→C
当n=2时：A→B,A→C,B→C
当n=3时：



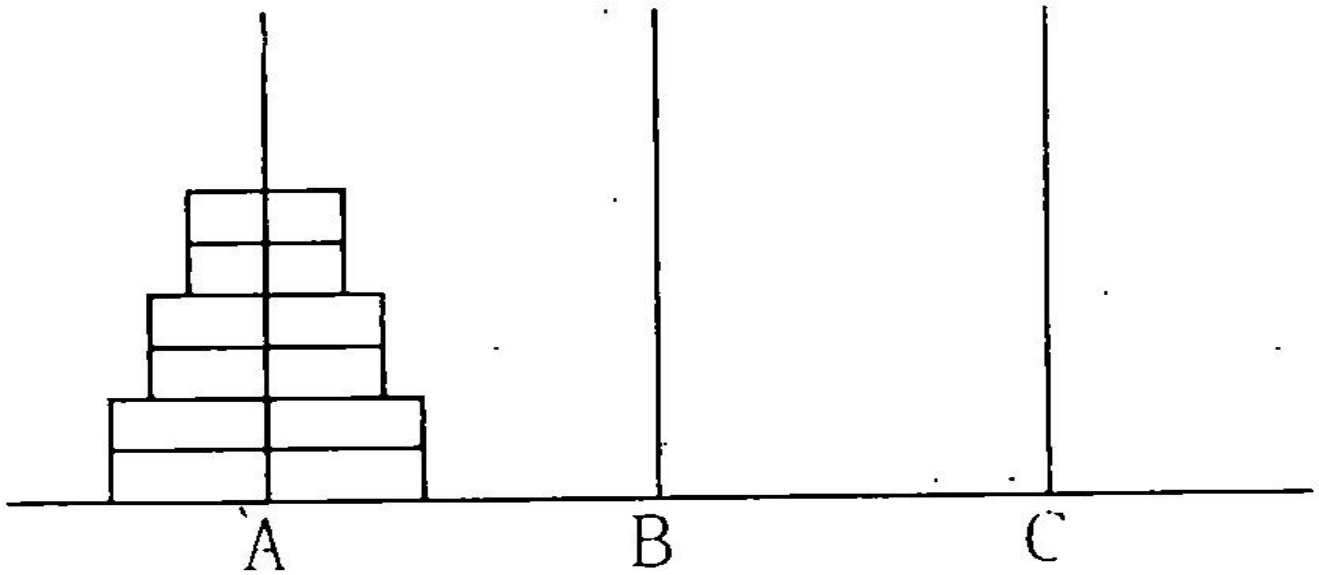
分析

- 设f(n)为n 个盘子从1柱移到3柱所需移动的最少盘次。
- 当n=1时，f(1)=1。
- 当n=2时，f(2)=3。
- 以此类推，当1柱上有n(n>2)个盘子时，我们可以利用下列步骤：
- 第一步：先借助3柱把1柱上面的n-1个盘子移动到2柱上，所需的移动次数为f(n-1)。
- 第二步：然后再把1柱最下面的一个盘子移动到3柱上，只需要1次盘子。
- 第三步：再借助1柱把2柱上的n-1个盘子移动到3上，所需的移动次数为f(n-1)。
- 由以上3步得出总共移动盘子的次数为：f(n-1)+1+ f(n-1)。
- 所以：f(n)=2 f(n-1)+1

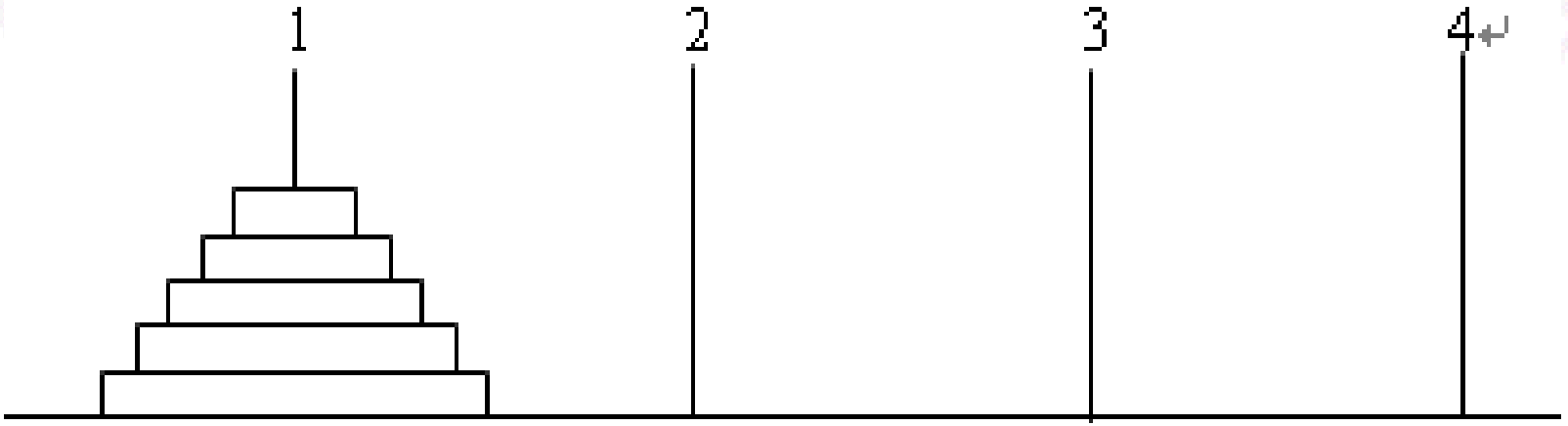
$h_n=2h_{n-1}+1=2^n-1$

边界条件：$h_1=1$

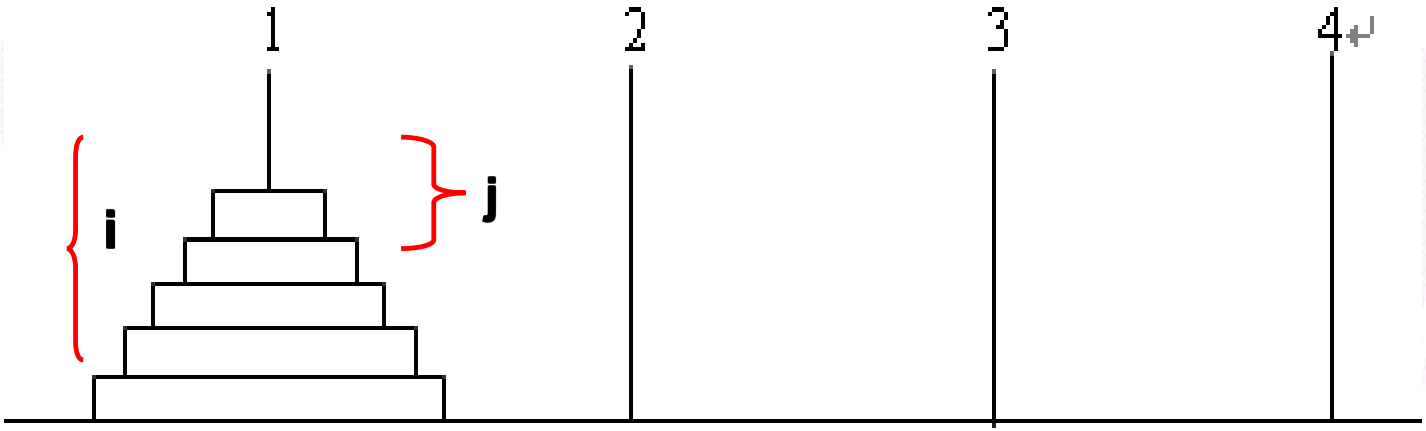
【扩展1】： *Hanoi*双塔问题



【扩展2】：四塔问题



【问题分析】令**f[i]**表示四个柱子时，把**i**个盘子从原柱移动到目标柱所需的最少移动次数。



■第一步：先把1柱上的前j个盘子移动到另外其中一个非目标柱（2或3柱均可，假设移到2柱）上，此时3和4柱可以作为中间柱。移动次数为：f[j]。

■第二步：再把原1柱上剩下的i-j个盘子在3根柱子（1、3、4）之间移动，最后移动到目标柱4上，因为此时2柱不能作为中间柱子使用，根据三柱问题可知，移动次数为：2^(i-j)-1。

■第三步：最后把非目标柱2柱上的j个盘子移动到目标柱上，次数为：f[j]。

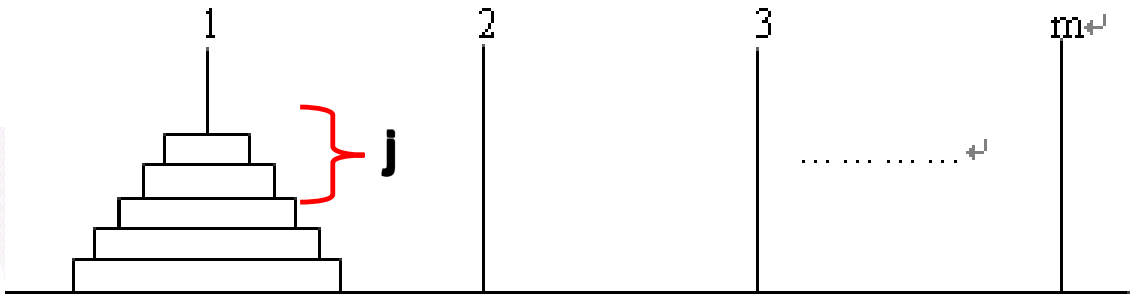
- 通过以上步骤我们可以初步得出：

  f[i] = 2*f[j]+2^(i-j)-1

- j可取的范围是1<=j<i，所以对于不同的j，得到的f[i]可能是不同的，本题要求最少的移动次数。

f[i]=min{2*f[j]+2^(i-j)-1}，其中1<=j<i

【扩展3】：m塔问题



【问题分析】 设F(m,n)为m根柱子,n个盘子时移动的最少次数：

- 1、先把1柱上的前j个盘子移动到另外其中一个除m柱以外的非目标柱上，移动次数为：f[m, j]；
- 2、再把原1柱上剩下的n-j个盘子在m-1根柱子之间移动，最后移动到目标柱m上，移动次数为：f[m-1, n-j]；
- 3、最后把非目标柱上的j个盘子移动到目标柱没柱上，移动次数为：f[m, j]。

$$F(m,n)=\min\{2*F(m,j)+F(m-1,n-j)\}\ (1<=j<n)$$

（一）递推的应用（一般递推问题）

例题4:数的计数

【问题描述】我们要求找出具有下列性质数的个数(包含输入的自然数n)，先输入一个自然数n(n≤1000)，然后对此自然数按照如下方法进行处理：

(1)、不作任何处理；

(2)、在它的左边加上一个自然数，但该自然数不能超过原数的一半；

(3)、加上数后，继续按此规则进行处理，直到不能再加自然数为止；

方法1：用递推

- 用h[n]表示自然数n所能扩展的数据个数，则：

  h[1]=1,h[2]=2,h[3]=2,h[4]=4,h[5]=4,

  h[6]=6,h[7]=6,h[8]=10,h[9]=10 。

- 分析上数据，可得递推公式：

  h[i]=1+h[1]+h[2]+ …+h[i/2]。

- 时间复杂度$O(n^2)$。

方法2：是对方法1的改进。我们定义数组s。

$$s(x)=h(1)+h(2)+\ldots+h(x)$$
$$=s(x-1)+h(x)$$
$$=s(x-1)+s(x/2)$$

$$h(x)=s(x)-s(x-1)=s(x/2)$$

此算法的时间复杂度可降到O(n)。

方法3：还是用递推。

- 只要做仔细分析，其实我们还可以得到以下的递推公式：

  (1)当i为奇数时,h(i)=h(i-1);

  (2) 当i为偶数时,h(i)=h(i-1)+h(i/2);

【思考】

1.若n<=10000怎么计算；

2.若n<=3000000怎么计算；

（一）递推的应用（一般递推问题）

- 例题5：猴子吃桃问题1538
- 【问题描述】猴子摘了一堆桃，第一天吃了一半，还嫌不过瘾，又吃了一个；第二天又吃了剩下的一半零一个；以后每天如此。到第n天，猴子一看只剩下一个了。问最初有多少个桃子？

【扩展练习】猴子分桃

【问题描述】有一堆桃子和N只猴子，第一只猴子将桃子平均分成了M堆后，还剩了1个，它吃了剩下的一个，并拿走一堆。后面的猴子也和第1只进行了同样的做法，请问N只猴子进行了同样做法后这一堆桃子至少还剩了多少个桃子(假设剩下的每堆中至少有一个桃子)？而最初时的那堆桃子至少有多少个？

【文件输入】输入包含二个数据，数据间用空格隔开。第一个数据为猴子的只数N(1≤N≤10)，第二个数据为桃子分成的堆数M(2≤M≤7)。

【文件输出】输出包含两行数据，第一行数据为剩下的桃子数，第二行数据为原来的桃子数。

【样例输入】3 2

【样例输出】

1

15

枚举+倒推法

（一）递推的应用（一般递推问题）

- **【例题6】传球游戏（NOIP2008普及）**
- **【问题描述】**上体育课的时候，小蛮的老师经常带着同学们一起做游戏。这次，老师带着同学们一起做传球游戏。

游戏规则是这样的：n（3<=n<=30）个同学站成一个圆圈，其中的一个同学手里拿着一个球，当老师吹哨子时开始传球，每个同学可以把球传给自己左右的两个同学中的一个（左右任意），当老师再吹哨子时，传球停止，此时，拿着球没传出去的那个同学就是败者，要给大家表演一个节目。

聪明的小蛮提出一个有趣的问题：有多少种不同的传球方法可以使得从小蛮手里开始传的球，传了m（3<=m<=30）次后，又回到小蛮手里。两种传球被视作不同的方法，当且仅当这两种方法中，接到球的同学按照顺序组成的序列是不同的。比如3个同学1号、2号、3号，并假设小蛮为1号，球传了3次回到小蛮手里的方式有1->2->3->1和1->3->2->1，共两种。

分析

- 设f[i,k]表示经过k次传到编号为i的人手中的方案数，传到i号同学的球只能来自于i的左边一个同学和右边一个同学，这两个同学的编号分别是i-1和i+1，所以可以得到以下的递推公式：

  f[i,k]=f[i-1,k-1]+f[i+1,k-1]

  f[1,k]=f[n,k-1]+f[2,k-1]，　当i=1时

  f[n,k]=f[n-1,k-1]+f[1,k-1]，　当i=1时

  边界条件：f[1,0]=1；

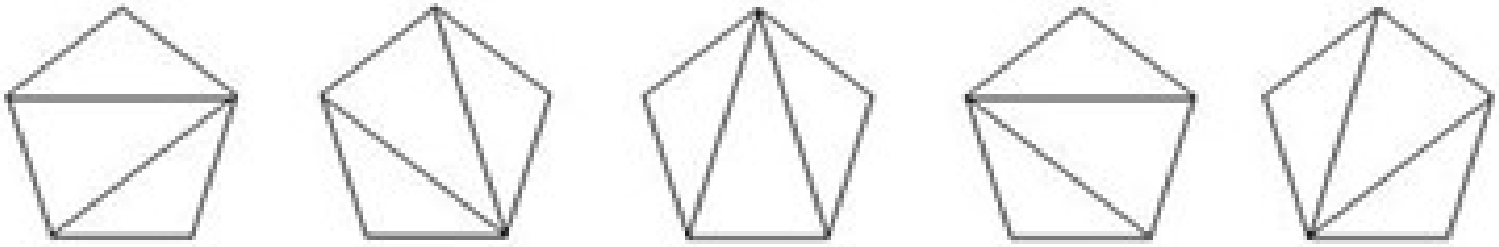
  answer=f[1,m]

参考代码

```
readln(n,m);
f[1,0]:=1;
for k:=1 to m do
 begin
    f[1,k]:=f[2,k-1]+f[n,k-1];
    for i:=2 to n-1 do f[i,k]:=f[i-1,k-1]+f[i+1,k-1];
    f[n,k]:=f[n-1,k-1]+f[1,k-1];
 end;
writeln(f[1,m]);
```

（二）递推的应用（组合计数）

- **Catalan数**
  - **例题7：凸n边形的三角形剖分**
  - 在一个凸n边形中，通过不相交于n边形内部的对角线，把n边形拆分成若干三角形，不同的拆分数目用f(n)表之，f(n)即为Catalan数。例如五边形有如下五种拆分方案，故f(5)=5。求对于一个任意的凸n边形相应的f(n)。

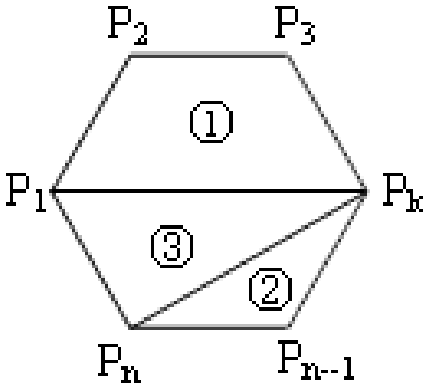


分析:

- 设f(n)表示凸n边形的拆分方案总数。由题目中的要求可知一个凸 n边形的任意一条边都必然是一个三角形的一条边，边P_1P_n也不例外，再根据“不在同一直线上的三点可以确定一个三角形”，只要在P_2，P_3，……，P_{n-1}点中找一个点P_k(1<k<n)，与P_1、P_n 共同构成一个三角形的三个顶点，就将n边形分成了三个不相交的部分(如图)，我们分别称之为区域①、区域②、区域③，其中区域③必定是一个三角形，区域①是一个凸k边形，区域②是一个凸n-k+1边形，区域①的拆分方案总数是f(k)，区域②的拆分方案数为f(n-k+1)，故包含$\triangle P_1P_kP_n$的n 边形的拆分方案数为f(k)*f(n-k+1)种，而P_k可以是P_2，P_3，……，P_{n-1}种任一点，根据加法原理，凸n边形的三角拆分方案总数为:

$$f(n)=\sum_{i=2}^{n-1} f(i)*f(n-i+1)$$

边界条件：f(2)=1



【扩展1】：二叉树数目

【问题描述】：求n个结点能构成不同二叉数的数目。

【问题分析】：

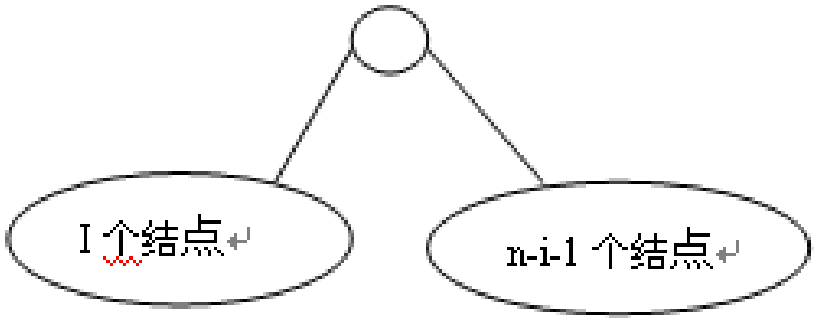
设F(n)为n个结点组成二叉树的数目。

容易知道：f(1)=1;f(2)=2,f(3)=5

选定1个结点为根，左子树结点的个数为i，二叉树数目f(i)种；右子树结点数目为n-i-1，二叉树数目f(n-i-1)种，i的可取范围[0，n-1]。所以有：

$$f(n)=\sum_{i=0}^{n-1} f(i)*f(n-i-1)$$

为了计算的方便：约定f(0)=1



【扩展2】：出栈序列

【问题描述】：N个不同元素按一定的顺序入栈，求不同的出栈序列数目。

【问题分析】设f(n)为n个元素的不同出栈序列数目。

容易得出：f(1)=1；f(2)=2。

第n个元素可以第i(1<=i<=n)个出栈，前面已出栈有i-1个元素，出栈方法：f(i-1)；后面出栈n-i 个元素，出栈方法为：f(n-i)。所以有：

$$\sum_{i=1} f(i-1) * f(n-i)$$

$$f(n) = \sum_{i=1}^{n} f(i-1) * f(n-i)$$

初始条件： F(0)=1

（二）递推的应用（组合计数）

例题10：错排问题（经典问题）

- n个数，分别为1~n，排成一个长度为n的排列。若每一个数的位置都与数的本身不相等，则称这个排列是一个错排。例如，n=3，则错排有２３１、３１２。编写程序，求n的错排个数

分析

- 我们设k个元素的错位全排列的个数记做：f(k)。
- 四个元素的错位排列f(4)我们用穷举法可以找到如下9个：

  (4,3,2,1);(3,4,1,2);(2,1,4,3)

  (4,3,1,2);(2,4,1,3);(2,3,4,1)

  (4,1,2,3);(3,4,2,1);(3,1,4,2)

- 它们有什么规律呢？

分析

■通过反复的试验，我们发现事实上有两种方式产生错位排列：

A、将k与(1，2，…，k-1)的某一个数互换，其他k-2个数进行错排，这样可以得到**(k-1)×f(k-2)**个错位排列。

B、另一部分是将前k-1个元素的每一个错位排列(有f(k-1)个)中的每一个数与k互换，这样可以得到剩下的**(k-1)×f(k-1)**个错位排列。

■根据加法原理，我们得到求错位排列的递推公式f(k):

$$f(k)=(k-1)*(f(k-1)+f(k-2))$$

（二）递推的应用（组合计数）

例题11：编码问题

- 【问题描述】编码工作常被运用于密文或压缩传输。这里我们用一种最简单的编码方式进行编码：把一些有规律的单词编成数字。字母表中共有26个小写字母{a,b,c….,z}。这些特殊的单词长度不超过6且字母按照升序排列。把所有这样的单词放在一起，按字典顺序排列，一个单词的编码就对应着它在字典中的位置，例如：a-1;b-2;z-26;ab-27;ac-28;你的任务就是对于所给的单词，求出它的编码。

（二）递推的应用（组合计数）

例题12：2^k进制数（NOIP2006）

见word文档

（三）递推的应用（博弈问题）

例题13：走直线棋问题。有如下所示的一个编号为 1 到 n 的方格：

1	2	3	4	5	…	…	N-1	N

现由计算机和人进行人机对奕，从 1 到 n，每次可以走 k 个方格，其中 k 为集 s ={a_1, a_2, a_3, ... a_m} 中的元素(m<=4)，规定谁最先走到第n格为胜，试设计一个人机对奕方案，摸拟整个游戏过程的情况并力求计算机尽量不败。

分析

- 题设条件：若谁先走到第N格谁将获胜，例如，假设S={1,2}，从第N格往前倒推，则走到第N-1格或第N-2格的一方必败，而走到第N-3格者必定获胜，因此在N，S确定后，棋格中每个方格的胜、负或和态（双方都不能到达第N格）都是可以事先确定的。将目标格置为必胜态，由后往前倒推每一格的胜负状态，规定在自己所处的当前格后，若对方无论走到哪儿都必定失败，则当前格为胜态，若走后有任一格为胜格，则当前格为输态，否则为和态。

分析

设1表示必胜态，-1表示必败态，0表示和态或表示无法到达的棋格。

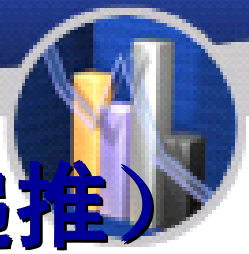
例如，设N＝10，S＝{1,2}，则可确定其每个棋格的状态如下所示：

1	-1	-1	1	-1	-1	1	-1	-1	1

而N＝10，S＝{2,3}时，其每格的状态将会如下所示：

0	-1	-1	0	1	0	-1	-1	0	1

有了棋格的状态图后，程序应能判断让谁先走，计算机选择必胜策略或双方和（双方均不能到达目标格）的策略下棋，这样就能保证计算机尽可能不败。



（四）递推的应用（动态规划中的递推）

- **例题14：最小伤害**
- 把儿站在一个N x N的方阵中最左上角的格子里。他可以从一个格子走到它右边和下边的格子里。每一个格子都有一个伤害值。他想在受伤害最小的情况下走到方阵的最右下角。

分析

- F[i,j]：设走到(i,j) 这格的最小伤害值,a[i][j]表示(i,j)这格的伤害值。
- F[i,j]=min(f[i-1,j],f[i,j-1])+a[i,j]
- 边界条件：f[1,1]=a[1,1]

  f[i,1]=f[i-1,1]+a[i,1](2<=i<=n)

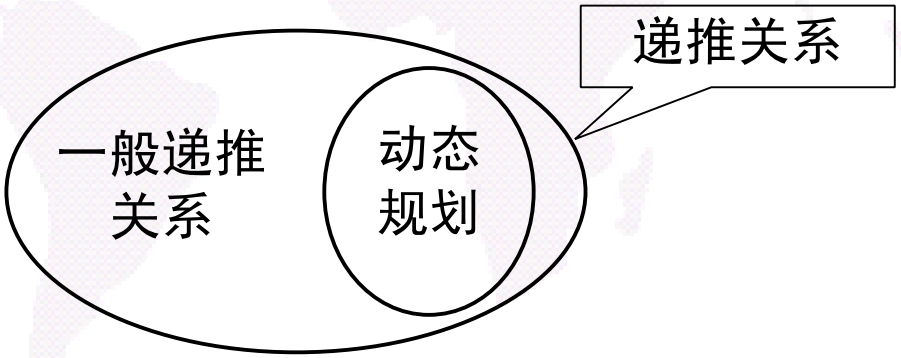
  f[1,i]=f[1,i-1]+a[1,i](2<=i<=n)

例题15：乌龟棋（NOIP2010提高）

- 见word文档

动态规划与递推的关系

- 上题实质上是采用动态规划来求解,那么递推与动态规划之间到底是什么关系呢?
- 我们不妨画个图(如下图)。而通常人们理解的递推关系就是一般递推关系，故认为动态规划与递推关系是两个各自独立的个体。



动态规划与递推的关系

- 1、一般递推边界条件很明显，动态规划边界条件比较隐蔽，容易被忽视；
- 2、一般递推数学性较强，动态规划数学性相对较弱；
- 3、一般递推一般不划分阶段，动态规划一般有较为明显的阶段；

第三部分

递归策略

一、递归的概念

- 一个函数、过程、概念或数学结构，如果在其定义或说明内部又直接或间接地出现有其本身的引用，则称它们是递归的或者是递归定义的。在程序设计中，过程或函数直接或者间接调用自己，就被称为递归调用。

二、递归的基本思想

- 递归是借助于一个递归工作**栈**来实现；**递归=递推+回归**；
- **1.递推：**问题向一极推进，这一过程叫做递推；这一过程相当于**压栈**。
- **2.回归：**问题逐一解决，最后回到原问题，这一过程叫做回归。这一过程相当于**弹栈**。

二、递归的基本思想

例题1：用递归算法求n!

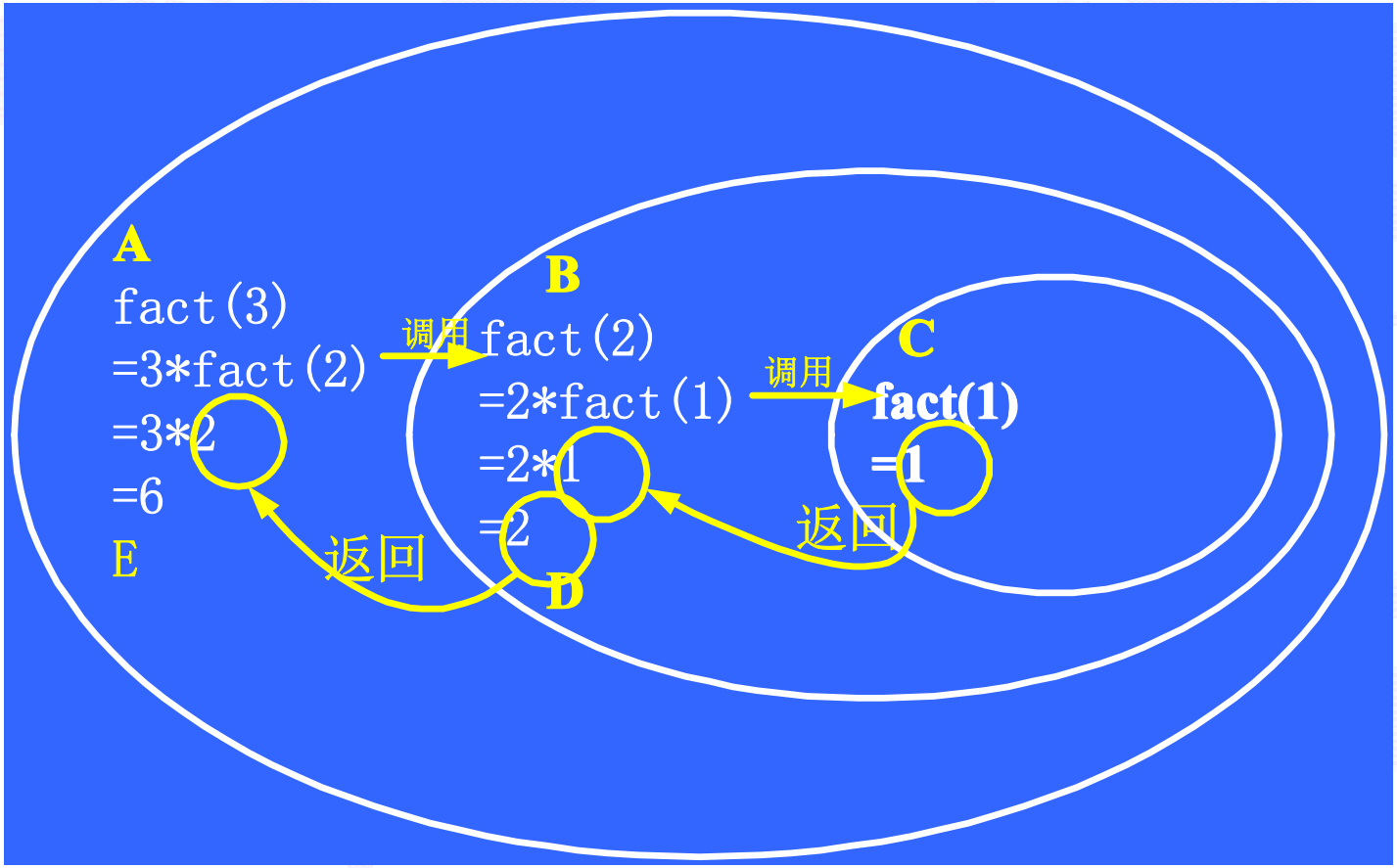
定义：函数 **fact(n)=n!**

fact(n-1)=(n-1)!

递推方程：**fact(n)=n*fact(n-1)**

边界条件：**fact(1)=1**

下面画出了调用和返回的递归示意图



三、递归的实现

- 递归实现的代价是巨大的栈空间的耗费，那是因为过程每向前递推一次，程序将本层的实在变量（值参和变参）、局部变量构成一个“工作记录”压入工作栈的栈顶，只有退出该层递归时，才将这一工作记录从栈顶弹出释放部分空间。
- 由此可以想到，减少每个“工作记录”的大小便可节省部分空间。例如某些变参可以转换为全局变量，某些值参可以省略以及过程内部的精简。

例题2：求a[1..n]的最大者。

有如下过程：

```
Procedure findmax(i:integer;var max:integer);
var j:integer;
begin
    max:=a[i];
    if i=n then exit
        else begin
                findmax(i+1,j);
                if j>max then max:=j;
             end;
end;
```

- 起始状态就是调用**findmax(1,max),**而像上述过程中的变参**max**完全可以省略。将上述方法修改可得下面的算法：

```
Procedure findmax(i:integer);
begin
   if i=n then exit
     else begin
            findmax(i+1);
            if a[i]>max then max:=a[i];
          end;
end;
```

■起始状态就是调用**findmax(1)**，**max**为全局变量，同时还减少了一个局部变量的使用。尽管这只是一个很简单的例子，在本例中不做精简，程序也还是能通过，但它精简的原则对其它使用递归的程序而言却是同样适用的。特别是在递归过程出现堆栈溢出情况时就应该考虑这一问题。

四、递归的优点与缺点

- 优点：采用递归方法编写的问题解决程序具有结构清晰，可读性强等优点，且递归算法的设计比非递归算法的设计往往要容易一些，所以当问题本身是递归定义的，或者问题所涉及到的数据结构是递归定义的，或者是问题的解决方法是递归形式的时候，往往采用递归算法来解决。
- 缺点：执行的效率很低，尤其在边界条件设置不当的情况下，极有可能陷入死循环或者内存溢出的窘境。

五、递归的应用

- 处理递归定义或解决方法为递归方式的问题
- 解决搜索问题和组合计数
- 实现分治思想
- 用于输出动态规划的中间过程

1、递归定义问题

- 树结构是由递归定义的。因此，在解决与树有关的问题时，常常可以采用递归的方法。

2、解决搜索问题

- 因为搜索产生的节点成树状结构，所以可以用递归方法解决。这类例子很多，如“N皇后”问题，全排列，哈密顿回路，图的可着色性等搜索问题。

例题3：全排列

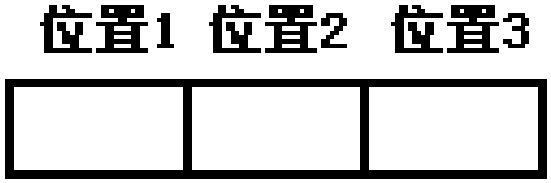
【问题描述】编程列举出1、2、...、n的全排列，要求产生的任一个数字序列中不允许出现重复的数字。

【文件输入】输入n(1<=n<=9)

【文件输出】有1到n组成的所有不重复数字的序列，每行一个序列

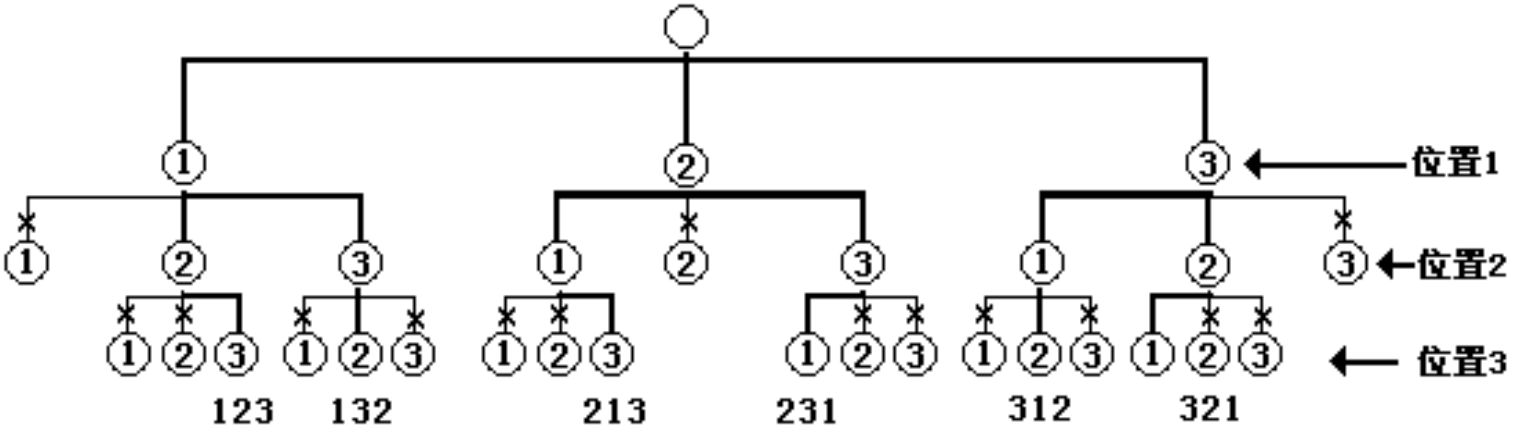
分析

- 我们假设n=3时，如下图：位置1可以放置数字1、2、3；位置2可以放置数字1、2、3；位置3可以放置数字1、2、3，但是当位置1放了数字1后位置2和位置3都不能在放1，因此画树的约束条件是：各位置的数字不能相同。



分析

- 我们画“解答树”时，根结点一般是一个空结点，根结点下面的第1、2、3三层分别对应位置1、位置2、位置3，用“╳”标示的分支表示该结点不满足约束条件，不能被扩展出来：



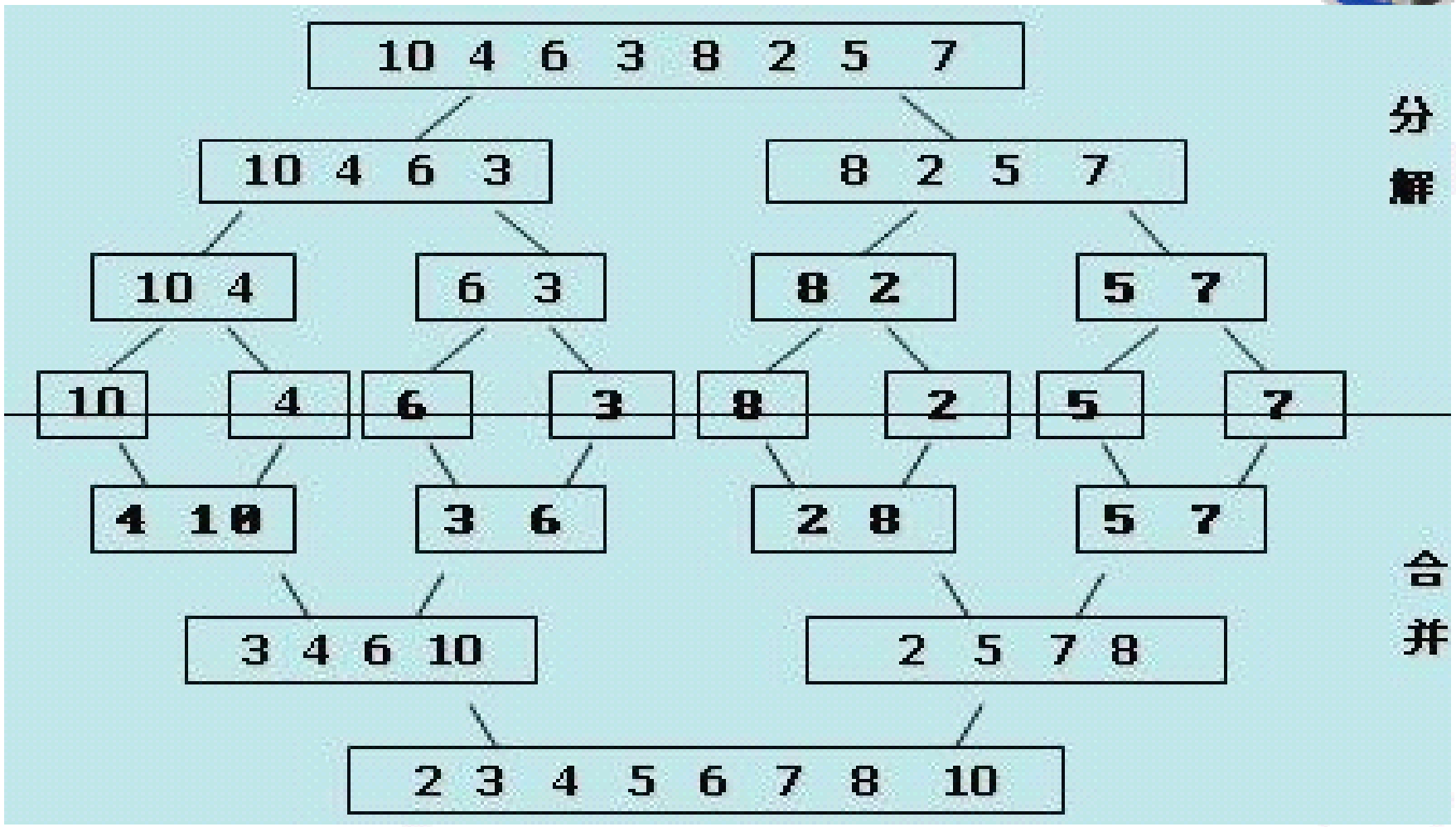
（解答树）

- 我们用递归过程来描述“解答树”的深度优先搜索

```
procedure f(k:integer)   //搜索第k层结点（向第k个位置放数）
begin
   var i:integer;
   if k=n+1 then begin for i:=1 to n do write(a[i]);writeln;end
         // 如果搜索到一条路径，则输出一种解
    else for i:=1 to n do//每一个结点可以分解出n个子结点；
            if b[i]=0 then //如果能生成第k层的第i个结点；
            begin
               a[k]:=i; //第k个位置为数字i;
                b[i]:=1; //标记数字i已用
                f(k+1); //扩展第k层的第i个结点(向第k+1个位置放数)
                b[i]:=0; //向上回溯，并恢复数据
            end;
end;
```

3、实现分治思想

- 不难发现，在各种时间复杂度为nlogn排序方法中，大都采用了递归的形式。
- 因此无论是归并排序，还是堆排序、快速排序，都存在有分治的思想。只要分开处理，就可以采用递归处理。其实进行分治，也是一个建树的过程。
- 如下图中归并排序的过程：


10 4 6 3 8 2 5 7
10 4 6 3
8 2 5 7
10 4
6 3
8 2
5 7
10
4
6
3
8
2
5
7
4 10
3 6
2 8
5 7
3 4 6 10
2 5 7 8
2 3 4 5 6 7 8 10
分解
合并

例题4：表达式求值

- 由键盘输入一个算术表达式，该表达式由数字，加(+)、减(-)、乘(*)、求商(/)运算符及小括号组成。
- 例如：6*(8-1)+(5-(12-7)/5)/2
- 请编写一个程序，计算输入表达式的值。

```
funtion tree(left,right:integer):integer;
begin
    var tmp,L1,L2,p:integer;
    tmp:=isnum(left,right);
    if tmp<>-1 begin tree:=tmp;exit;end;//返回值
    if (st[left]='(')and (poskh(left,right)=right) then
                    tree:=tree(left+1,right-1);
              //去掉最外层的括号
    p:=findlow(left,right);   //找运算级别最低的运算符
    L1:=tree(left,p-1);       //递归左子树
    L2:=tree(p+1,right);      //递归右子树
    tree:=cal(L1,st[p],L2);
end;
```

4、用于输出动态规划的中间过程

例题5：复制书稿

【问题描述】假设有m本书（编号为1,2,...m），想将每本复制一份，m本书的页数可能不同（分别是$p_1,p_2,...p_m$）。任务是将这m本书分给k个抄写员（k<=m），每本书只能分配给一个抄写员进行复制，而每个抄写员所分配到的书必须是连续顺序的。复制工作是同时开始进行的，并且每个抄写员复制的速度都是一样的。所以，复制完所有书稿所需时间取决于分配得到最多工作的那个抄写员的复制时间。

- 试找一个最优分配方案，使分配给每一个抄写员的页数的最大值尽可能小（如存在多个最优方案，输出结果排序最小的一种）。

分析

- 该题中m本书是顺序排列的，k个抄写员选择数也是顺序且连续的。不管以书的编号，还是以抄写员标号作为参变量划分阶段，都符合策略的最优化原理和无后效性。考虑到k<=m，以抄写员编号来划分阶段会方便些。
- 设f[i,j]为前i个抄写员复制前j本书的最小“页数最大数”。S[i]=a[1]+a[2]+…+a[i]，则状态转移方程为：
- f[i,j]=min{max(f[i-1,k],S[j]-S[k])}(i-1<=k<=j-1)
- d[i,j]=k；记录第i个人的最佳位置
- 边界条件 f[1,i]=S[i]。

■输出方案，则递归输出；

```
procedure Print(i,j:integer)
begin
    if i=1 then begin writeln(1,' ',j);exit;end;
    Print(i-1,d[i,j]);
    writeln(d[i,j]+1,' ',j);
end
```

第四部分

归纳策略

一、归纳法的基本思想

- **归纳法的基本思想：** 是通过列举少量的特殊情况，经过分析，最后找出一般的关系。
- 从本质上讲，归纳就是通过观察一些简单而特殊的情况，最后总结出有用的结论或解决问题的有效途径。

二、归纳法解题的过程

1. 细心的观察；
2. 丰富的联想；
3. 继续尝试；
4. 总结归纳出结论。

二、归纳法解题的过程

- 归纳是一种抽象，即从特殊现象中找出一般关系。
- 由于在归纳的过程中不可能对所有的可能情况进行枚举，因而最后得到的结论还只是一种猜测(即归纳假设)。所以，严格说来对于归纳假设还必须加以严格的证明。

三、归纳策略解题应注意的问题

1. 从问题的简单具体状态分析入手，目的是去寻求可以推广的一般性规律，因此应考虑简单状态与一般性状态之间的联系。
2. 从简单状态中分析出来的规律特征应能够被验证是正确的，不能想当然或任意地提出猜想，否则归纳出来的结论是错误的，必然导致整个问题的解是错解。

四、归纳策略的应用

例题1：求前n个自然数的平方之和：

$S=1^2+2^2+3^2+\cdots\cdots+n^2$

【分析】这本是一道很简单的题目，但如果能找出S值与n的关系，则此题将更进一步得到简化，由数学证明得知：

$$(1^2+2^2+3^2+\ldots+n^2)/(1+2+3+\ldots+n) =(2n+1)/3$$

又由于 $1+2+3+\ldots+n=n(n+1)/2$，因此得到：

$$1^2+2^2+3^2+\ldots+n^2=n(n+1)\ (2n+1)/6$$

但这只是通过总结归纳而得到的一种猜测，是否正确还需证明，对归纳假设的证明通常采用数学归纳法（证略）。

例题2：乘积最大

【问题描述】若干个正整数之和为n，其中：n<2000，试求它们乘积的最大值以及该最大值的位数k。

【分析】根据数学规律可知,若要使和固定的数的乘积最大，必须使这些数尽可能的多为3，于是可推得以下规律：

- 当N mod 3＝1 时，N可分解为一个4和若干个3的和；
- 当N mod 3＝2 时，N 可分解为一个2和若干个3的和；
- 当N mod 3＝0 时，N 直接分解为若干个3的和。

  按照这一分解方法，所有因数的乘积必定最大。

注意：因N 的最大值可达2000，乘积将超过长整型数据范围，所以需用高精度运算。

例题3：极值问题

已知m、n为整数，且满足下列两个条件：

① m、n∈1，2，…，K，（1≤K≤10^9）

② $(n^2-mn-m^2)^2=1$

编一程序，由键盘输入K，求一组满足上述两个条件的m、n，并且使m^2+n^2的值最大。例如，若K＝1995，则m＝987，n＝1597，则m、n满足条件，且可使m^2+n^2的值最大。

【分析】分析小数据会发现：m,n是Fibonacci数列的相邻两项。

因为：$(n^2-mn-m^2)^2=1$

故：$(m^2+mn-n^2)^2=1$

又：$m^2+mn-n^2=(m+n)^2-mn-2n^2$

$=(m+n)^2-(m+n)n-n^2$

故：$(m^2+mn-n^2)^2=[(m+n)^2-(m+n)n-n^2]^2$

即：$(n^2-mn-m^2)^2=[(m+n)^2-(m+n)n-n^2]^2$

【分析】由上述数学变换式可以得出，如果m和n为一组满足条件①和条件②的解，设n’=m+n，m’=n那么n’，m’也是一组满足条件①和条件②的一组解。

将所有满足条件①和条件②的m和n 按递增顺序排成一个Fibonacci数列 {1，1，2，3，5，8，……}

数列中小于k的最大两个相邻数即为试题所要求的一组m和n。

【算法】利用Fibonacci数列顺推m,n，求出在条件范围内的m,n最大值，此时m^2+n^2的值最大。

例题4：士兵排队问题

【问题描述】有N个士兵（1<=N<=100),编号依次为1,2,...,N。队列训练时，指挥官要把士兵从高到矮排成一行，但指挥官只知道“1 比2 高，7 比 5高”这样的比较结果。如果不能排成一排，则输出"No answer"。

【文件输入】第一行为一个整数N（N<=100），表示士兵的个数。接下来若干行每行两个数A和B，表示A高于B。(A,B属于 1 ..N)

【文件输出】输出文件只有一行为一个合法的排队序列。

【样例输入】

4
1 2
2 3
4 2
1 4

【样例输出】1 4 2 3

【思路点拨】由题意可知，所有士兵的身高关系对应一张图，每两个士兵的身高关系对应于一条边；A高于B对应于一条有向边，将B的入度加1，A→B这条边称之为A的出边。如果有环，显然不能得到排队方案；无环时，每次选入度为0的点，且删除与该点相连的边，再重复上述操作，即可得到士兵的排队方案。这种方法称为拓扑排序，能构造出最优解。