递推算法

递推法是一种重要的数学方法，在数学的各个领域中都有广泛的运用，也是计算机用于数值计算的一个重要算法。这种算法特点是：一个问题的求解需一系列的计算，在已知条件和所求问题之间总存在着某种相互联系的关系，在计算时，如果可以找到前后过程之间的数量关系（即递推式），那么，从问题出发逐步推到已知条件，此种方法叫逆推。无论顺推还是逆推，其关键是要找到递推式。这种处理问题的方法能使复杂运算化为若干步重复的简单运算，充分发挥出计算机擅长于重复处理的特点。

递推算法的首要问题是得到相邻的数据项间的关系（即递推关系）。递推算法避开了求通项公式的麻烦，把一个复杂的问题的求解，分解成了连续的若干步简单运算。一般说来，可以将递推算法看成是一种特殊的迭代算法。

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。

算法：

```
#include<iostream>
using namespace std;
int n,f[51];
int main(){
    cin>>n;
    f[0]=0;
    f[1]=1;
    for(int i=2;i<=n;i++){
        f[i]=f[i-1]+f[i-2];
    }
    cout<<f[n];
}
```

总结:

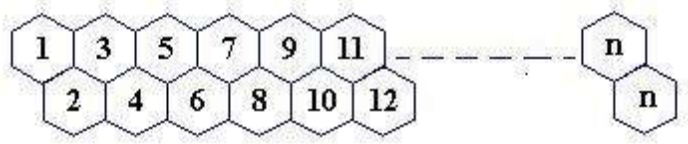
从这个问题可以看出，在计算裴波那契数列的每一项目时，都可以由前两项推出。这样，相邻两项之间的变化有一定的规律性，我们可以将这种规律归纳成如下简捷的递推关系式：Fn=g(Fn-1)，这就在数的序列中，建立起后项和前项之间的关系。然后从初始条件（或是最

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。

蜜蜂路线

【题目描述】
一只蜜蜂在下图所示的数字蜂房上爬动,已知它只能从标号小的蜂房爬到标号大的相邻蜂房,现在问你：蜜蜂从蜂房 M 开始爬到蜂房 N，M<N，有多少种爬行路线？



【输入】
输入 M，N 的值，1<=M<N<=50
【输出】
爬行有多少种路线。
【样例输入】
1 14
【样例输出】
377
【分析】
斐波那契数列数列。对于第 i 个蜂房，只能从第 i-1 个蜂房或者第 i-2 个蜂房爬过来，所以，设 f[i]表示爬到第 i 个蜂房的路线数目，则 f[i]=f[i-1]+f[i-2]。边界 :对于起点 f[m]=1,f[m+1]=1。
【代码】

```
#include<iostream>
using namespace std;
long long m,n,f[50];
int main(){
    cin>>m>>n;
    f[m]=1;
    f[m+1]=1;
    for(int i=m+2;i<=n;i++)f[i]=f[i-1]+f[i-2];
    cout<<f[n];
}
```

母牛生小牛

【题目描述】

有一头小母牛，从出生第四年起每年生一头小母牛，按此规律，第 N 年时有几头母牛？
【输入】
只有一个整数 N，独占一行。(1≤N≤50)
【输出】
对每组数据，输出一个整数（独占一行）表示第 N 年时母牛的数量 。
【样例输入】
样例输入 1
4
样例输入 2
20
【样例输出】
样例输出 1
2
样例输出 2
872
【分析】
斐波那契数列的变形。设 f[i]为第 i 年的母牛数目，则前一年的母牛会继续活到第 i 年，而且三年前的母牛，每头牛都会生出一头牛，则数量和三年前的数量一样，即 f[i]=f[i-1]+f[i-3]。
【代码】

```
#include<iostream>
using namespace std;
int n,f[51];
int main(){
    cin>>n;
    f[1]=f[2]=f[3]=1;
    for(int i=4;i<=n;i++){
        f[i]=f[i-1]+f[i-3];
    }
    cout<<f[n];
}
```

铺地砖

【题目描述】
一天，晨晨的数学老师布置了一道题目，大意如下：用 1×1 和 2×2 的瓷砖不重叠地铺满 n×3 的地板，共有多少种方案？
例如：
n=1 时：1×3 的地板方法就一个，直接由三个 1×1 的磁砖铺满。
n=2 时：2×3 的地板可以由下面 3 种方案铺满：

<table>
<tr><td>1×1</td><td rowspan="2">2×2</td></tr>
<tr><td>1×1</td></tr>
</table>

<table>
<tr><td rowspan="2">2×2</td><td>1×1</td></tr>
<tr><td>1×1</td></tr>
</table>

<table>
<tr><td>1×1</td><td>1×1</td><td>1×1</td></tr>
<tr><td>1×1</td><td>1×1</td><td>1×1</td></tr>
</table>

【输入】

第一行：一个整数 n（1≤n≤100）。

【输出】

输出铺满 n×3 的地板的方案数。

【样例输入】

3

【样例输出】

5

【数据范围】

对于 100%的数据，1≦n≦50;

【分析】

设 f[i]表示前 i 行的方案数，则 f[i]=f[i-1]+2*f[i-2]。

<table>
<tr><td rowspan="3">f[i-1]</td><td>1*1</td></tr>
<tr><td>1*1</td></tr>
<tr><td>1*1</td></tr>
<tr><td>前 i-1 行</td><td>第 i 行</td></tr>
</table>

<table>
<tr><td rowspan="2">F[i-2]</td><td colspan="2">2*2</td></tr>
<tr><td>1*1</td><td>1*1</td></tr>
<tr><td>前 i-2 行</td><td>第 i-1 行</td><td>第 i 行</td></tr>
</table>

<table>
<tr><td rowspan="2">F[i-2]</td><td>1*1</td><td>1*1</td></tr>
<tr><td colspan="2">2*2</td></tr>
<tr><td>前 i-2 行</td><td>第 i-1 行</td><td>第 i 行</td></tr>
</table>

【代码】

```
#include<iostream>
using namespace std;
int n;
long long f[51];
int main(){
    cin>>n;
    f[1]=1;
    f[2]=3;
    for(int i=3;i<=n;i++){
```

```
        f[i]=f[i-1]+2*f[i-2];
    }
    cout<<f[n];
}
```

Alpha 编码

【题目描述】
背景：Alice 和 Bob 之间要进行秘密通信，他们正在讨论如何对信息进行加密。
Alice：“不如采用一种很简单的加密方式：’A’替换成 1，’B’替换成 2，……，’Z’替换成 26。”
Bob：“这种加密方式太傻了，Alice。如果我想要传送一个单词’BEAN’给你，它加密后就是 25114。但你有很多种不同的方法来解密，从而得到许多单词！”
Alice：“你说的是没错，但是除了’BEAN’有意义以外，其他解密出来的’BEAAD’、’YAAD’、’YAN’、’YKD’和’BEKD’都没有任何含义。”
Bob：“是的，但是同一个加密后的数字序列，可能的得到数以亿计的不同解密方案。”
Alice：“是吗？有这么多吗？”
你要帮助 Bob 编写一个程序，来说服 Alice。对于一个加密后的数字序列，告诉她确切的解密方案数。
【输入】
有若干个加密后的数字序列，每行一个，行数不超过 10，每行的数字数量不超过 4100 个。序列一定是符合要求的，例如没有先导的零和连续两个零等情况。数字间没有空格。一行一个零表示输入结束，这是不需要处理的。
【输出】
对于每个加密后的数字序列，输出一行。一个整数，表示解密的不同方案数。结果保证在 32 位带符号整数（longint）范围内。
【样例输入】
25114
1111111111
3333333333
0
【样例输出】
6
89
1
【分析】
类似动态规划，设 f[i]表示以第 i 个字符结束所能构成的方案数，如果第 i 个字符独立当成一个字母，则 f[i]=f[i-1],前提是 s[i]不能为 0;如果 s[i-1]和 s[i]两个数字构成一个字母，前提是这两个数字所表示的数字在 10 到 26 之间，则 f[i]=f[i-2]。两种方案都可能，即 f[i]=f[i-1]+f[i-2],与斐波那契数列数列极像。
【代码】

```
#include<iostream>
#include<cstring>
using namespace std;
```

```
char s[5001];
int n,f[5001];
bool check(int i,int k){
    if(k==1){
        return s[i]!='0';
    }else{
        int x=(s[i-1]-'0')*10+s[i]-'0';
        return (x>=10)&&(x<=26);
    }
}
int main(){
    while(cin>>s+1){
        if(s[1]=='0')break;
        memset(f,0,sizeof(f));
        f[0]=f[1]=1;
        for(int i=2;i<=strlen(s+1);i++){
            f[i]=f[i-1]*check(i,1)+f[i-2]*check(i,2);
        }
        cout<<f[strlen(s+1)]<<endl;
    }
}
```

青蛙跳荷叶

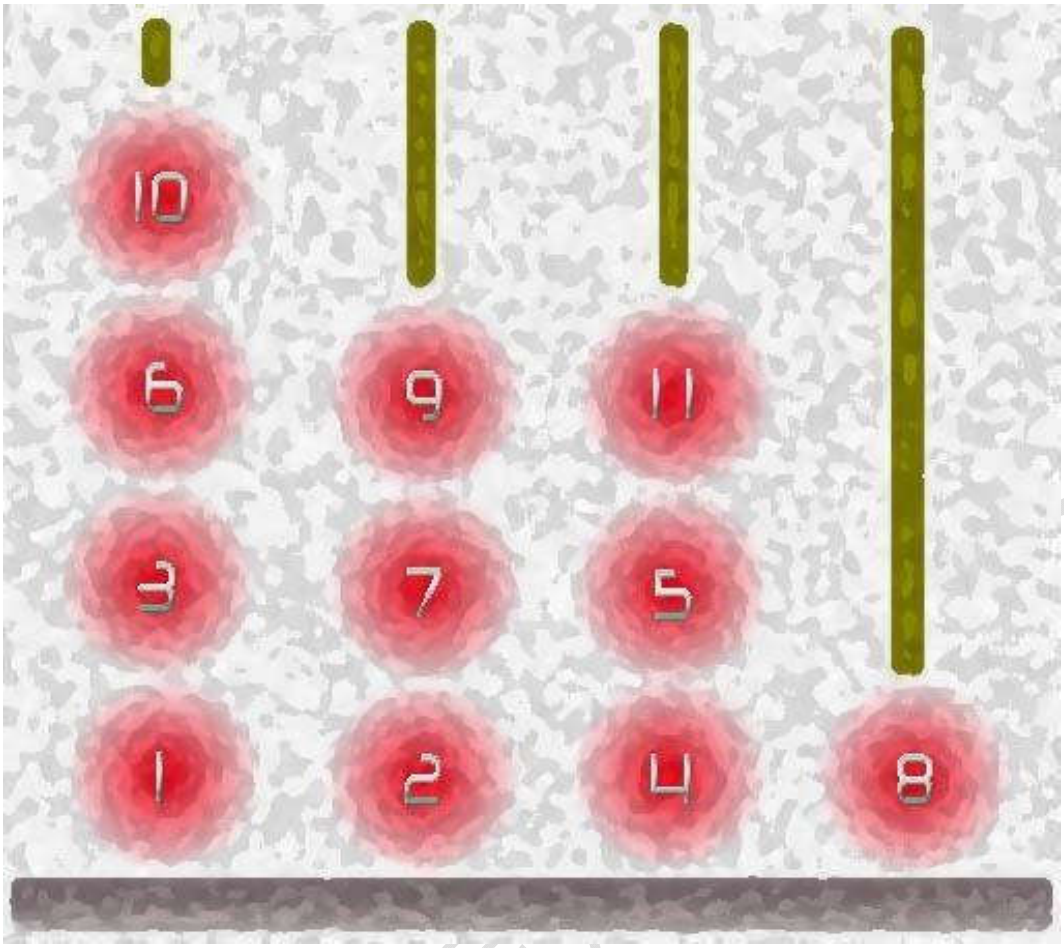
【题目描述】
从前，有一个小青蛙决定去荷叶上练习跳跃。
现在有 n 个荷叶排成一排，小青蛙一开始在最左边的荷叶(一号荷叶)上，当然，这个青蛙是很牛 X 的，可以在任意两个荷叶之间跳跃。
有一天这个青蛙突发奇想，想用一种奇怪的方式完成跳跃练习:
1.它希望每次跳到不同的荷叶上
2.每一次跳的距离不同
当然，作出这个决定是何其的简单，但是跳跃方式是何其的困难……,所以他希望你可以帮他解决这个问题。
下面给出这个问题严格的数学定义：
请给出 1 到 n 这 n 个自然数的一个排列 a1,a2,a3……an，使得
1：a1=1
2：对于任意的 i<>j(1<=i,j<=n-1)，有|ai-a(i+1)|<>|aj-a(j+1)|
其中 n 是给定的
【输入】
一行，一个数 n
【输出】
一行，n 个数，用一个空格隔开,末尾没有多余空格
【样例输入】

3
【样例输出】
1 3 2
【数据范围】
对于 20%的数据，1<n<=4
对于 100%的数据，1<n<=10000
另 |ai-a(i+1)|<>|aj-a(j+1)|即表示每次跳的距离要不一样不能重复，且只能从 1-n 之间选择。
【分析】
找规律。本题要求距离两两不同，我们找几个样例试下：
N=3 的排列：1 3 2
N=4 的排列：1 4 2 3
N=5 的排列：1 5 2 4 3
N=6 的排列：1 6 2 5 3 4
仔细观察，对于数列 n，相邻两数的差值分别为 n-1,n-2,n-3,……,3,2,1,而且第一个数为 1，这样，我们就可以模拟实现这个数列了。
【代码】

```
#include<iostream>
using namespace std;
int n,a[10001];
int main(){
    cin>>n;
    a[1]=1;
    int m=n-1;
    for(int i=2;i<=n;i++){
        if(i%2==0){
            a[i]=a[i-1]+m;
        }else{
            a[i]=a[i-1]-m;
        }
        m--;
    }
    for(int i=1;i<=n;i++){
        cout<<a[i]<<" ";
    }
}
```

又见汉诺塔

【题目描述】
A 先生发明了一个小游戏。游戏有 N 根柱子和很多球。球被编号为 1、2、3„，球看上去很普通，但它们实际上是有魔法的。如果上下相邻的放在一起的两个球编号之和不是平方数，他们将以一个强大的力量相互排斥，不能放在一起。



玩家每一次拿一颗球放在其中一根柱子上(从上往下放),玩家拿球按照 1 号,2 号,3 号.....这样的次序依次下去。你的任务是帮忙编程计算在 N 根柱子上最多可以放多少个球，注意当一根柱子上相邻的两个球编号之和不是平方数时是不能放在一起的。上图给出了四根柱子的最佳放法。

【输入】

输入一行一个整数 N ,表示柱子数量。

【输出】

输出一行，一个整数，表示最多可以放的球的数量，如果可以放无限多的球，请输出“-1”。

【输入样例 1】

4

【输入样例 2】

25

【输出样例 1】

11

【输出样例 2】

337

【数据规模】

对于 50%的数据： 1≤N≤50； 对于 100%的数据：1≤N≤5,000；

【分析】

对于本题，我们同样尝试找规律。

我们设 f[n]表示第 n 个柱子时能放的珠子个数，则：

F[1]=1
F[2]=3　与前一个数差值 2
F[3]=7　与前一个数差值 4
F[4]=11 与前一个数差值 4
F[5]=17 与前一个数差值 6
F[6]=23 与前一个数差值 6
依次递推，我们可以发现规律，差值每过两个数自增 2，由此可以模拟实现本题。
【代码】

```
#include<iostream>
using namespace std;
int n,f[5001],k=2;
int main(){
    cin>>n;
    f[1]=1;
    for(int i=2;i<=n;i++){
        if(i%2!=0)k+=2;
        f[i]=f[i-1]+k;
    }
    cout<<f[n];
}
```