

# 递推算法

# 递推基础知识

斐波那契数列是这样一系列数: 1 1 2 3 5 8 13 21 34 55 ..., 可以发现从第三项开始, 每一项是前两项之和。

And 斐波那契数列是怎么来的?

早在 1202 年, 意大利数学家 Leonardo Pisano, Fibonacci 在《算盘书》中提出兔子问题: 假设一对初生兔子一个月到成熟期, 一对成熟兔子每月生一对兔子, 那么由一对初生兔子开始, 6 个月后会多少对兔子?

每个月兔子数 = 每个月初生兔子数 + 每个月成熟兔子数  
= 上个月成熟兔子数 + (上个月初生兔子数 + 上个月成熟兔子数)  
= 上个月成熟兔子数 + 上个月初生兔子数 + (上上个月成熟兔子数 + 上上个月初生兔子数)  
= 上个月兔子数 + 上上个月兔子数

到第 6 个月兔子数是 8 对, 其中初生兔子 3 对, 成熟兔子 5 对。用  $f_n$  表示第  $n$  个月兔子的对数, 则二阶递推公式为:

从第三项起, 每项为前两项的和。

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

以上就是斐波那契数列, 斐波那契数列中每个数都是斐波那契数。

## 7.1-1

```
#include<bits/stdc++.h> // 斐波那契数列 , 兔子繁殖
```

```
using namespace std;
```

```
int n;
```

```
long long a[57];
```

```
int main()
```

```
{
```

```
    cin>>n;
```

```
    a[0]=0;
```

```
    a[1]=1;
```

```
    for (int i=2;i<=n;i++) a[i]=a[i-1]+a[i-2];
```

```
    cout<<a[n];
```

```
    return 0;
```

```
}
```

```

#include<bits/stdc++.h> // 递归解法，仅作参考
using namespace std;
long long tu[50];
long long fun(int n) // fun 意思是函数
{
    if(n==1 || n==2)    tu[n]=1;
    if(tu[n]!=0)        return tu[n];
    // 如果数值不为 0，表示数值已经计算出，不用再计算

    if(tu[n]==0)        return tu[n]=(fun(n-1)+fun(n-2));
    // 如果数值为 0，递归，计算出兔子数量
}
int main()
{
    long long n;
    cin>>n;

    cout<<fun(n);
    return 0;
}

```

```

#include<bits/stdc++.h> // 递归， 本程序超时 ， 不要使用， 仅做介绍。
using namespace std;
int fun(int n)
{
    if(n==1 || n==2) return 1;
    else if(n<=0) return 0;
        else return fun(n-1)+fun(n-2);
}
int main()
{
    int n;
    cin>>n;
    cout<<fun(n);
    return 0;
}

```

### 7.1-2-a

```
#include<bits/stdc++.h> // 母牛生小牛
using namespace std;
int n;
long long a[57];
int main()
{
    cin>>n;
    a[0]=0;
    a[1]=1;
    a[2]=1;
    a[3]=1;
    for (int i=4;i<=n;i++) a[i]=a[i-1]+a[i-3];
    cout<<a[n];
    return 0;
}
```

### 7.1-2-b

```
#include<iostream> // 母牛生小牛，函数（仅供参考）
using namespace std;
int cow(int year); // cow 牛
int main()
{
    int year;
    cin>>year;
    cout<<cow(year)<<endl;
    return 0;
}
int cow(int N)
{
    if(N<4) return 1;
    else return cow(N-1)+cow(N-3);
}
```

设有一头小母牛，从出生第四年起每年生一头小母牛，按此规律，问第 N 年时有几头母牛（假设牛是永生的）。  
[输入格式] 一行，一个整数 N，表示年份。

[输出格式] 一行，一个整数，表示第 N 年时母牛的数量。

输入：5 输出：3

输入：20 输出：872

[数据范围]

对于 100% 的数据： $n \leq 50$ 。

输入：5  
输出：3

输入：20  
输入：872

走楼梯，每次走一级台阶或者走二级台阶，走 n 多个台阶，问：有多少种走法？

### 7.1-3-a

```
#include<bits/stdc++.h> // 走楼梯
using namespace std;
int n;
long long a[57];
int main()
{
    cin>>n;
    a[0]=0;
    a[1]=1;
    a[2]=2;
    for (int i=3;i<=n;i++) a[i]=a[i-1]+a[i-2];
    cout<<a[n];
    return 0;
}
```

输入：5  
输出：8

走楼梯，有三种走法，每次走一级台阶，或者走二级台阶，或者走三级台阶，走 n 多个台阶，问：有多少种走法？

### 7.1-3-b

```
#include<iostream> // 骨牌（走楼梯）
using namespace std;
long long int a[10005];
int n, i;
int main()
{
    cin>>n;
    a[1]=1;
    a[2]=2;
    a[3]=4;
    for (i=4;i<=n;i++)
        a[i]=a[i-1]+a[i-2]+a[i-3];
    cout<<a[n];
    return 0;
}
```

有  $1 \times n$  的一个长方形，尝试用一个  $1 \times 1$ 、 $1 \times 2$  和  $1 \times 3$  的骨牌铺满方格。例如当  $n=3$  时为  $1 \times 3$  的方格。此时用  $1 \times 1$ 、 $1 \times 2$  和  $1 \times 3$  的骨牌铺满方格，共有四种铺法。

[输入格式]

一个整数 n，表示方格数。

[输出格式]

一个整数，表示方法总数。

输入：3 输出：4

输入：8 输出：81

输入：4  
输出：7  
输入：5  
输出：13  
输入：6  
输入：24

猴子吃桃问题：猴子第一天摘下若干个桃子，当即吃了一半，还不瘾，又多吃了一个；第二天早上又将剩下的桃子吃掉一半，又多吃了一个；以后每天早上都吃了前一天剩下的一半零一个；到第 10 天早上想再吃时，见只剩下一个桃子了。求第一天共摘了多少。

#### 7.1-4-a

```
#include<bits/stdc++.h> // 猴子吃桃子
using namespace std;
int n;
long long a[100];
int main()
{
    int i;
    a[10]=1;
    for (i=9; i>=1; i--) a[i]=(a[i+1]+1)*2;
    cout<<a[1];
    return 0;
}
```

输出：1534

一堆桃子不知数目，猴子第一天吃掉一半，又多吃了  $x$  个，第二天照此方法，吃掉剩下桃子的半又多  $x$  个，天天如此，到第  $n$  天后，猴子发现只剩 3 只桃子了，问这堆桃子在第  $t$  天还剩多少个？

[输入] 第一行两个整数  $n, x$ ，表示吃的天数和每天多吃的数量；

接下来若干行每行一个整数  $t$ ，询问第  $t$  天清晨桃子的数量

[输出] 若干行，每行一个整数，输出询问的答案

#### 7.1-4-b

```
#include<iostream> // 猴子吃桃
using namespace std;
int i, n, t, x, f[57];
int main()
{
    cin>>n>>x;
    f[n+1]=3;
    for (i=n; i>=1; i--) f[i]=(f[i+1]+x)*2;
    while (cin>>t) cout<<f[t]<<endl;
    return 0;
}
```

输入：  
5 3  
2  
3  
5  
输出：  
138  
66  
12

【题目描述】科学家在热带森林中发现了一种特殊的昆虫，这种昆虫的繁殖能力很强。每对成虫过  $x$  个月产  $y$  对卵，每对卵要过两个月长成成虫。假设每个成虫不死，第一个月只有一对成虫，且卵长成成虫后的第一个月不产卵（过  $x$  个月产卵），问过  $z$  个月以后，共有成虫多少对？ $0 \leq x \leq 20, 1 \leq y \leq 20, X \leq z \leq 50$ 。

【输入】 $x, y, z$  的数值。 【输出】过  $z$  个月以后，共有成虫对数。

【输入样例】1 2 8 【输出样例】37

### 7.1-5

```
#include<bits/stdc++.h>
```

```
using namespace std;
```

```
int x, y, z;
```

```
long long a[55]; // 成虫
```

```
long long b[55]; // 幼虫
```

```
int main()
```

```
{
```

```
    cin>>x>>y>>z;
```

```
    for (int i=1; i<=x; i++)
```

```
    {
```

```
        a[i]=1;
```

```
        b[i]=0;
```

```
    }
```

```
    /* 每对成虫过  $x$  个月产  $y$  对卵，每对卵要过两个月长成成虫。
```

```
    假设每个成虫不死，第一个月只有一对成虫，
```

```
    且卵长成成虫后的第一个月不产卵（过  $x$  个月产卵），
```

```
    问过  $z$  个月以后，共有成虫多少对 */
```

```
    // 每隔  $x$  个月循环一次 // 最后少于  $z$  个月，因为从 1 开始算，所以要加 1
```

```
    for (int i=x+1; i<=z+1; i++)
```

```
    {
```

```
        b[i] = y*a[i-x]; // 成虫是每隔  $x$  个月产  $y$  对卵
```

```
        a[i] = a[i-1] + b[i-2];
```

```
        // 这个月的成虫数等于上个月成虫数 + 前两个月的卵的和 .
```

```
    }
```

```
    cout<<a[z+1]<<endl;
```

```
    return 0;
```

```
}
```



## 7.1-6-b

```
#include<iostream>// 矩阵行走 递推
using namespace std;// 路径条数 =a[i-1][j]+a[i][j-1]
int main()
{
    int i, j, n, m;
    int a[100][100]={0}; // 存储图形
    cin>>n>>m;
    a[0][0]=1;
    for (i=0; i<=n; i++)
    {
        for (j=0; j<=m; j++)
        {
            if (i==0&&j==0) continue;
            if (i==0) a[i][j]=a[i][j-1];
            else if (j==0) a[i][j]=a[i-1][j];
            else a[i][j]=a[i-1][j]+a[i][j-1];
        }
    }
    for (i=0; i<=n; i++)
    {
        for (j=0; j<=m; j++)
        {
            cout<<a[i][j]<<" ";
        }
        cout<<endl;
    }
}
```

7.1-7

```
#include<iostream>//pell 数列
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int i, x;
```

```
    long long a[100];
```

```
    a[1]=1;
```

```
    a[2]=2;
```

```
    cin>>x;
```

```
    for (i=3; i<=x; i++)
```

```
        a[i]=a[i-1]*2+a[i-2];
```

```
    for (i=1; i<=x; i++)
```

```
        cout<<a[i]<<" ";
```

```
    return 0;
```

```
}
```

10

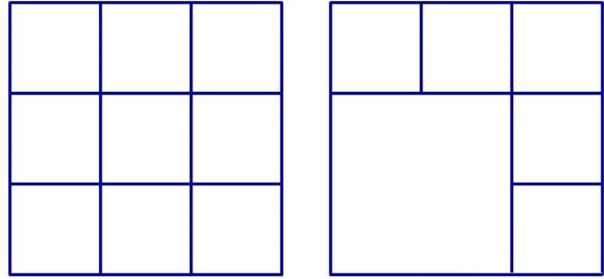
1 2 5 12 29 70 169

408 985 2378

7.1-8

#include<iostream>//n\*n 正方形包含几个正方形?

```
using namespace std;
int main()
{
    int i, n;
    int sum=0;
    cin>>n;
    for (i=n; i>=1; i--)
    {
        sum+=i*i;
    }
    cout<<sum;
    return 0;
}
```

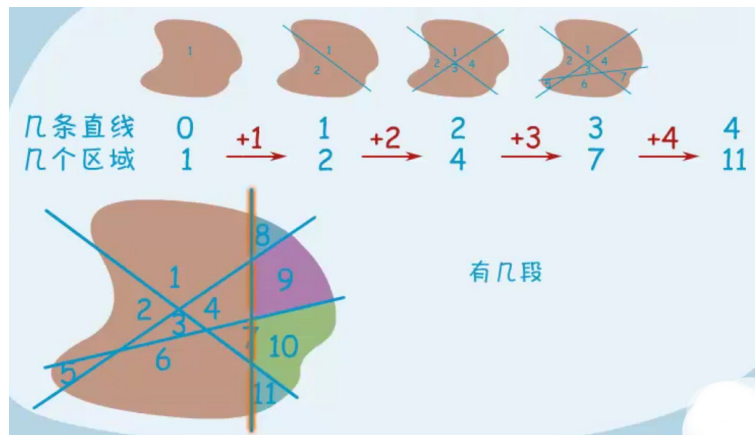


输入: 5  
输出: 55  
  
输入: 6  
输出: 91

7.1-9

#include<iostream>// 直线平分面

```
using namespace std;
int main()
{
    int i, n;
    int sum[100]={0};
    cin>>n;
    sum[0]=1;
    for (i=1; i<=n; i++)
    {
        sum[i]=sum[i-1]+i;
    }
    for (i=0; i<=n; i++)
        cout<<sum[i]<<" ";
    return 0;
}
```



7  
1 2 4 7 11 16 22 29

# 递推练习题

选自东方博宜

```
#include <bits/stdc++.h> //2-1146 递推 javacn
using namespace std;
int s = 0, x = 1; //x 代表每一项的值
int main()
{
    for (int i = 1; s < 5000; i++)
    {
        s = s + x; // 递推:  $A(n)=A(n-1)+(n-1)$ 
        x = x + i;
    }
    cout << s;
    return 0;
}
```

1146. 求 S 的值

问题描述

求  $S=1+2+4+7+11+16\cdots$   
的值刚好大于等于 5000  
时 S 的值。

输入

无

输出

一行，一个整数。

```
#include<bits/stdc++.h> //1146, 递归, 仅作参考
using namespace std;
long long ss[50];
long long fun(int n) { //fun 意思是函数
    if (n==0) ss[n]=1;
    if (ss[n]!=0) return ss[n];
    if (ss[n]==0) return ss[n]=(fun(n-1)+n);
}
int main() {
    long long s=0;
    int i=0;
    while (s<=5000) { s=s+fun(i); i++; }
    cout << s;
    return 0;
}
```

```

#include <bits/stdc++.h>//3-1147 递推 javacn
using namespace std;
long long a[50];
int n;
int main()
{
    cin>>n;
    a[1] = 1; // 求出斐波拉契数列的前 n+1 项
    a[2] = 1;
    for (int i = 3; i <= n + 1; i++)
        a[i] = a[i - 1] + a[i - 2];
    double ans = 0; // 求结果
    for (int i = 1; i <= n; i++)
        ans += a[i] * 1.0 / a[i+1];
    printf("%.3lf", ans);
    return 0;
}

```

```

#include<bits/stdc++.h>//1147. 递归, 仅做参考
using namespace std;
long long tu[50];
long long fun(int n) {
    if(n==1||n==2) tu[n]=1;
    if(tu[n]!=0) return tu[n];
    if(tu[n]==0) return tu[n]=(fun(n-1)+fun(n-2));
}
int main() {
    double s=0;// 总和
    int n;
    cin>>n;
    for (int i=1;i<=n;i++)
        s=s+fun(i)*1.0/fun(i+1);
    printf("%.3lf", s);
    return 0;
}

```

求  $1/1+1/2+2/3+3/5+5/8$   
 $+8/13+13/21+21/34 \dots$   
 的前  $n$  项的和。

输入

输入一个整数  $n$

( $1 \leq n \leq 30$ )。

输出

输出一个小数, 即前  $n$   
 项之和(保留 3 位小数)。

输入

20

输出

12.660

思路:

分子和分母分别是斐波拉  
 契数列

分子是第  $i$  项时, 分母是  
 第  $i+1$  项

因此如果要求前  $n$  项的  
 值, 需要求出斐波拉契数  
 列的前  $n+1$  项

### 1145. 数列求和

有一数列如下： 1 2 4 7 11 16 22 …… 试求该数列前 N 项之和。

输入 一个整数 N ( 0<N<1000 )。 输出 一个整数。

输入 6 输出 41

```
#include<bits/stdc++.h>//4-1145-1 递推 数列求和 w2016010182
```

```
using namespace std;
long long a[1005];
int main()
{
    int n, i;
    long long sum=1;
    cin>>n;
    a[0]=1;
    for (i=1; i<n; i++)// 所有项数和
    {
        a[i]=a[i-1]+i;
        sum=sum+a[i];
    }
    cout<<sum;
    return 0;
}
```

```
#include<bits/stdc++.h>//4-1145-2 递推 hasome
```

```
using namespace std;
int main() {
    int n, i, t=1, s=0;
    cin>>n;
    for (i=0; i<n; i++)
    {
        t=t+i;
        s=s+t;
    }
    cout<<s<<endl;
    return 0;
}
```

```
#include<bits/stdc++.h>//1145 递归, 仅作参考
```

```
using namespace std;
```

```
long long a[1005];
```

```
long long qh(int i)
```

```
{  
    if(i==0)  
        a[0]=1;  
    if(a[i]!=0)  
        return a[i];  
    else  
        return a[i]= qh(i-1)+i;  
}
```

```
int main()
```

```
{  
    int n, i;  
    long long sum=0;  
    cin>>n;  
    for (i=0; i<n; i++)// 所有项数和  
        sum=sum+qh(i);  
  
    cout<<sum;  
}
```

1148. 数数小木块 在墙角堆放着一堆完全相同的正方体小木块，如下图所示：  
因为木块堆得实在是太有规律了，你只要知道它的层数就可以计算所有木块的数量了。

输入：只有一个整数  $n$ ，表示这堆小木块的层数，已知  $1 \leq n \leq 100$ 。

输出：只有一个整数，表示这堆小木块的总数量。

输入 5 输出 35

```
#include <bits/stdc++.h>//6-1148-1
```

```
using namespace std;
```

```
int mu[100];
```

```
int main() {
```

```
    int n;
```

```
    cin>>n;
```

```
    int s=0;
```

```
    for (int i=1;i<=n;i++)
```

```
    {
```

```
        mu[i] = mu[i-1]+i;
```

```
        s+=mu[i];
```

```
    }
```

```
    cout<<s;
```

```
    return 0;
```

```
}
```

```
#include <bits/stdc++.h>//6-1148-2
```

```
using namespace std;
```

```
int main() {
```

```
    int n;
```

```
    cin>>n;
```

```
    int a = 1;
```

```
    int s = 1;
```

```
    for (int i=2;i<=n;i++)
```

```
    {
```

```
        a = a+i;
```

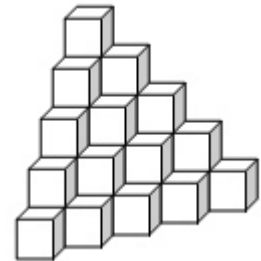
```
        s+=a;
```

```
    }
```

```
    cout<<s;
```

```
    return 0;
```

```
}
```



先找出每层方块的次数，然后找出规律，可以计算每层的次数。

1

1+2

1+2+3

1+2+3+4

1+2+3+4+5

## 1216. 数塔问题

有如下所示的数塔，要求从底层走到顶层，若每一步只能走到相邻的结点，则经过的结点的数字之和最大是多少？

输入：输入数据首先包括一个整数整数  $N$  ( $1 \leq N \leq 100$ )，表示数塔的高度，接下来用  $N$  行数字表示数塔，其中第  $i$  行有个  $i$  个整数，且所有的整数均在区间  $[0, 99]$  内。

输出：从底层走到顶层经过的数字的最大和是多少？

输入

5

7

3 8

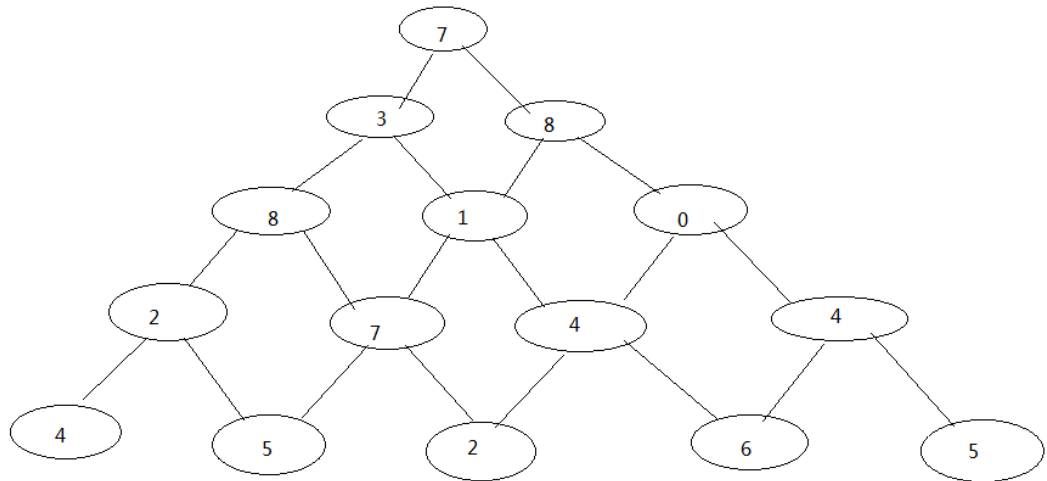
8 1 0

2 7 4 4

4 5 2 6 5

输出

30



```
#include<bits/stdc++.h>//7-1216  改自 javacn
```

```
#include<iomanip>
```

```
using namespace std;
```

```
int a[110][110];
```

```
int main()
```

```
{
```

```
    int n;
```

```
    cin>>n;
```

```
    for (int i=1;i<=n;i++)
```

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

```
            cin>>a[i][j];
```

```
// 从倒数第 2 层开始逆推，每个点累加下方的值和下方右边的值中的较大值
```

```
    for (int i=n-1;i>=1;i--)
```

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

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

```
    cout<<a[1][1];
```

```
    return 0;
```

```
}
```

思路：将数塔存入二维数组，从倒数第 2 层开始，递推计算出走到每个点最多能够累计的最大数字和，直到第 1 层，就能求出所经过结点的数字和的最大值

### 1224. 过河卒

A 点有一个过河卒，需要走到目标 B 点。

卒行走规则：可以向下、或者向右。同时在棋盘上的任一点有一个对方的马（如下图的 C 点），该马所在的点和所有跳跃一步可达的点称为对方马的控制点。

例如：下图 C 点可以控制 9 个点（图中的 P1, P2 …… P8 和 C ），卒不能通过对方马的控制点。棋盘用坐标表示，现给定 A 点位置为  $(0, 0)$ ，B 点位置为  $(n, m)$  ( $n, m$  为不超过 10 的整数)，马的位置 C 为  $(X, Y)$ （约定：C 点与 A 点不重叠，与 B 点也不重叠）。要求你计算出卒从 A 点能够到达 B 点的路径的条数。

输入

B 点的坐标  $(n, m)$  以及对方马的坐标  $(X, Y)$ ；（马的坐标一定在棋盘范围内，但要注意，可能落在边界的轴上）

输出

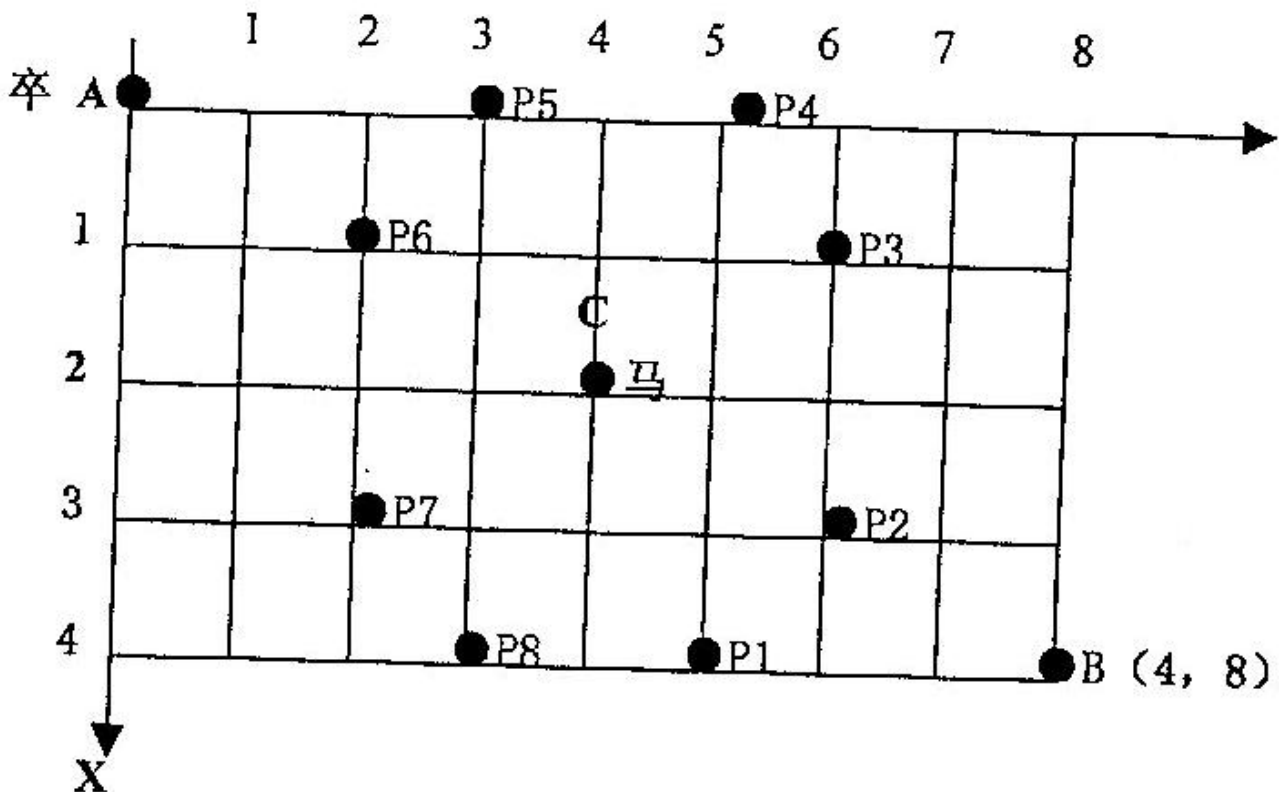
输出卒从 A 点能够到达 B 点的路径条数。

输入

6 6 3 2

输出

17



#include<iostream>//8-1224-1   jiangyf70   推荐学习

```
using namespace std;
long long a[25][25];
int main()
{
    int n, m, x, y;
    cin >> n >> m >> x >> y;
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= m; j++) a[i][j] = -1;
    }
    int dx[10] = {0, -1, -1, -2, -2, 1, 1, 2, 2};
    int dy[10] = {0, -2, 2, -1, 1, -2, 2, -1, 1};
    for (int i = 1; i <= 9; i++)
    {
        int ix = dx[i] + x;
        int iy = dy[i] + y;
        if (ix >= 0 && ix <= n && iy >= 0 && iy <= m) a[ix][iy] = 0;
    }

    a[0][0] = 1;
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= m; j++)
        {
            if (i == 0 && a[i][j] == -1) a[i][j] = a[i][j-1];
            else if (j == 0 && a[i][j] == -1) a[i][j] = a[i-1][j];
            else if (a[i][j] == -1) a[i][j] = a[i][j-1] + a[i-1][j];
        }
    }

    cout << a[n][m];
    return 0;
}
```

```
#include<bits/stdc++.h>//8-1224-2 javacn 推荐学习
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    long long a[30][30];
```

```
    int n,m,x,y,i,j;
```

```
    cin>>n>>m>>x>>y;
```

```
    for (i=0; i<=n; i++)
```

```
    {
```

```
        for (j=0; j<=m; j++)    a[i][j]=1;
```

```
    }
```

```
    a[x][y] = 0;    // 把控制点设置为 0
```

```
    if (x-1>=0&&y-2>=0) a[x-1][y-2] = 0;    // 如果控制点在棋盘内，设置为 0
```

```
    if (x-2>=0&&y-1>=0) a[x-2][y-1] = 0;
```

```
    if (x-2>=0&&y+1<=m) a[x-2][y+1] = 0;
```

```
    if (x-1>=0&&y+2<=m) a[x-1][y+2] = 0;    // 可以这样写    n>=x-1>=0&&0<=y+2<=m
```

```
    if (x+1<=n&&y+2<=m) a[x+1][y+2] = 0;
```

```
    if (x+2<=n&&y+1<=m) a[x+2][y+1] = 0;
```

```
    if (x+2<=n&&y-1>=0) a[x+2][y-1] = 0;
```

```
    if (x+1<=n&&y-2>=0) a[x+1][y-2] = 0;
```

```
    // 递推计算
```

```
    for (i=0; i<=n; i++)
```

```
    {
```

```
        for (j=0; j<=m; j++)
```

```
        {
```

```
            if (i==0&&j==0)    {    continue; }// 省略本次循环，直接下一个
```

```
            if (a[i][j]==0)    {    continue;    }
```

```
            if (i==0)    {    a[i][j] = a[i][j-1];    }
```

```
            else if (j==0)    {    a[i][j] = a[i-1][j];    }
```

```
                else    {    a[i][j] = a[i][j-1] + a[i-1][j];    }
```

```
        }
```

```
    }
```

```
    cout<<a[n][m];
```

```
    return 0;
```

```
}
```

## 1298. 摘花生问题

Hello Kitty 想摘点花生送给她喜欢的米老鼠。她来到一片有网格状道路的矩形花生地（如下图），从西北角进去，东南角出来。地里每个道路的交叉点上都有种着一株花生苗，上面有若干颗花生，经过一株花生苗就能摘走该它上面所有的花生。Hello Kitty 只能向东或向南走，不能向西或向北走。问 Hello Kitty 最多能够摘到多少颗花生。

```
#include<bits/stdc++.h> //9-1298   javacn
```

如输入：2 2

```
using namespace std;
```

1 1

```
int main()
```

3 4

```
{
```

代表有 2 行，每行有 2 株花生，那么摘能摘到的最多的花生就是：1->3->4，总和为 8 颗花生。

```
    int a[110][110]={0}, m, n, i, j;
```

```
    cin>>m>>n;
```

```
    for (i=1; i<=m; i++)
```

输入：第一行是两个整数 n 和 m（ $1 \leq n, m \leq 100$ ），代表了花生地里有 n 行，

```
    {
```

```
        for (j=1; j<=n; j++)
```

每行有 m 列的花生！后面 n 行，每行有 m 个整数代表了每行中，每株花生的数量！（每株花生数量  $\leq 10000$ ）

```
        {
```

```
            cin>>a[i][j];
```

```
        }
```

```
    }
```

```
    // 数组通过计算存储当前累计的最大值：
```

```
    // 当前最大值 = 当前值 +
```

```
    // ( 左边值和上面值中的较大值 )
```

输出：最多摘的花生总数

```
    for (i=1; i<=m; i++)
```

输入：2 2 输出：8

```
    {
```

```
        for (j=1; j<=n; j++)
```

1 1

```
        {
```

3 4

```
            a[i][j] = a[i][j] + max(a[i][j-1], a[i-1][j]);
```

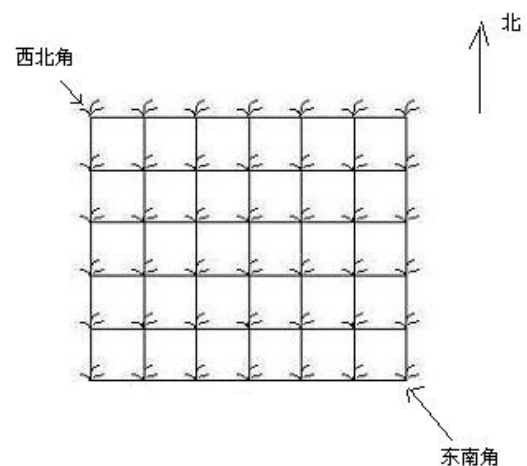
```
        }
```

```
    }
```

```
    cout<<a[m][n];
```

```
    return 0;
```

```
}
```



### 1374. 摘花生问题 (2)

Hello Kitty 又一次来到花生地里摘花生，从左上角进入花生地，从右下角出去，只能向右或者向下，请问 Hello Kitty 应该沿着什么样的路线走，能够摘到的花生数量最多（假设花生地里没有任何 2 株的花生一样多，也不存在多条路线能够摘到一样多的花生的情况）？

比如输入：

2 2

1 2

3 4

应该输出：1-3-4，也就是按照 1 3 4 这三株数量的花生摘过去，能够摘到最多的花生！

输入：第一行是 2 个整数  $m$  和  $n$  ( $2 \leq m, n \leq 100$ )，代表花生地有  $m$  行， $n$  列花生！后面  $m$  行，每行有  $n$  个整数代表了每行中，每株花生的数量。

输出：输出 Hello Kitty 按照走过的路线中，摘到每株花生的数量。

输入

2 2

1 2

3 4

输出

1-3-4

解题思路：

1、计算出走到每个点最多能摘到多少花生，由于每个点不是从左侧来就是从上方来，因此走到每个点能够摘到花生的数量 = 该点的花生数量 +  $\max$ (走到左侧能摘到的最多花生数量，走到上方能摘到的最多花生数量)

2、倒过来推导每个点是从哪个点来的；

```

#include<bits/stdc++.h>//10-1374-1  javacn
using namespace std;
int n,m,a[110][110];
int r[210];
int main() {
    cin>>n>>m; // 读入
    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++) // 计算走到每个点最多能摘到几个花生
        for (int j = 1;j <= m;j++)
            a[i][j] = a[i][j] + max(a[i][j-1], a[i-1][j]);

    int x = n,y = m;// 终点 // 仿照数塔问题，倒过来计算路线
    for (int i = 1;i <= n + m - 1;i++) // 一共经过 n+m-1 个点
    {
        if(a[x][y-1] > a[x-1][y]) // 如果从左侧来
        {
            r[i] = a[x][y] - a[x][y-1];
            y--; // 让 x, y 到左侧
        }
        else
        {
            r[i] = a[x][y] - a[x-1][y]; // 从上方来
            x--;
        }
    }
    for (int i = n + m - 1;i >= 1;i--) // 输出结果
    {
        cout<<r[i];
        if(i != 1) cout<<"-";
    }
    return 0;
}

```

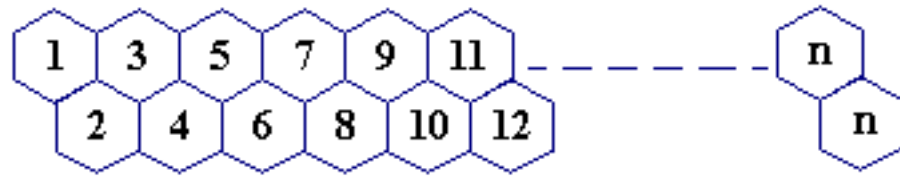
```

#include<iostream>//10-1374-2   jiangyf70
using namespace std;
int main()
{
    int a[105][105] = {0};
    int b[105][105] = {0};
    int n, m;
    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++)
        {
            b[i][j] = a[i][j] + max(b[i-1][j], b[i][j-1]);
        }
    }
    int c[200] = {0};
    c[0] = a[n][m];
    int i = n, j = m, k = 0;
    while(!(i == 1 && j == 1))
    {
        if(b[i-1][j] < b[i][j-1])    j--;
        else    i--;
        c[++k] = a[i][j];
    }
    for(i = k; i >= 0; i--)
    {
        if(i == k)    cout << c[i];
        else    cout << '-' << c[i];
    }
    return 0;
}

```

1368. 蜜蜂路线

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， $1 \leq M < N \leq 100$ ，有多少种爬行路线？



// 斐波那契数列，本题用于讲解原理，提交错误

```
#include <iostream>
using namespace std;
typedef unsigned long long ULL;
const int N = 1000;
ULL f[N + 1];
void setf()
{
    f[0] = 0, f[1] = 0, f[2] = 1, f[3] = 2;
    for (int i = 4; i <= N; i++)
        f[i] = f[i - 2] + f[i - 1];
}
int main()
{
    setf();
    int m, n;
    cin >> n >> m;
    cout << f[m - n + 1];
    return 0;
}
```

输入

输入 M, N 的值。 ( $1 \leq M < N \leq 100$ )

输出

爬行有多少种路线。

样例

输入

1 14

输出

377

输入

1 100

输出

354224848179261915075

```

#include<bits/stdc++.h>//11-1368 蜜蜂路线 javacn
using namespace std;
// 当 n-m 的值较大时需要高精度运算
string he(string s1, string s2)
{
    string r;// 存放总和
    int a[1000] = {0};
    int b[1000] = {0};
    int c[1000] = {0};
    // 逆序存入整数数组
    int i;
    for(i = 0; i < s1.size(); i++) { a[i] = s1[s1.size() - i - 1] - '0';}
    for(i = 0; i < s2.size(); i++) { b[i] = s2[s2.size() - i - 1] - '0';}

    int len = s1.size(); // 逐位相加, 逐位进位
    if(s2.size() > s1.size()) len = s2.size();
    for(i = 0; i < len; i++)
    {
        c[i] = c[i] + a[i] + b[i];
        if(c[i] >= 10)
        {
            c[i+1] = c[i+1] + c[i] / 10;
            c[i] = c[i] % 10;
        }
    }
    if(c[len] != 0) len++; // 判断是否多出 1 位
    // 逆序将 c 数组拼接成字符串
    for(i = len-1; i >= 0; i--)
    {
        // 将 c[i]+'0' 这个 ascii 码强制转换为 char 类型
        r = r + (char)(c[i] + '0');
    }
    return r;
}

```

```
int main()
{
    int m, n, i, j;
    cin >> m >> n;
    i = n - m;
    string x, y, z;
    x = "1";
    y = "1";
    if (i == 1)        {    cout << x;    }
    else if (i == 2)   {    cout << y;    }
    else
    {
        for (j = 3; j <= i + 1; j++)
        {
            z = he(x, y);
            x = y;
            y = z;
        }
        cout << z;
    }
    return 0;
}
```

### 1367. 骨牌铺方格

有  $1 \times n$  ( $n \leq 50$ ) 的一个长方形，用  $1 \times 1$ 、 $1 \times 2$  和  $1 \times 3$  的骨牌铺满方格，请问有多少种铺法？

例如当  $n=3$  时为  $1 \times 3$  的方格。此时用  $1 \times 1$ 、 $1 \times 2$  和  $1 \times 3$  的骨牌铺满方格，共有四种铺法。如下图：



输入：一个整数 ( $n \leq 50$ )

输出：骨牌的铺法。

输入

3

输出

4

`#include<bits/stdc++.h>`//12-1367-1 [骨牌铺方格](#) [推荐学习](#)

```
using namespace std;
```

```
// 1 2 4 7 13 24 44
```

```
//A(n) = A(n-1) + A(n-2) + A(n-3)
```

```
int a[100]={0};
```

```
int main()
```

```
{
```

```
    int i,n;
```

```
    cin>>n;
```

```
    a[1] = 1;
```

```
    a[2] = 2;
```

```
    a[3] = 4;
```

```
    for (i=4; i<=n; i++)
```

```
    {
```

```
        a[i]=a[i-1]+a[i-2]+a[i-3];
```

```
    }
```

```
    cout<<a[n];
```

```
    return 0;
```

```
}
```

因为砖有三种分别是  $1 \times 1$  ,  $1 \times 2$ ,  $1 \times 3$ ; 若铺设  $1 \times n$  的长方形, 那么先铺设好  $n-3$  块砖再加一块  $1 \times 3$  的砖即可, 或者铺设好  $n-2$  块砖 +  $1 \times 2$  的砖组成, 或者先铺设好  $n-1$  块砖再加一个  $1 \times 1$  的砖组成。

所以  $f(n) = f(n-1) + f(n-2) + f(n-3)$

```
#include<iostream>//12-1367-2 递归解法, 仅作参考 jiangyf70
```

```
using namespace std;
```

```
int f(int n)
{
    if(n == 1) return 1;
    if(n == 2) return 2;
    if(n == 3) return 4;
    return f(n-1) + f(n-2) + f(n-3);
}
```

```
int main()
{
    int n;
    cin >> n;
    cout << f(n);
    return 0;
}
```

```
#include<bits/stdc++.h>//12-1367-3 骨牌铺方格 仅作参考 javacn
```

```
using namespace std;
```

```
// 1 2 4 7 13 24 44
```

```
// $A(n) = A(n-1) + A(n-2) + A(n-3)$ 
```

```
int main()
```

```
{
```

```
    long long n, x, y, z, s=0, i;
```

```
    cin>>n;
```

```
    x = 1;
```

```
    y = 2;
```

```
    z = 4;
```

```
    if(n==1) { cout<<x; }
```

```
    else if(n==2) { cout<<y; }
```

```
    else if(n==3) { cout<<z; }
```

```
    else
```

```
    {
```

```
        for(i=4; i<=n; i++)
```

```
        {
```

```
            s = x + y + z;
```

```
            x = y;
```

```
            y = z;
```

```
            z = s;
```

```
        }
```

```
        cout<<s;
```

```
    }
```

```
    return 0;
```

```
}
```

### 1539. 小 X 放骨牌

小 X 喜欢下棋。

这天，小 X 对着一个长为  $N$  宽为  $M$  的矩形棋盘发呆，突然想到棋盘上不仅可以放棋子，还可以放多米诺骨牌。

每个骨牌都是一个长为 2 宽为 1 的矩形，当然可以任意旋转。小 X 想知道在骨牌两两不重叠的前提下，这个棋盘上最多能放多少个骨牌，希望你帮帮他。

输入

第一行包含用一个空格隔开的两个整数  $N, M$ 。

输出

第一行包含一个整数，表示该棋盘上最多能放的骨牌个数。

样例      输入    2 3      输出    3

说明

如图所示，三种颜色分别对应了三个骨牌。

数据范围

对于 30% 的数据， $N, M \leq 4$ 。

对于 60% 的数据， $N, M \leq 1000$ 。

对于 100% 的数据， $1 \leq N, M \leq 40000$ 。



小 X 放骨牌

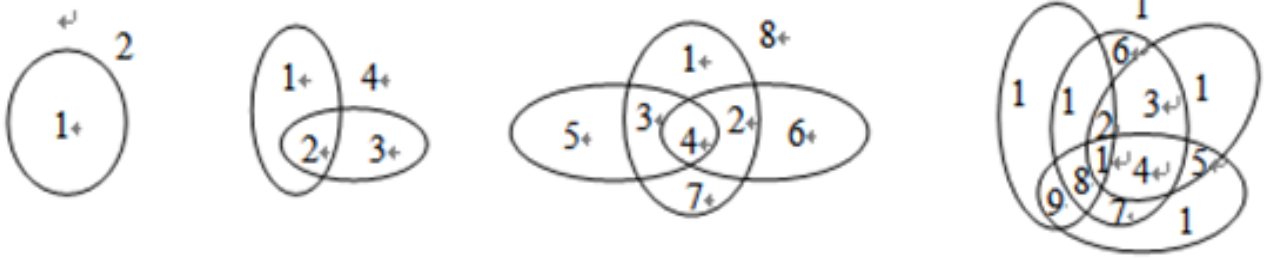
```
#include<iostream>//13-1539  jiangyf70
```

```
using namespace std;
```

```
int main()  
{  
    int n, m;  
    cin >> n >> m;  
    cout << n * m / 2;  
  
    return 0;  
}
```

1366. 平面分割问题

设有  $n$  条封闭曲线画在平面上，而任何两条封闭曲线恰好相交于两点，且任何三条封闭曲线不相交于同一点，问这些封闭曲线把平面分割成的区域个数。



```
#include<bits/stdc++.h>//14-1366-1 javacn
```

```
using namespace std;
int main() {
    int n, s, t;
    cin>>n;
    s = 2, t=2;
    for (int i=2; i<=n; i++)
    {
        s = s + t;
        t = t + 2;
    }
    cout<<s;
    return 0;
}
```

输入

一个整数  $n$   
( $n \leq 10000$ ), 代表封闭曲线的条数

输出

$n$  条曲线分割区域的个数

样例:

输入 2

输出 4

// 2 4 8 14 22 32 44

// 2 4 6 8 10 1

```
#include<bits/stdc++.h>//14-1366-2
```

```
using namespace std;
int a[10000]={0}, b[10000]={0};
int main() {
    int i, n;
    cin>>n;
    a[2] = 2;
    for (i=3; i<=n; i++) a[i]=a[i-1]+2;
    b[1] = 2;
    for (i=2; i<=n; i++) b[i]=b[i-1]+a[i];
    cout<<b[n];
    return 0;
}
```

```

#include <bits/stdc++.h> //15-1369-1 javacn
using namespace std;
//An = A(n-1)*2 + A(n-2)
// 求两个高精度的整数的和
string he(string s1, string s2) {
    string r; // 存放总和
    int a[1000] = {0};
    int b[1000] = {0};
    int c[1000] = {0};

    // 逆序存入整数数组
    int i;
    for (i = 0; i < s1.size(); i++) {
        a[i] = s1[s1.size() - i - 1] - '0';
    }

    for (i = 0; i < s2.size(); i++) {
        b[i] = s2[s2.size() - i - 1] - '0';
    }

    // 逐位相加，逐位进位
    int len = s1.size();
    if(s2.size() > s1.size()) len = s2.size();

    for (i = 0; i < len; i++) {
        c[i] = c[i] + a[i] + b[i];
        if(c[i] >= 10) {
            c[i+1] = c[i+1] + c[i] / 10;
            c[i] = c[i] % 10;
        }
    }

    // 判断是否多出 1 位

```

1369. Pell 数列

有一种数列，它的前 10 项的值分别为：

1	2	5	12
29	70	169	
408	985	2378	

这个数列被称为 Pell 数列，请问该数列的第 n 项的值是多少？（ $n \leq 1000$ ）

输入

一个整数 n

输出

第 n 项的值。

样例

输入

10

输出

2378

Pell 数列

解法一：将高精度求和、高精度 \* 单精度定义为函数（其实高精度 \* 单精度的函数可以没有，因为一个数 \* 2，可以转换为：该数 + 该数，参考解法二）。

```
// 判断是否多出 1 位
```

```
if(c[len] != 0) len++;
```

```
// 逆序将 c 数组拼接成字符串
```

```
for (i = len-1; i >= 0; i--) {
```

```
    // 将 c[i]+'0' 这个 ascii 码强制转换为 char 类型
```

```
    r = r + (char)(c[i] + '0');
```

```
}
```

```
return r;
```

```
}
```

```
// 求一个高精度的整数 * 2 的积
```

```
string cheng(string s) {
```

```
    string r;
```

```
    int a[1000] = {0};
```

```
    int i;
```

```
// 逆序存入 a 数组
```

```
for (i = 0; i < s.size(); i++) {
```

```
    a[i] = s[s.size() - i - 1] - '0';
```

```
}
```

```
// 逐位 *2
```

```
for (i = 0; i < s.size(); i++) {
```

```
    a[i] = a[i] * 2;
```

```
}
```

```
// 逐位进位
```

```
for (i = 0; i < s.size(); i++) {
```

```
    if(a[i] >= 10) {
```

```
        a[i+1] = a[i+1] + a[i] / 10;
```

```
        a[i] = a[i] % 10;
```

```
    }
```

```
}
```

```
// 判断是否多一位
```

```

// 判断是否多一位
int len = s.size();
if(a[len] != 0) len++;
// 逆序拼接到字符串 r 上
for (i = len - 1; i >= 0; i--) {
    r = r + to_string(a[i]);
}

return r;
}

int main() {
    //z: 代表计算结果, xy 代表 z 的前两项
    string x, y, z;
    int i, n;
    cin >> n;
    //A(n)=A(n-1)*2+A(n-2)
    x = "1";
    y = "2";
    if(n == 1) {
        cout << x;
    } else if(n == 2) {
        cout << y;
    } else {
        // 从第 3 项开始递推
        for (i = 3; i <= n; i++) {
            z = he(cheng(y), x);
            // 修改 xy 的值, 逐步向后推导
            x = y;
            y = z;
        }

        cout << z;
    }
}

```

解法二：定义高精度求和函数，将高精度 \*2，转换为高精度 \* 高精度。

```
#include <bits/stdc++.h> //15-1369-2 javacn
using namespace std;
/*
    
$$A_n = A_{(n-1)} * 2 + A_{(n-2)}$$

*/
// 求两个高精度的整数的和
string he(string s1, string s2) {
    string r; // 存放总和
    int a[1000] = {0};
    int b[1000] = {0};
    int c[1000] = {0};

    // 逆序存入整数数组
    int i;
    for (i = 0; i < s1.size(); i++) {
        a[i] = s1[s1.size() - i - 1] - '0';
    }

    for (i = 0; i < s2.size(); i++) {
        b[i] = s2[s2.size() - i - 1] - '0';
    }

    // 逐位相加，逐位进位
    int len = s1.size();
    if (s2.size() > s1.size()) len = s2.size();

    for (i = 0; i < len; i++) {
        c[i] = c[i] + a[i] + b[i];
        if (c[i] >= 10) {
            c[i+1] = c[i+1] + c[i] / 10;
            c[i] = c[i] % 10;
        }
    }

    // 判断是否多出1位
```

```

// 判断是否多出 1 位
if(c[len] != 0) len++;

// 逆序将 c 数组拼接成字符串
for (i = len-1; i >= 0; i--) {
    // 将 c[i]+'0' 这个 ascii 码强制转换为 char 类型
    r = r + (char)(c[i] + '0');
}

return r;
}

int main() {
    //z: 代表计算结果, xy 代表 z 的前两项
    string x, y, z;
    int i, n;
    cin >> n;
    //A(n)=A(n-1)*2+A(n-2)
    x = "1";
    y = "2";
    if(n == 1) {
        cout << x;
    } else if(n == 2) {
        cout << y;
    } else {
        // 从第 3 项开始递推
        for (i = 3; i <= n; i++) {
            z = he(he(y, y), x);
            // 修改 xy 的值, 逐步向后推导
            x = y;
            y = z;
        }
        cout << z;
    }
}

```

## Pell 数列

```
#include<bits/stdc++.h>//15-1369-3 jiangyf70
```

```
using namespace std;
```

```
string mult(string s, int b)
```

```
{  
    int a[500], c[500];  
    memset(a, 0, sizeof(a));  
    memset(c, 0, sizeof(c));  
    for(int i = s.size() - 1; i >= 0; i--) a[s.size() - 1 - i] = s[i] - '0';  
    int len = s.size();  
    int t = 0;  
    for(int i = 0; i < len; i++)  
    {  
        t += a[i] * b;  
        c[i] = t % 10;  
        t /= 10;  
    }  
    if(t) c[len++] = t;  
    string s1;  
    for(int i = len - 1; i >= 0; i--)  
    {  
        s1 += c[i] + '0';  
    }  
    return s1;  
}
```

```

string add(string s1, string s2)
{
    int a[500], b[500], c[500];
    memset(a, 0, sizeof(a));
    memset(b, 0, sizeof(b));
    memset(c, 0, sizeof(c));
    if(s1.size() < s2.size()) swap(s1, s2);
    for(int i = 0; i < s1.size(); i++) a[s1.size() - 1 - i] = s1[i] - '0';
    for(int i = 0; i < s2.size(); i++) b[s2.size() - 1 - i] = s2[i] - '0';
    int len = s1.size();
    int t = 0;
    for(int i = 0; i < len; i++)
    {
        t += a[i] + b[i];
        c[i] = t % 10;
        t /= 10;
    }
    if(t) c[len++] = t;
    string s3;
    for(int i = len - 1; i >= 0; i--) { s3 += c[i] + '0'; }
    return s3;
}

int main()
{
    int n;
    cin >> n;
    string a = "1", b = "2", c = "5";
    for(int i = 2; i <= n; i++)
    {
        a = b;
        b = c;
        c = add(mult(b, 2), a);
    } // c = add(add(b, b), a); 只用 add 加法函数也是可以的。
    cout << a << endl;
    return 0; }

```

汉诺塔的问题大家都已经很熟悉了，有三个柱子，每个柱子上有一些大小不一的金片，要把金片从 A 柱移动到 C 柱，可以借助 B 柱，请问 n 个金片的情况下，需要最少移动多少次？

输入：输入一个整数 n 代表金片的数量 ( $1 \leq n \leq 20$ )。

输出：一个整数，代表 n 个金片的移动次数。

```
#include <bits/stdc++.h> //1223 递推 汉诺塔的移动次数 javacn
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int i, n;
```

```
    int a[100];
```

```
    a[1]=1;
```

```
    cin>>n;
```

```
    for (i=2; i<=n; i++)
```

```
    {
```

```
        a[i]=a[i-1]*2+1;
```

```
    }
```

```
    cout<<a[n];
```

```
    return 0;
```

```
}
```

通过递推发现规律：

n 个金片要移动次数 = (n-1) 个金片要移动的次数 \* 2 + 1

用 A(n) 表示 n 个金片移动的步数：

$A(n) = A(n-1) * 2 + 1$

因为：

1. 需要先将 (n-1) 个顶部的金片从 A，借助于 C，移动到 B (A(n-1) 步)；
2. 把最底层的最大金片移动到 C (1 步)；
3. (n-1) 个顶部的金片从 B，借助于 A，移动到 C (A(n-1) 步)

因此得出结论： $A(n) = A(n-1) * 2 + 1$

```
#include <bits/stdc++.h> //1223 递归, javacn
```

```
using namespace std;
```

```
int fun(int n) {
```

```
    if(n == 1) return 1; //1 个金片不需要递归
```

```
    else return fun(n-1)*2+1;
```

```
}
```

```
int main() {
```

```
    int n;
```

```
    cin>>n;
```

```
    cout<<fun(n);
```

```
    return 0;
```

```
}
```