递推

统计每个月兔子的总数

```
#include<bits/stdc++.h>//1-1238-1    w2016010182
using namespace std;
long long b[50]={0};
long long tz(int  n)
{

      if(1<=n&&n<=2)
      {
            b[n]=1;
      }
      if(b[n]!=0)
      {
            return b[n];
      }
      if(b[n]==0)
      {
            return b[n]=(tz(n-1)+tz(n-2));
      }
}
int main()
{   int n;
      long long x;
      cin>>n;
//    for(int i=1;i<=50;i++)
//    {
//          cout<<tz(i)<<endl;
//    }
      x=tz(n);
      cout<<x;
      return 0;
}
```

```cpp
#include<bits/stdc++.h>//1-1238-2    jiangyf70
using namespace std;
int main(){
    int n;
    cin >> n;
    long long a = 1, b = 1, c = 2;
    for(int i = 2; i<= n; i++)
    {
        a = b;
        b = c;
        c = a + b;
    }
    cout << a;
    return 0;
}
```

```cpp
#include<bits/stdc++.h>//1-1238-3    javacn
using namespace std;
int main(){
      int n;
      cin>>n;
      //数组
      long long  a[n];
      a[0]=1,a[1]=1;
      for(int i=2;i<=n-1;i++){
            a[i] = a[i-1] + a[i-2];
      }
      cout<<a[n-1];
      return 0;
}
```

本题可以递推求解，使用变量 x 代表每一项的值。

```
#include <bits/stdc++.h>//2-1146    javacn
using namespace std;
//x 代表每一项的值
int s = 0,x = 1;
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;
}
```

求 1/1+1/2+2/3+3/5+5/8+8/13+13/21……的前 n 项的和

```
#include<stdio.h>//3-1147-1 475214719
double f(int n)
{
	double f1=1.0,f2=1.0,f3;
	int i;
	if(n==1||n==2) return 1.0;
	else
		return f(n-1)+f(n-2);
}
int main()
{
	int n,i;
	double sum=0;
	scanf("%d",&n);
	for(i=1;i<=n;i++)
		sum=sum+f(i)/f(i+1);
	printf("%.3lf",sum);
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//3-1147-2  w2016010182
using namespace std;
int fz[40];
int fm[40];
double qs(int n)
{
	double sum=0.0;
	if(n==1)
	{
		fz[1]=1;
		fm[1]=1;
	}
	else if(n==2)
	{
		fz[2]=1;
		fm[2]=2;
	}
	else{
	fz[n]=fz[n-1]+fz[n-2];
	fm[n]=fm[n-1]+fm[n-2];
	}

	sum=fz[n]*1.0/fm[n]*1.0;
	return sum;
}
int main()
{
	int n;
	double sum=0.0;
	cin>>n;
	for(int i=1;i<=n;i++)	{	sum=sum+qs(i);	}

	printf("%.3lf",sum);
}
```

```cpp
#include<bits/stdc++.h>//3-1147-3   w2016010182
using namespace std;
int b[35]={0};
int fz(int n)
{
	if(n==1 || n==2)  {     b[n]=1;     }
	if(b[n]!=0) {     return b[n];      }
	if(b[n]==0) {     return b[n]=(fz(n-1)+fz(n-2));      }
}
int c[35]={0};
int fm(int n)
{
	if(1==n)     {     c[n]=1;     }
	if(2==n)     {     c[n]=2;     }
	if(c[n]!=0) {     return c[n];      }
	if(c[n]==0)
	{
		c[n]=(fm(n-1)+fm(n-2));
		return c[n];
	}
}
int main()
{
	int n;
	double s;
	cin>>n;
	for(int i=1;i<=n;i++)
	{
		s+=(fz(i)*1.0)/(fm(i)*1.0);

	}
	printf("%.3lf",s);
	return 0;
}
```

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，需使用数组递推出斐波拉契数列前 n+1 项的值。

```
#include <bits/stdc++.h>//3-1147-4    javacn
using namespace std;

/*
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
*/
int a[50];
int n;

int main(){
	cin>>n;
	// 求出斐波拉契数列的前 n+1 项
	a[1] = 1;
	a[2] = 1;

	for(int i = 3;i <= n + 1;i++){
		a[i] = a[i - 1] + a[i - 2];
	}

	// 求结果
	double ans = 0;
	for(int i = 1;i <= n;i++){
		ans += a[i] * 1.0 / a[i+1];
	}

	printf("%.3lf",ans);
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//4-1145-1    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;
}
```

```cpp
#include<bits/stdc++.h>//4-1145-2   w2016010182
using namespace std;
int a[1005];
long long  qh(int n)//求某一项
{
	if(n==1)
	{
		a[1]=1;
		return a[1];
	}
	if(a[n]!=0)
	{
		return a[n];
	}
	else
	{
		a[n]= qh(n-1)+n-1;
		return a[n];
	}

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

```cpp
#include<bits/stdc++.h>//4-1145-3    w2016010182
using namespace std;
long long a[1005];
long long qh(int i)
{
	if(i==1)
	{
		a[1]=1;
	}
	if(a[i]!=0)
	{
		return a[i];
	}
	else
	{

		return a[i]= qh(i-1)+i-1;
	}
}

int main()
{
	int  n,i;
	long long sum=0;
	cin>>n;
	for(i=1;i<=n;i++)// 所有项数和
	{
		sum=sum+qh(i);
	}
//	for(i=1;i<=n;i++)// 所有项数
//	{
//		cout<<a[i]<<endl;
//	}
	cout<<sum;
}
```

```cpp
#include<bits/stdc++.h>//4-1145-4    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;
}
```

思路：已知第十天剩下 1 个桃子，从第十天往前推，得出上一天剩下的桃子数量，以此逆循环得出第一天的数量

```cpp
#include<bits/stdc++.h>//5-1082-1   javacn
using namespace std;

int main()  {
	int i;// 代表第几天吃桃子
	int n=1;// 代表剩下桃子的数量
	for(i=9;i>=1;i--){
		// 昨天剩下的桃子数量；
		n = (n+1)*2;
	}
	// 输出 n
	cout<<n;
	return 0;
}
```

猴子吃桃子

```
#include<iostream>//5-1082-2  jiangyf70
using namespace std;
int hz(int n, int s)
{
    if(n > 1)
    {
        hz(n - 1, (s + 1) * 2);
    }
    else return s;
}

int main()
{
    cout << hz(10, 1);
    return 0;
}
```

写法二：

```
#include<iostream>//5-1082-3  jiangyf70
using namespace std;
int hz(int n, int s)
{
    if(n == 1) return s;
    hz(n - 1, (s + 1) * 2);
}

int main()
{
    cout << hz(10, 1);
    return 0;
}
```

数数小木块

```
#include <bits/stdc++.h>//6-1148-1    1989444607
using namespace std;
int main(){
	// An = A(n-1) +n
	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;
}
```

```
#include <bits/stdc++.h>//6-1148-2  1989444607
using namespace std;
/**@brief 计算第 n 层木块数量
 * @param n 第几层
 * 找到层数与前一层数量的关系（An = A(n-1) + n）
 * @return 第 n 层的木块数量
 */
int num(int n){
	if(n==1){   return 1;   }
	else{ return num(n-1)+n;      }
}
int main(){
	int n;
	cin>>n;
	int s = 0;
	for(int i=1;i<=n;i++){  s+=num(i);  }
	cout<<s;
	return 0;
}
```

#3 思路：

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

一共 35 个

可以直接找到规律：x(x+1)(x+2)/6

数塔问题

```
#include<bits/stdc++.h>//7-1216    javacn
using namespace std;
int a[110][110];
// 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
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++){
			if(a[i+1][j]>a[i+1][j+1]){
				a[i][j] = a[i][j] + a[i+1][j];
			}
			else{
				a[i][j] = a[i][j] + a[i+1][j+1];
			}
		}
	}
	cout<<a[1][1];
	return 0;
}
```

过河卒

```
#include<iostream>//8-1224-1    jiangyf70
using namespace std;
int 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 = 0; 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;
}
```

```cpp
#include <bits/stdc++.h>//8-1224-2   w2ql
using namespace std;
int n,m,x,y;//B 点的位置（n,m）  C 马的位置（x,y）
int a[30][30];
int sum=0;
void dfs(int x,int y){
	//m 和马的控制点走不通
	if(a[x][y]==0)return ;
	// 走到地图外面则结束
	if(x>n || y>m)return ;
	// 走到（n,m），则结束
	if(x==n&&y==m){
		sum++;
		return ;
	}
	// 注意，因为不可能走重复的位置，所以没有回溯
	dfs(x+1,y);// 向下走
	dfs(x,y+1);// 向右走
}
```

```
int main(){
	cin>>n>>m>>x>>y;

	fill(a[0],a[0]+30*30,1);// 默认全部的位置都是 1
	// 标记马和马控制点的位置为 0，表示从（1,1）到达这些位置的路径数为 0
	// 标记马的位置
	a[x][y]=0;
	// 标记 8 个控制点的位置，注意可能会越界，如果越界则忽略
	if(x-1>=0&&y-2>=0)a[x-1][y-2]=0;//p6
	if(x-2>=0&&y-1>=0)a[x-2][y-1]=0;//p5
	if(x-2>=0&&y+1<=m)a[x-2][y+1]=0;//p4
	if(x-1>=0&&y+2<=m)a[x-1][y+2]=0;//p3
	if(x+1<=n&&y+2<=m)a[x+1][y+2]=0;//p2
	if(x+2<=n&&y+1<=m)a[x+2][y+1]=0;//p1
	if(x+2<=n&&y-1>=0)a[x+2][y-1]=0;//p8
	if(x+1<=n&&y-2>=0)a[x+1][y-2]=0;//p7
	/* 输出测试
	for(int i=0;i<=n;i++){
		for(int j=0;j<=m;j++)
			cout<<a[i][j]<<" ";
		cout<<endl;
	}*/
	dfs(0,0);
	cout<<sum;
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//8-1224-3    javacn
using namespace std;
int main(){
    int 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;
       }
    }
    // 把控制点设置为 0
    a[x][y] = 0;
    // 如果控制点在棋盘内，设置为 0
    if(x-1>=0&&y-2>=0) a[x-1][y-2] = 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;
    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; }// 省略本次循环下方的语句直接开始下一个循环
           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;
}
```

摘花生问题

```
#include<bits/stdc++.h>//9-1298    javacn
using namespace std;

int main(){
    int a[110][110]={0},m,n,i,j;
    cin>>m>>n;
    for(i=1;i<=m;i++){
      for(j=1;j<=n;j++){
            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）
      for(i=1;i<=m;i++){
            for(j=1;j<=n;j++){
                  a[i][j] = a[i][j] + max(a[i][j-1],a[i-1][j]);
            }
      }
      cout<<a[m][n];
    return 0;
}
```

摘花生问题（2）

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() {
	// 读入
	cin>>n>>m;
	for(int i = 1;i <= n;i++){
		for(int j = 1;j <= m;j++){
			cin>>a[i][j];
		}
	}
	// 计算走到每个点最多能摘到几个花生
	for(int i = 1;i <= n;i++){
		for(int j = 1;j <= m;j++){
			a[i][j] = a[i][j] + max(a[i][j-1],a[i-1][j]);
		}
	}
	// 仿照数塔问题，倒过来计算路线
```

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

        // 输出结果
        for(int i = n + m - 1;i >= 1;i--){
                cout<<r[i];
                if(i != 1){
                        cout<<"-";
                }
        }
        return 0;
}
```

```cpp
#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};
    int i = n, j = m, k = 0;
    c[0] = a[n][m];
    while(!(i == 1 && j == 1))
    {
        if(b[i-1][j] < b[i][j-1] && i >= 1 && j >= 1) j--;
        else if(b[i-1][j] > b[i][j-1] && i >= 1 && j >= 1)i--;
        c[++k] = a[i][j];
    }
    for(i = k; i >= 0; i--)
    {
        if(i == k) cout << c[i];
        else cout << '-' << c[i];
    }
    return 0;
}
```

蜜蜂路线

```
#include<bits/stdc++.h>//11-1368  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';
	}
	// 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;
		}
	}
	if(c[len] != 0) len++;  // 判断是否多出 1 位
	// 逆序将 c 数组拼接乘字符串
	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;
}
```

骨牌铺方格

```
#include<bits/stdc++.h>//12-1367-1  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;
}
```

因为砖有三种分别是 1x1 ， 1x2， 1x3；若铺设 1xn 的长方形，那么先铺设好 n-3 块砖再加一块 1x3 的砖即可，或者铺设好 n-2 块砖 + 1x2 的砖组成，或者先铺设好 n-1 块砖再加一个 1x1 的砖组成。

所以 f(n) = f(n-1) + f(n-2) + f(n-3)

```
#include<iostream>//12-1367-2  jiangyf70
using namespace std;

int main()
{
    int n;
    cin >> n;
    int a = 1, b = 2, c = 4, d;
    for(int i = 4; i <= n; i++)
    {
        d = a + b + c;
        a = b;
        b = c;
        c = d;
    }
    if(n == 1) cout << a;
    else if(n == 2) cout << b;
    else if(n == 3) cout << c;
    else cout << d;
    return 0;
}
```

递归解法

```
#include<iostream>//12-1367-3    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;
}
```

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;
}
```

平面分割问题

```
#include<bits/stdc++.h>//14-1366  javacn
using namespace std;
// 2 4 8 14 22 32 44
//  2 4 6 8 10 12
int main() {
    int n,s,t=2;
      cin>>n;
      s = 2;
      for(int i=2;i<=n;i++){
            s = s + t;
            t = t + 2;
      }
      cout<<s;
      return 0;
}
```

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）。

```
#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
```

```
        //判断是否多出 1 位
        if(c[len] != 0) len++;

        //逆序将 c 数组拼接乘字符串
        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;
/*
  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
```

```
    //判断是否多出1位
    if(c[len] != 0) len++;

    //逆序将c数组拼接乘字符串
    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 < s1.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;
}
```