动态规划系列专题讲义

专题一：斐波那契数列

/\*

Name: 动态规划专题之斐波那契数列

Copyright: 巧若拙

Author:

Date: 22-03-17 08:56

Description: 1755\_菲波那契数列

描述：斐波那契数列是指这样的数列: 数列的第一个和第二个数都为1，接下来每个数都等于前面2个数之和。给出一个正整数a，要求斐波那契数列中第a个数是多少。

输入：第1行是测试数据的组数n，后面跟着n行输入。每组测试数据占1行，包括一个正整数a(1 <= a <= 20)

输出：输出有n行，每行输出对应一个输入。输出应是一个正整数，为菲波那契数列中第a个数的大小

样例输入

4

5

2

19

1

样例输出

5

1

4181

1

\*/

#include<iostream>

#include<cmath>

using namespace std;

const int MAXN = 50;

int F1[MAXN];//Fibonacci数列

int F2[MAXN] = {0, 1};//Fibonacci数列

int Fibonacci(int n); //递归算法

int Fibonacci\_1(int n); //备忘录：自顶而下

int Fibonacci\_2(int n);//动态规划：自底而上

int Fibonacci\_3(int n);//动态规划：降维优化

int main()

{

int n, a;

Fibonacci\_2(MAXN); //动态规划，先记录所有子问题的解

cin >> n;

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

{

cin >> a;

cout << Fibonacci(a) << endl;

cout << Fibonacci\_1(a) << endl;

cout << F2[a] << endl;

cout << Fibonacci\_3(a) << endl;

}

return 0;

}

算法1：递归算法，没有记录任何中间结果。

int Fibonacci(int n)

{

if (n == 0 || n == 1) //递归出口

{

return //语句1

}

return Fibonacci( ) + Fibonacci( ); //语句2

}

问题1：将语句1和语句2补充完整。

参考答案：

问题1：语句1：return n;

语句2：return Fibonacci(n-1) + Fibonacci(n-2);

算法2：备忘录算法：自顶而下，需要用到全局变量F1 [MAXN]。

int Fibonacci\_1(int n)

{

if (F1[n] > 0) //如果这个问题曾经计算过，直接返回

{

return //语句1

}

if (n == 0 || n == 1) //递归出口

{

F1[n] = //语句2

}

else

{

F1[n] = //语句3

}

return F1[n];

}

问题1：将语句1，语句2和语句3补充完整。

问题2：与算法1（递归算法）相比，算法2（备忘录算法）有哪些优越之处？

参考答案：

问题1：语句1：return F1[n];

语句2：F1[n] = n;

语句3：F1[n] = Fibonacci\_1(n-1) + Fibonacci\_1(n-2);

问题2：递归算法进行了重复计算，而备忘录算法利用一维数组F1[n]记录了子问题的解，无需重复计算，大大提高了效率。

算法3：动态规划：自底而上，需要用到全局变量int F2[MAXN] = {0, 1};。

int Fibonacci\_2(int n)

{

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

{

F2[i] = //语句1

}

return F2[n];

}

问题1：将语句1补充完整。

问题2：与算法2（备忘录算法）相比，算法3（动态规划）有哪些异同？

参考答案：

问题1：语句1：F2[i] = F2[i-1] + F2[i-2];

问题2：备忘录和动态规划算法都是利用递推表达式获得子问题的解，并记录了子问题的解，用空间换时间，提高了时间效率。但是二者的思考方向不同，备忘录算法是自顶而下，从最终解出发，采用递归的方式来求解；动态规划算法是自底而上，从最小的子问题出发，逐步向上求出较大问题的解，直到获得最终解。

算法4：/动态规划：降维优化，使用3个变量代替一维数组。

int Fibonacci\_3(int n)

{

int cur, pre1, pre2;

pre1 = 0, cur = pre2 = 1; //初始化

for (int i=2; i<=n; i++) //自底向上，迭代更新变量值

{

cur = //语句1

pre1 = //语句2

pre2 = //语句3

}

return cur;

}

问题1：将语句1，语句2和语句3补充完整。

问题2：与算法3（基本动态规划算法）相比，算法4（动态规划降维优化）有哪些异同？

参考答案：

问题1：语句1：cur = pre1 + pre2;

语句2：pre1 = pre2;

语句3：pre2 = cur;

问题2：二者同属动态规划算法，都利用额外的变量（或数组）记录了各个子问题的解，都是从最小的子问题出发，逐步向上求出较大问题的解，直到获得最终解。算法3保留了所有子问题的解，虽然需要较多的空间，但是一次计算之后，就可以直接输出任意解了；算法4利用斐波那契数列递推公式的特性，只保留了每个元素的当前值和它前面的2个元素值，对计算过程中产生的子问题解用过即弃，所需空间较少，但每次求解新元素的值，都需要从头开始计算，适用于只需要求某一个元素值的情形。

课后练习：

练习1：1788\_Pell数列

描述：Pell数列a1, a2, a3, ...的定义是这样的，a1 = 1, a2 = 2, ... , an = 2 \* an-1 + an - 2 (n > 2)。

给出一个正整数k，要求Pell数列的第k项模上32767是多少。

输入：第1行是测试数据的组数n，后面跟着n行输入。每组测试数据占1行，包括一个正整数k (1 ≤ k < 1000000)。

输出：n行，每行输出对应一个输入。输出应是一个非负整数。

样例输入

2

1

8

样例输出

1

408

练习2：3089\_爬楼梯

描述：树老师爬楼梯，他可以每次走1级或者2级，输入楼梯的级数，求不同的走法数。

例如：楼梯一共有3级，他可以每次都走一级，或者第一次走一级，第二次走两级也可以第一次走两级，第二次走一级，一共3种方法。

输入：输入包含若干行，每行包含一个正整数N，代表楼梯级数，1 <= N <= 30

输出：不同的走法数，每一行输入对应一行输出

样例输入

5

8

10

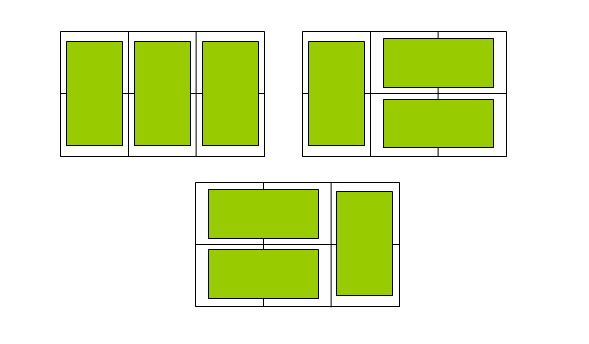
样例输出

8

34

89

练习3：2046\_骨牌铺方格

描述：在2×n的一个长方形方格中,用一个1× 2的骨牌铺满方格,输入n ,输出铺放方案的总数。例如n=3时,为2× 3方格，骨牌的铺放方案有三种,如下图：  


输入：输入数据由多行组成，每行包含一个整数n,表示该测试实例的长方形方格的规格是2×n (0<n<=50)。

输出：对于每个测试实例，请输出铺放方案的总数，每个实例的输出占一行。

样例输入

1

3

2

样例输出

1

3

2

练习4：2718\_移动路线

描述：桌子上有一个m行n列的方格矩阵，将每个方格用坐标表示，行坐标从下到上依次递增，列坐标从左至右依次递增，左下角方格的坐标为(1,1)，则右上角方格的坐标为(m,n)。

小明是个调皮的孩子，一天他捉来一只蚂蚁，不小心把蚂蚁的右脚弄伤了，于是蚂蚁只能向上或向右移动。小明把这只蚂蚁放在左下角的方格中，蚂蚁从左下角的方格中移动到右上角的方格中，每步移动一个方格。蚂蚁始终在方格矩阵内移动，请计算出不同的移动路线的数目。

对于1行1列的方格矩阵，蚂蚁原地移动，移动路线数为1；对于1行2列（或2行1列）的方格矩阵，蚂蚁只需一次向右（或向上）移动，移动路线数也为1……对于一个2行3列的方格矩阵，如下图所示：

-------------------

|(2,1)|(2,2)|(2,3)|

-------------------

|(1,1)|(1,2)|(1,3)|

-------------------

蚂蚁共有3种移动路线：

路线1：(1,1) → (1,2) → (1,3) → (2,3)

路线2：(1,1) → (1,2) → (2,2) → (2,3)

路线3：(1,1) → (2,1) → (2,2) → (2,3)

输入

输入只有一行，包括两个整数m和n（0<m+n<=20），代表方格矩阵的行数和列数，m、n之间用空格隔开

输出

输出只有一行，为不同的移动路线的数目。

样例输入

2 3

样例输出

3

提示：移动路线类似爬楼梯问题，但不是一维路径，而是二维路径，可以使用二维数组来记录到各个位置的路线数量。熟练掌握基本的动态规划算法后，可以考虑实现降维优化。