**青蛙跳台阶问题**

**解题思路：**

当n = 1， 只有1中跳法；当n = 2时，有两种跳法；当n = 3 时，有3种跳法；当n = 4时，有5种跳法；当n = 5时，有8种跳法；

类似斐波那契数列，

step1: f(n) = 1, n=1

step2: f(n) = 2, n=2

step3: f(n) = f(n-1) + f( n-2) , n >2

**解题代码：**

#include<stdio.h>

int Fib(int n)

{

if (n <= 0)

{

printf("error\n");

return -1;

}

if (1 == n)

{

return 1;

}

else if (2 == n)

{

return 2;

}

else

{

return Fib(n - 1) + Fib(n - 2);

}

}

int main() {

int n;

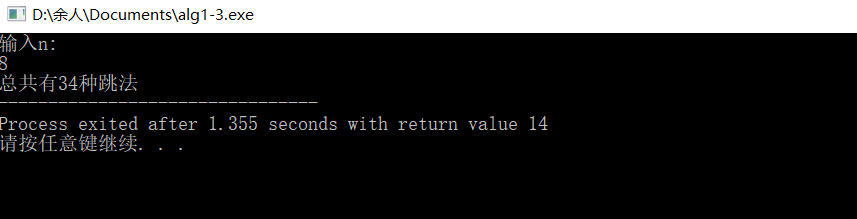
printf("输入n:\n");

scanf("%d", &n);

printf("总共有%d种跳法", Fib(n));

}

**运行结果：**



**算法分析：**

时间复杂度：O（2^n）

斐波那契数列采用递归的方法，直接或间接的调用自己，将复杂的问题逐步分解简化，转化为可求解的步骤。但与此同时，由于递归需要调用系统工作栈，会造成内存的巨大消耗，当递归深度过大（即n很大）时，可能会发生栈溢出的现象，因此该递归算法对n值有所限制。

**最大公约数问题**

**方法一：辗转相减法**

**解题思路：**

求两个数的最大公约数时，先用较大数除以较小数，如果能整除，最大公约数就等于较小数；否则用较小数除以第一步的余数，如果能整除，最大公约数就等于第一步的余数；否则，用当前获得的余数除以上一步的余数，直到能整除为止。此时作为除数的那个数就是最开始那两个数的最大公约数。

gcd(A, B) = gcd(B, A mod B)

step1:

m <- 105, n <- 252

step2:

while n!=0 do

r <- m % n

m <- n

n <- r

end

step3:gcd <- m

**解题代码：**

#include <stdio.h>

int gcd(int m, int n);

int main() {

int m,n;

printf("输入m和n \n");

scanf("%d%d",&m,&n);

gcd(m,n);

}

int gcd(int m, int n) {

int r; //余数

while(n!=0) { //当其中一个数(即余数)为0，另一个数就是两数的最大公约数

r = m%n;

m = n;

n = r;

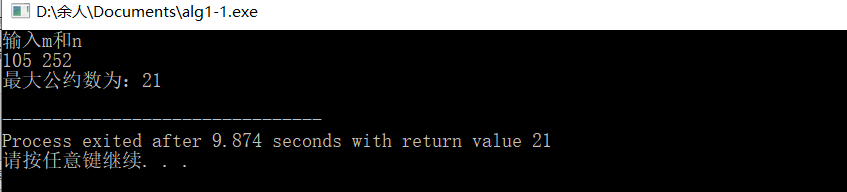
}

printf("最大公约数为：%d\n", m);

return m;

}

**运行结果：**



**算法分析:**

时间复杂度：O（logn）

采用递归的思想，直观上不易理解，计算上较为复杂繁琐，且当数字较大时容易产生较大存储量，消耗内存。

**方法二：更相减损法**

**解题思路：**

step1：任意给定两个正整数；判断它们是否都是偶数。若是，则用2约简；若不是则执行第二步。

step2：以较大的数减较小的数，接着把所得的差与较小的数比较，并以大数减小数。继续这个操作，直到所得的减数和差相等为止。

step3:用第一步中约掉的若干个2与第二步中等数的乘积就是所求的最大公约数。

**解题代码：**

#include <stdio.h>

int gcd(int m, int n);

void sort(int \*m, int \*n); //小数为减数

int main() {

int m, n;

printf("输入m和n \n");

scanf("%d%d",&m,&n);

gcd(m,n);

}

int gcd(int m, int n) {

int gcd, index = 0, c;

//比大小

sort(&m, &n);

//如果m, n 都为偶数

while(m % 2 == 0 && n % 2==0) {

m = m / 2;

n = n / 2;

index++;

}

//当减数和差相等时结束

c = m - n;

while(n != c) {

m = n;

n = c;

sort(&m, &n);

c = m - n; //大数减小数

}

if(index == 0) {

gcd = c;

}

else {

gcd = c \* 2 \* index;

}

printf("最大公约数为：%d", gcd);

return gcd;

}

void sort(int \*m, int \*n) {

int temp;

//对m ,n比较，m为较大数，n为较小数

if(\*m < \*n) {

temp = \*m;

\*m = \*n;

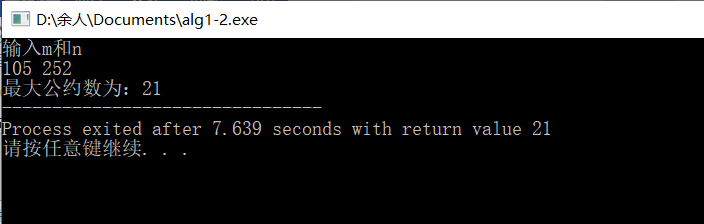
\*n = temp;

}

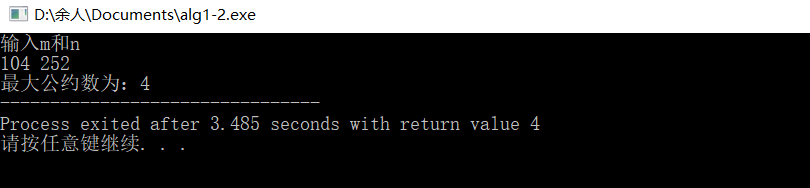
}

**运行结果：**

不全为偶数



全为偶数：



**算法分析：**

时间复杂度：O（logn）

计算上辗转相除法以除法为主，更相减损法以减法为主。计算次数上辗转相除法计算次数相对较少，特别当两个数字大小区别较大时计算次数的区别较明显。

要求多个数的最大公约数，可以先求出其中任意两个数的最大公约数，再求这个最大公约数与第三个数的最大公约数，依次求下去，直到最后一个数为止。最后所得的那个最大公约数，就是所有这些数的最大公约数。