递归：无非是自己调用自己，无形中完成循环而已。

其中递归函数主要分2部分，1终止条件，2循环体。PS：超级类似于循环。

递归可以用非递归方法堆栈模拟实现。

分治：将一个难以解决的大问题分割为一些规模较小的相同问题，以便各个击破。

Fibonacci

PS:（不含推导）

Fn=(1/ sqrt(5)) \* pow((1+sqrt(5))/2,n)-(1/sqrt(5))\*pow((1-sqrt(5))/2,n)

来源卢卡斯数列与Fibonacci数列的关系

费波纳茨数列Fn：0、1、1、2、3、5、8、13、21、34、55、89、144、233…

卢卡斯数列…Ln：1、3、4、7、11、18、29、47、76、123、199、322…

卢卡斯数列的构成为相邻两Fibonacci数之和的集合。即Ln=Fn-1+Fn+1

1876年卢卡斯在研究pow(x, 2) -x -1 = 0的两个根X1=（1+sqrt（5））/2，X2=（1-sqrt（5））/2时 {1/X=X/(1-X)}得出了两个重要的推论结果

Fn=(1/sqrt(5))\*pow((1+sqrt(5))/2,n)-(1/sqrt(5))\*pow((1-sqrt(5))/2,n)

Ln=pow((1+sqrt(5))/2,n)+pow((1-sqrt(5))/2,n)

其中，方程1/x = x/(1-x)的正根为无理数∮=（1+sqrt(5)）/2 = 1.618 即黄金分割比。

比较费波纳茨数列与卢卡斯数列，对相邻两数的比值取n趋向无穷大的极限，比值趋向黄金分割比∮

　　Fn/Fn+1-->0.618

　　Ln/Ln+1-->0.618

因此，结论是两数列的本质是一致的，都与黄金分割比有着密切的关系。

int Fib(int n) {

if(n<=1) return 1;

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

}

int Fib2(int n) {

int i=1, j=1, c;

while(n-- >0) {

c = i+j;

i = j;

j = c;

}

return i;

}

int Fib3(int n) { //SP

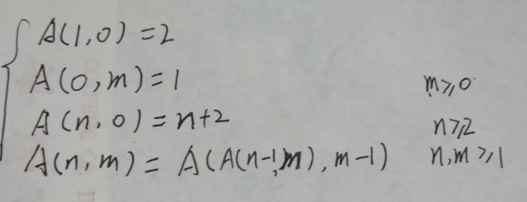
return (pow((1+sqrt(5))/2, n+1) - pow((1-sqrt(5))/2, n+1))

/ sqrt(5);

}

Ackerman

函数定义



//双递归函数Ackerman

//增长极快，很快无法计算

int Ackerman(int n, int m) {

if(n==1 && !m) return 2;

if(!n && m>=0) return 1;

if(n>=2 && !m) return n+2;

if(n>=1 && m>=1) return Ackerman(Ackerman(n-1, m), m-1);

}

Permutation

R={r1, r2, r3 ,,,}为进行排列的n个元素，求出该R的全排列

当n=1时，r为R的全排列

当n>1时，Perm(R) = Perm(R-r1)+Perm(R-r2)+Perm(R-r3)+…

By Python3

def Permutation(list, end):

result = []

re = []

#print("list", list)

if 0 == end:

re.append(list)

else:

for i in list:

list\_tmp = list[:] # 完全复制

list\_tmp.remove(i)

#print("list\_tmp", list\_tmp)

for x in (Permutation(list\_tmp, end-1)):

result = []

result.append(i)

result.extend(x)

#print("result", result)

re.append(result)

#print("re", re)

return re

pass

函数严格按照定义，每次递归会自动补全该元素开头的排列列表

By C++

template<typename T\_>

inline void Swap(T\_ &a, T\_ &b){

T\_ item = a;

a = b;

b = item;

}

template<typename T\_>

void Permutation(T\_ list[], int i\_start, int i\_end){

if(i\_start == i\_end){

for(int i=0; i<=i\_end; ++i)

cout<<list[i]<<' ';

cout<<endl;

}

else {

for(int i=i\_start; i<=i\_end; ++i){

Swap<T\_>(list[i\_start], list[i]);

Permutation<T\_>(list, i\_start+1, i\_end);

Swap<T\_>(list[i\_start], list[i]);

}

}

}

此函数每次递归即会隔离头部元素，而swap实现每次循环隔离不同的元素