12.5-递归函数（二）

代码示例1：

二分查找算法:对于一个**有序的**数组，它的搜索过程从数组的中间元素开始：

如果中间元素正好是要查找的元素，则搜索过程结束；

如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。

如果在某一步骤数组为空，则代表找不到。

这种搜索算法每一次比较都使搜索范围缩小一半。请编写程序实现在一个数组中完成二分查找。

输入：

有三行，第一行为数组元素个数N（N的范围是1到100），第二行为N个整数，作为数组元素（这些元素可能是有序的，也可能是无序的，但没有重复元素），第三行为一个整数num，表示要查找的元素。

输出：

如果能找到num，输出num在有序数组中的具体位置；若找不到，则输出-1

//非递归

#include<iostream>

using namespace std;

void swap(int &a,int &b)

{

int tmp;

tmp=a;

a=b;

b=tmp;

return;

}

void order(int a[],int size) //数组排序

{

int i;

bool jud=true; //默认数组是按照升序排列的

for(i=0;i<size;i++)

{

if(a[i]>a[i+1])

{

//没有按照升序排序

jud=false;

break;

}

}

if(jud==false)

{

int j;

for(i=0;i<size-1;i++)

{

for(j=0;j<size-1-i;j++)

{

if(a[j]>a[j+1])

{

swap(a[j],a[j+1]);

}

}

}

}

return;

}

int main()

{

int N; //数组大小

cin>>N;

int a[101];

int i,j;

for(i=0;i<N;i++)

{

cin>>a[i];

}

int num;

cin>>num; //要查找的元素

order(a,N);

/\*for(i=0;i<N;i++)

{

//输出按升序顺序排序后的数组

cout<<a[i];

if(i!=N-1)

{

cout<<" ";

}

}

cout<<endl;\*/

int position=-1; //num在有序数组中的位置（找不到为-1）

int left=0,right=N-1; //left和right分别为二分寻找的下限和上限

int temp;

while(left<=right)

{

temp=(left+right)/2;

if(a[temp]==num)

{

position=temp; //确定元素位置

break;

}

else

{

if(a[temp]>num)

{

//当前元素比目标数值更大，在前半部分寻找,上限变更

right=temp-1;

}

else

{

//在后半部分寻找，下限变更

left=temp+1;

}

}

}

cout<<position<<endl;

return 0;

}

//递归方法

#include<iostream>

using namespace std;

void swap(int &a,int &b)

{

int tmp;

tmp=a;

a=b;

b=tmp;

return;

}

void order(int a[],int size) //数组排序

{

int i;

bool jud=true; //默认数组是按照升序排列的

for(i=0;i<size;i++)

{

if(a[i]>a[i+1])

{

//没有按照升序排序

jud=false;

break;

}

}

if(jud==false)

{

int j;

for(i=0;i<size-1;i++)

{

for(j=0;j<size-1-i;j++)

{

if(a[j]>a[j+1])

{

swap(a[j],a[j+1]);

}

}

}

}

return;

}

int BinarySearch(int a[],int x,int left,int right)

{

//在数组a中寻找x，left和right分别为下限和上限

if(left>right)

return -1;

else

{

if(a[(left+right)/2]==x)

return (left+right)/2;

else if(x>a[(left+right)/2])

return BinarySearch(a,x,(left+right)/2+1,right);

else

return BinarySearch(a,x,left,(left+right)/2-1);

}

}

int main()

{

int N; //数组大小

cin>>N;

int a[101];

int i,j;

for(i=0;i<N;i++)

{

cin>>a[i];

}

int num;

cin>>num; //要查找的元素

order(a,N);

int position=BinarySearch(a,num,0,N-1);

cout<<position<<endl;

return 0;

}

代码示例2：

如果一头母牛从出生起第4个年头（即出生3年后）开始每年生一头母牛，按此规律，第1年只有一头母牛的话，到第n年时，总共会有多少头母牛？

输入：

第一行输入n (0<n<20)，代表有n组测试数据。

接下来n行每行都是一个整数m （0<m<30）

输出：

输出n行，每行为根据输入的年数，计算当年母牛的总数量。

n输入有误直接输出"WRONG!"并退出

m输入有误显示"WRONG!"并继续

//非递归方法

#include<iostream>

using namespace std;

int main()

{

int n,m;

cin>>n;

if(n>0 && n<20)

{

int i,j;

int a[5];

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

{

cin>>m;

if(m<=0 || m>=45)

{

cout<<"WRONG!"<<endl;

continue;

}

a[1]=1;

a[2]=a[3]=a[4]=0;

for(j=2;j<=m;j++)

{

/\*

a[j]代表已经j岁的奶牛，a[4]表示已经具有生育能力的母牛

\*/

a[4]+=a[3]; //三岁的奶牛具有生育能力了

a[3]=a[2]; //两岁变三岁

a[2]=a[1]; //一岁变两岁

a[1]=a[4]; //所有具有生育能力的牛都会生下一头小牛

}

cout<<a[1]+a[2]+a[3]+a[4]<<endl;

}

}

else

{

cout<<"WRONG!"<<endl;

}

return 0;

}

//递归方法

#include<iostream>

using namespace std;

long long cow(int y)

{

if(y==1 || y==2 || y==3)

return 1;

else

return cow(y-1)+cow(y-3);

}

int main()

{

int n,m;

cin>>n;

if(n>0 && n<20)

{

int i;

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

{

cin>>m;

if(m<=0 || m>=45)

{

cout<<"WRONG!"<<endl;

continue;

}

cout<<cow(m)<<endl;

}

}

else

{

cout<<"WRONG!"<<endl;

}

return 0;

}

代码示例3：

有N级的台阶，你一开始在底部，每次可以向上迈最多3级台阶（最少1级），问到达第N级台阶有多少种不同方式。

输入：

5

输出：

13

输入：

6

输出：

24

分析：

5=1+1+1+1+1 （1种）

=2+1+1+1 （4种）

=2+2+1 （3种）

=3+1+1 （3种）

=3+2 （2种）

#include<iostream>

using namespace std;

long long step(int n)

{

if(n==1)

{

return 1;

}

if(n==2)

{

//2=2=1+1

return 2;

}

if(n==3)

{

//3=3=2+1=1+2=1+1

return 4;

}

return step(n-1)+step(n-2)+step(n-3);

}

int main()

{

int n;

cin>>n;

cout<<step(n)<<endl;

return 0;

}

代码示例4：

把m个同样的苹果放在n个同样的盘子里，允许有的盘子空着不放，问共有多少种不同的分法？3，2，1和2，1，3 是同一种分法。

输入：

输入m,n，分别表示苹果的个数和盘子的个数

输出：

一个整数，表示把m个同样的苹果放在n个同样的盘子里的分法数

分析：

1.有空盘子：f(m,n)=f(m,n-1) //有空盘子

很多人会有疑问，这不是只有一个空盘子的情况吗？那2个3个空盘子呢？这就需要递归的思想，随着一步一步的将n换成n-1你就会发现那就是2,3个空盘子的情况。

2.没有空盘子：f(m,n)=f(m-n,n) //没有空盘子

我们可以看成先给每一个盘子放一个苹果，则还剩下m-n个苹果，剩下的问题就是把这m-n个苹果放到n个盘子里的问题了，也许有人会问，m-n个苹果放到n个盘子也会出现空盘子的情况啊，不是和前面的有空盘子重复了？确实，会出现空盘子的情况，但是请注意，他们并不是真的空盘子，因为他们最开始已经放了一个，他们在这里的空代表着这个盘子只有最开始放的一个苹果。

因此：f(m,n)=f(m,n-1)+f(m-n,n)       m>=n

上面的表达式并不完整，当m<n时的情况没有考虑，当m<n的时候，肯定最少有n-m个空盘子，不过幸好，这些空盘子并不影响最后的结果，因为每种方法都带有着些空盘子，剩下的问题就是把m个苹果放到m个盘子有多少种方法了。

因此：f(m,n)=f(m,m)                m<n

样例1：

6 3

7

分析：6=6+0+0=5+1+0=4+2+0=3+3+0=4+1+1=3+2+1=2+2+2

f(6,3)=f(6,2)+f(3,3)

=[f(6,1)+f(4,2)]+[f(3,2)+f(0,3)]

=1+[f(4,1)+f(2,2)]+[f(3,1)+f(1,2)]+1

=1+1+[f(2,1)+f(0,2)]+1+f(1,1)+1

=1+1+1+1+1+1+1

=7

//f(6,2)为有至少1个空盘的情况，(3,3)为每个盘子至少装了1个苹果的情况

//f(6,1)为有至少2个空盘的情况，(4,2)为，有一个空盘的情况下，另两个盘子中至少装了1个苹果的情况

//f(3,2)表示在每个盘子至少装了1个苹果的情况下，其中一个盘子只装1个苹果的情况；(0,3)表示每个盘子至少装了2个苹果的情况

样例2：

4 8

5

分析：4=1+1+1+1=2+1+1+0=2+2+0+0=3+1+0+0=4+0+0+0

#include<iostream>

using namespace std;

long long apple(int m,int n)

{

//m个苹果放入n个盘子中

if(n==1 || m==0 || m==1)

{

/\*

只有一个盘子——只有一种放法

没有苹果——全空，一种放法

只有一个苹果——一种放法

\*/

return 1;

}

if(m>=n)

{

return apple(m,n-1)+apple(m-n,n);

}

else

{

return apple(m,m);

}

}

int main()

{

int m,n;

cin>>m>>n;

cout<<apple(m,n)<<endl;

return 0;

}

总结：在做递归的题目时，不要纠结于小细节，要整体把握，只要是递归边界正确，递归方法正确就可以，不要太在意是怎么运作的。太在乎细节，反而不利于题目的求解。