递归

---Edit by Qycccc.

1.引入

前言：递归其实是一种编程思想...

在面对问题时，人的思维总是从简单深入到困难，直到解决这个问题。比如我们学习数学，一定是先学加减法，再学乘除法，直到高等数学。这样从简单的起点出发，再到复杂一般情况的过程，我们称为**迭代**，数学归纳法就是一个很好的例子。

而**递归**则不同，基本上与迭代的思想相反，也就说它是一个反人类常规思维的思考方式。它总是从大问题本身出发，尝试将大问题的规模缩小成同类子问题，直到子问题简单到我们能够解决。

2.深入理解递归

【名词解释】子问题：与大问题描述类似的问题

在计算机科学中，递归是指函数在定义时，使用函数本身的一种方法。

> Eg.递归的定义：递归的定义参见递归的定义。

递归所要解决的问题一定是可以被**缩小为同类的子问题**，这也是用递归解决问题的基本条件。

值得一提的是，“递”和“归”是解决问题的两个过程，“递”表示将问题缩小为子问题进行解决；“归”表示把子问题的答案往上合并，最终得到大问题的答案。

例1. 求1~6的和

这是一道很简单的问题，我们尝试用递归的思想来解决，便于大家理解递归解题的思路

既然我们要求和，那么我们定义函数f(n)，来统计1~n的和

```
返回值 f(int n){    //定义f为求1~n的和
 //函数体
}
```

此时n=6，我们要求的是f(6)，那么函数体里面应该写什么呢?

根据递归的思想：**把大问题缩小为规模更小的同类的子问题**

f(6)能否被缩小为f(5)呢？假如可以，那么：

f(6)=f(5)+6;

f(5)=f(4)+5;

f(4)=f(3)+4;

...以此类推，可以发现求和1~6的问题在不断的变成求和1~5，1~4...的问题,规模在缩小

$$\text{根据以上的过程，就会得到这样的一个式子：}f(n)=f(n-1)+n$$

这就是我们的**递归式**，也就是我们把大问题转换成同类子问题的数学公式。其实大家做题最难的就是找到递归式。

递归式不断缩小得到的新f(n)'我们称为一个**状态**

还可以发现，当计算出f(5)后，f(6)需要f(5)算出的值，所以f(5)需要给f(6)返回它得到的结果，也就是**状态f(5)需要给上一层状态f(6)返回值**。所以递归函数需要有**返回值**，每层递归需要给上一层返回它得到的结果。(留个坑，这里会补一张递归状态关系图，待填坑ing)

```
int f(int n){    //定义f为求1~x的和
 return f(n-1)+n;   //递归式
}
```

将递归式作为函数里面，会发现一个神奇的事情：**函数在自己调用自己!** 这与刚才所讲的递归定义,不谋而合。

> 在计算机科学中，递归是指函数在定义时，使用函数本身的一种方法。

但是深入思考后会发现，f(n)是由f(n-1)得到，f(n-1)由f(n-2)得到，这样的递归过程什么时候停下来?

我们已经知道：f(1)=1,所以当n=1的时候，递归就不应该再进行下去，我们应当把这个条件加进去作为递归的结束条件，这个条件也被称为递归的**边界条件**（通常是我们的已知条件），即在什么情况下，递归应该停下来。

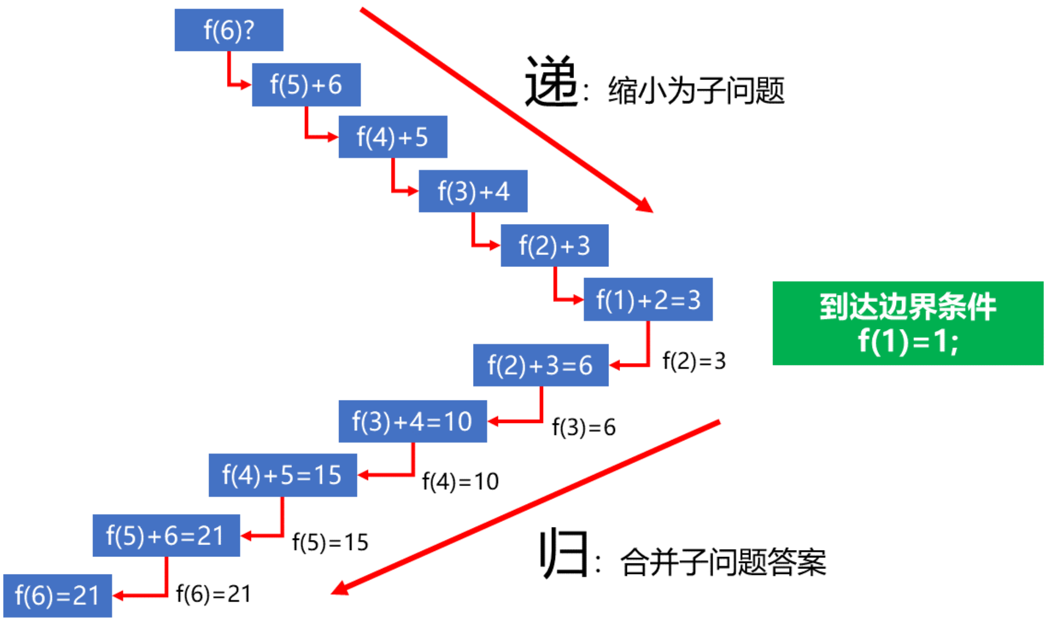
注意：在有些题目中，边界条件可能不止一个

```
int f(int n){    //定义f为求1~x的和，返回和
    if(n==1) return 1;   //边界条件
    return f(n-1)+n;    //递归式
}
```

这样，我们一个完整的递归函数就得到啦，定义式如下：

$$f(n)=\begin{cases}1 & (n==1)\\ f(n-1)+n & (n>1)\end{cases}$$

如果对递归过程不懂的同学，可以结合下面的图进行理解：



总结一下：

得到一个递归函数应该考虑以下三点：

1.定义问题，得到递归式

2.确定每层递归做什么事情，是否需要给上一层返回值，什么值？

3.递归需要有递归边界

最后，递归本身就是反常规思维的，短时间想要弄懂的确很困难。

例2. 走楼梯

有 n 级的台阶，你一开始在底部，每次可以向上迈级一级或两级台阶，问到达第 n 级台阶有多少种不同方式

输入样例#1：

```
4
```

输出样例#1：

```
5
```

(1).得到递归式

直接从题目看是得不到上面信息的，不妨模拟一下样例的具体的走的方案。大概能得到如下图的步数方案：

阶梯编号	走法1	走法2	走法3	走法4	走法5
1	1				
2	11	2			
3	111	12	21		
4	1111	121	112	211	22

从图中可以发现，当阶梯数n>2的时候，方案数里面只包含了1步和2步的走法，这就是我们缩小问题的关键。

设f(n)代表n级阶梯的方案数，n=4，考虑第一次走的情况：

情况一：第一次走了1步，那么还有3个阶梯要走，f(4)=f(3);

情况二：第一次走了2步，那么还有2个阶梯要走，f(4)=f(2);

根据**累加原理**，我们应该把这两种情况加起来，所以f(4)=f(3)+f(2);

那么我们的递归式也就呼之欲出了：$f(n) = f(n-1) + f(n-2)$;

(2).递归什么时候结束（边界条件）

从图里不难发现，当n=1和n=2的时候，问题是无法再被拆分的，并且方案数我们也是知道。

所以递归条件就是：$f(1)=1, f(2)=2$

那么最终的递归函数f(n)就得到了：

$$f(n)=\begin{cases}1 & (n==1)\\2 & (n==2)\\f(n-1)+f(n-2) & (n>2)\end{cases}$$

(3).递归是否需要返回值给上一层调用?

从递归式就可以发现，f(n)需要利用到f(n-1),f(n-2)的值，所以需要返回值。

所以最终的函数代码：

```
int f(int n){  //求解n级楼梯的方案数
    if(n==1) return 1;
    if(n==2) return 2;  //两个边界条件
    return f(n-1)+f(n-2); //递归式
}
```

本题完。

例3.汉诺塔问题

设a，b，c是三个塔座，开始时，在塔座a上有一叠共n个圆盘，这些圆盘自下而上，由大到小地叠放在一起，各圆盘从小到大编号为1,2,3，…，n。现要求将塔座a上的一叠圆盘移到塔座b上，并仍按同样顺序叠置。在移动圆盘是应遵守以下移动规则：

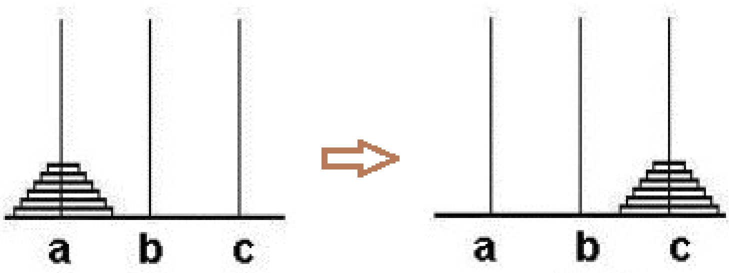
（1）每次只能移动一个圆盘；

（2）任何时刻都不允许将较大的圆盘压在较小的圆盘之上；

（3）在满足移动规则（1）和（2）的前提下，可以将圆盘移至a，b，c中任一塔座上。

要求打印出出若干行，每行表示盘子的一次移动

如：1 a->c 表示将1号圆盘从a塔座移到c塔座



样例输入

```
2
```

样例输出

```
1 A->C
2 A->B
1 C->B
```

(1).得到递归式

先考虑简单的情况。

当n=1时，那么直接就是1号圆盘，从A直接到B，记为1 A->B;

当n=2是，那么就是

Step1：先将1号圆盘，从A移到C;记为1 A->C;

Step2：将2号圆盘，从A移到B;记为2 A->B;

Step3：最后将1号圆盘，从C移到B;记为1 C->B;

当n=3的时候呢？模拟出移动步骤也是很容易，这里我们引入数学上的**整体法**来考虑这种情况。

如果我们从上往下编号(1~n)，我们把n号圆盘作为一块圆盘，剩下的(n-1)块整体作为一块圆盘，会出现什么样的情况呢?

是不是就是f(2)的情况？那么就可以按照n=2的样子把n号圆盘，和(n-1)块组成的2号圆盘，通过各种移动摆成我们想要的样子。下面仿造n=2的情况，文字描述一下n块圆盘的转移。

n块圆盘的移动步骤(n>1)

> 1.把(n-1)块组成的第二块圆盘移到C上去，期间要借助B转移
>
> 2.把n号圆盘从A移到B上去
>
> 3.把(n-1)块组成的第二块圆盘从C移到B，期间要借助A转移

也就是说，n块可以变成移动n-1块，n-1块可以变成移动n-2块...所以，这个问题是可以被缩小规模的，这就是用递归解决它的原理。

递归式是什么呢？这里的递归式就不是一个等式了，只是一个关系式。

递归式：$f(n)->f(n-1)$

(2).递归什么时候结束（边界条件）

很明显，当n=1时，直接从A到B就可以了。

即n=1，1号圆盘从A->B

(3).递归是否要给上层返回值?

这里就是要重点讲的了，也是与上两题不同的地方。

我们先讨论下，递归函数应该如何去定义呢？f(int n)?

汉诺塔递归求解的时候，是问题的缩小和转移，但同时我们还要输出圆盘是如何移动的，即是从哪个柱子到哪个柱子。

所以汉诺塔的递归函数的参数应该有4个：块数n，起始柱A，中间柱B，目标柱C，即f(n,A,B,C)

那么递归需不需要给上一层返回值呢？答案是不需要，为什么呢？因为本题只需要得到输出的情况，只要每一层递归的状态把该层的结果输出了就是问题的答案，所以上一层状态不需要得到下层状态的情况。

最终递归函数得到：

```
void f(int n,char a,char b,char c) {   //这里的A,B,C是代表的起始柱，中间柱，目标柱。与题目含义不同
    if (n==1) printf("%d %c->%c\n",n,a,c);//只有一个盘子，直接将其移到C
    else{
        f(n-1,a,c,b);  //第一步，A借助C，将n-1个盘子移到B;
        printf("%d %c->%c\n",n,a,c);  //第二步，将A上剩余的一个盘移到C,输出移动情况;
        f(n-1,b,a,c);  //第三步，将B上的n-1个盘子移到C。
    }
}
```

3.参考习题

(1)基础练习题

走楼梯

汉诺塔

数组求和

二分查找

（2）进阶练习题

数的组合
自然数拆分
波兰表达式
集合的划分
分解因数
数的组合
幂次方