## 场景

　　 递归：你打开面前这扇门，看到屋里面还有一扇门。你走过去，发现手中的钥匙还可以打开它，你推开门，发现里面还有一扇门，你继续打开它。若干次之后，你打开面前的门后，发现只有一间屋子，没有门了。然后，你开始原路返回，每走回一间屋子，你数一次，走到入口的时候，你可以回答出你到底用这你把钥匙打开了几扇门。

　 循环：你打开面前这扇门，看到屋里面还有一扇门。你走过去，发现手中的钥匙还可以打开它，你推开门，发现里面还有一扇门（若前面两扇门都一样，那么这扇门和前两扇门也一样；如果第二扇门比第一扇门小，那么这扇门也比第二扇门小，你继续打开这扇门，一直这样继续下去直到打开所有的门。但是，入口处的人始终等不到你回去告诉他答案。

## 2什么是递归

所谓递归，即递去、归来

**递去**

递归问题必须可以分解为若干个规模较小，与原问题形式相同的子问题，这些子问题可以用相同的解题思路来解决

**归来**

在将递归问题分解成若干子问题时会有一个临界条件用来原路返回

## 3. 递归的三要素

明确递归终止条件

给出递归终止时的的处理方法

提出重复逻辑，缩小问题规模

## 4.递归的变成模型

### **模型一： 在递去的过程中解决问题**

function recursion(大规模){

if (end\_condition){ // 明确的递归终止条件

end; // 简单情景

}else{ // 在将问题转换为子问题的每一步，解决该步中剩余部分的问题

solve; // 递去

recursion(小规模); // 递到最深处后，不断地归来

}

}

### **模型二： 在归来的过程中解决问题**

function recursion(大规模){

if (end\_condition){ // 明确的递归终止条件

end; // 简单情景

}else{ // 先将问题全部描述展开，再由尽头“返回”依次解决每步中剩余部分的问题

recursion(小规模); // 递去

solve; // 归来

}

## 5.经典问题

## 阶乘

## 斐波纳契数列

## 杨辉三角取值

## 汉诺塔问题

## 二叉树深度

## 递归生成树形数组

 public function tree($table,$p\_id='0') {  
       $tree = array();  
       foreach($table as $row){  
           if($row['parent\_id']==$p\_id){  
               $tmp = $this->tree($table,$row['id']);  
               if($tmp){  
                   $row['children']=$tmp;  
              }else{  
                   $row['leaf'] = true;  
              }  
               $tree[]=$row;                  
          }  
      }  
       Return $tree;          
}

## 递归力扣题

### [递归乘法](https://leetcode-cn.com/problems/recursive-mulitply-lcci/)

递归乘法。 写一个递归函数，不使用 \* 运算符， 实现两个正整数的相乘。可以使用加号、减号、位移，但要吝啬一些。

示例1:

输入：A = 1, B = 10

输出：10

示例2:

输入：A = 3, B = 4

输出：12

class Solution {

    /\*\*

     \* @param Integer $A

     \* @param Integer $B

     \* @return Integer

     \*/

    function multiply($A, $B) {

       $tmp=0;

        if($B!=1){

            $B--;

           $tmp=$this->multiply($A,$B);

        }

      return $A+$tmp;

        // if($B!=0){

        //     $B--;

        //    return $A+$this->multiply($A,$B);

        // }

        // if($B==0)

        //  return 0;

        //  $B--;

        // return $A+$this->multiply($A,$B);

    }

}