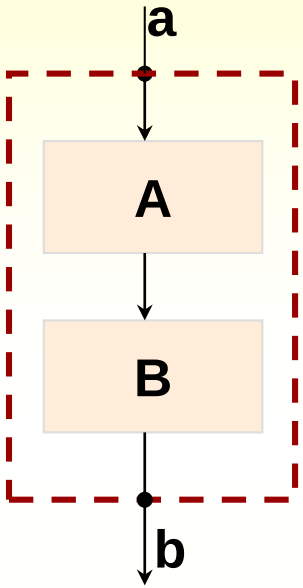
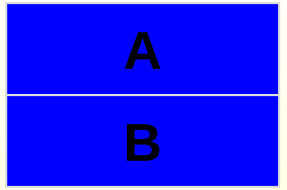
第2章　算法运用

第05讲 算法结构

算法是计算步骤的有序集合，自然构成算法是按照一定顺序排列的语句集合（顺序结构）。

在算法的语句集合中，反映算法复杂性的主要体现在构成算法的具体语句结构之中，主要包括选择结构、迭代结构和递归结构，也是分析计算复杂性的重点所在。（N-S盒图表示是由美国学者Nassi和Shneiderman于1973年提出地一种新的流程图。）



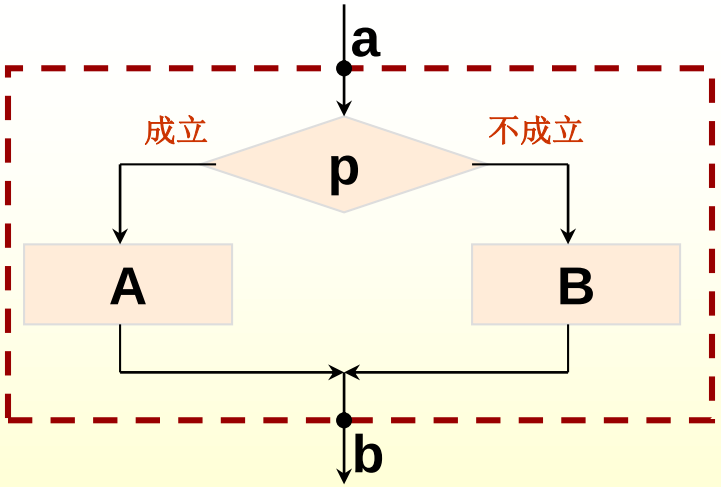


N-S盒图表示

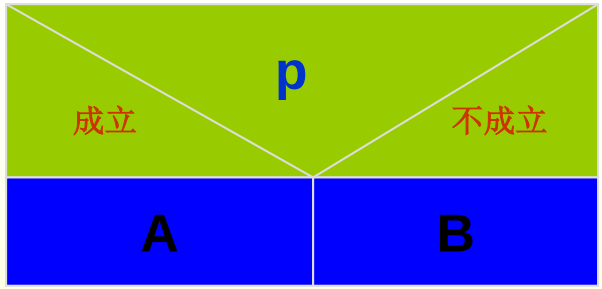
1、算法的选择结构

首先是选择结构，在算法实现中需要考虑多种可能情况的不同处理策略时，就会采用算法的选择结构，具体表示一般采用条件语句（选择结构1），有时也会采用情况语句（选择结构2）。

选择结构本身不会增加计算复杂性，但是确实算法实现中非常重要的表达方式，可以有效地解决人们思维中的选择机制问题。



选择结构1

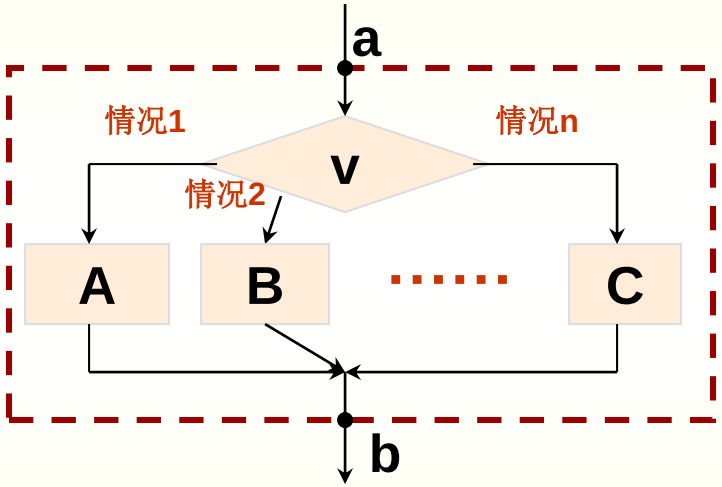


N-S盒图表示

1、算法的选择结构

情况语句（选择结构2）的伪码原语，一般定义如下：

switch（变量）（
  case “取值1”（活动1）
  case “取值2”（活动2）
  ……
  case “取值n”（活动n）
）



选择结构2

其中**switch**、**case**均为保留词。

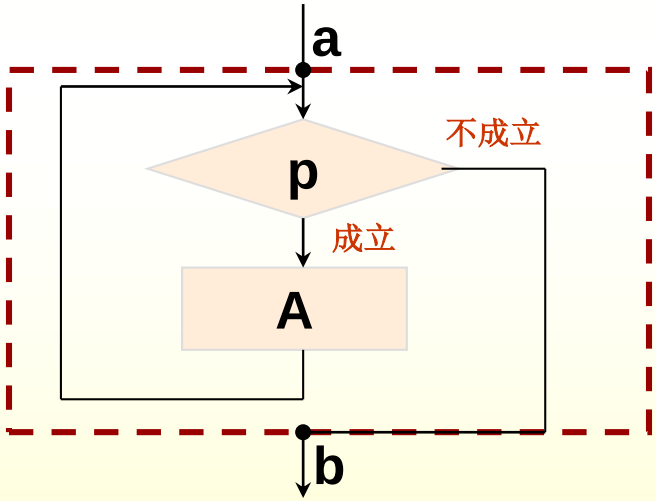
1、算法的选择结构

比如对于给定的一个数据和一张数据表，要确定该数据是否在这张表中，就需要构造一个算法来解决这个问题，这样的算法就是数据查找算法。比如查找某个人的姓名是否出现在给定的名单中，就属于这样一种数据查找问题。一般实现这一过程的伪码可以采取选择结构来构建。具体算法表示如下所示。

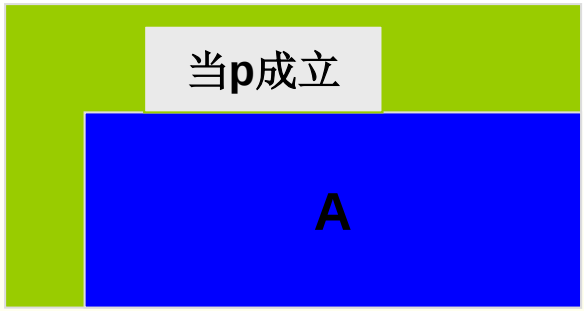
```
Procedure Search（list, TargetValue）
    if（list为空）then（返回失败）
       else（
         TestEntry←在list中选择第一个表项;
         while（TargetValue≠TestEntry 且
                TestEntry不是最后一个表项）
         do（TestEntry←在list中选择下一个表项）;
         if（TargetValue=TestEntry）then（返回成功）
            else（返回失败）
    ）end if
```

2、算法的迭代结构

其次就是迭代结构。在算法的迭代结构中，一组指令以循环方式重复执行。当然，除了while语句外（循环结构1），也可以采用repeat语句（循环结构2）来实现循环控制过程。



循环结构1



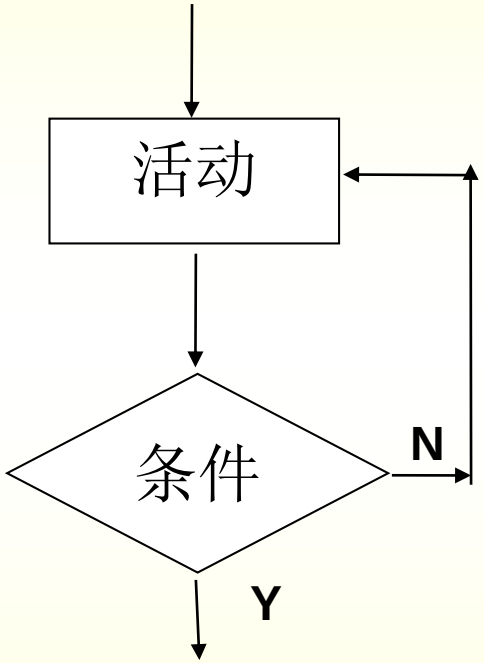
N-S盒图表示

2、算法的迭代结构

跟while语句不同采用repeat语句来实现循环控制过程，其伪码原语的使用规定如下：

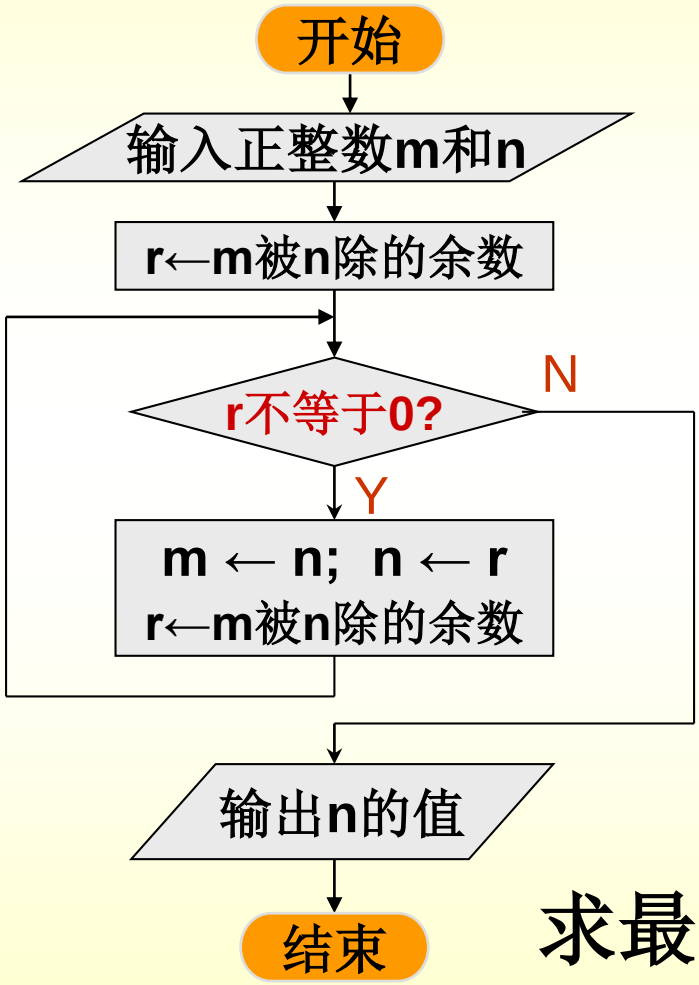
repeat（活动）until（条件）

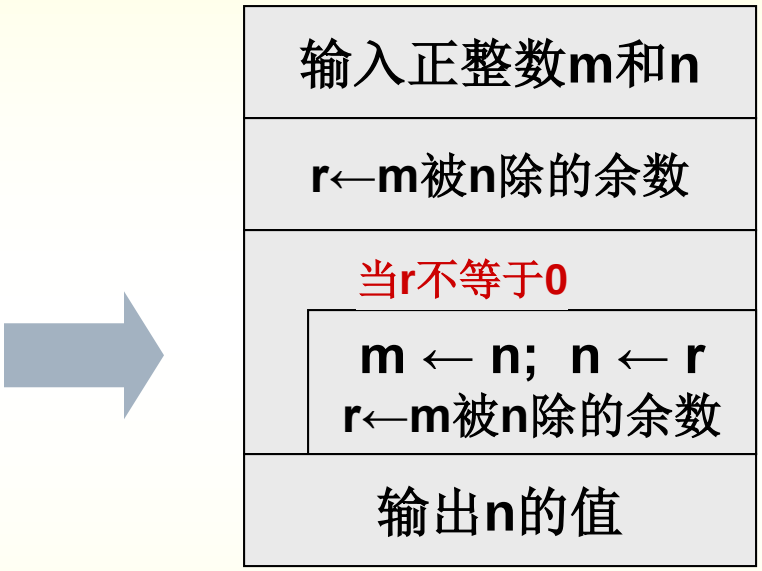
其中repeat、until都是保留词。repeat语句的含义如图所示，跟while语句不同之处是先执行循环体，然后进行条件检查。



repeat语句流程图

2、算法的迭代结构





N-S盒图表示

求最大公约数流程图

2、算法的迭代结构

为了更好地理解这种迭代结构，我们再以上述数据查找算法为例，来详细分析迭代结构算法的特点。在上述算法中，为了解决姓名查找问题，我们是从名单首列开始依次逐一将待查姓名与名单中出现的姓名进行比较，找到了就查找成功；如果名单结束也没有找到，则查找失败。现在，如果名单是按照字母顺序排列的，那么只须按照字母顺序查找即可，不必比较所有的表项。这样，实现这一过程的伪码可以表示如下所示。

```
Procedure Search（list，TargetValue）
  if（list为空） then（返回失败）
     else（
        TestEntry←在list中选择第一个表项；
        while（TargetValue>TestEntry 且
              TestEntry不是最后一个表项）
        do（TestEntry←在list中选择下一个表项）；
        if（TargetValue=TestEntry）
           then（返回成功）
           else（返回失败）
  ）end if
```

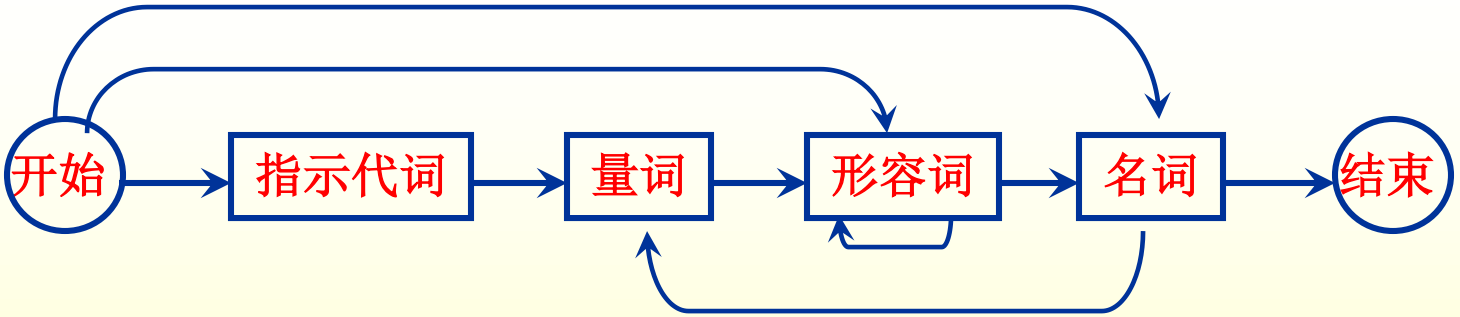
2、算法的迭代结构

上述算法是按照字母排列顺序来查找的，因此称其为顺序查找（sequential search）算法，简单分析可知，计算代价主要体现在**while**语句，如果表的长度为n的话，那么平均需要计算n/2计算步，因此算法的计算复杂性为O（n）。实际上，该算法中**while**语句就是一个迭代结构，而其中的指令“TestEntry←在list中选择下一个表项”是以重复的方式被执行的。

一般，循环控制由状态初始化、条件检查和状态修改三个环节部分组成，其中每个环节都决定着循环的成功与否。其中状态初始化设置一个初始状态，并且这一状态是可以被修改的，直到满足终止条件；条件检查是将当前状态与终止条件比较，如果符合就终止循环；状态修改就是对当前状态进行有规律地改变，使其朝着终止条件发展。

3、算法的递归结构

最后就是递归结构。比循环结构还要复杂的算法结构是递归结构，也可以实现重复计算任务。如果说循环是通过重复执行同样一组指令的方式来进行的，那么递归则是通过将一组指令当作自身的一个子程序进行调用来进行的。



“花哨名词”递归迁移网（RTN）

3、算法的递归结构

为了直观起见，我们下面通过一个名叫折半查找（binary search）算法来说明算法中的递归结构。对于同样的名字查找问题，除了顺序依次比较方法外，为了提高查找的效率，也可以采取折半查找方法来查找。通过问题的分析，具体构造的算法如下页所示。

注意，算法中引入了新的语句**Switch case**语句，即分情况语句，属于条件语句的一种变形形式，其中**Switch**、**case**均为保留词。

3、算法的递归结构

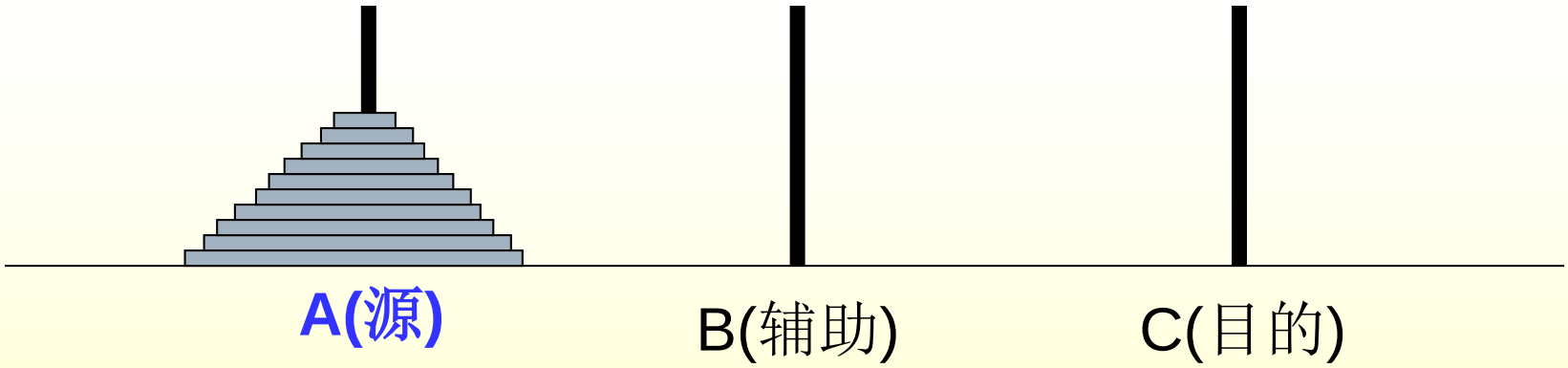
```
procedure Search(List, TargetValue)
  if(list) then （返回失败）
    else(
      TestEntry←选择List的“中间”值;
      Switch(TargetValue)(
        case(TargetValue=TestEntry)(返回成功)
        case(TargetValue>TestEntry) (
           BList←位于TestEntry前半部分List;
           返回procedure Search(BList, TargetValue)的返回值)
        case(TargetValue<TestEntry) (
           AList←位于TestEntry后半部分List;
           返回procedure Search(AList, TargetValue)的返回值)
      )
  ) end if
```

3、算法的递归结构

上述算法中，出现了过程自身调用的情况，这便是递归结构。递归过程的控制主要是通过过程调用参数的变化来进行的。在上述算法中这样的参数就是列表本身。如果追踪某个递归过程，那么就会发现，其计算过程是一层一层递进的，然后再一层一层返回。对于递归结构而言，在动态执行过程中，递进的最大层数就称为该递归结果的递归深度。对上述折半算法分析可知，递归算法的计算复杂性与递归深度密切关联，如果表的长度为n的话，那么平均需要计算的递归深度为$\log_2 n$，因此算法的计算复杂性为$O(\log_2 n)$。显然，折半查找算法的效率要高于顺序查找算法的效率。

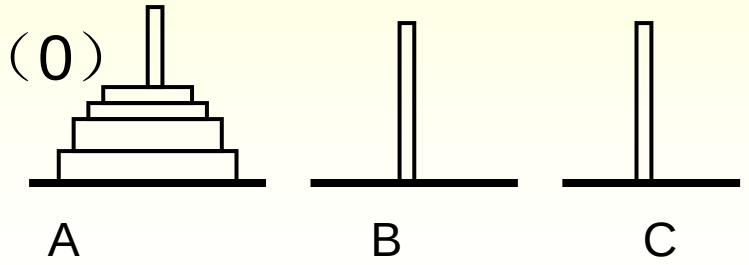
3、算法的递归结构

另一个需要运用递归结构来设计求解算法的就是著名的汉诺塔问题：将A柱子上的n个圆盘借助于B柱子移到C柱子上去，问如何移动圆盘？约束规则是（1）每次只能移动一张圆盘；（2）圆盘只能在这三个柱上存放；（3） 大盘不能压在小盘上面。

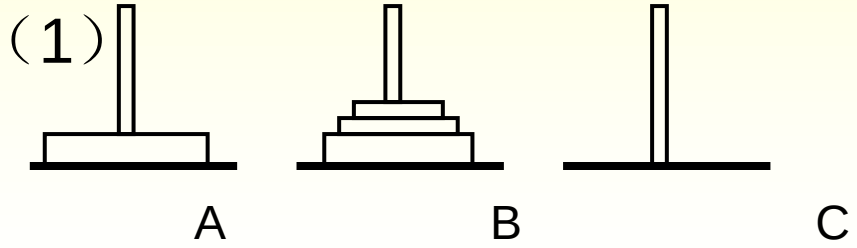


汉诺塔问题示意图

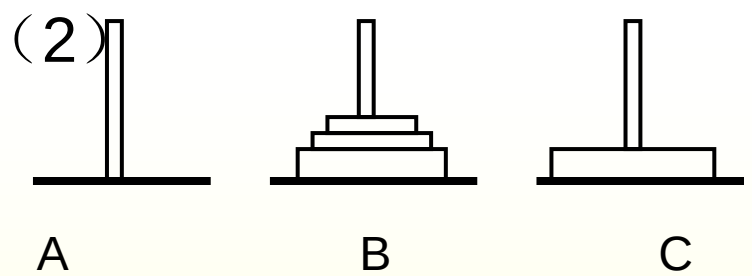
3、算法的递归结构



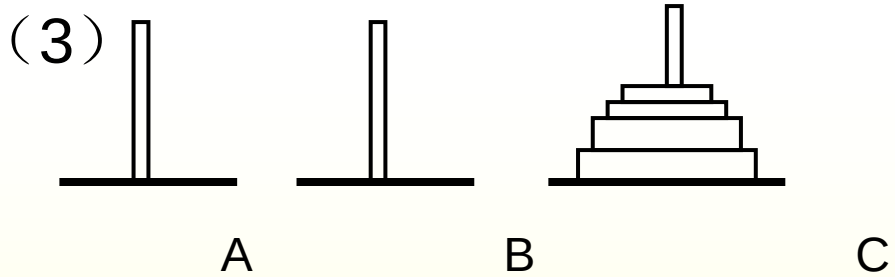
初始状态



借助C，将A上方的三个盘移到B



将A上的最后一个盘直接移到C



借助A，将B上的三个盘移到C

汉诺塔问题递归解法思路（设n=4）

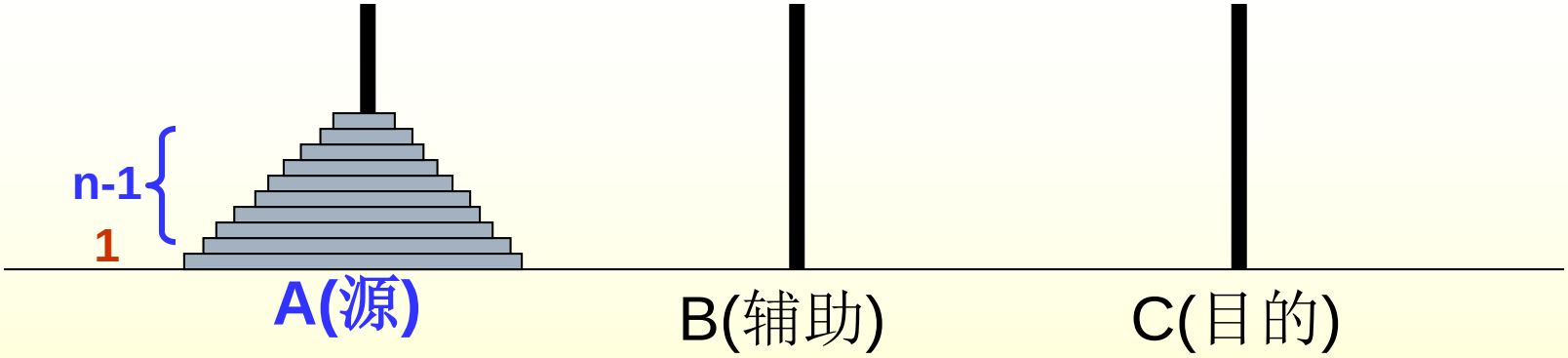
3、算法的递归结构

根据上面的分析，对于任意n，求解的方法是：将A柱子上的n个圆盘分成两部分：1个园盘和n-1个园盘，然后按照如下移动园盘步骤:

第一步：将A柱子的上面n-1个园盘移动到B柱子上面；

第二步：将A柱子上剩下的那个园盘移动到C柱子上面；

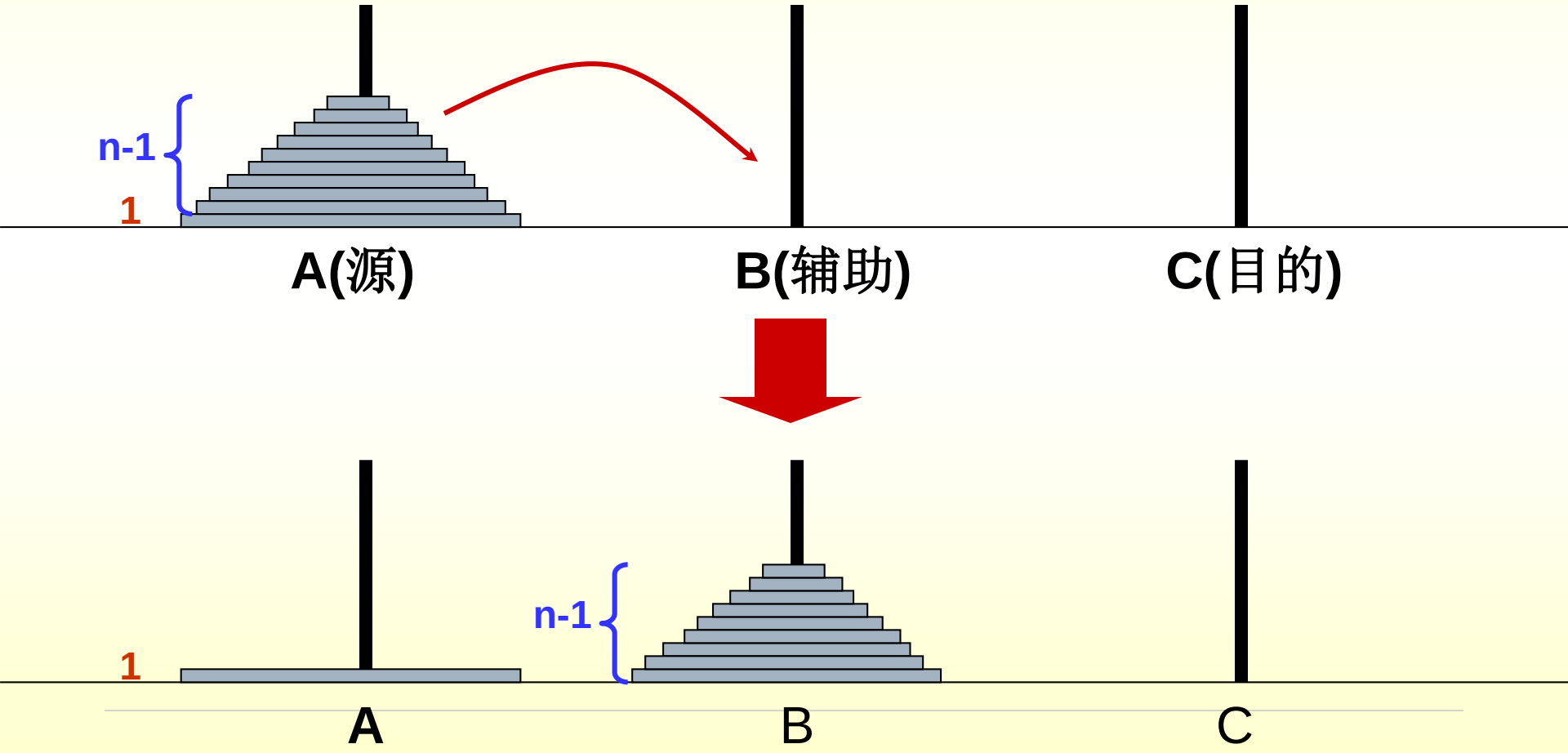
第三步：将B柱子上的n-1个园盘移动到C柱子上面。



汉诺塔问题示意图

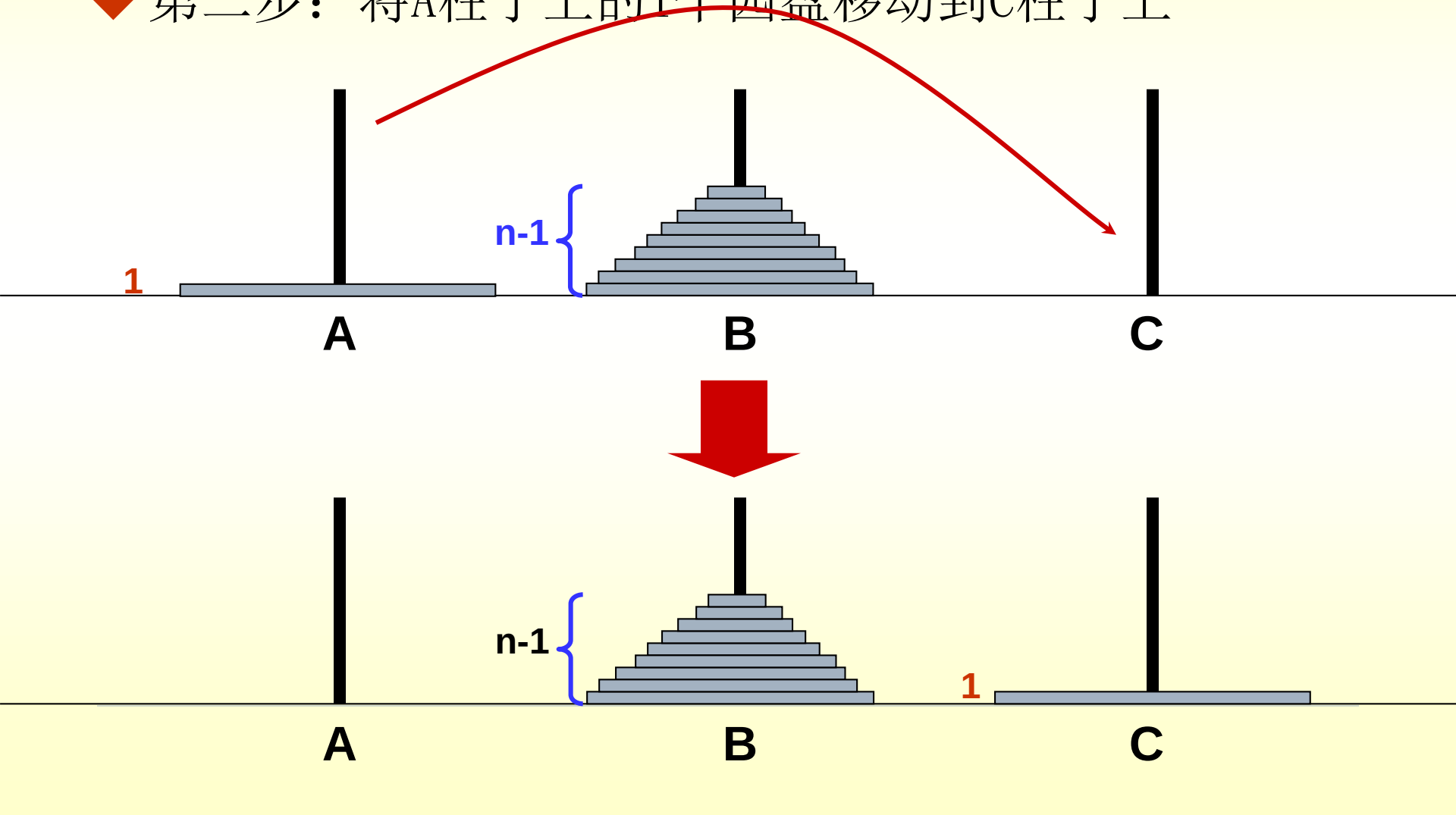
3、算法的递归结构

- 第一步：将A柱子的上面n-1个园盘移动到B柱子上面



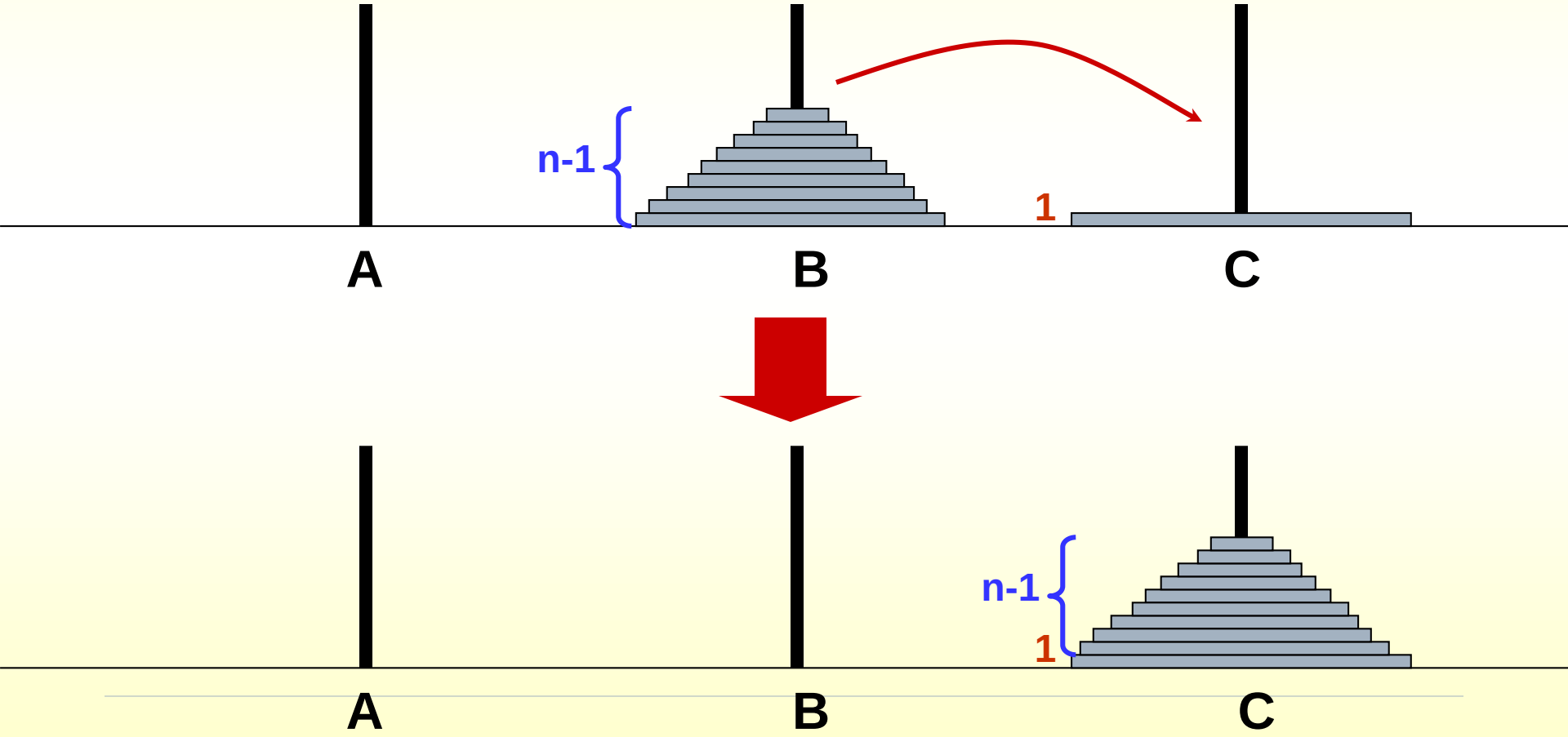
3、算法的递归结构

- 第二步：将A柱子上的1个圆盘移动到C柱子上



3、算法的递归结构

- 第三步：将B柱子上的n-1个园盘移动到C柱子上



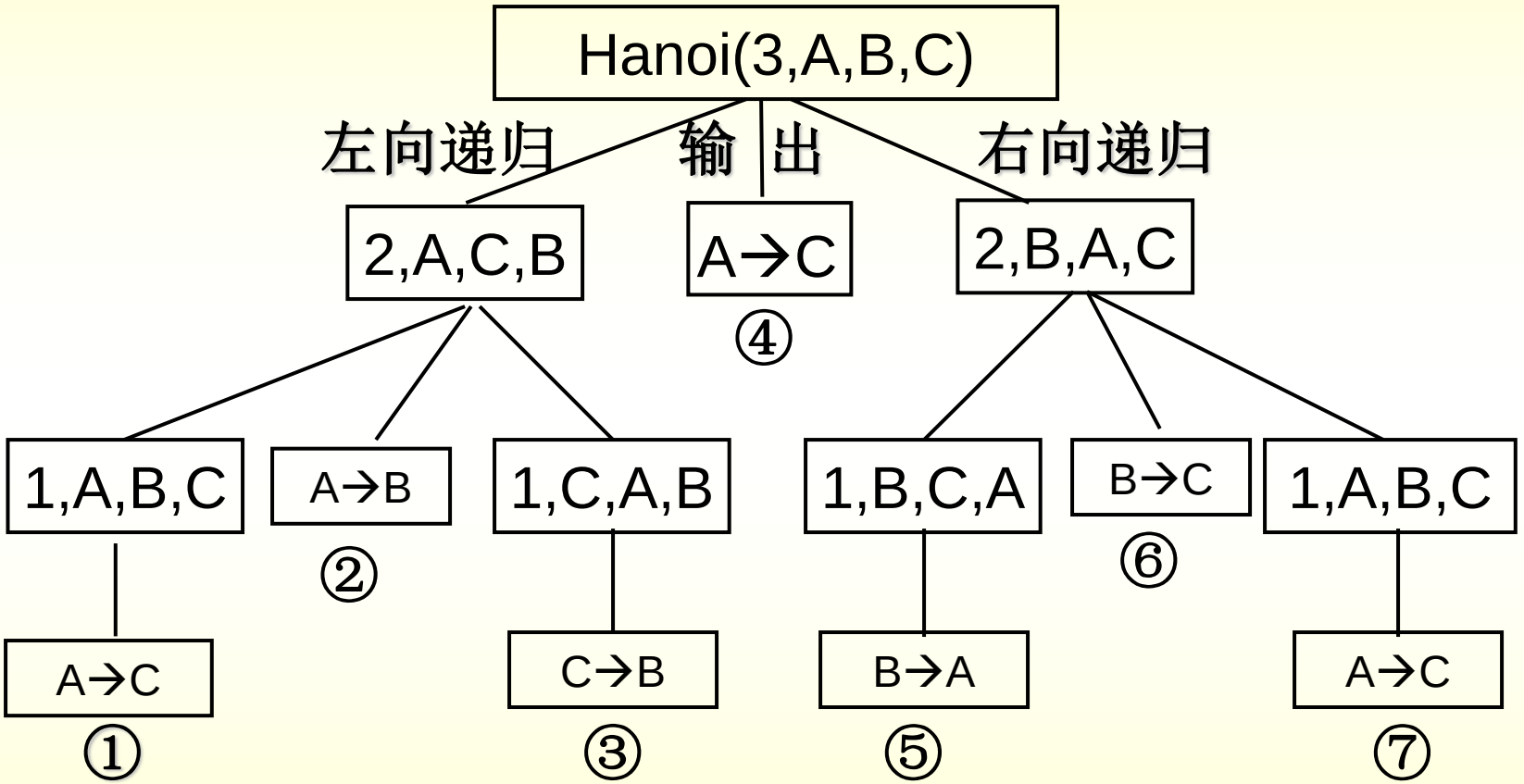
3、算法的递归结构

这样，剩下的问题就是n-1个盘子的汉诺塔问题，可以通过递归来求解，直道n=1为止。因此，整个递归算法如下：

```
procedure hanoi(int n, char A ,char B,char C)
if(n=1) then printf("%c-->%c\n", A, C); //输出
   else (
      hanoi(n-1,A,C,B); //左向递归
      printf("%c-->%c\n", A, C); //输出
      hanoi(n-1, B, A, C);  //右向递归
  )
```

其中“左向递归”是借助C，将A柱上方的n-1个盘子移到B柱；“右向递归”则是借助A，将B柱上 n-1个盘子移到C柱。

3、算法的递归结构



算法的运行图解(设 n=3)

3、算法的递归结构

对于递归算法，保证递归过程终止的条件也是需要考虑的问题。比如上述折半查找算法中表列为空或者找到目标，可以保证递归过程的终止。而在汉诺塔问题中，盘子数降到1，算法也会终止。与循环终止条件的显式表示不同，递归终止条件往往是隐含表示的，因此在设计递归算法时需要格外注意。因为，有时不恰当的递归也会产生悖论，使计算无法终止。

3、算法的递归结构

最后，我们强调指出，从理论上讲，所有算法的结构，都可以看作是并化解为递归结构，关键在于递归的深度。其中作为计算理论的之一递归函数论，就是以递归的思想，建立起完整的计算理论模型的。这就是为什么，我们在算法构造的原语介绍中，一开始就将各种算法步骤的描述方式为递归语义结构的原因所在。

在算法中，递归的思想非常重要，如果说对于计算学科而言，玩的就是算法，那么对于算法而言，玩的就是递归。从更为广泛的视野讲，递归也是大自然的一种普遍现象，从自然界的分形，到语言、音乐、绘画中的嵌套结构，无不体现着递归的本性。正是从这个意义上讲，通过递归，算法的方法可以被应用到自然与人文的各个方面，特别是可以应用到解决我们的智能机智仿造之中。