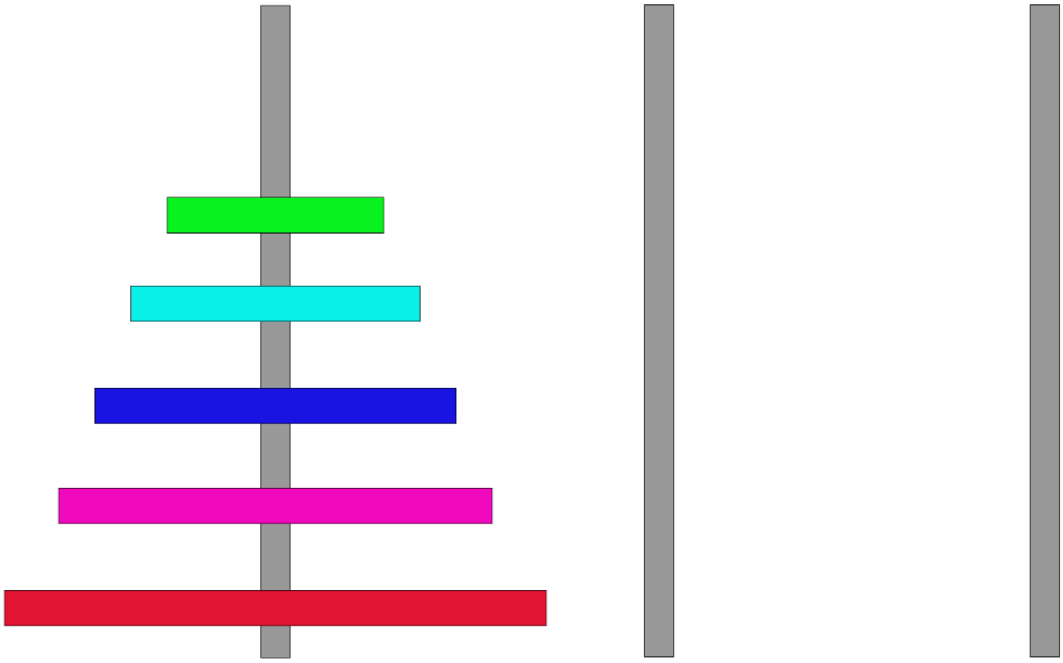
Hanoi

规则

1. 目标：把左边柱子上的所有盘子移到最右边
2. 每次只能移动一个盘子
3. 小盘必须在大盘上



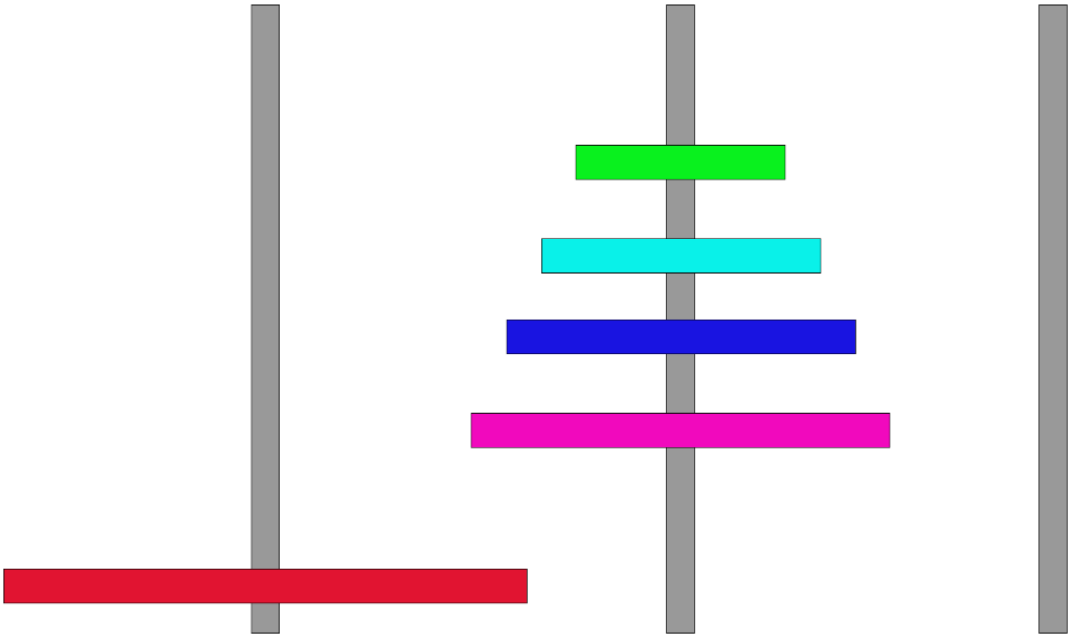
解决逻辑：

拆分：基础塔和小塔

每个汉诺塔可以拆分成最大的盘子和一个小的汉诺塔，这给了我们一种简化问题的可能。

移动：从大盘开始

我们发现达到我们的目的需要将整座汉诺塔从起始柱子移动到最终柱子，我管这个大操作叫做“move”，等下就会用到。显然，想要完成这个操作，我们需要把大盘子移动到目标柱子的最下面。自然地，我们拆分汉诺塔，把整个基础塔拆成一个大盘和一个小盘。并且如果达到下面的情况：



我们就可以按照规则来移动大盘，使的大盘位于目标柱子的最底部。

刚刚是不是移动了一个塔？

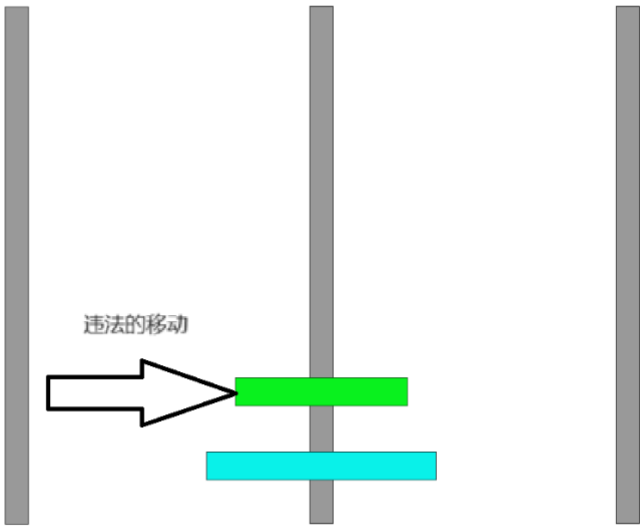
没错，想要达成上面的状况，需要做的是什么呢？移动一个小的汉诺塔。还记得我们管这个操作叫什么吗？叫“move”。问题回到了开头。怎么样跳出这一个循环呢？答案是，当汉诺塔只剩下一个盘子的时候，汉诺塔就不能被拆分了，问题也就得到了解决。显然一个盘子是可以自由移动的。

何为递归

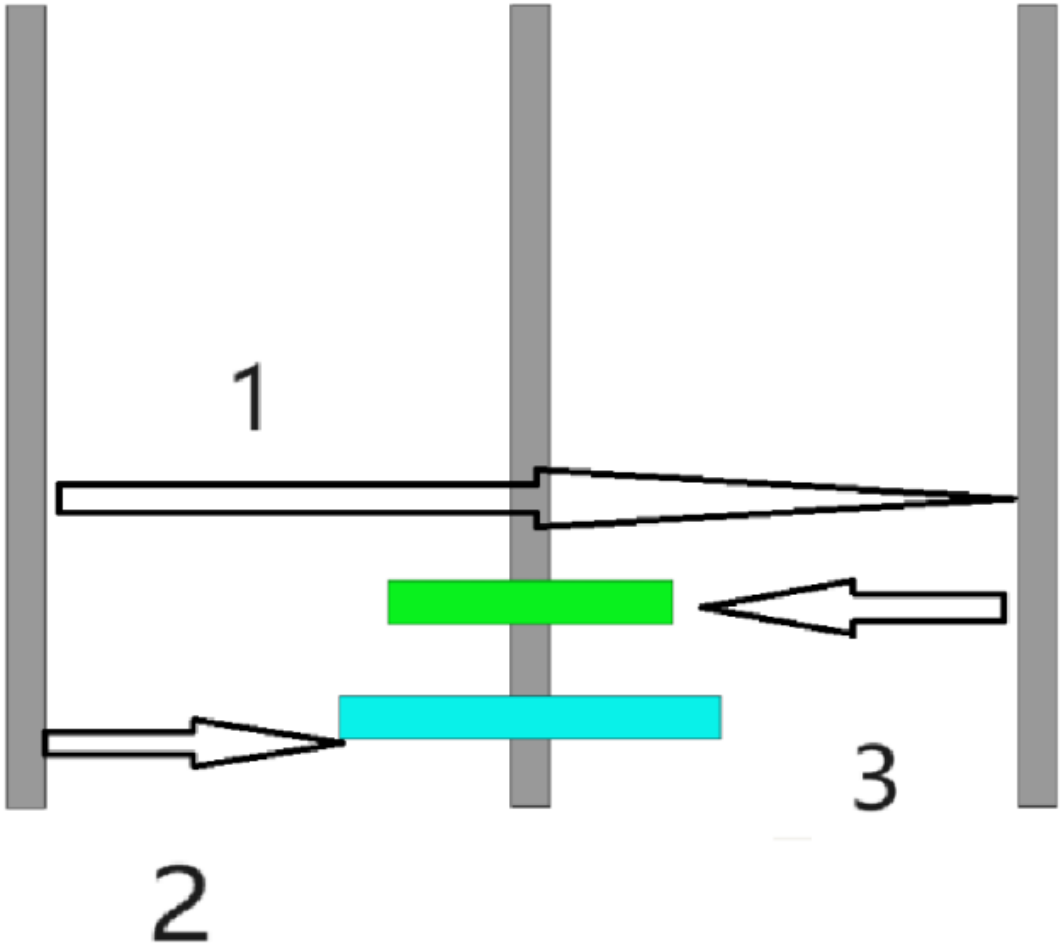
像这种，把问题层层向下传递直到最后一层，最后一层的答案再层层返回最终解决问题的流程，就是广义上的递归。正如汉诺塔的解法当中，执行一个move的时候，停下来对一个小部分再执行一次move，直到最后一层的move执行完毕，执行完成的结果再作为上一层move的一步，最终实现解决整个问题。递归的目的是，把一个层层嵌套的问题简化成一个问题。

什么是中转柱子

但是根据规则，我们不能直接移动一整个塔，解决办法是多一根柱子作为中转柱子，这样可以通过一个个盘子的移动来最终实现整摞盘子的移动。用两个盘子的汉诺塔来举例子：



加上中转柱子之后，就可以按照规则合法移动：



用程序解决汉诺塔

在理清逻辑之后，我们可以用代码来解决这个汉诺塔问题。还记得操作move吧，我们把他定义成一个函数move不就好了。

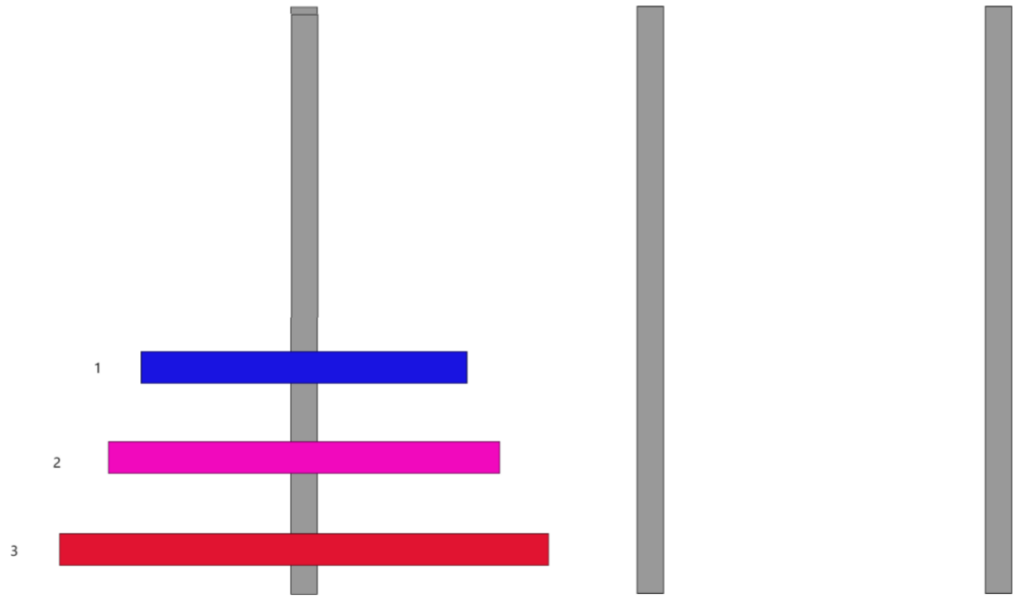
定义函数：move(num, sta, temp, goal)

其中num为第一层盘子的数量，sta,temp,goal分别是三根柱子,构建起一个最初的函数。为了方便说明，下面把三根柱子绝对命名为：一柱，二柱，三柱。用起始柱，中转柱，目标柱表示移动过程中柱子的相对关系。

拆分塔：

按照我们的步骤：先拆分，再移动小塔，于是可以写出的代码来：

```
1  def move(num,sta,temp,goal):  #n为汉诺塔数，sta为初始柱子，temp为辅助柱子，goal为目标柱子
2      if (num>0):
3          move(num-1,sta,goal,temp)  # 移动n-1个盘子到辅助柱子上
```



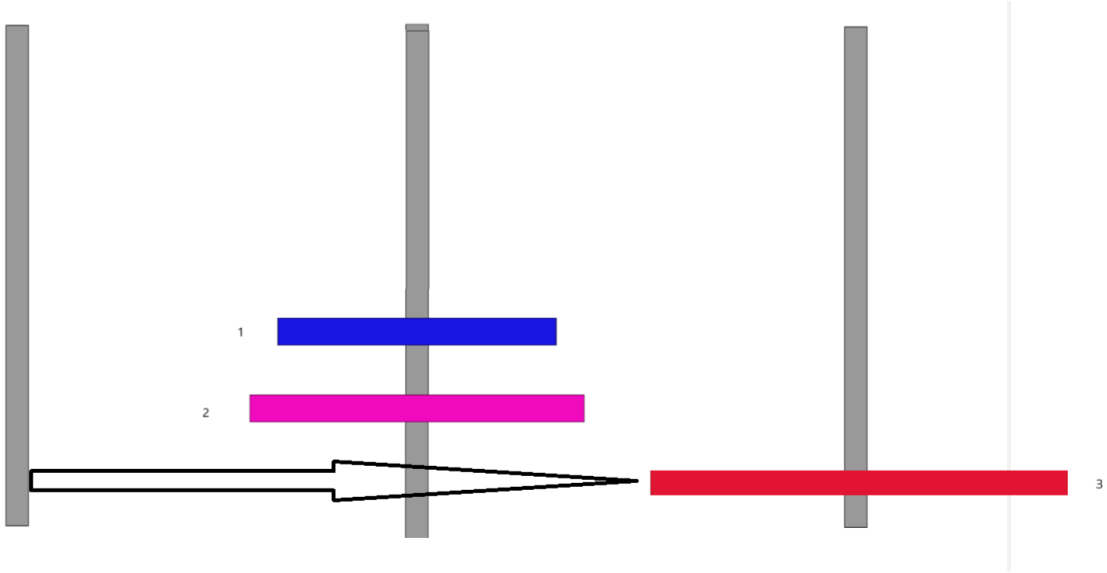
n>0的意思很简单：只有第一层上有盘子才能移动嘛。我们需要移动一柱上除了最大盘子的汉诺小塔，即盘子数为n-1的小塔移动到二柱上，这样才能将最大的盘子移动到三柱上。

移动大盘

接下来移动大盘子到三柱上：

```python
def move(num,sta,temp,goal):  #n为汉诺塔数，sta为初始柱子，temp为辅助柱子，goal为目标柱子
    if (num>0):
        move(num-1,sta,goal,temp)  # 移动n-1个盘子到辅助柱子上
        print("Move disk", num, "from", sta, "to", goal)  # 移动最后一个盘子到目标柱子上
```

用图形表示一下就是：



移动小塔

最后把小塔从二柱移动到三柱，此时一柱空置，可以作为中转柱子使用：

```
def move(num,sta,temp,goal):  #n为汉诺塔数，sta为初始柱子，temp为辅助柱子，goal为目标柱子
    if (num>0):
        move(num-1,sta,goal,temp)  # 移动n-1个盘子到辅助柱子上
        print("Move disk", num, "from", sta, "to", goal)  # 移动最后一个盘子到目标柱子上
        move(num-1,temp,sta,goal)  # 移动n-1个盘子到目标柱子上
```

这样整个函数就结束了。接下来的事情就是输入汉诺塔的层数，并且调用这个函数执行就好了。你会发现在这个函数当中，函数调用了他自己，这是编程当中实现递归的常用手段之一。

```
def move(num,sta,temp,goal):  #n为汉诺塔数，sta为初始柱子，temp为辅助柱子，goal为目标柱子
    if (num>0):
        move(num-1,sta,goal,temp)  # 移动n-1个盘子到辅助柱子上
        print("Move disk", num, "from", sta, "to", goal)  # 移动最后一个盘子到目标柱子上
        move(num-1,temp,sta,goal)  # 移动n-1个盘子到目标柱子上

while(1):    # 测试代码
    a=int(input("您已经进行汉诺塔教程，请选择汉诺塔的数"))
    move(a, 'A', 'B', 'C')
```

附录：用列表解决问题

以上的代码是老师的代码，这个代码很小巧，但是无法在移动盘子的各个阶段显示三根柱子的具体情况。我想到的一种解决办法是，把三根柱子定义成三个列表，并且用列表的追加弹出等操作模拟盘子的移动。下面是我的代码：

```python
def hanoi(num,sta,temp,goal):      #l1为起始，l为中转，l2为目标
    if num >0:
        hanoi(num-1,sta,goal,temp)         #小塔从l1，以l2为中转，移到l

        y = sta.pop(-1)       #将大盘从l1移到l2
        goal.append(y)
        print(f'l1 = {l1} \t\t l = {l} \t\t l2 = {l2}')

        hanoi(num-1,temp,sta,goal)       #小塔从l，以l1为中转，移到l2

while 1:
    Jud = 'Y'
    if Jud == 'Y':
        l1 = []
        l = []
        l2 = []
        num = int(input("输入hanoi的层数:>"))
        for i in range(1,num+1):
            l1.append(i)
        l1.sort(reverse=True)

        print(f'l1 = {l1} \t\t l = {l} \t\t l2 = {l2}')
        hanoi(num,l1,l,l2)
        Jud = input("again? Y or N:")
    else:
        exit()
```

使用列表的好处是，在这之后需要调用某个状态的时候，可以通过访问列表来实现，例如需要将移动过程做成动画。