实例 3：汉诺塔

汉诺塔是一个可以使用递归解决的经典问题，它源于印度一个古老传说：大梵天创造世界的时候做了三根金刚石柱子，其中一根柱子从下往上按照从大到小的顺序摞着 64 片黄金圆盘，大梵天命令婆罗门把圆盘从下面开始按照从大到小的顺序重新摆放在另一根柱子上，并规定：小圆盘上不能放大圆盘，三根柱子之间一次只能移动一个圆盘。问一共需要移动多少次，才能按照要求移完这些圆盘。三根金刚柱子与圆盘摆放方式如图 1 所示。

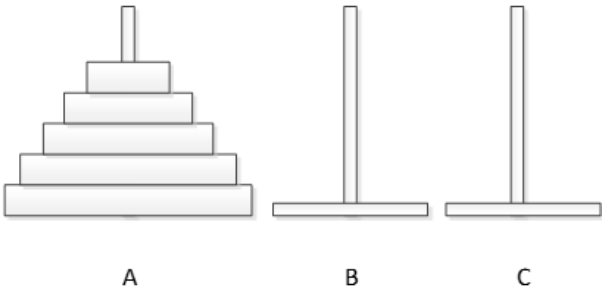

图 1 汉诺塔格局图

本实例要求编写程序，实现输出汉诺塔移动过程的功能。

实例目标

- 明确递归函数的递归公式与边界条件
- 熟练地定义递归函数

实例分析

我们先来分析一下移动 n 个圆盘的过程：

（1）假设现在 A 柱子上只有 1 个圆盘，此时无需 B 柱子中转而直接将圆盘从 A 柱子移动到 C 柱子上。

（2）假设 A 柱子上有 2 个圆盘，此时可以先将小盘子移动到 B 柱子上，再将大盘子移动到 C 柱子上，最后将小盘子移动到 C 柱子上。我们既可以借助柱子 B 将 2 个盘子从 A 移动到 C，也可以借助 C 将 2 个盘子从 A 移动到 B。

（3）假设 A 柱子上有 3 个圆盘，则可以根据移动 2 个盘子的过程，先借助柱子 C 将柱子 A 上的两个盘子先移动到柱子 B，将 A 上的大盘子移动到柱子 C 上，此时 A 成为空柱子，再借助柱子 A，将柱子 B 上两个盘子移动到 C。

（4）以此类推，假设 A 柱子上有 n 个盘子，则可以将 n-1 个盘子看作一个整体，也就是递归中的子问题。例如，借助柱子 C 先将柱子 A 上的 n-1 个盘子移动到柱子 B 上，再将柱子 A 上最大的盘子移动到柱子 C 上，此时 A 成为空柱子，最后借助柱子 A，将柱子 B 上 n-2 个盘子移动到柱子 A 上，将柱子 B 上最大的盘子移动到柱子 C 上，此时 B 成为空柱子.....

如此往复。

经过上述分析的过程，递归函数边界条件和递归公式分别如下：

（1）边界条件：当 n 为 1 时，直接将盘子从 A 移动到 C；

（2）递归公式：当 n 为 n 时，直接将 n-1 个盘子先从 A 移动到空柱子 B 上，再将第 n 个盘子移动到柱子 C 上，最后将柱子 B 上的 n-1 个盘子移动到柱子 C 上。

代码实现

```
def hanoi(n, ch1, ch2, ch3):
    if n == 1:
        print(ch1, '->', ch3)
    else:
        hanoi(n - 1, ch1, ch3, ch2)
        print(ch1, '->', ch3)
        hanoi(n - 1, ch2, ch1, ch3)
plate_nums = int(input("请输入盘子的数量："))
hanoi(plate_nums, 'A', 'B', 'C')
```

以上代码首先定义了一个递归函数 hanoi()，该函数包含 n、 ch1、ch2、ch3 共 4 个参数，其中参数 n 表示用户输入的盘子数量，ch1、ch2、ch3 分别表示柱子 A、B、C 的名称，该函数中分别处理了 1 个盘子和 n-1 个盘子的情况，然后接收用户输入的盘子数量 plate_nums，最后调用 hanoi()函数输出 plate_nums 个盘子的移动过程。

代码测试

运行程序，在控制台输入“1”之后的结果如下所示：

```
请输入盘子的数量：1
A -> C
```

再次运行程序，在控制台输入“2”之后的结果如下所示：

```
请输入盘子的数量：2
A -> B
A -> C
B -> C
```

再次运行程序，在控制台输入“3”之后的结果如下所示：

```
请输入盘子的数量：3
A -> C
A -> B
C -> B
A -> C
B -> A
```

```
B -> C
A -> C
```