拉勾教育
—互联网人实战大学—

递归：求解汉诺塔问题

简单/递归、列表

学习目标

拉勾教育
—互联网人实战大学—

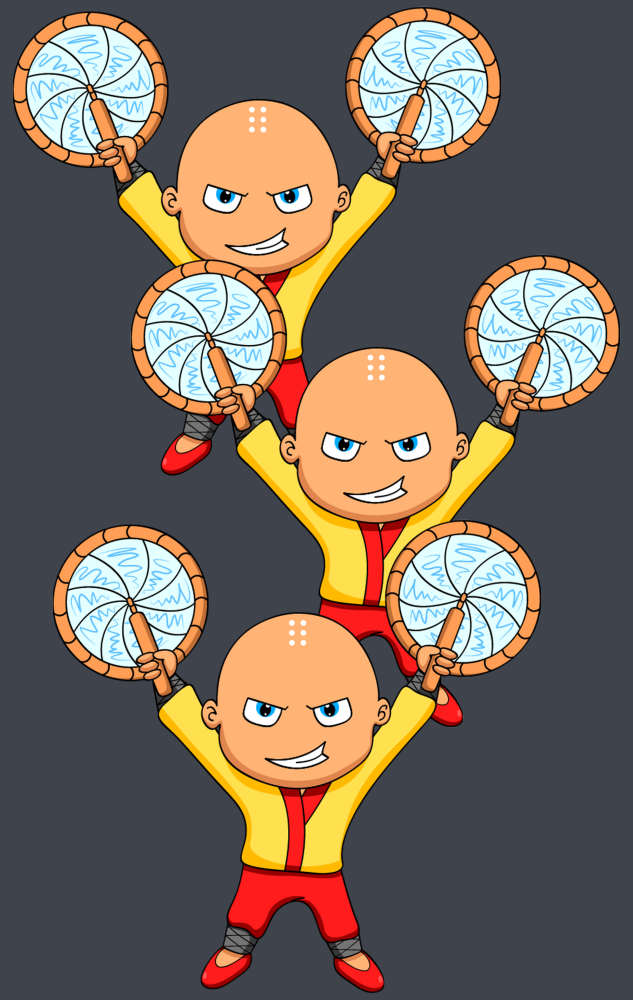
- 使用循环解决汉诺塔问题
- 递归算法的思想
- 递归算法三要素及编码模板
- 运用递归算法解决汉诺塔问题

相传在很久以前

有个寺庙里的几个和尚整天不停地移动着 64 个盘子

日复一日，年复一年

据说，当 64 个盘子全部移完的那一天就是世界末日...



题目描述

在经典汉诺塔问题中，有 3 根柱子及 N 个不同大小的穿孔圆盘，盘子可以滑入任意一根柱子。一开始，所有盘子自上而下按升序依次套在第一根柱子上(即每一个盘子只能放在更大的盘子上面)。移动圆盘时受到以下限制:

(1) 每次只能移动一个盘子;

(2) 盘子只能从柱子顶端滑出移到下一根柱子;

(3) 盘子只能叠在比它大的盘子上。

输入：A = [2, 1, 0], B = [], C = []
输出：C = [2, 1, 0]

请编写程序，用栈将所有盘子从第一根柱子移到最后一根柱子。

你需要原地修改栈。

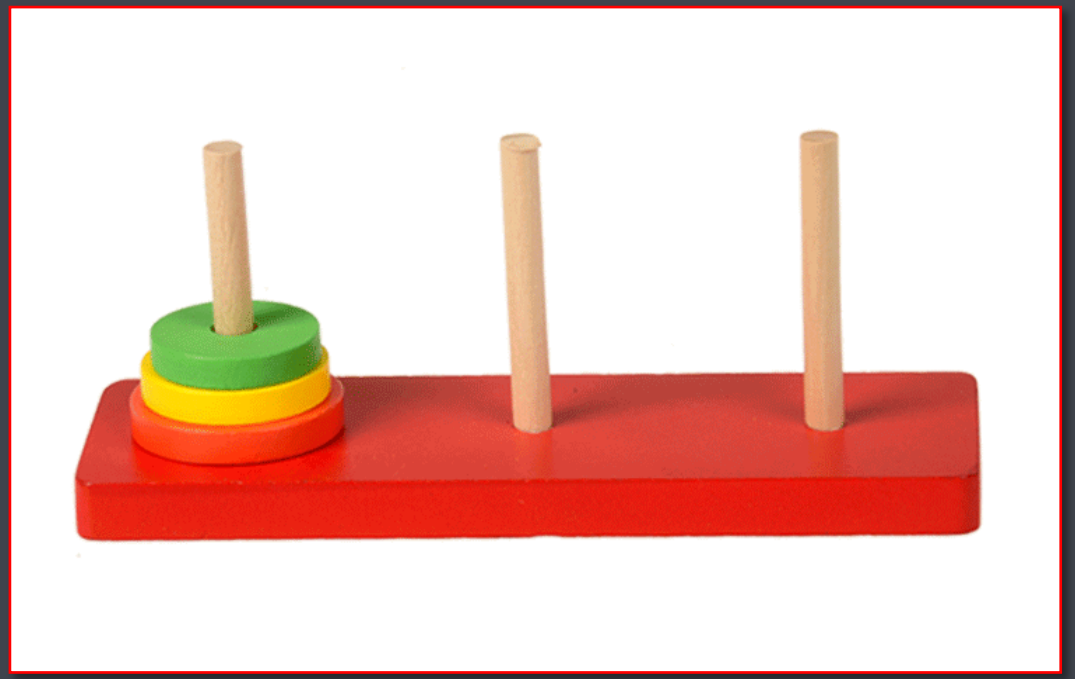
A中盘子的数目不大于14个。

输入：A = [1, 0], B = [], C = []
输出：C = [1, 0]

一. Comprehend 理解题意

经典益智类问题

- 每次只能移动一个盘子
- 盘子只能从柱子顶端滑出移到下一根柱子
- 盘子只能叠在比它大的盘子上



一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

题目主干

- 通过程序解决汉诺塔问题
- 圆盘数量不固定

附加限制

- 原地修改

细节问题

- 严格遵守汉诺塔规则

宽松限制

- A中盘子的数目不大于14个

二. Choose 数据结构及算法思维选择

数据结构选择

- 输入和输出的参数类型已经限定为List
- 其中 “盘子只能从柱子顶端滑出移到下一根柱子”
  - 与栈的后进先出的特点一致
  - 但输入输出参数均为List
  - 每次都在List尾部添加或者移除（模拟栈的方式）

```
1  class Solution {
2      public void hanota(List<Integer> A, List<Integer> B, List<Integer> C) {
3
4      }
5  }
```

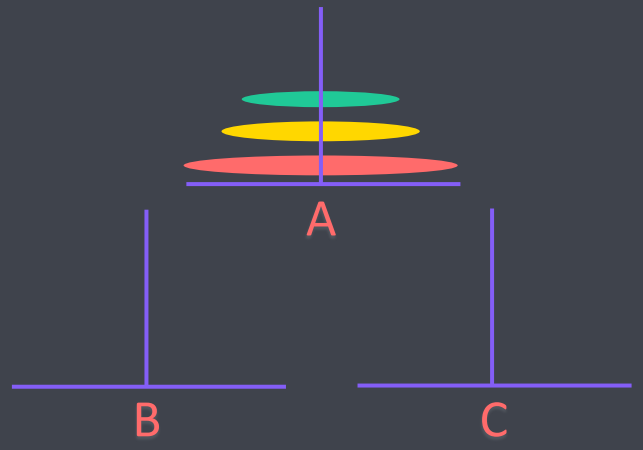
二. Choose 数据结构及算法思维选择

算法思维选择（美国学者提出的一种两步操作法）

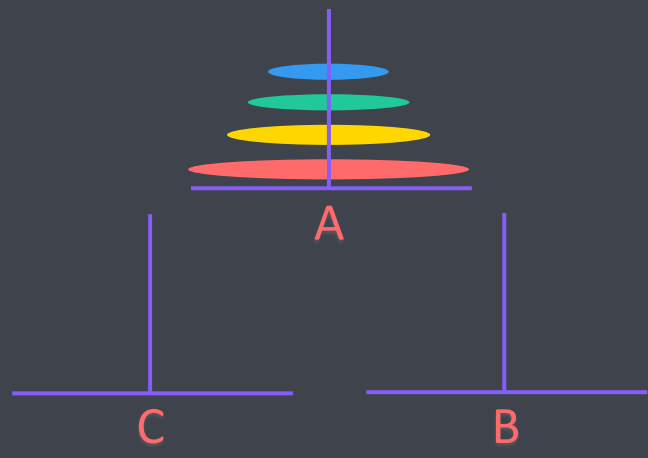
将柱子摆成品字型，摆放顺序由圆盘数量奇偶性决定

思考1层汉诺塔、2层汉诺塔如何移动

若n为奇数，按顺时针方向依次摆放 A C B



若n为偶数，按顺时针方向依次摆放 A B C



二. Choose 数据结构及算法思维选择

算法思维选择（美国学者提出的一种两步操作法）

1. 按顺时针方向把圆盘1从现在的柱子移动到下一根柱子
2. 把另外两根柱子上可以移动的圆盘移动到新的柱子上
   - 把非空柱子上的圆盘移动到空柱子上
   - 当两根柱子都非空时，移动较小的圆盘

反复进行1、2两步操作，最后就能按规定完成汉诺塔的移动

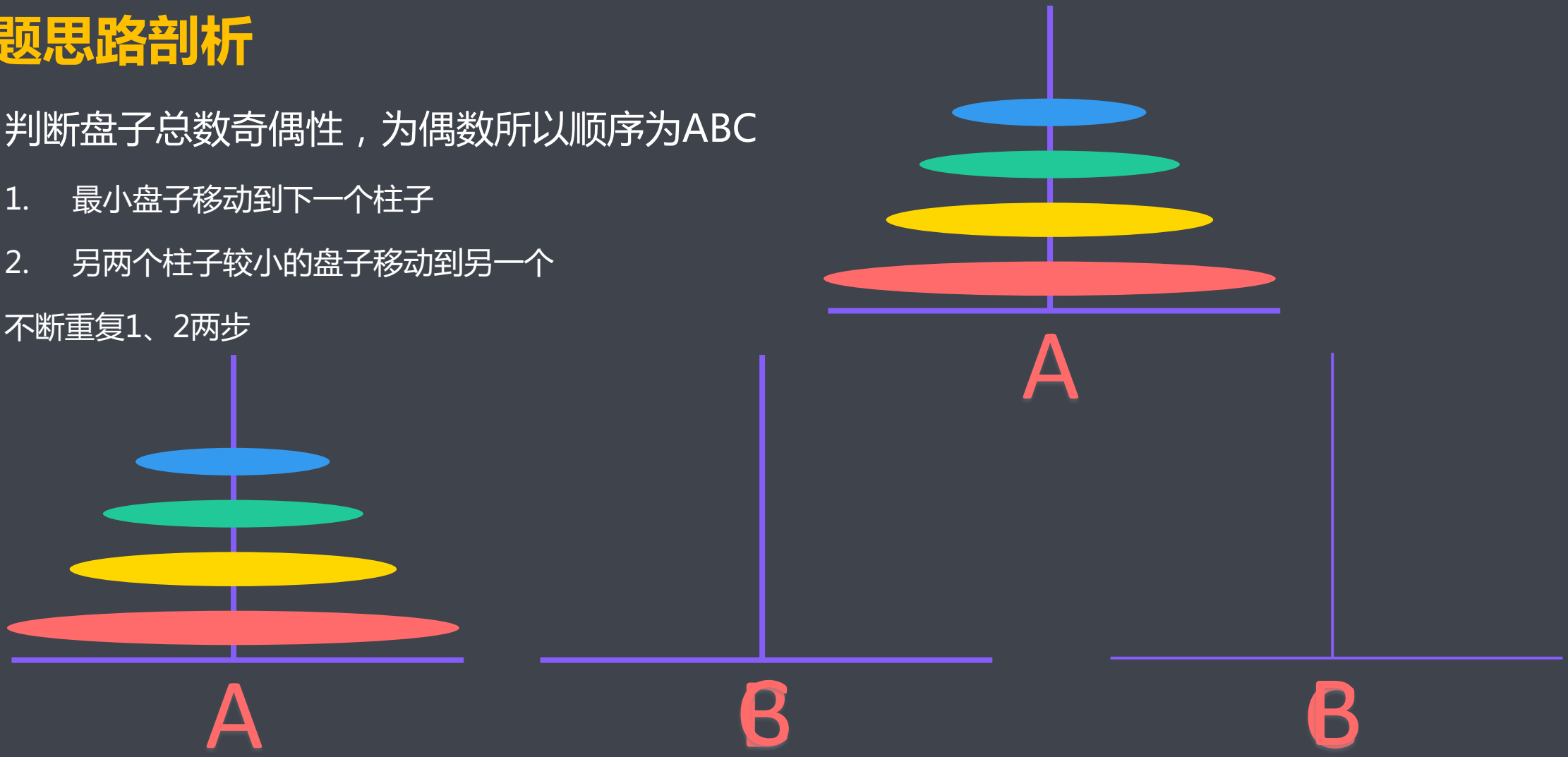
三. Code 基本解法及编码实现

解题思路剖析

判断盘子总数奇偶性，为偶数所以顺序为ABC

1. 最小盘子移动到下一个柱子
2. 另两个柱子较小的盘子移动到另一个

不断重复1、2两步



三. Code 基本解法及编码实现

复杂度分析

- 时间复杂度
  - 我们需要多次比较盘子大小以及移动盘子
  - 通过在代码中进行移动盘子的地方增加计数器的方式，我们可以得出移动次数为2^n-1
  - 同时我们在每次移动前还需要进行比较，比较次数也为2^n-1
  - 因此时间复杂度为$O(2^n)$
- 空间复杂度
  - 空间消耗方面我们只需要使用常数级的变量空间，因此空间复杂度为O(1)

三. Code 基本解法及编码实现

Java编码实现

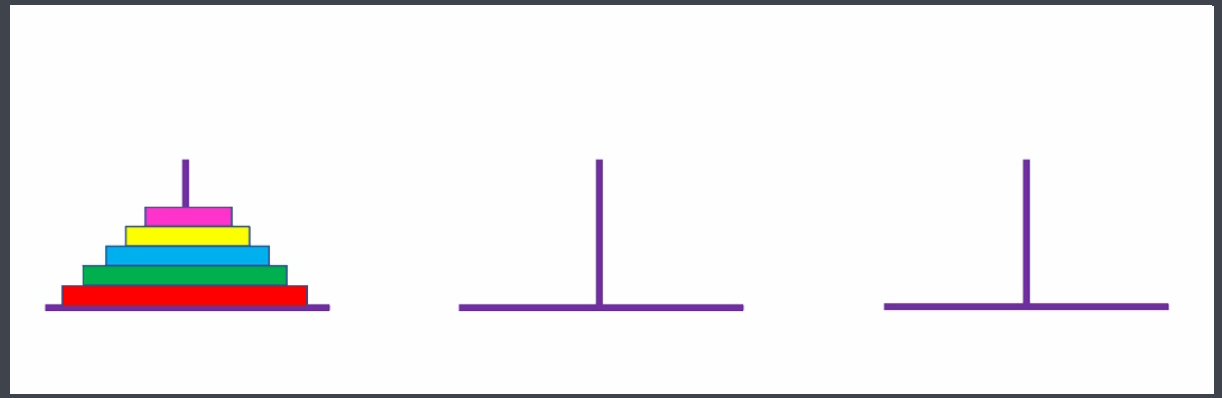
解答成功:
执行耗时:6 ms,击败了7.21% 的Java用户
内存消耗:37.4 MB,击败了77.97% 的Java用户

```java
public void hanota(List<Integer> A, List<Integer> B, List<Integer> C) {
    int size = A.size();
      List<Integer>[] lists = new List[3];
    lists[0] = A;
    if (size % 2 == 0) {//若n为偶数，按顺时针方向依次摆放 A B C;
        lists[1] = B;
        lists[2] = C;
    } else {      // 若n为奇数，按顺时针方向依次摆放 A C B
        lists[1] = C;
        lists[2] = B;
    }
     int currentIndex = 0; //记录最小盘子所在的柱子下标
    while (C.size() < size) {
        List<Integer> current = lists[currentIndex];
            currentIndex = (currentIndex + 1) % 3; //编号最小盘子所在柱子的下一个柱子
        List<Integer> next = lists[currentIndex];
            List<Integer> pre = lists[(currentIndex + 1) % 3]; //编号最小盘子所在柱子的上一个柱子
        int preSize = pre.size();
        int curSize = current.size();
            next.add(current.remove(--curSize)); //最小的圆盘   移动到下一个柱子
        //另外两根柱子上可以移动的圆盘移动到新的柱子上     当两根柱子都非空时，移动较小的圆盘
            int plateToMove1 = preSize == 0 ? Integer.MAX_VALUE : pre.get(preSize - 1);
        int plateToMove2 = curSize == 0 ? Integer.MAX_VALUE : current.get(curSize - 1);
        if (plateToMove1 < plateToMove2) {current.add(pre.remove(preSize - 1));}
        else if (plateToMove2 < plateToMove1) {pre.add(current.remove(curSize - 1));}
    }
}
```

四. Consider 思考更优解

是否存在无效代码或者无效空间消耗

是否有更好的算法思维



N层汉诺塔我们该如何思考呢？

- 如果我们能将上面的N-1层移动到B上
- 把N层移动到C，再把B上N-1层移动到C上就可以解决问题了
- 问题变为如何解决N-1层汉诺塔的移动问题
- 继续思考一直到N-1等于1时，我们可以直接将1层汉诺塔移动目的位置

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

关键知识点：递归

- 概念

  数学与计算机科学范畴，是指在函数的定义中使用函数自身的方法

  递归算法是一种直接或者间接调用自身函数的算法

- 本质

  递归，去的过程叫"递"，回来的过程叫"归"

  递是调用，归是结束后回来

  是一种循环，而且在循环中执行的就是调用自己

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

关键知识点：递归

- 三要素
  - 结束条件
  - 函数主功能
  - 函数的等价关系式（参数、返回值、关系）
- 递归方法模板

```java
public int recursion(int n) {
    //结束条件
    if (n == 1) {
        return 1;
    }
    //do something 函数主功能
    System.out.println(n);
    //等价关系式f(n)=n+f(n-1) 转换为简单问题
    return n + recursion(n - 1);
}
```

五. Code 最优解思路及编码实现

解题思路剖析

- 递归结束条件：移动一个盘子
- 递归函数主功能
  - 移动N-1个盘子到中间柱子
  - 移动第N个盘子到目标柱子
  - 将N-1个盘子从中间柱子移动到目标柱子
- 函数的等价关系式
  - 参数：本轮盘子数量、三个柱子（List）
  - 返回值：无返回值，使用List中本地删除的方式
  - 等价关系：

    $f(n , A , B , C)=f(n-1,A,C,B)+M(A,C)+f(n-1, B,A,C)$

五. Code 最优解思路及编码实现

复杂度分析

- 时间复杂度：使用迭代法分析

设递归函数的运行时间 T(n)

每一轮递归中都会再调用两次递归函数，每次都使问题的规模减少 1 个， $T(n) = 2 \times T(n - 1)$

所以最终的复杂度为$O(2^n)$，其中比较次数为0。

$T(n - 1) = 2 \times T(n - 2) + 1$:

$T(n) = 2 \times (2 \times T(n - 2) + 1) + 1 = 2^2 \times T(n - 2) + (2 + 1)$

$T(n) = 2 \times (2 \times (2 \times T(n - 3) + 1) + 1) + 1 = 2^3 \times T(n - 3) + (4 + 2 + 1)$

$T(n) = 2 \times (2 \times (2 \times (2 \times T(n - 4) + 1) + 1) + 1) + 1 = 2^4 \times T(n - 4) + (8 + 4 + 2 + 1)$

...

$T(n) = 2^k \times T(n - k) + (2^k - 1)$

其中，$1 + 2 + 4 + 8 + \ldots$ 是一个等比数列，由求和公式得到 $2^k - 1$。

当 k 等于 n 的时候，$T(n) = 2^n \times T(0) + (2^n - 1)$，

由于 T(0) 等于 1，所以最终 $T(n) = 2 \times 2^n - 1$。

五. Code 最优解思路及编码实现

复杂度分析

- 空间复杂度
  - 函数中我们不需要主动创建额外存储
  - 但递归调用本身需要进行堆栈的存储
  - 空间复杂度和递归的深度有关系
  - 递归深度为n，所以空间复杂度为O(n)

五. Code 最优解思路及编码实现

Java编码实现

info
解答成功:
执行耗时:1 ms,击败了84.31% 的Java用户
内存消耗:37.7 MB,击败了13.28% 的Java用户

```java
public void hanota(List<Integer> A, List<Integer> B, List<Integer> C) {
    movePlate(A.size(), A, B, C);
}
/**
 * @param size  需要移动盘子的数量
 * @param start  起始柱子
 * @param auxiliary  辅助柱子
 * @param target  目标柱子
 */
private void movePlate(int size, List<Integer> start, List<Integer>
auxiliary, List<Integer> target) {
    if (size == 1) {    // 只剩一个盘子时，直接从第一个柱子移动到第三个柱子 即可
        target.add(start.remove(start.size() - 1));
        return;
    }
    // 将 n-1 个盘子，从 第一个柱子 移动到 第二个柱子
    movePlate1(size - 1, start, target, auxiliary);
    // 将第 n个盘子，从 第一个柱子 移动到 第三个柱子
    target.add(start.remove(start.size() - 1));
    // 再将 n-1 个盘子，从 第二个柱子 移动到 第三个柱子
    movePlate1(size - 1, auxiliary, start, target);
}
```

六. Change 变形延伸

拉勾教育
—互联网人实战大学—

题目变形

如果规则变为大盘必须放在小盘上该如何实现？

延伸扩展

递归一种基本的编程思维

- 快速排序、归并排序、二分查找、树……

实际工作中应用很广泛

- 按照组织架构统计公司某部门人数
- 按职级关系统计销售人员提成

总结

- 使用循环解决汉诺塔问题
- 递归算法的思想
- 递归算法三要素及编码模板
- 运用递归算法解决汉诺塔问题

课后练习

1. 使用递归计算阶乘（ n!=(n-1)!×n /简单）
2. 使用递归求解 斐波那契数（Leetcode 509/简单）
3. 使用递归求解第 N 个泰波那契数（ Leetcode 1137 /简单）
4. 使用递归求解 Pow(x, n) --数值的整数次方（ Leetcode 50 /中等）

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容