E2-E problem

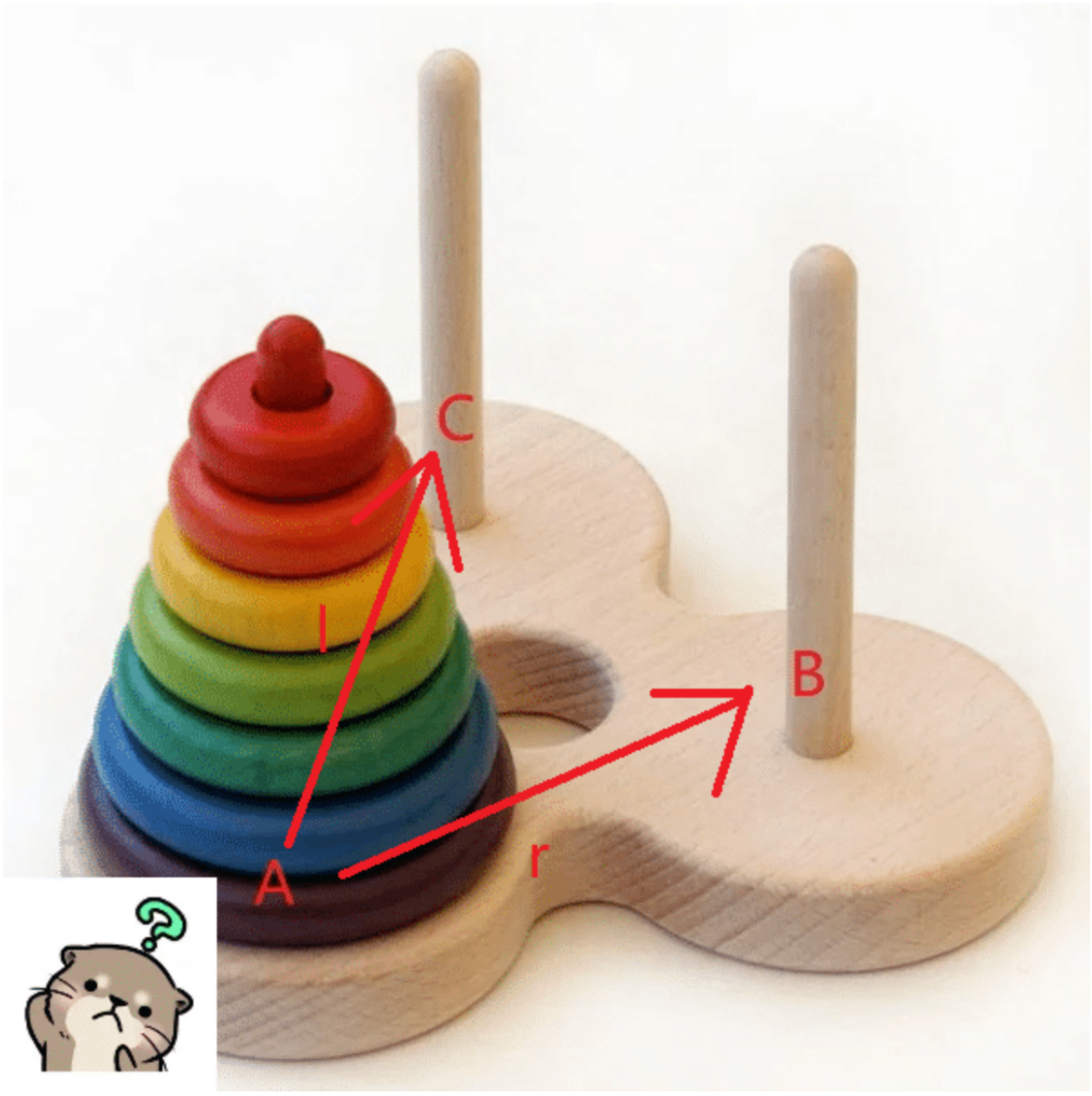
题目描述

小水獭学算法学累了，于是开始玩汉诺塔。它了解到，汉诺塔是一个有名的益智游戏，一共有三根柱子，其中一根柱子上从下往上按照大小顺序摞着 n 片圆盘，要求圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。

小水獭非常聪明，它很快明白了普通汉诺塔的玩法。然而它还没有玩够，于是它构想了一个奇怪的汉诺塔。如下图所示，三柱均匀分布在一个圆形基座上，规定初始柱为 A 柱，目标柱为 C 柱，中间柱为 B 柱，且每个圆盘仅能依次顺时针旋转或逆时针旋转，其他规则和普通的汉诺塔相同。

举个栗子，若圆盘 22 规定为仅能顺时针旋转，那么它仅能以 A→C→B→A 的顺序移动，若此时圆盘 2 在 A 柱，而圆盘 1 在 C 柱，则圆盘 2 无法移动，只能先设法移开圆盘 1 才能移动。

小水獭想问你如何以最少的操作次数，使所有圆盘从 A 柱移动到 C 柱。



题解思路

- 此题为经典汉诺塔的变种，限制了没片圆盘的移动方向。只需要稍微修改汉诺塔的递归函数和移动函数即可。
- 首先设计函数`void move(int n, char from, char to, char direct)`解决高度为1的汉诺塔的移动问题，需要考虑移动目标与圆盘移动方向的关系，需要分为目标与圆盘移动方向相同、相反两种情况考虑，两种情况圆盘分别需要移动1次、2次。
- 之后设计递归函数，使用direct存储所有圆盘限制的移动方向。对于高度为n的汉诺塔的从from杆，移动到to杆的问题，根据n号盘移动目标与n号盘移动方向的关系，分为目标与n号圆盘移动方向相同、相反两种情况考虑。【规定除了from,to杆以外的另一个杆为mid杆】
    - 若目标与n号圆盘移动方向相同，则按如下步骤操作：

      1、将n-1高度的汉诺塔移动到mid杆

      2、将n号圆盘一步移动到to杆

      3、将n-1高度的汉诺塔移动到to杆。
    - 若目标与n号圆盘移动方向相反，则按如下步骤操作：

      1、将n-1高度的汉诺塔移动到to杆

      2、将n号圆盘移动到mid杆

      3、将n-1高度的汉诺塔移动到from杆

      4、将n号盘移动到to杆

      5、将n-1高度的汉诺塔移动到to杆

代码

```
#include <iostream>
#include<bits/stdc++.h>

using namespace std;

char pos[3] = {'A', 'B', 'C'};

void move(int n, char from, char to, char direct) {
    if (to == pos[((from - 'A') + 2) % 3] && direct == 'l') {
        printf("move %d from %c to %c\n", n, from, to);
    }
    else if (to == pos[((from - 'A') + 1) % 3] && direct == 'r') {
        printf("move %d from %c to %c\n", n, from, to);
    }
    else if (to == pos[((from - 'A') + 2) % 3] && direct == 'r') {
        printf("move %d from %c to %c\n", n, from, pos[((from - 'A') + 1) % 3]);
        printf("move %d from %c to %c\n", n, pos[((from - 'A') + 1) % 3], to);
    }
    else if (to == pos[((from - 'A') + 1) % 3] && direct == 'l') {
        printf("move %d from %c to %c\n", n, from, pos[((from - 'A') + 2) % 3]);
        printf("move %d from %c to %c\n", n, pos[((from - 'A') + 2) % 3], to);
    }
    else {
        printf("move error!");
    }
}

void hanoi(int n, char from, char to, char *direct) {
```

```
    if (n == 1) {
        move(1, from, to, direct[0]);
    }
    else {
        if (to == pos[((from - 'A') + 2) % 3] && direct[n-1] == 'l') {
            hanoi(n-1, from, pos[((from - 'A') + 1) % 3], direct);
            printf("move %d from %c to %c\n", n, from, to);
            hanoi(n-1, pos[((from - 'A') + 1) % 3], to, direct);
        }
        else if (to == pos[((from - 'A') + 1) % 3] && direct[n-1] == 'r') {
            hanoi(n-1, from, pos[((from - 'A') + 2) % 3], direct);
            printf("move %d from %c to %c\n", n, from, to);
            hanoi(n-1, pos[((from - 'A') + 2) % 3], to, direct);
        }
        else if (to == pos[((from - 'A') + 2) % 3] && direct[n-1] == 'r') {
            hanoi(n-1, from, to, direct);
            printf("move %d from %c to %c\n", n, from, pos[((from - 'A') + 1) %
3]);
            hanoi(n-1, to, from, direct);
            printf("move %d from %c to %c\n", n, pos[((from - 'A') + 1) % 3],
to);
            hanoi(n-1, from, to, direct);
        }
        else if (to == pos[((from - 'A') + 1) % 3] && direct[n-1] == 'l') {
            hanoi(n-1, from, to, direct);
            printf("move %d from %c to %c\n", n, from, pos[((from - 'A') + 2) %
3]);
            hanoi(n-1, to, from, direct);
            printf("move %d from %c to %c\n", n, pos[((from - 'A') + 2) % 3],
to);
            hanoi(n-1, from, to, direct);
        }
    }
}

int main() {
    int n;
    while (scanf("%d ", &n) != EOF) {
        char direction[21];
        scanf("%s", direction);
        hanoi(n, 'A', 'C', direction);
        printf("\n");
    }

    return 0;
}
```