基本解法

Java代码

```
public void hanota(List<Integer> A, List<Integer> B, List<Integer> C) {
    int size = A.size();
    //准备：根据圆盘的数量确定柱子的排放顺序（使用数组存放）：
    //若n为偶数，按顺时针方向依次摆放 A B C；
    // 若n为奇数，按顺时针方向依次摆放 A C B
    List<Integer>[] lists = new List[3];
    lists[0] = A;
    if (size % 2 == 0) {
        lists[1] = B;
        lists[2] = C;
    } else {
        lists[1] = C;
        lists[2] = B;
    }
    //记录最小盘子所在的柱子下标
    int currentIndex = 0;
    while (C.size() < size) {
        List<Integer> current = lists[currentIndex];
        //编号最小盘子所在柱子的下一个柱子
        currentIndex = (currentIndex + 1) % 3;
        List<Integer> next = lists[currentIndex];
        //编号最小盘子所在柱子的上一个柱子
        List<Integer> pre = lists[(currentIndex + 1) % 3];
        int preSize = pre.size();
        int curSize = current.size();
        //最小的圆盘  移动到下一个柱子
        next.add(current.remove(--curSize));

        //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));
        }
    }
}
```

优化解法

递归模板代码

```
public int recursion(int n) {
    //结束条件
    if (n == 0) {
        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);
}
```

Java代码

```
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) {
    //结束条件 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
    if (size == 1) {
        target.add(start.remove(start.size()-1));
        return;
    }
    //函数主功能：移动n-1个盘子，移动第n个盘子，移动n-1个盘子
    // 等价关系式 f(n , A , B , C)=f(n-1,A,C,B)+M(A,C)+f(n-1, B,A,C)
    // 将 n-1 个盘子，从 第一个柱子 移动到 第二个柱子
    movePlate(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
    movePlate(size-1,auxiliary,start,target);
}
```

C代码

```c
#define MAX_LEN 200
typedef struct {
    int* data;
    int size;
}Stack;
int cnt = 0;
void Push(Stack* s, int d)
{
    s->data[s->size++] = d;
}
int Pop(Stack* s){
    return s->data[--s->size];
}
void hanota(int* A, int ASize, int* B, int BSize, int** C, int* CSize){
    if(ASize <= 0) return;
    Stack X,Y,Z;
    X.data = A;
    X.size = ASize;
    Y.data = (int*)malloc(sizeof(int)*MAX_LEN);
    Y.size = 0;
    int *p = (int*)malloc(sizeof(int)*MAX_LEN);
    Z.data = p;
    Z.size = 0;
    hannuota(ASize, &X, &Y, &Z);
    *C = p;
    *CSize = Z.size;
    free(Y.data);
}

void hannuota(int n, Stack* A, Stack* B, Stack* C){
    if(n == 1){
        cnt ++;
        Push(C, Pop(A));
        return;
    }
    hannuota(n-1, A, C, B);
    hannuota(1, A, B, C);
    hannuota(n-1, B, A, C);
}
```

C++代码

```c
class Solution {
```

```
public:
    void hanota(vector<int>& A, vector<int>& B, vector<int>& C) {
        int n = A.size();
        //n 个盘子通过 B 从 A 移向 C
        move(n, A, B, C);
    }
    void move(int n, vector<int>& A, vector<int>& B, vector<int>& C){
        if(n == 1){
            C.push_back(A.back());
            A.pop_back();
        } else{
            //n - 1 个盘子通过 C 从 A 移向 B
            move(n - 1, A, C, B);
            //最底下的盘子移向 C
            move(1, A, B, C);
            //n - 1 个盘子通过 A 从 B 移向 C
            move(n - 1, B, A, C);
        }
    }
};
```

Python代码

```
class Solution:
    def hanota(self, A: List[int], B: List[int], C: List[int]) -> None:
        self.movePlate(len(A), 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
    def movePlate(self, size:int, start: List[int], auxiliary: List[int],
target: List[int]) -> None:
        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.append(start.pop())
        else:
            #将 n-1 个盘子，从 第一个柱子 移动到 第二个柱子
            self.movePlate(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.append(start.pop());
            #再将 n-1 个盘子，从 第二个柱子 移动到 第三个柱子
            self.movePlate(size - 1, auxiliary, start, target)
```