Java方向编程题答案

####

day17

[编程题]36901-火车进站

链接：https://www.nowcoder.com/questionTerminal/97ba57c35e9f4749826dc3befaeae109

【题目解析】

该题目中的字典排序的意思是：n辆火车有多少种出站的可能顺序，即数据结构中的栈有多少种出栈的顺序。

【解题思路】

思路为用三个变量分别存储：

- 待进站火车
- 站中火车（用Stack存储）
- 已出站火车

具体实现思路：

第一种：

采用递归的方法，递归函数的参数为当前待进站火车、站中火车、已出站火车的值所组成的三元组，递归结束条件是，**未进站火车和站中火车均为空**，此时输出已出站火车即为所有出站的一种可能，递推关系为对于当前情况有让下一辆火车进站或让站中的一辆火车出站两种可能，对于两种可能分别调用递归函数，即可得出问题的解。

第二种：

采用先对火车编号进行排列组合，计算出所有可能的出站情况。但是火车出站的情况需要满足栈的出栈顺序，所以通过**火车编号的顺序**，**排列组合的顺序**进行出栈和入栈来比较排列组合中的一组顺序是否满足条件，如果满足，则该排序就是有效的出栈顺序。

（下面的代码实现采用第二种思路）

【示例代码】

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Scanner;
import java.util.Set;
import java.util.TreeSet;

public class Main {

public static void main(String[] args) {
   Scanner in = new Scanner(System.in);
   while(in.hasNext()){
       //输入火车数量
       int n = in.nextInt();
       //输入火车编号
       int[] A = new int[n];
```

```
        for(int i=0;i<n;i++){
            A[i] = in.nextInt();
        }
        int start = 0;
        //计算n个火车的出站的编号的排列组合
        ArrayList<int[]> result = new ArrayList<int[]>();
        Permutation(A,start,n,result);
        //出栈的结果，一个元素一个记录，例如：1 2 3 ；  1 3 2
        Set<String> sortResult = new TreeSet<String>();
        //循环排列组合
        for(int[] out : result){
            //判断是否满足出栈要求（后进先出）
            if(isLegal(A,out,n)){
                //满足的组合，输入结果，每一个编号用空格分隔
                StringBuilder sb = new StringBuilder();
                for(int i=0;i<n-1;i++){
                    sb.append(out[i]+" ");
                }
                sb.append(out[n-1]);
                sortResult.add(sb.toString());
            }
        }
        //最后输出所有的符合出栈要求的组合
        for(String list:sortResult){
            System.out.println(list);
        }
    }
    in.close();

}

/*
 in  :  火车编号数组
 out :  火车出站顺序
 n   :  火车数量
*/
private static boolean isLegal(int[] in,int[] out,int n){
    //栈：存储进站的火车编号
    LinkedList<Integer> stack = new LinkedList<Integer>();
    int i=0;
    int j=0;
    while(i<n){ // in 还有元素的时候都需要判断
        if(in[i] == out[j]){ //相等时候就不用入栈，直接后移
            i++;
            j++;
        }else{
            if(stack.isEmpty()){ //空stack 就只有入栈了
                stack.push(in[i]);
                i++;
            }else{
                int top = stack.peek(); // 栈顶元素相等，进行出栈
                if(top ==out[j]){
                    j++;
                    stack.pop();
                }else if(i<n){ //不相等时候入栈，说明还有待进站的车
                    stack.push(in[i]);
                    i++;
                }
```

```
            }
        }
    }
    while(!stack.isEmpty() && j<n){ // in 的结束后，栈中元素进程出栈序列应该和out剩余的元素相同
        int top = stack.pop();
        if(top == out[j]){
            j++;
        }else{
            return false;
        }
    }
    return true;

}
/**
     * 求出所有排列
     * @param A
     * @param start
     * @param n
     * @param result
*/
private static void Permutation(int[] A,int start,int n,ArrayList<int[]>
result){
    if(start == n){
        return;
    }
    if(start == n-1){
        int[] B = A.clone();
        result.add(B);
        return;
    }
    for(int i=start;i<n;i++){
        swap(A,start,i);
        Permutation(A,start+1,n,result);
        swap(A,start,i);
    }

}
private static void swap(int[] A,int i,int j){
    int t = A[i];
    A[i] = A[j];
    A[j] = t;
}
}
```

[编程题]23270-二叉树的镜像

链接：https://www.nowcoder.com/questionTerminal/564f4c26aa584921bc75623e48ca3011

【题目解析】

无

【解题思路】

源二叉树：

```
        8
       /  \
      6   10
     / \  / \
    5  7 9 11
```

镜像二叉树：

```
    8
       /  \
      10   6
     / \  / \
    11 9 7   5
```

通过上述图示，可以看出，源二叉树镜面成像变成镜像二叉树。 一个节点的左右节点进行互换，可以通过递归来实现。

【示例代码】

```
public class Solution {
public void Mirror(TreeNode root) {
    //节点为null 不处理
    if(root == null)
        return;
    //节点的左右子节点为null(即就是节点为叶子节点）同样不处理
    if(root.left == null && root.right == null)
        return;
    //节点的左右子节点交换
    TreeNode pTemp = root.left;
    root.left = root.right;
    root.right = pTemp;
    //递归处理
    if(root.left != null)
        Mirror(root.left);
    if(root.right != null)
        Mirror(root.right);
}
}
```