Class 19 加强练习 I

Q1 Element deduplication/removal in an array.

隔板题：

基本思想：用两个变量，一个变量记录当前指针位置 (= fast index)，一个变量记录隔板位置 (= slow index)。

性质1：slow 隔板左边是处理好的元素，当前指针 fast 右边是未处理的元素，隔板和当前指针之间的区域是无用的元素。每次只要分析当前元素性质是否要加入或者移动 slow 隔板就可以了。

性质2：用快慢2个指针，同向而行，处理完毕之后， return 的结果中，**每个integer/char的相对位置不变。**

Discussion

1. 隔板，同向而行, Two pointers moving in the same direction
    a. 基本思想：用两个变量，一个变量记录左边隔板位置 (= slow index)，一个变量记录右边隔板位置 (= fast index)

    b. 性质1：slow 左边是处理好的元素，right右边是未知探索区域，两个隔板中间是do not care
    c. 性质2：处理完毕之后， return 的结果中，每个integer/char的相对位置不变。
2. 隔板，相向而行, Two pointers moving in opposite direction
    a. 基本思想：用两个变量，一个变量记录左边隔板位置 (= left index)，一个变量记录右边隔板位置 (= right index)
    b. 性质1：left左边是处理好的元素，right右边也是处理好的元素，两个隔板中间是未处理区域。
    c. 性质2：处理完毕之后， return 的结果中，每个integer/char的相对位置可能发生变化。

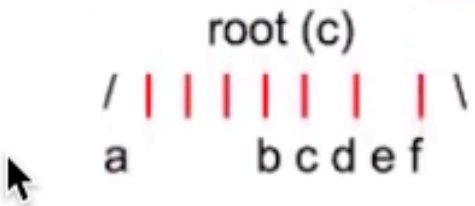
Q5.4 LCA for two nodes in k-naryTree

LCA(a, b)

a b

```
class TreeNode {
        int val;
        List<TreeNode> children;    // TreeNode[] children;
}
```

Q5.5 LCA for k nodes in k-naryTree **LCA(a, b)**




```
class TreeNode {
        int val;
        List<TreeNode> children;  // TreeNode[] children;
}
```

```
public TreeNode LCA(TreeNode root, Set<TreeNode> nodes) {
      if (root == null || nodes.contains(root)) {
            return root;
      }

      int counter = 0;
```

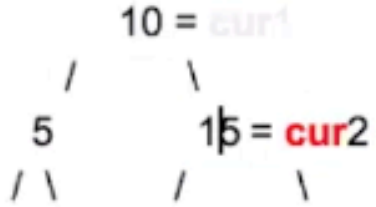
```
    TreeNode temp = null;
    for (TreeNode child : root.children){ //  step 1
          TreeNode node = LCA(child, nodes);
          if (node != null){                    //    + step2
                counter++;
                if (counter > 1){
                      return root;
                }
                temp = node;
          }
    }
    return temp;                                //  step 3
}
```

```
class TreeNode {
        int val;
        List<TreeNode> children;    // TreeNode[] children;
}

public TreeNode LCA(TreeNode root, TreeNode a, TreeNode b) {
      if (root == null || a == root || b == root) {
            return root;
      }

      int counter = 0;
      TreeNode temp = null;
      for (TreeNode child : children){        //  step 1
            TreeNode node = LCA(child, a, b);
            if (node != null){                //    + step2
                  counter++;
                  if (counter == 2){
                        return root;
                  }
                  temp = node;
            }
      }
      return temp;                            //  step 3
}
```

Q5.7: LCA in Binary Search Tree



2 7 12=a 20=b

1

```
public TreeNode solution(TreeNode root, TreeNode one, TreeNode two){
        if (root == null)
                return root;
        if (root.value < one.value && root.value < two.value){
                return solution(root.right, one, two);
        } else if (root.value > one.value && root.value > two.value){
                return solution(root.left, one, two);
        } else {
                return root;
```