常见算法总结 - 数组篇

作者：valarchie
原文链接：https://ld246.com/article/1588212614084
来源网站：链滴

1.给定一个数值在1-100的整数数组，请找到其中缺少的数字。

找到丢失的数字 利用byte数组的1或0标记该数字是否被删除，例如byte数组下标为0的数值为1的话代表数字1存在

```
public static void findMissNumber1(int[] ints) {
        // 声明一个byte数组
        byte[] isExist = new byte[100];
        for (int i = 0; i < ints.length; i++) {
            // 由于数值比下标大1， 0位置其实代表的是数字1
            isExist[ints[i] - 1] = 1;

        }
        for (int i = 0; i < isExist.length; i++) {
            if (isExist[i] == 0) {
                System.out.println("删除的数字是:" + ++i );
            }
        }
}
```

我们可以利用1-100的总和为5050，我们依次减掉数据内的所有值，得到的差值即为删除的值

```
public static void findMissNumber2(int[] ints) {
        int sum = 5050;
        for (int i = 0; i < ints.length; i++) {
            sum -= ints[i];
        }
        System.out.println("删除的数字：" + sum);
}
```

2.从数组中找出给定目标数字的两数组合。例如数组为，给定数字给17，那么组内的3+14=17。

先利用Set存储对应的数字，然后再遍历数组，假设遍历到数字3的时候，检查set中是否存在（17-3=14这个数字，如果存在的话，即存在这个组合。

```
public static void findPairNumber(int[] arrays, int target) {

        Set<Integer> existIntegers = new HashSet<>();
        for (int i = 0; i < arrays.length; i++) {
            existIntegers.add(arrays[i]);
        }
        for (int i = 0; i < arrays.length; i++) {
            if (existIntegers.contains(target - arrays[i])) {
                System.out.println("找到对应的数字组合：" + arrays[i] + "和" + (target - arrays[i]));
                // 去除掉已使用过的数字
                existIntegers.remove(arrays[i]);
            }
        }
    }
```

3.将数组进行反转。

只需要将数组按中间位置为对称轴进行位置交换即可

```
public static void reverseArray(int[] arrays) {

    for (int i = 0; i < arrays.length/2; i++) {

        int tmp = arrays[i];
        arrays[i] = arrays[arrays.length -1 - i];
        arrays[arrays.length -1 - i] = tmp;

    }

}
```

4.寻找数组中前K个最大的数。

我们新建一个长度为K的数组，遍历原数组，然后将大于K数组内的数字，放进K数组里，然后冒泡淘末尾的数字，最后剩下的就是前K个最大的数字。

```
public static int[] findKMaxInts(int K, int[] ints) {

    int[] kInts = new int[K];

    for (int i = 0; i < ints.length; i++) {
        // 如果大于kInts 数组中的最后一个数字的话，直接插入
        if (ints[i] > kInts[K - 1]) {
            kInts[K - 1] = ints[i];
        }
        // 再进行冒泡
        bulb(kInts);
    }

    return kInts;
}


public static void bulb(int[] ints) {

    for (int i = ints.length-1; i >= 1; i--) {

        // 进行冒泡
        if (ints[i] > ints[i - 1]) {

            int tmp = ints[i];
            ints[i] = ints[i - 1];
            ints[i-1]=tmp;

        }

    }
```

```
}
```

5.求一个数组中连续子向量的最大和

遍历数组，从第一个大于0的数字开始累加，保存最大值，如果累加后的值小于等于0的话，就抛这一标，从下一个下标开始累积，如果超过之前的最大值的话就进行替换。使用sum和max变量进行操作

```
public static Integer findMaxSum(int[] ints) {
        if (ints.length == 0) {
            return null;
        }
        int currentSum = 0;
        int maxSum = 0;
        for (int i = 0; i < ints.length; i++) {
            currentSum += ints[i];
            if (currentSum < 0) {
                currentSum = 0;
            }
            if (currentSum > maxSum) {
                maxSum = currentSum;
            }
        }
        return maxSum;
}
```

6.一个数组中数字都两两重复，只有一个数字没有重复，如何找到那个数字?

我们可以将所有数字进行异或，两两重复的数字，都会被抵消掉，剩余最后的数字就是单个不重复的字

```
public static Integer findSingleNumber(int[] ints) {

        if (ints.length == 0) {
            return null;
        }
        int singleNumber = ints[0];
        for (int i = 1; i < ints.length; i++) {
            singleNumber ^= ints[i];
        }
        return singleNumber;
    }
```

7.将一个二维数组顺时针旋转90度.

先找出旋转90度，下标的变换规律。然后从外圈向内圈递归旋转变换。

```
public static void rollMatrix90(int[][] matrix, int circle) {

        // 当circle为0的时候为最外圈
```

```
        if (matrix.length - circle * 2 <= 0) {
            return;
        }

        int bound = matrix.length - circle - 1;

        for (int i = circle; i < matrix.length - 1 - circle; i++) {

            int a = matrix[circle][i];
            int b = matrix[i][bound];
            int c = matrix[bound][bound - i + circle];
            int d = matrix[bound - i + circle][circle];

            matrix[circle][i] = d;
            matrix[i][bound] = a;
            matrix[bound][bound - i + circle] = b;
            matrix[bound - i + circle][circle] = c;

        }

        rollMatrix90(matrix, circle + 1);

    }
```

原数组

```
1,2,3,4,5,
6,7,8,9,10,
11,12,13,14,15,
16,17,18,19,20,
21,22,23,24,25,
```

翻转后的数组

```
21,16,11,6,1,
22,17,12,7,2,
23,18,13,8,3,
24,19,14,9,4,
25,20,15,10,5,
```

8.找出数组的中位数。

使用快排的思想，但是只往中间位置进行排序，当最后的排序下标抵达数组中间位置的时，就是中位。

```
public static int findMiddleNumber(int[] array, int begin, int end) {

        int beginRange = begin;
        int endRange = end;

        if (begin >= end) {
            return begin;
```

```
        }

        // 选择begin位置的数字作为分界点
        int splitNumber = array[begin];

        while (begin + 1 < end) {

            if (array[begin + 1] < splitNumber) {
                begin++;
                continue;
            }

            if (array[end] > splitNumber) {
                end--;
                continue;
            }

            int tmp = array[begin + 1];
            array[begin + 1] = array[end];
            array[end] = tmp;

        }

        int tmp = array[beginRange];
        array[beginRange] = array[begin];
        array[begin] = tmp;

        if (begin > array.length / 2) {

            return findMiddleNumber(array, beginRange, begin);

        } else {
            return findMiddleNumber(array, end, endRange);

        }

}
```

9.两个有序数组的合并排序

同时遍历两个数组，每次对比两个数组当前坐标的值哪个小，小的存入新的数组，数组下表往后移一，大的那边坐标不变，继续下次大小对比。依次循环。

```
public static int[] mergeSortArray(int[] arrayA, int[] arrayB) {
        int[] sortArray = new int[arrayA.length + arrayB.length];
        int aIndex = 0;
        int bIndex = 0;
        for (int i = 0; i < sortArray.length && aIndex <= arrayA.length && bIndex <= arrayB.leng
h; i++) {
            if (aIndex == arrayA.length) {
                sortArray[i] = arrayB[bIndex++];
                continue;
            }
            if (bIndex == arrayB.length) {
```

```
            sortArray[i] = arrayA[aIndex++];
            continue;
        }
        if (arrayA[aIndex] < arrayB[bIndex]) {
            sortArray[i] = arrayA[aIndex];
            aIndex++;
        } else {
            sortArray[i] = arrayB[bIndex];
            bIndex++;
        }
    }
    return sortArray;
}
```