0805拼多多

# 算法题经验总结：

就拿这次算法的第二题：一开始看起来挺困难的，但是仔细把情况分析下，发现很有规律可循。虽然一开始，走了弯路：笨方法，不过总的来说实现出来了，这就是第一步，也是最重要的一步。在原来的规律上进一步归纳总结，就可以简化。

所以以后，遇到算法题，大胆分类，在草稿纸上分类归纳总结。

大胆，不要怕，这是最重要的。

# 旋转字符串

## 题目要求

2018年8月5日 拼多多笔试算法第1道

时间限制：C/C++ 1秒，其他语言2秒；

空间限制：C/C++ 32768K，其他语言65536K；

64bit IO Format:%IId

题目描述：

给定一个字符串，按顺时针顺序输出一个正方形，具体规则如下：

1. 从上边开始，上边从左到右；
2. 然后到右边，右边从上到下；
3. 然后是下边，下边从右到左；
4. 最后是左边，左边从下到上。

输入描述：

输入一行，包含4K(K为整数，1<=K<=10)个小写字母。

输出描述：

输出K+1行，按上面的规则输出正方形，正方形内部用空格填充。

示例：

输入：

abcdefghijklmnop

输出：

abcde

p f

o g

n h

mlkj i

## 思路分析

难度：Easy

（1）首先分析一共是4\*K个字符，需要拼成的矩形字符是K+1行，且每行K+1个字符；

（2）按照行分类：第1行和K+1行比较特殊，需要单独实现；第2-K行规律类似，看成一种情况，共三种情况分析：line=1；line=K+1;1<line<K+1;(代码中行号从0开始）；

(3)第 1行：nums[0]---nums[K];第K+1行：nums[2\*K]--nums[3\*K]逆序；

第2-K行：nums[len-lineNum+1],K-1个空格，nums[K+lineNum-1]

## Java代码

/\*\*

\* the first problem

\* 旋转的字符串

\*/

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String str = scanner.nextLine();

char[] chars = str.toCharArray();

int K = chars.length/4;

char[] spaces = new char[K-1];

for(int i = 0;i<K-1;i++){

spaces[i] = ' ';

}

String spaceStr = new String(spaces);

for(int i = 0;i < K+1;i++){//控制行数：分成三种情况

**if(i == 0){**

**System.out.println(str.substring(0,K+1));**

**}else if(i >0 && i < K){**

**System.out.println(chars[4\*K-i]+spaceStr+chars[K+i]);**

**}else{**

**System.out.println(new StringBuilder(str.substring(2\*K,3\*K+1)).reverse().toString());**

**}**

}

}

# 有趣的变换

## 题目要求

2018年8月5日 拼多多笔试算法第2道

时间限制：C/C++ 1秒，其他语言2秒；

空间限制：C/C++ 32768K，其他语言65536K；

64bit IO Format:%IId

题目描述：

字符串形式的正整数(可能包含前缀0,1<=length<=10)，先将这个字符串拆分成两部分，接着可以在这两部分中分别加入一个小数点也可以不加入，分别形成一个整数或小数。找出所有经“拆分”和“变化”两次操作后所有可能组合的数目。

要求：

1). 对于新形成的整数和小数，不可包含多余的前缀0，比如010和010.1不合法；

2). 对于小数，不可包含多余的后缀0，比如0.10不合法；

3). .1和1.这样的小数不合法。

输入描述：

输入为一行，包含一个字符串形式的正整数。

输出描述：

输出为一行，找出经过“拆分”和“变化”后的所有组合的数目。

示例1：

输入： 123

输出： 4

说明：[[1,23],[12,3],[1.2,3],[1,2.3]]

示例2：

输入： 00011

输出： 2

可能的组合如下：

[[0.001,1],[0,0.011]]

## 思路分析

难度：Medium

第二种：优化思路（先看第一种）

提供halfCount函数，统计当前段的符合规范的情况。

首先：一位数，返回1；

若开头和结尾都是0，则返回0；

若开头或结尾有一个为0，则返回1；因为开头为0，只能为0.xxx；结尾为0，只能为整数；

最后，可能情况就是字符的个数，如203：203,2.03,20.3

第一种：笨重思路

（1）首先将整数字符串转换成字符数组char[] nums，其长度为len;

（2）遍历分割方式：[1,len-1]、[2,len-2].....、[len-1,1];

（3）在上述的每一种情况下，分别计算前一段、后一段符合规则的数目preCount和sufCount，然后将preCount\*sufCount乘积累加起来；输出sum即为答案。

（4）分割方式采用遍历方法，对于分割后的一段，由于小数点的移动会带来多种可能，因此构建一个方法isLegal用来判断当前小数点下是否合法：boolean isLegal(char[] nums,int from ,int end,int point)；具体实现思路：

首先判断是否是一位数，若是一位数，返回true；

再判断是否是多位整数：若是多位整数且不以0开头，则返回true；否则返回false；

最后判断小数：小数以0结尾，必须为false；小数若以0开头必须是0.xxx，若为0.xxx返回true，否则返回false。

## Java代码

### 优化代码

/\*\*

\* the second problem

\* 有趣的变换

\* 比较优化的方法

\*/

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String numStr = scanner.nextLine();

char[] nums = numStr.toCharArray();

int count = 0;

for(int i = 0;i < nums.length-1;i++){

int leftCount = halfCount(nums,0,i);

if(leftCount == 0) continue;//提高效率；左段不存在；右段就不用统计了

int rightCount = halfCount(nums,i + 1,nums.length-1);

count += leftCount\*rightCount;

}

System.out.println(count);

}

public static int halfCount(char[] nums,int from ,int end){

if(from == end) return 1;//一位数肯定就一种

if(nums[from] == '0'&& nums[end] == '0') return 0;//前后都有0，肯定不合法

if(nums[from] == '0'|| nums[end] == '0') return 1;//以0开头只能为0.xxx;以0结尾只能为整数

return end-from+1;//如345,：345,3.45,34.5

}

### 笨重方法

/\*\*

\* the second problem

\* 比较笨重的方法

\* 有趣的变换

\*/

public static void main2(String[] args) {

Scanner scanner = new Scanner(System.in);

String numStr = scanner.nextLine();

char[] nums = numStr.toCharArray();

int count = 0;

int len = nums.length;

for(int i = 0;i < len -1;i++){//从[1,len-1],[2,len-2],...,[len-1,1]

int preCount = 0,sufCount = 0;

for(int point = -1;point < i; point ++) {

if (isLegal(nums, 0, i, point)) preCount ++;

}

for(int point2 = i; point2 < len-1; point2 ++){

if (isLegal(nums, i+1, len-1, point2)) sufCount++;

}

count += preCount\*sufCount;

}

System.out.println(count);

}

/\*\*

\* 判断nums[from]--nums[end]组成的数字是否合法,point表示小数点位置

\* @return

\*/

public static boolean isLegal(char[] nums,int from ,int end,int point){

//一位数情况：肯定返回ture;(此时必有point<from，但是point<from不一定只是一位数)

if(from == end) return true;//只有1位数，肯定返回true

//以下是多位数情况

if(point < from) {//多位整数

return nums[from] != '0'? true :false;

}

//下面是小数情况

if(nums[end] == '0') return false;//如果以0结尾，必定返回false

if(nums[from]=='0'){//如果以0开头，必须0.xxx才可以

if(point != from ) return false;//当前必须是小数点，否则返回false

}

return true;

}

# 多多社交推荐

## 题目要求

2018年8月5日 拼多多笔试算法第3道

时间限制：C/C++ 1秒，其他语言2秒；

空间限制：C/C++ 32768K，其他语言65536K；

64bit IO Format:%IId

题目描述：

给定一个含有N个用户的朋友列表，对于一个指定用户，找出这个用户最可能认识的人。最可能认识的人的定义为这个人和当前用户不是朋友关系，但有最多的共同朋友。

朋友关系是相互的(如果A列出B为朋友，B也会列出A为朋友)，如果两个用户都有同样多的共同朋友，返回用户序号(从0开始)小的用户。如果用户和所有人都没有共同朋友，返回-1。

输入描述：

第一行两个数，第一个数表示用户数目N(N小于等于100)，第二个数为需要判断的用户序号。第2至N+1行表示序号为0到序号为N-1的每个用户的朋友序号列表，每个列表的长度小于100。

输出描述：

给定用户最可能认识的人的用户序号。

示例：

输入：

5 0

1 2 3

0 4

0 4

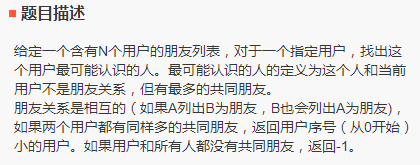
0 4

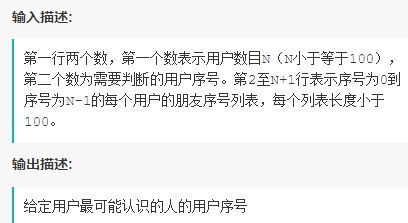
1 2 3

输出：

4

说明：用户0与用户1、2、3都相互认识，用户4与用户1、2、3都相互认识。





## Java代码

# 升序降序取数游戏

## 题目要求

2018年8月5日 拼多多笔试算法第4道

时间限制：C/C++ 1秒，其他语言2秒；

空间限制：C/C++ 32768K，其他语言65536K；

64bit IO Format:%IId

题目描述：

多多鸡正在玩一个取卡片的游戏，有n个标有正整数的卡片，从左到右一次排列，每轮取卡多多鸡必须满足升序规则和降序规则中的一种：

升序规则：取出的右边卡片数值大于左边卡片数值；

降序规则：取出的右边卡片数值小于左边卡片数值；

帮多多鸡算算最少需要多少轮游戏可以取完所有的卡片。

输入描述：

输入为两行，第一行为卡片个数n(1<= n<=50)，第二行为长度为n的正整数序列。

输出描述：

一个整数，代表最少的轮次。

示例1：

输入：

5

3 5 2 4 1

输出：

2

说明：

第一轮321，第二轮54

示例2：

输入：

6

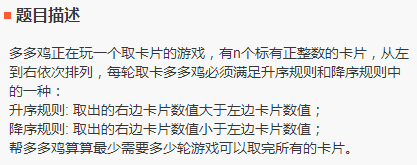
1 2 4 3 3 3

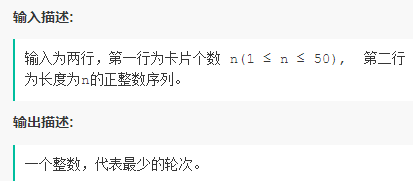
输出：

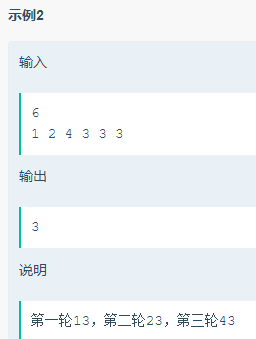
3

说明：

第一轮13，第二轮23，第三轮43





## Java代码