数据结构与算法题解（3）：字符串题解

字符串相关题解java实现

一、替换空格（剑4）

请实现一个函数，把字符串中的每个空格替换成"%20"。例如输入“We are happy.”，则输出“We%20are%20happy”

网络编程中，要把特殊符号转换成服务器可识别的字符。转换的规则是在“%”后面跟上ASCII码的两位十六进制的表示。比如空格的ASCII码是32，即十六进制的0X20，因此空格被替换成“%20”。

问题1：替换字符串，是在原来的字符串上做替换，还是新开辟一个字符串做替换！
问题2：在当前字符串替换，怎么替换才更有效率（不考虑java里现有的replace方法）。从前往后替换，后面的字符要不断往后移动，要多次移动，所以效率低下；从后往前，先计算需要多少空间，然后从后往前移动，则每个字符只为移动一次，这样效率更高一点。

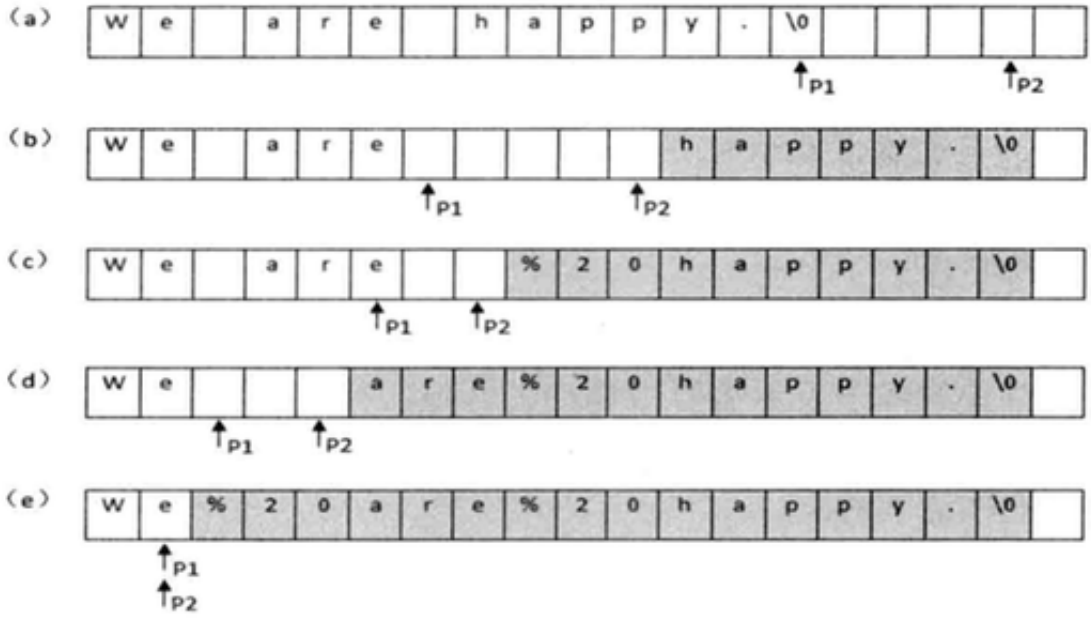


图 2.4　从后往前把字符串中的空格替换成“%20”的过程

```
public class Solution {
    public String replaceSpace(StringBuffer str) {
        int spacenum = 0;//spacenum为计算空格数
        for(int i=0;i<str.length();i++){
            if(str.charAt(i)==' ')
```

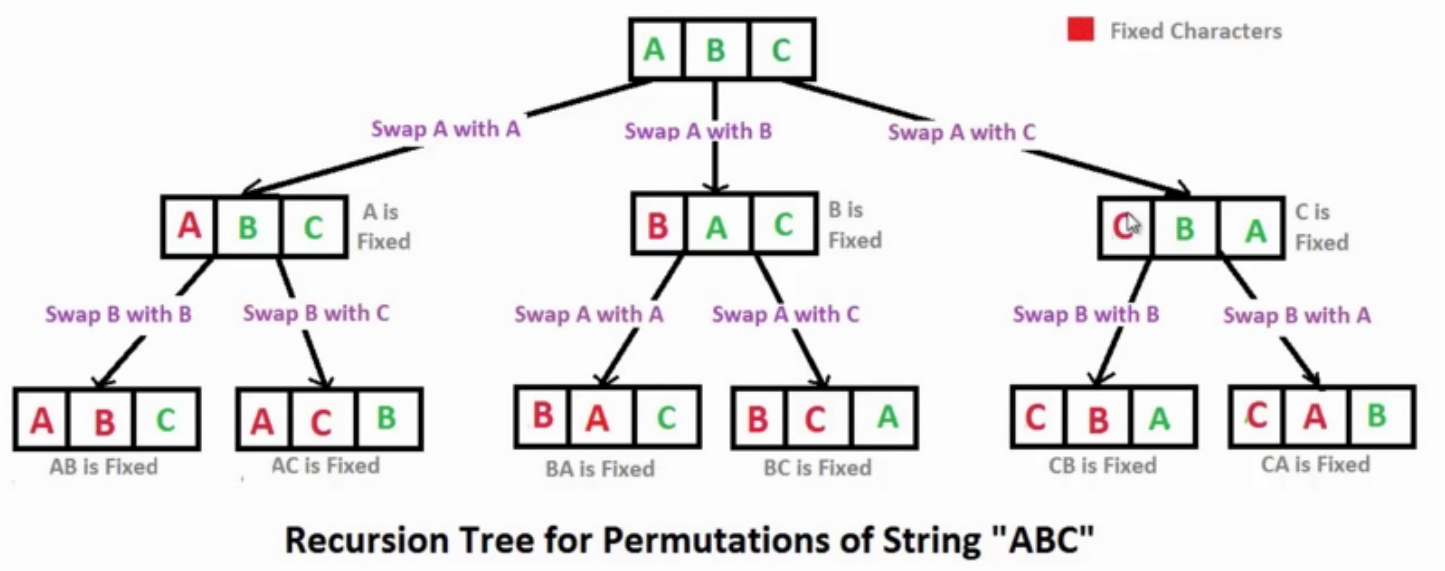
```
                spacenum++;
        }
        int indexold = str.length()-1;//indexold为为替换前的str下标
        int newlength = str.length()+2*spacenum;//计算空格转换成%20之后的str长度
        int indexnew = newlength-1;//indexold为为把空格替换为%20后的str下标
        str.setLength(newlength);//使str的长度扩大到转换成%20之后的长度,防止下标越界,setLength方法
        for(;indexold>=0&&indexold<newlength;--indexold){
            if(str.charAt(indexold)==' '){//charAt方法
                str.setCharAt(indexnew--,'0');
                str.setCharAt(indexnew--,'2');
                str.setCharAt(indexnew--,'%');
            }
            else{
                str.setCharAt(indexnew--,str.charAt(indexold));
            }
        }
        return str.toString();
    }
}
```

二、字符串的排列（剑28）

输入一个字符串,按字典序打印出该字符串中字符的所有排列。例如输入字符串abc,则打印出由字符a,b,c所能排列出来的所有字符串abc,acb,bac,bca,cab和cba。

首先我要打印abc的全排列，就是第一步把a 和bc交换（得到bac,cab），这需要一个for循环，循环里面有一个swap，交换之后就相当于不管第一步了，进入下一步递归，所以跟一个递归函数， 完成递归之后把交换的换回来，变成原来的字串

```
abc 为例子:
1. 固定a，求后面bc的全排列： abc，acb。 求完后，a 和 b交换； 得到bac,开始第二轮
2. 固定b，求后面ac的全排列： bac，bca。 求完后，b 和 c交换； 得到cab,开始第三轮
3. 固定c，求后面ba的全排列： cab，cba
 即递归树:
         str:     a            b             c
               ab ac         ba bc          ca cb
       result:     abc acb       bac bca          cab cba
```



Recursion Tree for Permutations of String "ABC"

> java

```java
import java.util.ArrayList;
import java.util.*;
public class Solution {
    public ArrayList<String> Permutation(String str) {
        ArrayList<String> list = new ArrayList<>();
        if(str.length()==0)
            return list;
        char[] array = str.toCharArray();
        permutation(array,0,list);
        Collections.sort(list);
        return list;
    }
    public void permutation(char[] array,int begin,ArrayList<String> list) {
        if(begin == array.length-1) {
            list.add(String.valueOf(array));
        }else {
            for(int i=begin;i<array.length;++i) {
                if(i==begin || array[i]!=array[begin]) {
                    swap(array,begin,i);
                    permutation(array,begin+1,list);
                    swap(array,begin,i);
                }
            }
        }
    }
    public void swap(char[] array,int i,int j) {
        char temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
}
```

三、第一次只出现一次的字符（字符串）

在一个字符串(1<=字符串长度<=10000，全部由字母组成)中找到第一个只出现一次的字符,并返回它的位置.

我们可以使用一个容器来存放每个字符的出现次数。在这个数据容器中可以根据字符来查找出现的次数，也就是这个容器的作用是把一个字符映射成一个数字。在常用的数据容器中，哈希表正是这个用途。

为了解决这个问题，我们可以定义哈希表的键值（Key）是字符，而值（Value）是该字符出现的次数。同时我们还需要从头开始扫描字符串两次。第一次扫面字符串时，每扫到一个字符就在哈希表的对应项把次数加1.接下来第二次扫描时，每扫描到一个字符就能在哈希表中得到该字符出现的次数，这样第一个只出现一次的字符就是符合要求的输出。

需要涉及到Java中HashMap工作原理及实现，资料链接

> java

```
import java.util.HashMap;
public class Solution {
    public int FirstNotRepeatingChar(String str) {
        HashMap<Character,Integer> map = new HashMap<Character,Integer>();
        for(int i=0;i<str.length();i++){
            char c = str.charAt(i);//charAt方法，获得位置i的串
                if(map.containsKey(c)){//HashMap的containKey方法;
                    int time = map.get(c);//HashMap的get方法，得到Key c的Value;
                    time++;
                    map.put(c,time);//HashMap的put方法，将Key c的Value置为time;
                }else{
                    map.put(c,1);
                }
        }
        for(int i=0;i<str.length();i++){
            char c = str.charAt(i);
            if(map.get(c)==1){
                return i;
            }
        }
        return -1;
    }
}
```

四、翻转单词顺序（剑42.1）

输入一个英文句子，翻转句子中单词的顺序，但单词内字符的顺序不变。为简单起见，标点符号和普通字母一样处理。例如输入字符串“I am a student.”，则输出“student. a am I”。
可以先翻转整个句子，然后，依次翻转每个单词。依据空格来确定单词的起始和终止位置

> java

```
public class Solution {
    public String ReverseSentence(String str) {
        char[] chars = str.toCharArray();
        reverse(chars,0,chars.length-1);
        int blank = -1;
        for(int i =0;i<chars.length-1;i++){
            if(chars[i]==' '){
                int nextblank = i;
                reverse(chars,blank+1,nextblank-1);
                blank = nextblank;
            }
        }
        reverse(chars,blank+1,chars.length-1);//单独翻转最后一个单词
        return new String(chars);

    }
    public void reverse(char[] chars,int low,int high){
        while(low<high){
            char temp = chars[low];
            chars[low]=chars[high];
            chars[high]=temp;
            low++;
            high--;
        }
    }
}
```

五、左旋转字符串（剑42.2）

汇编语言中有一种移位指令叫做循环左移（ROL），现在有个简单的任务，就是用字符串模拟这个指令的运算结果。对于一个给定的字符序列S，请你把其循环左移K位后的序列输出。例如，字符序列S=”abcXYZdef”,要求输出循环左移3位后的结果，即“XYZdefabc”。

以“abcdefg”为例，我们可以把它分为两部分。由于想把它的前两个字符移到后面，我们就把钱两个字符分到第一部分，把后面的所有字符都分到第二部分。然后先翻转这两部分，于是就得

到“bagfedc”。接下来在翻转整个字符串，得到的"cdefgab"刚好就是把原始字符串左旋转2位的结果。

```
public class Solution {
    public String LeftRotateString(String str,int n) {
        char[] chars = str.toCharArray();
        if(chars.length < n) return "";
        reverse(chars, 0, n-1);
        reverse(chars, n, chars.length-1);
        reverse(chars, 0, chars.length-1);
return new String(chars);
    }

    public void reverse(char[] chars,int low,int high){
        char temp;
        while(low<high){
            temp = chars[low];
            chars[low] = chars[high];
            chars[high] = temp;
            low++;
            high--;
        }
    }
}
```

六、 把字符串转换成整数（剑49）

将一个字符串转换成一个整数，要求不能使用字符串转换整数的库函数。 数值为0或者字符串不是一个合法的数值则返回0。
问题不难，但是要把很多特殊情况都考虑进去，却并不容易。需要考虑的特殊情况有以下几个：

1. 空指针null
2. 字符串为空
3. 正负号
4. 上下溢出 Integer.MAX_VALUE (2^{31-1}) Integer.MIN_VALUE(-2^{31})

> java

```
```

```
public class Solution {
    public int StrToInt(String str) {
        if(str==null||str.length()==0){return 0;}//空指针或空字符串
        char[] c = str.toCharArray();
        boolean minus=false;
        int i=0;
        //正负号
        if(c[i]=='+'){
            i++;
        }else if(c[i]=='-'){
            i++;
            minus=true;
        }
        int num=0;
        if(i<c.length){
            num = StrToIntCore(c,minus,i);
        }else{
            return num;
        }
        return num;

    }
    int StrToIntCore(char[] str,boolean minus,int i){
        int num=0;
        for(int j=i;j<str.length;j++){
            if(str[j]>='0'&&str[j]<='9'){
                int flag = minus?-1:1;
                num = num*10+flag*(str[j]-'0');
                if((!minus&&num>Integer.MAX_VALUE)||minus&&num<Integer.MIN_VALUE){/
/上下溢出
                    num=0;
                    break;
                }
            }else{//非法数值
                num=0;
                break;
            }

        }
        return num;
    }
}
```

七、正则表达式匹配（剑53）

当模式中的第二个字符不是“*”时：

1. 如果字符串第一个字符和模式中的第一个字符相匹配，那么字符串和模式都后移一个字符，然后匹配剩余的。
2. 如果字符串第一个字符和模式中的第一个字符相不匹配，直接返回false。

而当模式中的第二个字符是“*”时：

如果字符串第一个字符跟模式第一个字符不匹配，则模式后移2个字符，继续匹配。如果字符串第一个字符跟模式第一个字符匹配，可以有3种匹配方式：

1. 模式后移2字符，相当于$x*$被忽略；
2. 字符串后移1字符，模式后移2字符；
3. 字符串后移1字符，模式不变，即继续匹配字符下一位，因为*可以匹配多位；

> java

```
public class Solution {
    public boolean match(char[] str, char[] pattern) {
    if (str == null || pattern == null) {
        return false;
    }
    int strIndex = 0;
    int patternIndex = 0;
    return matchCore(str, strIndex, pattern, patternIndex);
}

public boolean matchCore(char[] str, int strIndex, char[] pattern, int patternIndex
) {
    //有效性检验：str到尾，pattern到尾，匹配成功
    if (strIndex == str.length && patternIndex == pattern.length) {
        return true;
    }
    //pattern先到尾，匹配失败
    if (strIndex != str.length && patternIndex == pattern.length) {
        return false;
    }
    //模式第2个是*，且字符串第1个跟模式第1个匹配,分3种匹配模式；如不匹配，模式后移2位
    if (patternIndex + 1 < pattern.length && pattern[patternIndex + 1] == '*') {
        if ((strIndex != str.length && pattern[patternIndex] == str[strIndex]) || (
pattern[patternIndex] == '.' && strIndex != str.length)) {
            return matchCore(str, strIndex, pattern, patternIndex + 2)//模式后移2,
视为x*匹配0个字符
                    || matchCore(str, strIndex + 1, pattern, patternIndex + 2)//视为
模式匹配1个字符
                    || matchCore(str, strIndex + 1, pattern, patternIndex);//*匹配1
个，再匹配str中的下一个
        } else {
```

```
            return matchCore(str, strIndex, pattern, patternIndex + 2);
        }
    }
    //模式第2个不是*，且字符串第1个跟模式第1个匹配，则都后移1位，否则直接返回false
    if ((strIndex != str.length && pattern[patternIndex] == str[strIndex]) || (patt
ern[patternIndex] == '.' && strIndex != str.length)) {
        return matchCore(str, strIndex + 1, pattern, patternIndex + 1);
    }
    return false;
    }
}
```

八、表示数值的字符串（剑54）

请实现一个函数用来判断字符串是否表示数值（包括整数和小数）。例如，字符串"+100","5e2","-123","3.1416"和"-1E-16"都表示数值。 但是"12e","1a3.14","1.2.3","+-5"和"12e+4.3"都不是。

> java

```
public class Solution {
    boolean isNumeric(char[] s) {
        if(s.length==0) return false;
        if((s.length==1)&&(s[0]<'0'||s[0]>'9')) return false;
        if(s[0]=='+'||s[0]=='-'){
            if(s.length==2&&(s[1]=='.')) return false;
        }else if((s[0]<'0'||s[0]>'9')&&s[0]!='.') return false;//首位既不是符号也不是
数字还不是小数点，当然是false
        int i = 1;
        while((i<s.length)&&(s[i]>='0'&&s[i]<='9')) i++;
        if(i<s.length&&s[i]=='.'){
            i++;
            //if(i>=s.length) return false;
            while((i<s.length)&&(s[i]>='0'&&s[i]<='9')) i++;
        }
        if(i<s.length&&(s[i]=='e'||s[i]=='E')){
            i++;
            if((i<s.length)&&(s[i]=='+'||s[i]=='-')){
                i++;
                if(i<s.length) while((i<s.length)&&(s[i]>='0'&&s[i]<='9')) i++;
                else return false;
            }else if(i<s.length){
                while((i<s.length)&&(s[i]>='0'&&s[i]<='9')) i++;
            }else return false;
        }
```

```
        if(i<s.length) return false;
        return true;
    }
}
```

九、字符流中第一个不重复的数组（剑55）

使用一个HashMap来统计字符出现的次数，同时用一个ArrayList来记录输入流，每次返回第一个出现一次的字符都是在这个ArrayList（输入流）中的字符作为key去map中查找。

> java

```
import java.util.*;
public class Solution {
    //HashMap来统计字符出现的次数
    HashMap<Character, Integer> map=new HashMap();
    //ArrayList来记录输入流
    ArrayList<Character> list=new ArrayList<Character>();
    //Insert one char from stringstream
    public void Insert(char ch)
    {
        if(map.containsKey(ch)){
            int time = map.get(ch);
            time++;
            map.put(ch,time);
        }else{
            map.put(ch,1);
        }

        list.add(ch);
    }

    //return the first appearence once char in current stringstream
    public char FirstAppearingOnce()
    {   char ch='#';
        for(char k : list){//list迭代
            if(map.get(k)==1){
                ch=k;
                break;//得到第一个结果即可break
            }
        }

        return ch;
    }
}
```

十、最长无重复字符子串

给定一个字符串，找字符中的最大非重复子串 。

基本思路是维护一个窗口，每次关注窗口中的字符串，在每次判断中，左窗口和右窗口选择其一向前移动。同样是维护一个HashSet, 正常情况下移动右窗口，如果没有出现重复则继续移动右窗口，如果发现重复字符，则说明当前窗口中的串已经不满足要求，继续移动有窗口不可能得到更好的结果，此时移动左窗口，直到不再有重复字符为止，中间跳过的这些串中不会有更好的结果，因为他们不是重复就是更短。因为左窗口和右窗口都只向前，所以两个窗口都对每个元素访问不超过一遍，因此时间复杂度为O(2*n)=O(n),是线性算法。空间复杂度为HashSet的size,也是O(n). 用start记录当前处理的开始位置历遍字符串，当当前字符从开始位置start开始已经出现过的时候，子串开始位置+1，否则更新map中的hash值为当前位置 。代码如下：

> java

```
import java.util.HashMap;
public class Solution {
       public int lengthOfLongestSubstring(String s) {
        if (s.length()==0) return 0;
        HashMap<Character, Integer> map = new HashMap<Character, Integer>();
        int max=0;
        int lens=s.length();
        for (int i=0, start=0; i<lens; ++i){
            char ch = s.charAt(i);
            if (map.containsKey(ch)){
                start = Math.max(start,map.get(ch)+1);
            }
            map.put(ch,i);
            max = Math.max(max,i-start+1);
        }
        return max;
    }
}
```

十一、最长回文字符串

已整理

十二、KMP算法

已整理

面试出现的字符串题

已整理

- 三、寻找字符串中第一个只出现一次的字符；寻找一个字符串中第一个只出现一次的字符
- 五、左旋转字符串；手写算法:字符串反转；翻转一个英文字符串中的单词位置，单词间以空格分隔，但不改变每个单词本身的顺序。如输入“Ha Mo”，输出“Mo Ha”；字符串反转；请实现一个函数将“I am a student”转为“student a am I”。
- 六、实现atoi函数，即字符串转整型；就是那个字符串转换成整数，题目不难，但考虑的细节特别多。。。没写出来；写程序 str2Int
- 七、写 find 函数，在目标串中匹配模式串（要考虑中文字符的情况）。
- 十一、最长回文子串：判断一个数字是否为回文数（此处需注意，面试官一直问我有没有更优的方法，我当时已经说出了2-3个方法，囧），
- 十二、KMP算法

未整理

字符串由大小写字母组成，要求去重，只允许使用几个int临时变量，要求时间复杂度尽可能少。
左右括号组成的字符串，去除最少使得剩余的字符串是合法的
统计一个字符串中英文字母、空格、数字的个数，考察代码风格是否规范
然后要求手写纯C字符串拼接，当时笔者想到了三个细节（1：const char* str 2: 空串判断 3：返回新串还是原有串），写完代码之后面试就结束了。在出门的那一刹那，我想起了代码中一个问题，空间申请啊，内心是崩溃的。。。
字符串分割
字符串排序
字符串中字符替换
两个字符串的复制（除了字符串地址重叠的情况，也要注意判断字符串本身的空间足够不足够，对于异常情况要考虑全面）
写code去除字符串S1中的字符使得最终的字符串S2不包含’ab’和’c’