> 授权转载!
>
> - 本文作者：wwwxmu
> - 原文地址:https://www.weiweiblog.cn/13string/

考虑到篇幅问题，我会分两次更新这个内容。本篇文章只是原文的一部分，我在原文的基础上增加了部分内容以及修改了部分代码和注释。另外，我增加了爱奇艺 2018 秋招 Java：求给定合法括号序列的深度 这道题。所有代码均编译成功，并带有注释，欢迎各位享用!

1. KMP 算法

谈到字符串问题，不得不提的就是 KMP 算法，它是用来解决字符串查找的问题，可以在一个字符串（S）中查找一个子串（W）出现的位置。KMP 算法把字符匹配的时间复杂度缩小到 O(m+n) ,而空间复杂度也只有 O(m)。因为"暴力搜索"的方法会反复回溯主串，导致效率低下，而KMP算法可以利用已经部分匹配这个有效信息，保持主串上的指针不回溯，通过修改子串的指针，让模式串尽量地移动到有效的位置。

具体算法细节请参考：

- **字符串匹配的KMP算法:** http://www.ruanyifeng.com/blog/2013/05/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm.html
- **从头到尾彻底理解KMP:** https://blog.csdn.net/v_july_v/article/details/7041827
- **如何更好的理解和掌握 KMP 算法?:** https://www.zhihu.com/question/21923021
- **KMP 算法详细解析:** https://blog.sengxian.com/algorithms/kmp
- **图解 KMP 算法:** http://blog.jobbole.com/76611/
- **汪都能听懂的KMP字符串匹配算法【双语字幕】:** https://www.bilibili.com/video/av3246487/?from=search&seid=17173603269940723925
- **KMP字符串匹配算法1:** https://www.bilibili.com/video/av11866460?from=search&seid=12730654434238709250

除此之外，再来了解一下BM算法!

> BM算法也是一种精确字符串匹配算法，它采用从右向左比较的方法，同时应用到了两种启发式规则，即坏字符规则 和好后缀规则 ，来决定向右跳跃的距离。基本思路就是从右往左进行字符匹配，遇到不匹配的字符后从坏字符表和好后缀表找一个最大的右移值，将模式串右移继续匹配。《字符串匹配的KMP算
> 法》:http://www.ruanyifeng.com/blog/2013/05/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm.html

2. 替换空格

> 剑指offer：请实现一个函数，将一个字符串中的每个空格替换成“%20”。例如，当字符串为We Are Happy.则经过替换之后的字符串为We%20Are%20Happy。

这里我提供了两种方法：①常规方法；②利用 API 解决。

```
//https://www.weiweiblog.cn/replacespace/
public class Solution {

  /**
   * 第一种方法：常规方法。利用String.charAt(i)以及String.valueOf(char).equals(" "
   * )遍历字符串并判断元素是否为空格。是则替换为"%20",否则不替换
   */
  public static String replaceSpace(StringBuffer str) {

    int length = str.length();
    // System.out.println("length=" + length);
    StringBuffer result = new StringBuffer();
    for (int i = 0; i < length; i++) {
      char b = str.charAt(i);
      if (String.valueOf(b).equals(" ")) {
        result.append("%20");
      } else {
        result.append(b);
      }
    }
    return result.toString();

  }

  /**
   * 第二种方法：利用API替换掉所用空格，一行代码解决问题
   */
  public static String replaceSpace2(StringBuffer str) {

    return str.toString().replaceAll("\\s", "%20");
  }
}
```

3. 最长公共前缀

> Leetcode: 编写一个函数来查找字符串数组中的最长公共前缀。如果不存在公共前缀，返回空字符串 ""。

示例 1:

```
输入: ["flower","flow","flight"]
输出: "fl"
```

示例 2:

```
输入: ["dog","racecar","car"]
输出: ""
解释: 输入不存在公共前缀。
```

思路很简单！先利用Arrays.sort(strs)为数组排序，再将数组第一个元素和最后一个元素的字符从前往后对比即可！

```
public class Main {
    public static String replaceSpace(String[] strs) {

        // 如果检查值不合法及就返回空串
        if (!checkStrs(strs)) {
            return "";
        }
        // 数组长度
        int len = strs.length;
        // 用于保存结果
        StringBuilder res = new StringBuilder();
        // 给字符串数组的元素按照升序排序(包含数字的话，数字会排在前面)
        Arrays.sort(strs);
        int m = strs[0].length();
        int n = strs[len - 1].length();
        int num = Math.min(m, n);
        for (int i = 0; i < num; i++) {
            if (strs[0].charAt(i) == strs[len - 1].charAt(i)) {
                res.append(strs[0].charAt(i));
            } else
                break;

        }
        return res.toString();

    }

    private static boolean chechStrs(String[] strs) {
        boolean flag = false;
        if (strs != null) {
            // 遍历strs检查元素值
            for (int i = 0; i < strs.length; i++) {
                if (strs[i] != null && strs[i].length() != 0) {
                    flag = true;
                } else {
                    flag = false;
                    break;
                }
            }
        }
```

```
            return flag;
        }

        // 测试
        public static void main(String[] args) {
            String[] strs = { "customer", "car", "cat" };
            // String[] strs = { "customer", "car", null };//空串
            // String[] strs = {};//空串
            // String[] strs = null;//空串
            System.out.println(Main.replaceSpace(strs));// c
        }
    }
```

4. 回文串

4.1. 最长回文串

> LeetCode: 给定一个包含大写字母和小写字母的字符串，找到通过这些字母构造成的最长的回文串。在构造过程中，请注意区分大小写。比如"Aa"不能当做一个回文字符串。注 意:假设字符串的长度不会超过 1010。

> 回文串：“回文串”是一个正读和反读都一样的字符串，比如“level”或者“noon”等等就是回文串。——百度百科 地址：https://baike.baidu.com/item/%E5%9B%9E%E6%96%87%E4%B8%B2/1274921?fr=aladdin

示例 1:

```
输入:
"abccccdd"

输出:
7

解释:
我们可以构造的最长的回文串是"dccaccd", 它的长度是 7。
```

我们上面已经知道了什么是回文串？现在我们考虑一下可以构成回文串的两种情况：

- 字符出现次数为双数的组合
- **字符出现次数为偶数的组合+单个字符中出现次数最多且为奇数次的字符** （参见 **issue665** ）

统计字符出现的次数即可，双数才能构成回文。因为允许中间一个数单独出现，比如“abcba”，所以如果最后有字母落单，总长度可以加 1。首先将字符串转变为字符数组。然后遍历该数组，判断对应字符是否在hashset中，如果不在就加进去，如果在就让count++，然后移除该字符！这样就能找到出现次数为双数的字符个数。

```
//https://leetcode-cn.com/problems/longest-palindrome/description/
class Solution {
```

```
  public  int longestPalindrome(String s) {
    if (s.length() == 0)
      return 0;
    // 用于存放字符
    HashSet<Character> hashset = new HashSet<Character>();
    char[] chars = s.toCharArray();
    int count = 0;
    for (int i = 0; i < chars.length; i++) {
      if (!hashset.contains(chars[i])) {// 如果hashset没有该字符就保存进去
        hashset.add(chars[i]);
      } else {// 如果有,就让count++（说明找到了一个成对的字符），然后把该字符移除
        hashset.remove(chars[i]);
        count++;
      }
    }
    return hashset.isEmpty() ? count * 2 : count * 2 + 1;
  }
}
```

4.2. 验证回文串

> LeetCode: 给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。
> 说明：本题中，我们将空字符串定义为有效的回文串。

示例 1:

```
输入: "A man, a plan, a canal: Panama"
输出: true
```

示例 2:

```
输入: "race a car"
输出: false
```

```
//https://leetcode-cn.com/problems/valid-palindrome/description/
class Solution {
  public  boolean isPalindrome(String s) {
    if (s.length() == 0)
      return true;
    int l = 0, r = s.length() - 1;
    while (l < r) {
      // 从头和尾开始向中间遍历
      if (!Character.isLetterOrDigit(s.charAt(l))) {// 字符不是字母和数字的情况
        l++;
      } else if (!Character.isLetterOrDigit(s.charAt(r))) {// 字符不是字母和数字的情
况
        r--;
```

```
      } else {
        // 判断二者是否相等
        if (Character.toLowerCase(s.charAt(l)) !=
Character.toLowerCase(s.charAt(r)))
          return false;
        l++;
        r--;
      }
    }
    return true;
  }
}
```

4.3. 最长回文子串

> Leetcode: LeetCode: 最长回文子串 给定一个字符串 s，找到 s 中最长的回文子串。你可以假设 s 的最大长度为1000。

示例 1:

```
输入: "babad"
输出: "bab"
注意: "aba"也是一个有效答案。
```

示例 2:

```
输入: "cbbd"
输出: "bb"
```

以某个元素为中心，分别计算偶数长度的回文最大长度和奇数长度的回文最大长度。给大家大致花了个草图，不要嫌弃！

```
//https://leetcode-cn.com/problems/longest-palindromic-substring/description/
class Solution {
  private int index, len;

  public String longestPalindrome(String s) {
    if (s.length() < 2)
      return s;
    for (int i = 0; i < s.length() - 1; i++) {
      PalindromeHelper(s, i, i);
      PalindromeHelper(s, i, i + 1);
    }
    return s.substring(index, index + len);
  }
```

```
  public void PalindromeHelper(String s, int l, int r) {
    while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {
      l--;
      r++;
    }
    if (len < r - l - 1) {
      index = l + 1;
      len = r - l - 1;
    }
  }
}
```

4.4. 最长回文子序列

> LeetCode: 最长回文子序列 给定一个字符串s，找到其中最长的回文子序列。可以假设s的最大长度为1000。 **最长回文子序列和上一题最长回文子串的区别是，子串是字符串中连续的一个序列，而子序列是字符串中保持相对位置的字符序列，例如，"bbbb"可以是字符串"bbbab"的子序列但不是子串。**

给定一个字符串s，找到其中最长的回文子序列。可以假设s的最大长度为1000。

示例 1:

```
输入:
"bbbab"
输出:
4
```

一个可能的最长回文子序列为 "bbbb"。

示例 2:

```
输入:
"cbbd"
输出:
2
```

一个可能的最长回文子序列为 "bb"。

动态规划： dp[i][j] = dp[i+1][j-1] + 2 if s.charAt(i) == s.charAt(j) otherwise, dp[i][j] = Math.max(dp[i+1][j], dp[i][j-1])

```
class Solution {
    public int longestPalindromeSubseq(String s) {
        int len = s.length();
        int [][] dp = new int[len][len];
        for(int i = len - 1; i>=0; i--){
```

```
                dp[i][i] = 1;
                for(int j = i+1; j < len; j++){
                    if(s.charAt(i) == s.charAt(j))
                        dp[i][j] = dp[i+1][j-1] + 2;
                    else
                        dp[i][j] = Math.max(dp[i+1][j], dp[i][j-1]);
                }
            }
            return dp[0][len-1];
        }
    }
```

5. 括号匹配深度

> 爱奇艺 2018 秋招 Java： 一个合法的括号匹配序列有以下定义:
>
> 1. 空串""是一个合法的括号匹配序列
> 2. 如果"X"和"Y"都是合法的括号匹配序列,"XY"也是一个合法的括号匹配序列
> 3. 如果"X"是一个合法的括号匹配序列,那么"(X)"也是一个合法的括号匹配序列
> 4. 每个合法的括号序列都可以由以上规则生成。

> 例如: "","()","()()","((()))"都是合法的括号序列 对于一个合法的括号序列我们又有以下定义它的深度:
>
> 1. 空串""的深度是0
> 2. 如果字符串"X"的深度是x,字符串"Y"的深度是y,那么字符串"XY"的深度为max(x,y)
> 3. 如果"X"的深度是x,那么字符串"(X)"的深度是x+1

> 例如: "()()()"的深度是1,"((()))"的深度是3。牛牛现在给你一个合法的括号序列,需要你计算出其深度。

```
输入描述:
输入包括一个合法的括号序列s,s长度length(2 ≤ length ≤ 50),序列中只包含'('和')'。

输出描述:
输出一个正整数,即这个序列的深度。
```

示例:

```
输入:
(())
输出:
2
```

思路草图:

代码如下:

```
import java.util.Scanner;

/**
 * https://www.nowcoder.com/test/8246651/summary
 *
 * @author Snailclimb
 * @date 2018年9月6日
 * @Description: TODO 求给定合法括号序列的深度
 */
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String s = sc.nextLine();
    int cnt = 0, max = 0, i;
    for (i = 0; i < s.length(); ++i) {
      if (s.charAt(i) == '(')
        cnt++;
      else
        cnt--;
      max = Math.max(max, cnt);
    }
    sc.close();
    System.out.println(max);
  }
}
```

6. 把字符串转换成整数

> 剑指offer: 将一个字符串转换成一个整数(实现Integer.valueOf(string)的功能，但是string不符合数字要求时返回0)，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。

```
//https://www.weiweiblog.cn/strtoint/
public class Main {

  public static int StrToInt(String str) {
    if (str.length() == 0)
      return 0;
    char[] chars = str.toCharArray();
    // 判断是否存在符号位
    int flag = 0;
    if (chars[0] == '+')
      flag = 1;
    else if (chars[0] == '-')
      flag = 2;
    int start = flag > 0 ? 1 : 0;
    int res = 0;// 保存结果
    for (int i = start; i < chars.length; i++) {
      if (Character.isDigit(chars[i])) {// 调用Character.isDigit(char)方法判断是否是数字，是返回True，否则False
```

```
          int temp = chars[i] - '0';
          res = res * 10 + temp;
        } else {
          return 0;
        }
      }
     return flag != 2 ? res : -res;

    }

    public static void main(String[] args) {
      // TODO Auto-generated method stub
      String s = "-12312312";
      System.out.println("使用库函数转换: " + Integer.valueOf(s));
      int res = Main.StrToInt(s);
      System.out.println("使用自己写的方法转换: " + res);

    }

  }
```