1，字符串

2018年4月25日 19:03

- 字符串简介
- 面试题总体分析
- 一些例题
- 例1 0-1串交换排序
- 例2 字符的替换和复制
- 例3 交换星号
- 例4 子串变位词
- 例5 单词（字符串）翻转
- 总结
- 1，字符串(String)简介
  - 通常把它作为字符数组
  - java : String内置类型，不可更改，要更改的话可考虑转StringBuffer, StringBuilder, char []之类
  - C++ : std::string可更改，也可以考虑用char[] (char*)
  - C： 只有char[]
  - 注意
    - C++中“+”运算符，复杂度未定义，但通常认为是线性的，（因此，在for循环中若有str的+，时间复杂度可能退化成n^2,比如在建立一个a-z的字符串的时候，不断使用字符串加法，一个一个加上去的话，时复为26^2,而char数组赋值，时复只有26）
    - C++ std::string substr和java的String的subString参数含义不同，c++为起始位置和长度
    - 字符范围:
      - C/C++ [-128..+127], 我们通常转化为unsigned 变

为[0..+255]（我们可以将字符转化为ascii码，作为数组的下标，从而统计字符数量）

- Java: [0..65535]

2，面试题总体分析

- 和数组相关，内容广泛
  - 概念理解：字典序
  - 简单操作：插入、删除字符，旋转
  - 规则判断（罗马数字转换 是否是合法的整数（atoi）、浮点数）
  - 数字运算（大数加法、二进制加法）
  - 排序、交换（partition过程：把比分区元素小的放到一边，比其大的放到另一边）
  - 字符计数（hash)：变位词（组成字母完全相同的词，比如abc和acb）
  - 匹配（正则表达式、全串匹配、KMP、周期判断）
  - 动态规划(LCS、编辑距离、最长回文子串）
  - 搜索（单词变换、排列组合）
- 3，一些例题
- 例1 0-1串交换排序

```
/*
问题：把一个0-1串（只包含0和1的串）进行排序，你可以交换任意两个位置，问最少交换的次数？

分析：明显，排好序之后0在左边，1在右边，因此左边的0和右边的1都可以不考虑，
      维持两个指针，分别从左到右扫第一个1的位置，和从右往左扫第1个0的位置，交换即可
      这里甚至不需真的交换，只需记录下交换的次数即可
      时复:o(len)
*/
#include<bits/stdc++.h>
using namespace std;


class Solution
{
public:
    int count(string s)
    {


        int c = 0;
```

```
        for(int i = 0,j = s.size()-1;i<j;i++,j--)
        {
            for(;i<j && s[i] =='0';i++);
            for(;i<j && s[j] =='1';j--);
            if(i<j) c++;
        }
        return c;
    }
};


int main()
{
    string s = "010101";//1次
    Solution so;
    cout<<so.count(s)<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

- **例2 字符的替换和复制**

```
/*
问题：删除一个字符串所有的a,并且复制所有的b。注：字符数组足够大

分析：数组足够大，意味着无需新建数组，只需在原数组上做修改即可
    首先，删除a，只需重新维持一个idx（从0开始），不断将非a的元素放入即可，在这个过程中，同时统计b的数目有多少
    然后复制b，通过上一步，我们可以得到新数组的长度应为idx(更新后的)+nums(b),然后从后往前遍历，从新数组的末尾开始放即可

*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    void solve(string &s)
    {
        int n = s.size();
        int idx = 0;
        int numsb = 0;
        for(int i = 0;i<n;i++)//删除a，并统计b的数量
        {

            if(s[i] != 'a') s[idx++] = s[i];
            if(s[i] == 'b') numsb++;
        }
        int new_len = idx+numsb;//新索引
        for(int j = new_len-1,i=idx-1;i>=0;i--)//这里要注意索引都是从0开始的。
```

```
        {
            s[j--] = s[i];
            if(s[i] == 'b') s[j--] = s[i];
        }
    }
};


int main()
{
    string s ="aabbc";
    Solution so;
    so.solve(s);
    cout<<s<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

- **思考题：如何把字符串的空格变成”%20”?同样，字符数组足够大！**

- **例3 交换星号**

```
/*
问题：一个字符串只包含*和数字，请把它的*号都放开头。

分析：既有partition，又有倒着复制的思想
    如果只有partition，那么数字的相对位置会发生变化，这个思想如下：设[0,i-1]都是*,[i,j-1]都是数字，[j,n-1]为未探测
    for(int i = 0,j = 0;j<n;j++)
        if(s[j] == '*') swap(s[i++],s[j]);//j在j到n的区间遍历，如果其是数字，不管，是*,就和i处交换，i同时++。
        i其实是*和数字的分界点
    再加上倒着复制的思想，
    int j = n-1;//新索引
    for(int i = n-1;i>=0;i--)
        if(isdigit(s[i])) s[j--] = s[i];//倒着来，如果是数字，不管，直接放即可。这样结束，后面全是数字了
    //下面再将前面全赋值为*即可
    for(;j>=0;j--)
        s[j] = '*';
*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    void solve(string &s)
    {
        int n = s.size();
        int j = n-1;
        for(int i = n-1;i>=0;i--)
```

```
            if(isdigit(s[i])) s[j--] = s[i];
        for(;j>=0;j--)
            s[j] = '*';
    }
};


int main()
{
    string s = "*3*90*9932**";
    Solution so;
    so.solve(s);
    cout<<s<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

- 例4 子串变位词
- 给定两个串a和b，问b是否是a的子串的变位词。例如输入a = hello, b = lel, lle, ello都是true,但是b = elo是false。（国外某公司最新面试题）

```
/*
问题：给定两个串a和b，问b是否是a的子串的变位词。例如输入a = hello, b = lel, lle, ello都是true,但是b = elo是false。


分析：这里涉及到滑动窗口和词频统计的思想
    定义一个和b长度大小相同的窗口，统计词频，分析和b是否一样，但这样的话，每滑动一次都要比较一次，有没有更好的方法呢？
    暂且假定我们的输入都是小写字母，因此我们定义一个[0,25]的数组表示b中每个单词出现多少次。同时，统计下b中多少个字母是非0的。
    int nonzero = 0;
    for(int i=0;i<lenb;i++)
        if(++nums[b[i]-'a'] == 1) ++nonzero;
    然后，我们用b中的次数，减去a中一个窗口内的字符种类，如果结果都为0,那么找到这样的字串了。注意，这里nums的含义变成了字符种类差
    第一个窗口为[0,lenb-1]，（注意lena<lenb是无解的）
    for(int i =0;i<lenb;i++)
    {
        int c = a[i] - 'a';
        --nums[c];
        if(nums[c] == 0) --nonzero;
        else if(nums[c] == -1) ++nonzero;
    }
 if(nonzero == 0) return true;
*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    bool solve(string a,string b)
    {
```

```
        if(a.size()<b.size()) return false;
        int nonzero = 0;
        int nums[26];
        for(int i = 0;i<b.size();i++)//nums先统计b中词频
        {
            if(++nums[b[i]-'a'] == 1) nonzero++;
        }
        int temp = nonzero;
        //再来逐个和a中的每个窗口比对。这里窗口如何滑动呢？首先原窗口为[i-lenb,i-1],新窗口为[i-lenb+1,i],相当于扔掉最左边的，再加入a[i].
        //这个思想无比精妙，节省了很多时间复杂度
        for(int i = b.size();i<a.size();i++)//i为每个窗口的结束位置。第一个窗口为[0,lenb-1]
        {
            int c = a[i-b.size()] -'a';//原窗口最左边元素,不属于当前窗口了，还原回去，++
            ++nums[c];
            if(nums[c] == 1) nonzero++;
            else if(nums[c] == 0) nonzero--;

            c = a[i]-'a';//新窗口加入的元素
            --nums[c];
            if(nums[c] == 0) nonzero--;
            else if(nums[c] == -1) nonzero++;
            if(nonzero == 0) return true;
        }
        return false;
    }
};

int main()
{
    string a = "hello";
    string b = "elo";
    Solution so;
    cout<<so.solve(a,b)<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

- 思考题 Leetcode 3

- 例5 单词（字符串）翻转

```
/*
问题：翻转句子中全部的单词，单词内容不变，例如I’m a student.  变为student. a I’m

分析：while(i<j) swap(s[i++],s[j--]);这种效果可以翻转整个句子 ： .tneduts a m’I，
再对每个单词单独反转，即可得到结果
```

拓展题：字符串循环移位abcd

如果长度为n，移位m，那么m=m%n；前m位翻转，后n-m位翻转，再整体翻转。
比如这里m=1,那么a，dcb ->bcda
*/

```
#include<bits/stdc++.h>
using namespace std;


class Solution
{
public:
    void reverse(string &s)
    {
        int i = 0,j = s.size()-1;
        rev(s,i,j);

        i = 0;//i,j为一个单词的开始和结束位置
        for(int k = 0;k<s.size();k++)
        {
            if(s[k] == ' ')
            {
                j = k-1;
                rev(s,i,j);
                i = k+1;


            }


        }
        rev(s,i,s.size()-1);//反转最后一个单词
    }


    void rev(string& s,int i,int j)//反转s中从i到j位置的内容
    {
        while(i<j) swap(s[i++],s[j--]);
    }
};


int main()
{
    string s = "I'm a student.";
    Solution so;
    so.reverse(s);
    cout<<s<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

- **总结**

- 我理解的in-place (原地）
  - 本身O(1)空间
  - 递归，堆栈空间可以不考虑
- 原地相关的问题
  - 字符串循环左移、右移动
  - 快排partition相关
- 滑动窗口
  - 能达到O(n)的的时间复杂度
  - O(1)的空间复杂度
- 规则相关——细致
- 匹配（暴力）:KMP比较少见
- Manacher——要求比较高的笔试