算法分析之其他题目

717. 1比特与2比特字符

```
/*
暴力
由哈夫曼编码知识可知这种编码是唯一的(这就是哈夫曼编码中的变长码)
if 是1：删除两个字符
else 是0：删除一个字符
*/
#include <iostream>
#include <vector>
using namespace std;

class Solution {
public:
    bool isOneBitCharacter(vector<int>& bits) {
        int n = bits.size();
        int i = 0;

        //只计到倒数第二个字符处
        //如果倒数第二个字符和最后一个字符0是一体的，最后i会加2等于n
        //如果不是一体的，i就停留在最后一个字符位置上，等于n-1
        while(i<n-1) {
            if(bits[i]==1) i += 2;  //是1跳两个
            else i++; //是0跳一个
        }
        return i==n-1;
    }
};

int main() {
    Solution solu;
    vector<int> bits = {1, 0, 1, 0};
    cout << boolalpha << solu.isOneBitCharacter(bits);
    return 0;
}
```

43. 字符串相乘

```
#include <iostream>
#include <vector>
```

```
using namespace std;

class Solution {
public:
    string multiply(string num1, string num2) {
        int len1 = num1.length();
        int len2 = num2.length();
        vector<int> sum(len1+len2,0);
        for(int i=0; i<len2; i++) {//用第二个数乘以第一个数,每次一位
            for(int j=0; j<len1; j++) {
                sum[i+j] += (num2[len2-1-i]-'0') * (num1[len1-1-j]-'0');
//先乘起来,后面统一进位
            }
        }

        for(int i=0; i<len1+len2-1; i++) {//统一处理进位问题
            if(sum[i]>=10) {
                sum[i+1] += sum[i]/10;
                sum[i]%=10;
            }
        }

        string result;
        int i=len1+len2-1;
        while(i>=0 && sum[i]==0) i--;
        if(i<0) return "0";
        while(i>=0) result+=sum[i--]+'0';
        return result;
    }
};

int main() {
    Solution solu;
    cout<<solu.multiply("123","456");
    return 0;
}
```

71. 简化路径

```
/*
使用stringstream来分隔字符串
*/
#include <iostream>
#include <vector>
#include<sstream>
using namespace std;

class Solution {
public:
```

```
    string simplifyPath(string path) {
        stringstream ss(path);
        string t;
        vector<string> v;
        while(getline(ss, t, '/')) {
            if(t=="" || t==".") continue;
            if(t==".."){
                if(!v.empty()) v.pop_back();
            } else
                v.push_back(t);
        }
        string res;
        for(auto e:v) res += "/"+e;
        return res.empty() ? "/" : res;
    }
};

int main() {
    Solution solu;
    cout<<solu.simplifyPath("/a/./b/../../c/");
    return 0;
}
```

151. 翻转字符串里的单词

```
/*
更巧妙的方法：使用字符串流类stringstream的解法
我们先把字符串装载入字符串流中，然后定义一个临时变量tmp，然后把第一个单词赋给s，
如果含有非空格字符，那么每次>>操作就会提取连在一起的非空格字符，那么我们每次将其加在s前面即可；
如果原字符串为空，那么就会被略过；
这里需要注意的是，如果原字符串为许多空格字符连在一起，那么第一个>>操作就会提取出这些空格字符放入s中，所以要加判断。
*/
#include<iostream>
#include<sstream>
#include<string>
using namespace std;

class Solution {
public:
    void reverseWords(string &s) {
        istringstream is(s);
        string tmp;
        is >> s;
        while(is >> tmp) s = tmp + " " + s;
        if(!s.empty() && s[0]==' ') s="";
    }
};
```

```
int main() {
    Solution solu;
    string s="the sky is blue";
    solu.reverseWords(s);
    cout<<s;
    return 0;
}
```

567. 字符串的排列

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
using namespace std;

class Solution {
public:
    bool checkInclusion(string s1, string s2) {
        int len1=s1.length(),len2=s2.length();
        if(len1>len2) return false;

        vector<int> target(26, 0); //记录s1字符串中 a~z 26个字符个数情况
        vector<int> window(26, 0); //从s2中取与s1等大的窗口，也做如此处理
        for(int i=0; i<len1; i++){
            target[s1[i]-'a']++;
            window[s2[i]-'a']++;
        }
        for(int i=len1; i<len2&&target!=window; i++){ //核心 target==windo
w ，判断字符个数情况是否一样
            //滑窗，左边踢出一个，右边加入一个
            window[s2[i-len1]-'a']--;
            window[s2[i]-'a']++;
        }
        return target==window;
    }
};

int main() {
    Solution solu;
    cout<<solu.checkInclusion("adc","dcda");
    return 0;
}
```

无重复字符的最长子串

```
/*思路
    用 map记录字符出现位置
    right 是遍历到当前位置
    left 是满足不重复条件最左位置
    拿 abcbde 来说
    遍历一次整个字符串：
        遇到第一个b时，   m['b'] 记录其位置，即2
        遇到第二个b时，此时left即指向第一个b位置，什么意思呢？
        就是说后面子串有了第二个b，那么第一个b就要抛弃，就是说bcbde不满足，但是cbde
满足
*/
#include<iostream>
#include<string>
#include<map>
#include<algorithm>
using namespace std;

class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        map<char,int> m;
        int result = 0;

        for(int right=0, left=0; right<s.length(); right++) {
            char tmpc = s[right];
            //cout<<m[tmpc];

            left = max(left, m[tmpc]);
            result = max(result, right+1-left);
            m[tmpc] = right+1;
        }
        return result;
    }
};

int main() {
    Solution solu;
    string s="abcacb";
    cout << solu.lengthOfLongestSubstring(s) ;
    return 0;
}
```

多关键字排序和排名

```
#include <algorithm>
```

```
#include <string>
#include <iostream>
#include <cstdio>
#include <iomanip>
using namespace std;

struct student {
    string no;
    string name;
    int toil;
    int sub;
    double cor;
};

bool comp(const student &a,const student &b) {
    if(a.toil>b.toil) return true;
    else if(a.toil<b.toil) return false;

    if(a.cor>b.cor) return true;
    else if(a.cor<b.cor) return false;

    if(a.no<b.no) return true;
    else return false;
}

int main() {
    int n;
    cin>>n;

    struct student stu[n];
    for(int i=0; i<n; i++) {
        cin >> stu[i].no >> stu[i].name >> stu[i].toil >> stu[i].sub;
        stu[i].cor = stu[i].toil*100.0/stu[i].sub*1.0;
    }

    sort(stu, stu+n, comp);

    int rank=1;
    for(int i=0; i<n; i++) {
        if(i>0 && (fabs(stu[i].cor-stu[i-1].cor)>1e-6 || stu[i].toil!=stu
[i-1].toil)) rank=i+1;
        cout<<left<<setw(4)<<rank;
        cout<<left<<setw(12)<<stu[i].no;
        cout<<left<<setw(11)<<stu[i].name;
        cout<<right<<setw(6)<<stu[i].toil;
        cout<<right<<setw(6)<<stu[i].sub;
        cout<<right<<setw(8)<<fixed<<setprecision(2)<<stu[i].cor<<'%'<<en
dl;
    }
    return 0;
}
```