C++方向编程题答案

第四周

day23

> 题目ID：26026-微信红包
>
> 链接：https://www.nowcoder.com/practice/fbcf95ed620f42a88be24eb2cd57ec54?tpId=49&&tqId=29311&rp=1&ru=/activity/oj&qru=/ta/2016test/question-ranking
>
> **【题目解析】：**
>
> 本题题意明确
>
> **【解题思路】：**
>
> 本题两种思路，第一种排序思路，如果一个数出现次数超过一半了，排序过后，必然排在中间，则最后遍历整个数组查看是否符合即可。第二种思路可以用map统计每个数字出现的次数，最后判断有没有超过一半的数字。
>
> **【示例代码】**

```
class Gift {
public:
    int getValue(vector<int> gifts, int n) {
        sort(gifts.begin(), gifts.end());
        //超过一半的数排序之后必然排在中间
        int middle = gifts[n / 2];
        int count = 0;

        for(int i = 0; i < n; i++)
        {
            //统计排在中间的数的个数
            if(gifts[i] == middle)
            {
                count++;
            }
        }
        //如果个数大于一半，则存在超过一半的数
        if(count > n / 2)
            return middle;
        else
            return 0;
        }
};

/*思路二：map统计*/
class Gift {
public:
    int getValue(vector<int> gifts, int n) {

        map<int,int> count;
```

```
        int middle = gifts.size() / 2;
        for(const auto& e : gifts)
        {
            ++count[e];
        }
        for(const auto& e : count)
        {
            if(e.second >= middle)
                return e.first;
        }
        return 0;
    }
};
```

题目ID:36876-计算字符串的距离

链接：https://www.nowcoder.com/practice/3959837097c7413a961a135d7104c314?tpId=37&&tqId=21275&rp=1&ru=/activity/oj&qru=/ta/huawei/question-ranking

【题目解析】：

本题题意明确

【解题思路】：

本题需要用动态规划解题 状态： 子状态：word1的前1，2，3，...m个字符转换成word2的前1，2，3，...n个字符需要的编辑距离

F(i,j):word1的前i个字符于word2的前j个字符的编辑距离 状态递推： F(i,j) = min { F(i-1,j) +1, F(i,j-1) +1, F(i-1,j-1) +(w1[i]==w2[j]?0:1) } 上式表示从删除，增加和替换操作中选择一个最小操作数 F(i-1,j): w1[1,...,i-1]于w2[1,...,j]的编辑距离，删除w1[i]的字符--->F(i,j) F(i,j-1): w1[1,...,i]于w2[1,...,j-1]的编辑距离，增加一个字符--->F(i,j) F(i-1,j-1): w1[1,...,i-1]于w2[1,...,j-1]的编辑距离，如果w1[i]与w2[j]相同， 不做任何操作，编辑距离不变，如果w1[i]与w2[j]不同，替换w1[i]的字符为w2[j]--->F(i,j) 初始化： 初始化一定要是确定的值，如果这里不加入空串，初始值无法确定 F(i,0) = i :word与空串的编辑距离，删除操作 F(0,i) = i :空串与word的编辑距离，增加操作 返回结果：F(m,n)

【示例代码】

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

int minDistance(string word1, string word2) {
        // word与空串之间的编辑距离为word的长度
        if (word1.empty() || word2.empty()) {
            return max(word1.size(), word2.size());
        }

        int len1 = word1.size();
        int len2 = word2.size();
        // F(i,j)初始化

        vector<vector<int> >  f(1 + len1, vector<int>(1 + len2, 0));
```

```
        for (int i = 0; i <= len1; ++i) {
            f[i][0] = i;
        }
        for (int i = 0; i <= len2; ++i) {
            f[0][i] = i;
        }

        for (int i = 1; i <= len1; ++i) {
            for (int j = 1; j <= len2; ++j) {
                // F(i,j) = min { F(i-1,j) +1, F(i,j-1) +1, F(i-1,j-1) +
  //(w1[i]==w2[j]?0:1) }
                // 判断word1的第i个字符是否与word2的第j个字符相等
                if (word1[i - 1] == word2[j - 1]) {
                    f[i][j] = 1 + min(f[i][j - 1], f[i - 1][j]);
                    // 字符相等，F(i-1,j-1)编辑距离不变
                    f[i][j] = min(f[i][j], f[i - 1][j - 1]);
                }
                else {
                    f[i][j] = 1 + min(f[i][j - 1], f[i - 1][j]);
                    // 字符不相等，F(i-1,j-1)编辑距离 + 1
                    f[i][j] = min(f[i][j], 1 + f[i - 1][j - 1]);
                }
            }
        }

        return f[len1][len2];
    }

int main(){
 string a,b;
 while(cin>>a>>b)
     cout<<minDistance(a, b)<<endl;
 return 0;
}
```