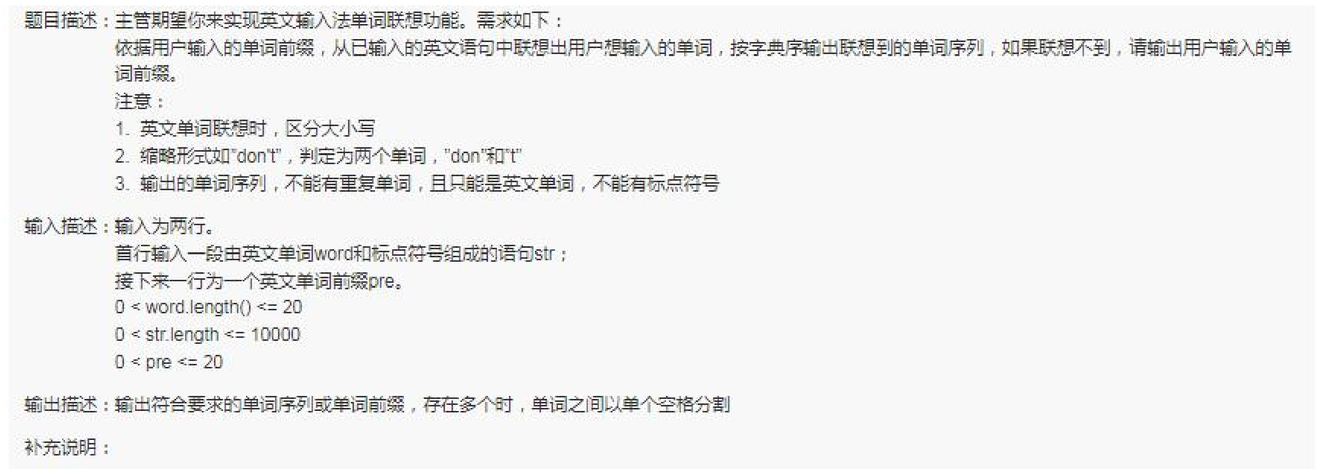
一、编程题

ACM：英文输入法

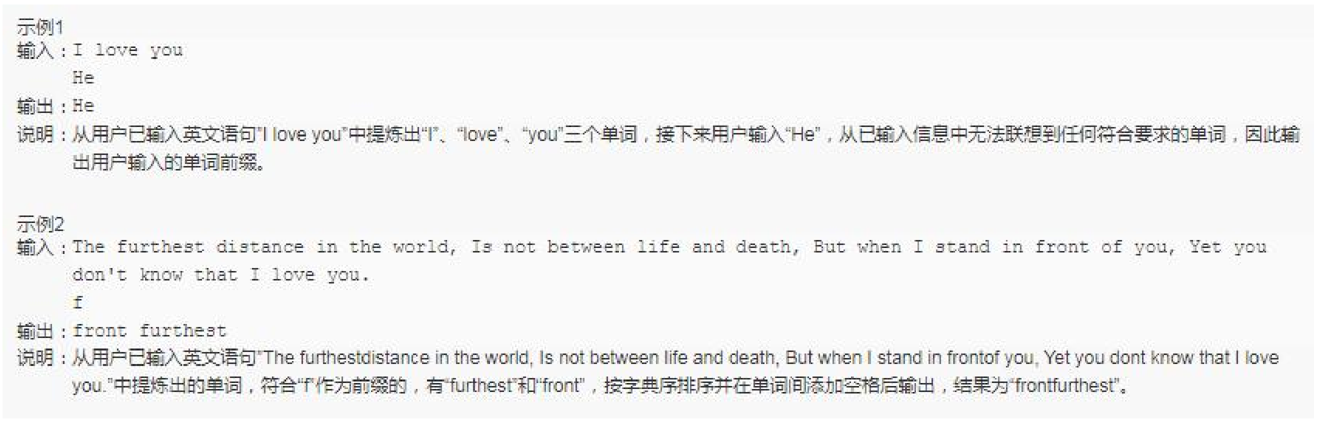


题目描述：主管期望你来实现英文输入法单词联想功能。需求如下：
依据用户输入的单词前缀，从已输入的英文语句中联想出用户想输入的单词，按字典序输出联想到的单词序列，如果联想不到，请输出用户输入的单词前缀。
注意：
1. 英文单词联想时，区分大小写
2. 缩略形式如"don't"，判定为两个单词，"don"和"t"
3. 输出的单词序列，不能有重复单词，且只能是英文单词，不能有标点符号

输入描述：输入为两行。
首行输入一段由英文单词word和标点符号组成的语句str；
接下来一行为一个英文单词前缀pre。
0 < word.length() <= 20
0 < str.length <= 10000
0 < pre <= 20

输出描述：输出符合要求的单词序列或单词前缀，存在多个时，单词之间以单个空格分割

补充说明：



示例1
输入：`I love you`
`He`
输出：`He`
说明：从用户已输入英文语句"I love you"中提炼出"I"、"love"、"you"三个单词，接下来用户输入"He"，从已输入信息中无法联想到任何符合要求的单词，因此输出用户输入的单词前缀。

示例2
输入：`The furthest distance in the world, Is not between life and death, But when I stand in front of you, Yet you don't know that I love you.`
`f`
输出：`front furthest`
说明：从用户已输入英文语句"The furthestdistance in the world, Is not between life and death, But when I stand in frontof you, Yet you dont know that I love you."中提炼出的单词，符合"f"作为前缀的，有"furthest"和"front"，按字典序排序并在单词间添加空格后输出，结果为"frontfurthest"。

代码：

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

struct TrieNode {
    unordered_map<char, TrieNode*> um;
    bool is_end = false;
};

class TrieTree {
public:
    TrieTree() {
        root = new TrieNode;
    }

    void insertNewWord(const string& word) {
        if(word.empty()){
            return;
        }
        TrieNode *next = root;
        TrieNode *pre = root;
        int len = word.size();
        for(int i = 0; i < len; ++i) {
                if(next->um.find(word[i]) != next->um.end()) {
                        if(i == len -1) {
                                next->is_end = true;
                        }
                        next = next->um[word[i]];
                }else {
                        TrieNode* node = new TrieNode;
                        next->um[word[i]] = node;
                        if(i == len -1) {
                                next->is_end = true;
                        }
                        next = node;
                }
        }
}

void geAllWord(TrieNode *root_, string s, vector<string>& result) {
        if(!root_) {
                return;
        }
        for(const auto& item : root_->um) {
                if(root_->is_end) {
                        result.push_back(s + item.first);
                }
                geAllWord(item.second, s + item.first, result);
        }
        return ;
}

void getWordWithPrefix(const string& prefix, vector<string>& result) {
        TrieNode *next = root;
        for(int i = 0; i < prefix.size(); ++i) {
                if(next->um.find(prefix[i]) != next->um.end()) {
                        next = next->um[prefix[i]];
                }else if(i != prefix.size()) {
                        result.push_back(prefix);
                        return;
                }
        }
        geAllWord(next, prefix, result);
```

```
    }
private:
        TrieNode * root = nullptr;
};

int main() {
    string str;
    string pre;
    getline(cin, str);
    cin >> pre;
    TrieTree tree;
    for(int i = 0; i < str.size(); ++i) {
        string tmp;
        while(i < str.size() && isalpha(str[i])) {
            tmp += str[i];
            i++;
        }
        if(!tmp.empty()) {
            tree.insertNewWord(tmp);
        }
    }
    vector<string> res;
    tree.getWordWithPrefix(pre, res);

    sort(res.begin(), res.end());
    auto it = unique(res.begin(), res.end());
    for(int i = 0; i < it - res.begin(); ++i) {
        if(i != res.size() - 1) {
            cout << res[i] << " ";
        }else {
            cout << res[i];
        }
    }
    return 0;
}
// 64 位输出请用 printf("%lld")
```