题目描述

主要就是仿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

用例

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。

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."中提炼出时间符合要求的单词，因此输出用户输入的单词前缀。 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"front furthest"。

题目解析

简单的逻辑题，应该是主要考察数组去重，数组字典排序，过滤等知识

JavaScript算法源码

```javascript
// JavaScript Node ACM模式 控制台输入获取 ==
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const str = lines[0];
    const pre = lines[1];

    console.log(lenovo(str, pre));

    lines.length = 0;
  }
});

function lenovo(str, pre) {
  const cache = [...new Set(str.split(/[^a-zA-Z]/))];

  const res = cache.filter((ele) => ele.startsWith(pre)).sort();

  if (res.length === 0) return pre;
  return res.join(" ");
}
```

Java算法源码

```java
import java.util.Collections;
import java.util.Scanner;
import java.util.StringJoiner;
import java.util.TreeSet;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String str = sc.nextLine();
    String pre = sc.nextLine();
    System.out.println(getResult(str, pre));
  }

  public static String getResult(String str, String pre) {
    String[] tmp = str.split("[^a-zA-Z]");
    TreeSet<String> cache = new TreeSet<>();
    Collections.addAll(cache, tmp);

    StringJoiner sj = new StringJoiner(" ");
    cache.stream().filter(s -> s.startsWith(pre)).forEach(s -> sj.add(s));

    String ans = sj.toString();
    if (ans.length() > 0) {
      return ans;
    } else {
      return pre;
    }
  }
}
```

Python算法源码

```python
import re

# 输入获取
s = input()
pre = input()


# 算法入口
def getResult(s, pre):
    tmp = re.split("[^a-zA-Z]", s)
    cache = list(set(tmp))
    cache.sort()
    cache = list(filter(lambda x: x.startswith(pre), cache))
    if len(cache) > 0:
        return " ".join(cache)
    else:
        return pre


# 算法调用
print(getResult(s, pre))
```