61、快速人名查找，考点 or 实现——回溯算法

题目描述

给一个字符串，表示用','分开的人名。

然后给定一个字符串，进行快速人名查找，符合要求的输出。

快速人名查找要求：人名的每个单词的连续前几位能组成给定字符串，一定要用到每个单词。

输入描述

第一行是人名，用','分开的人名
第二行是 查找字符串

输出描述

输出满足要求的人名

用例

输入	zhang san,zhang san san zs
输出	zhang san
说明	无

输入	zhang san san,zhang an sa,zhang hang,zhang seng,zhang sen a zhas
输出	zhang an sa,zhang seng
说明	无

题目解析

本题暂时没有想到更好的解法，只能通过暴力法求解。

我的解题思路如下：

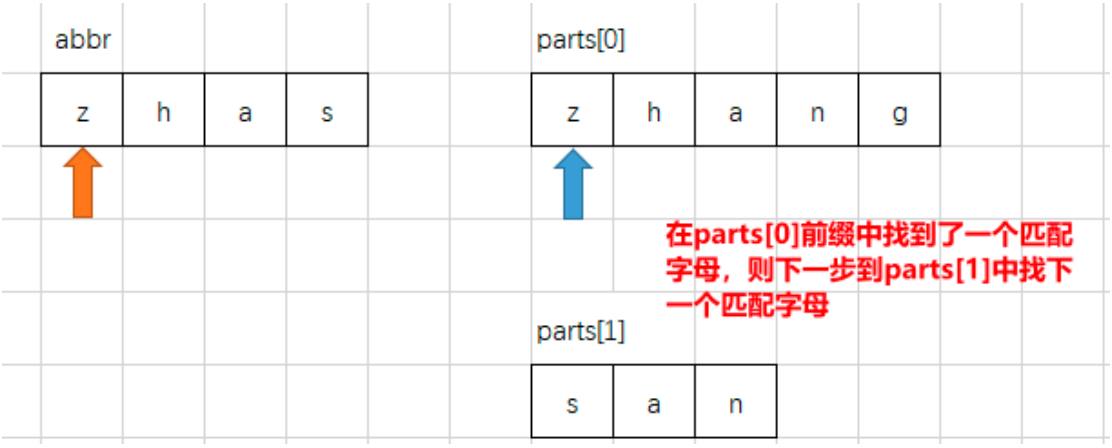
比如人名为：zhang seng，前缀缩写为：zhas

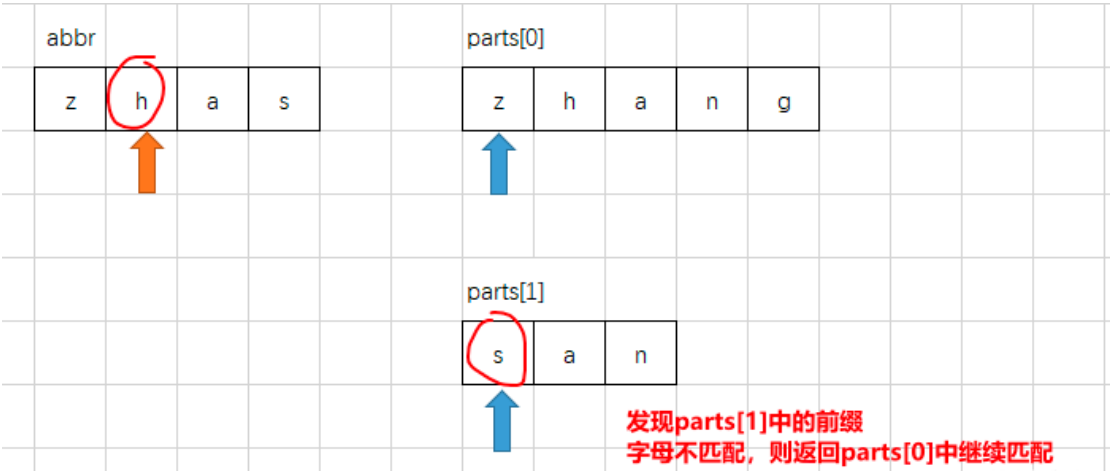
首先将人名字符串按照空格分隔为 字符串数组，如Sting parts = ["zhang", "san"]

前缀缩写定义为 String abbr = "zhas"，用一个start指针指向abbr的被检查的字母，初始时start=0，即指向abbr的第0个字母。

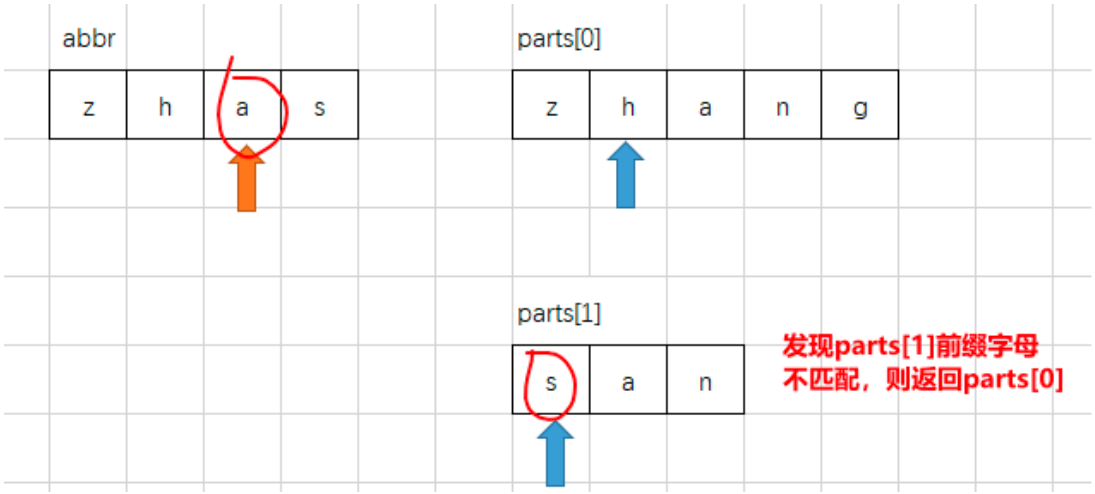
然后按照下图逻辑开始进行人名的每个part的前缀部分和abbr的匹配，匹配逻辑如下图所示：

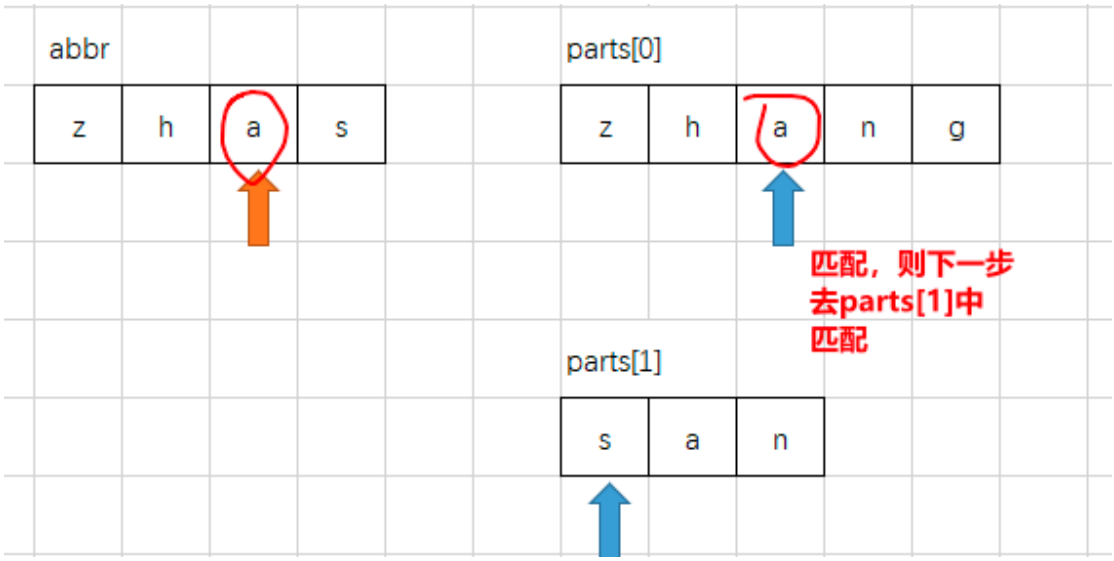
> abbr开头字母必须首先要和parts[0]的开头字母匹配，如果无法匹配否则就表示当前人名无法缩写为abbr。


abbr
z h a s
parts[0]
z h a n g
在parts[0]前缀中找到了一个匹配字母，则下一步到parts[1]中找下一个匹配字母
parts[1]
s a n


abbr
z h a s
parts[0]
z h a n g
parts[1]
s a n
发现parts[1]中的前缀字母不匹配，则返回parts[0]中继续匹配


abbr
z h a s
parts[0]
z h a n g
发现parts[0]匹配字母，则下一步去parts[1]中继续查找
parts[1]
s a n


abbr
z h a s
parts[0]
z h a n g
parts[1]
s a n
发现parts[1]前缀字母
不匹配，则返回parts[0]


abbr
z h a s
parts[0]
z h a n g
匹配，则下一步
去parts[1]中
匹配
parts[1]
s a n


abbr
z h a s
parts[0]
z h a n g
parts[1]
s a n
发现parts[1]前缀字母
可以匹配
且此时刚好匹配完abbr所有字母，且
abbr最后一个字母 是在 parts[1]
（即人名最后一个单词）中
匹配完成的，因此可以判定
人名“zhang san” 可以匹配缩写"zhas"

通过上面图示，我们可以发现，parts[0]每匹配到一个abbr字母，就把下一次的匹配权让给了parts[1]，这是为什么呢？

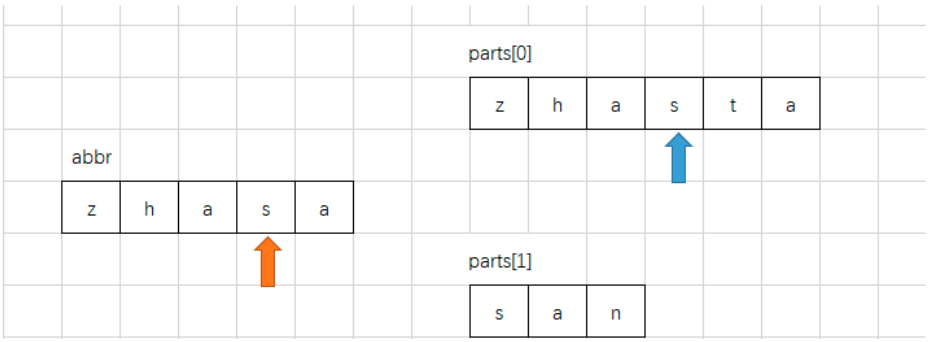
因为，这样可以避免abbr的扫描指针的回溯。啥意思呢？

我们看看下面这个例子：

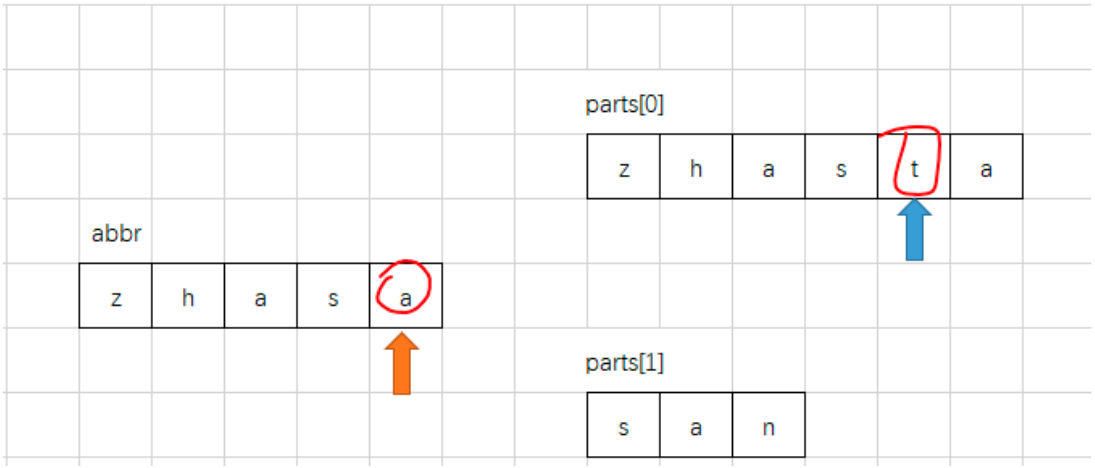
人名：zhasta san

缩写：zhasa

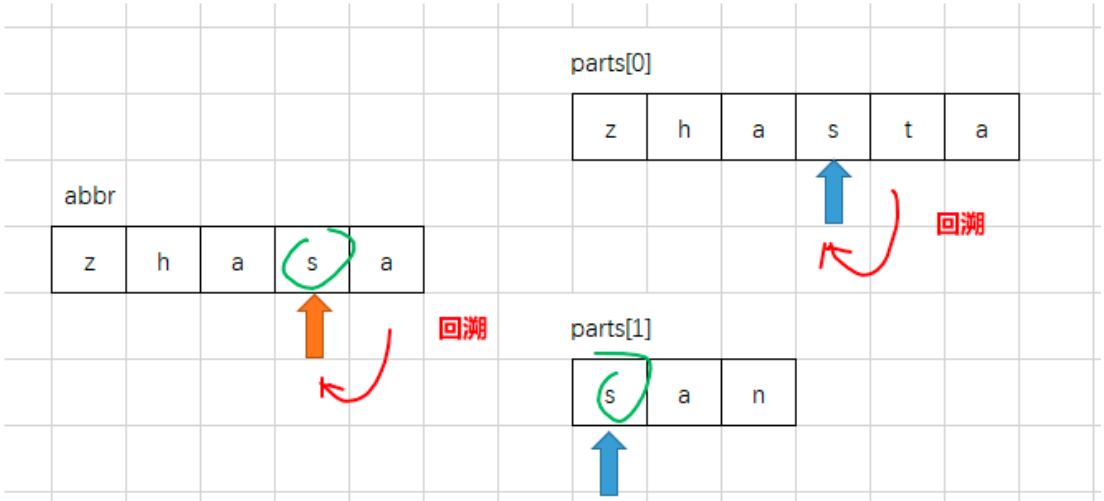
如果优先parts[0]匹配的话，则必然会走到下面这种情况



然后下一步匹配，就会出问题



此时，我们就需要进行abbr的扫描指针回溯，然后将匹配权交到parts[1]手上。



可能有人觉得这样匹配速度反而会更快一点，但是如果用例是下面这种呢？

- 人名：zhass san san
- 缩写：zhass

即我们可以在一个part中就完成了缩写匹配，但是题目要求所有的单词part都要使用到，此时我们应该如何回溯呢？

这样的回溯操作逻辑将变得十分复杂，且性能不佳。因此我们需要尽量避免abbr的扫描指针回溯行为。

如果，大家对上面部分都理解了，那么本题差不多就搞定了。

上面例子都是两个part的人名，本题并没有说人名由几个part组成，因此我们无法确定要几重for循环解题，此时只能依赖于回溯递归来帮助我们。具体递归逻辑请看代码。

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 names = lines[0].split(",");
    const abbr = lines[1];

    console.log(getResult(names, abbr));
    lines.length = 0;
  }
});

function getResult(names, abbr) {
  const ans = [];

  for (let name of names) {
    const parts = name.split(" ");
    if (parts.length > abbr.length) continue;

    const res = dfs(parts, 0, abbr, 0);
    if (res) {
      ans.push(name);
    }
  }

  return ans.join(",");
}

function dfs(parts, index, abbr, start) {
  if (start >= abbr.length) return index >= parts.length;

```

```
  for (let i = index; i < parts.length; i++) {
    const part = parts[i];

    for (let j = 0; j < part.length; j++) {
      if (start < abbr.length && part[j] == abbr[start]) {
        const res = dfs(parts, i + 1, abbr, ++start);
        if (res) return true;
      } else {
        return false;
      }
    }
  }

  return false;
}
```

Java算法源码

```
import java.util.ArrayList;
import java.util.Scanner;
import java.util.StringJoiner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    String[] names = sc.nextLine().split(",");
    String abbr = sc.nextLine();

    System.out.println(getResult(names, abbr));
  }

  public static String getResult(String[] names, String abbr) {
    ArrayList<String> ans = new ArrayList<>();

    for (String name : names) {
      String[] parts = name.split(" ");
      if (parts.length > abbr.length()) continue;

      boolean res = dfs(parts, 0, abbr, 0);
      if (res) {
        ans.add(name);
      }
    }
```

```
27
28      StringJoiner sj = new StringJoiner(",");
29      for (String an : ans) {
30        sj.add(an);
31      }
32      return sj.toString();
33    }
34
35    public static boolean dfs(String[] parts, int index, String abbr, int start) {
36      if (start >= abbr.length()) return index >= parts.length;
37
38      for (int i = index; i < parts.length; i++) {
39        String part = parts[i];
40
41        for (int j = 0; j < part.length(); j++) {
42          if (start < abbr.length() && part.charAt(j) == abbr.charAt(start)) {
43            boolean res = dfs(parts, i + 1, abbr, ++start);
44            if (res) return true;
45          } else {
46            return false;
47          }
48        }
49      }
50
51      return false;
52    }
53  }
```

Python算法源码

```python
# 输入获取
names = input().split(",")
abbr = input()


# 深度优先搜索
def dfs(parts, index, abbr, start):
    if start >= len(abbr):
        return index >= len(parts)

    for i in range(index, len(parts)):
        part = parts[i]

        for j in range(len(part)):
            if start < len(abbr) and part[j] == abbr[start]:
                start += 1
                res = dfs(parts, i + 1, abbr, start)
                if res:
                    return True
            else:
                return False

    return False


# 算法入口
def getResult(names, abbr):
    ans = []

    for name in names:
        parts = name.split()

        if len(parts) > len(abbr):
            continue

        res = dfs(parts, 0, abbr, 0)
        if res:
            ans.append(name)

    print(",".join(ans))


# 算法调用
getResult(names, abbr)
```