38、服务失效判断，考点 or 实现——数据结构/并查集

题目描述

某系统中有众多服务，每个服务用字符串（只包含字母和数字，长度<=10）唯一标识，服务间可能有依赖关系，如A依赖B，则当B故障时导致A也故障。

依赖具有传递性，如A依赖B，B依赖C，当C故障时导致B故障，也导致A故障。

给出所有依赖关系，以及当前已知故障服务，要求输出所有正常服务。

依赖关系：服务1-服务2 表示“服务1”依赖“服务2”

不必考虑输入异常，用例保证：依赖关系列表、故障列表非空，且依赖关系数，故障服务数都不会超过3000，服务标识格式正常。

输入描述

半角逗号分隔的依赖关系列表 **(换行)**

半角逗号分隔的故障服务列表

输出描述

依赖关系列表中提及的所有服务中可以正常工作的服务列表，用半角逗号分隔，按依赖关系列表中出现的次序排序。

特别的，没有正常节点输出单独一个**半角逗号**。

用例

输入	a1-a2, a5-a6, a2-a3 a5, a2
输出	a6,a3
说明	a1依赖a2，a2依赖a3，所以a2故障，导致a1不可用，但不影响a3；a5故障不影响a6。 所以可用的是a3、a6，在依赖关系列表中a6先出现，所以输出:a6,a3。

输入	a1-a2 a2
输出	,
说明	a1依赖a2，a2故障导致a1也故障，没有正常节点，输出一个逗号。

题目解析

我的解题思路是：

根据第一行输入的依赖关系，统计出每个服务的直接子服务，记录在next中。

另外由于题目输出描述中说：输出的服务要按照：

> 按依赖关系列表中出现的次序排序。

因此，这里我还定义了一个first，用于记录每个服务第一次出现的位置。

当上面统计好后，就可以遍历第二行输入的故障服务列表了。

每遍历到一个故障服务，则删除next中对应服务，但是删除前，需要先记录将要删除服务的所有子服务。删除当前故障服务后，继续递归删除其子服务。

这样next剩下的就是正常服务了。

此时再按照first记录的出现位置对剩余的正常服务排序即可，

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 relatons = lines[0].split(",").map((p) => p.split("-")); // 先决条件
    const breakdowns = lines[1].split(","); // 故障机器

    console.log(getNormalMachine(relatons, breakdowns));

    lines.length = 0;
  }
});

function getNormalMachine(relatons, breakdowns) {
  const next = {}; // 属性是父服务，属性值是子服务集合
  const first = {}; // 记录服务第一次出现的位置

  let i = 0;
  for (let [c, f] of relatons) {
```

```
    if (!next[c]) next[c] = new Set();
    if (!next[f]) next[f] = new Set();
    next[f].add(c);

    if (!first[c]) first[c] = i++;
    if (!first[f]) first[f] = i++;
  }

  for (let s of breakdowns) {
    remove(next, s);
  }

  const ans = Object.keys(next);
  if (ans.length == 0) return ",";
  return ans.sort((a, b) => first[a] - first[b]).join(",");
}

// 由于服务s是故障服务，因此s服务本身，和其所有子孙服务都无法运行
function remove(next, s) {
  if (next[s]) {
    const need_remove = next[s];
    delete next[s];

    for (let ss of need_remove) {
      remove(next, ss);
    }
  }
}
```

Java算法源码

```
import java.util.*;

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

    String[][] relations =
        Arrays.stream(sc.nextLine().split(",")).map(s -> s.split("-")).toArray(String[][]::new);

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

    System.out.println(getResult(relations, breakdowns));
  }
```

```java

  public static String getResult(String[][] relations, String[] breakdowns) {
    HashMap<String, HashSet<String>> next = new HashMap<>(); // 属性是父服务，属性值是子服务集合
    HashMap<String, Integer> first = new HashMap<>(); // 记录服务第一次出现的位置

    int i = 0;
    for (String[] relation : relations) {
      String c = relation[0];
      String f = relation[1];

      next.putIfAbsent(c, new HashSet<>());
      next.putIfAbsent(f, new HashSet<>());

      next.get(f).add(c);

      first.putIfAbsent(c, i++);
      first.putIfAbsent(f, i++);
    }

    for (String s : breakdowns) {
      remove(next, s);
    }

    String[] ans = next.keySet().toArray(new String[0]);
    if (ans.length == 0) return ",";

    Arrays.sort(ans, (a, b) -> first.get(a) - first.get(b));
    StringJoiner sj = new StringJoiner(",");
    for (String an : ans) sj.add(an);
    return sj.toString();
  }

  // 由于服务s是故障服务，因此s服务本身，和其所有子孙服务都无法运行
  public static void remove(HashMap<String, HashSet<String>> next, String s) {
    if (next.containsKey(s)) {
      HashSet<String> need_remove = next.get(s);
      next.remove(s);

      for (String ss : need_remove) {
        remove(next, ss);
      }
    }
  }
}
```

Python算法源码

```python
# 输入获取
relations = [s.split("-") for s in input().split(",")]
breakdowns = input().split(",")


def remove(next, s):
    if next.get(s) is not None:
        need_remove = next[s]
        del next[s]

        for ss in need_remove:
            remove(next, ss)


# 算法入口
def getResult():
    next = {}
    first = {}

    i = 0
    for c, f in relations:
        if next.get(c) is None:
            next[c] = set()

        if next.get(f) is None:
```

```
            next[f] = set()

        next[f].add(c)

        if first.get(c) is None:
            first[c] = i
            i += 1

        if first.get(f) is None:
            first[f] = i
            i += 1

    for s in breakdowns:
        remove(next, s)

    ans = list(next.keys())
    if len(ans) == 0:
        return ","

    ans.sort(key=lambda x: first[x])

    return ",".join(ans)


# 算法调用
print(getResult())
```