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。

补充说明：

```
示例1
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


示例2
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
```

```
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main {
    //记录服务的工作状态
    static Map<String,Boolean> work = new LinkedHashMap<>();
    //记录依赖关系
    static Map<String,ArrayList<String>> map = new LinkedHashMap<>();
```

```
public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        String input = in.nextLine();
        String mis = in.nextLine();
        String[] inputs = input.split("\\,");
        for(int i=0;i<inputs.length;i++){
            String[] res=inputs[i].split("-");
            if(!work.containsKey(res[0])){
                work.put(res[0],true);
            }
            if(!work.containsKey(res[1])){
                work.put(res[1],true);
            }
            if(map.containsKey(res[1])){
                map.get(res[1]).add(res[0]);
            }else{
                ArrayList<String> link = new ArrayList<>();
                link.add(res[0]);
                map.put(res[1],link);
            }
        }
        Queue<String> errors = new LinkedList<String>();
        String[] t = mis.split("\\,");
        for(int i=0;i<t.length;i++){
            errors.add(t[i]);
        }
        changeState(errors);
        StringBuffer result = new StringBuffer();
        for(String s:work.keySet()){
            if(work.get(s)){
                result.append(s).append(",");
            }
        }
        if(result.length()==0){
            result.append(",");
        }else{
            result.delete(result.length()-1,result.length());
        }

        System.out.println(result.toString());
}

public static void changeState(Queue<String> queue){
        while(!queue.isEmpty()){
```

```
                String s=queue.poll();
                // System.out.println(s);
                if(work.get(s)){
                    work.replace(s,false);
                }
                if(map.containsKey(s)){
                    ArrayList<String> list =map.get(s);
                    for(String k:list){
                        if(work.get(k)){
                            queue.add(k);
                        }
                    }
                }
            }
        }
}
```