题目描述：
一个应用启动时，会有多个初始化任务需要执行，并且任务之间有依赖关系，例如 A 任务依赖 B 任务，那么必须在 B 任务执行完成之后，才能开始执行 A 任务。

现在给出多条任务依赖关系的规则，请输入任务的顺序执行序列，规则采用贪婪策略，即一个任务如果没有依赖的任务，则立刻开始执行，如果同时有多个任务要执行，则根据任务名称字母顺序排序。

例如：B 任务依赖 A 任务，C 任务依赖 A 任务，D 任务依赖 B 任务和 C 任务，同时，D 任务还依赖 E 任务。那么执行任务的顺序由先到后是：A 任务，E 任务，B 任务，C 任务，D 任务。这里 A 和 E 任务都是没有依赖的，立即执行

输入描述：
输入参数每个元素都表示任意两个任务之间的依赖关系，输入参数中符号“->”表示依赖方向，例如 A->B 表示 A 依赖 B，多个依赖之间用单个空格分割
输出描述：
输出为排序后的启动任务列表，多个任务之间用单个空格分割
示例 1
输入：
A->B C->B
输出：
B A C
说明：
输入参数每个元素都表示任意两个任务之间的依赖关系，输入参数中符号“->”表示依赖方向，例如 A->B 表示 A 依赖 B，多个依赖之间用单个空格分割，输出的多个任务之间也用单个空格分割

```
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        String line = in.nextLine();
        String[] depArr = line.split(" ");
        Map<String, List<String>> depMap = new TreeMap<>();
        List<String> result = new ArrayList<>();
        for (String dep : depArr) {
            String[] task = dep.split("->");
            String left = task[0];
            String right = task[1];
            if (!depMap.containsKey(left)) {
                depMap.put(left, new ArrayList<>());
            }
```

```
                depMap.get(left).add(right);
                if (!depMap.containsKey(right)) {
                    depMap.put(right, new ArrayList<>());
                }
            }
            while (!depMap.isEmpty()) {
                executeTask(depMap, result);
                for (String key : depMap.keySet()) {
                    List<String> depList = depMap.get(key);
                    Iterator<String> depIterator = depList.iterator();
                    while (depIterator.hasNext()) {
                        if (!depMap.containsKey(depIterator.next())) {
                            depIterator.remove();
                        }
                    }
                }
            }
            System.out.println(String.join(" ", result));
        }

        public static void executeTask(Map<String, List<String>> depMap, List<String> result) {
            Iterator<Map.Entry<String, List<String>>> iterator = depMap.entrySet().iterator();
            while (iterator.hasNext()) {
                Map.Entry<String, List<String>> entry = iterator.next();
                if (entry.getValue().isEmpty()) {
                    result.add(entry.getKey());
                    iterator.remove();
                }
            }
        }
    }
```