

题目描述：

VLAN 是一种对局域网设备进行逻辑划分的技术，为了标识不同的 *VLAN*，引入 *VLAN ID*(*1-4094* 之间的整数)的概念。定义一个 *VLAN ID* 的资源池(下称 *VLAN* 资源池)，资源池中连续的 *VLAN* 用开始 *VLAN*-结束 *VLAN* 表示，不连续的用单个整数表示，所有的 *VLAN* 用英文逗号连接起来。现在有一个 *VLAN* 资源池，业务需要从资源池中申请一个 *VLAN*，需要你输出从 *VLAN* 资源池中移除申请的 *VLAN* 后的资源池。

输入描述：

第一行为字符串格式的 *VLAN* 资源池，第二行为业务要申请的 *VLAN*，*VLAN* 的取值范围为[*1,4094*]之间的整数。

输出描述：

从输入 *VLAN* 资源池中移除申请的 *VLAN* 后字符串格式的 *VLAN* 资源池，输出要求满足题目描述中的格式，并且按照 *VLAN* 从小到大升序输出。

如果申请的 *VLAN* 不在原 *VLAN* 资源池内，输出原 *VLAN* 资源池升序排序后的字符串即可。

补充说明：

输入 *VLAN* 资源池中 *VLAN* 的数量取值范围为[*2-4094*]间的整数，资源池中 *VLAN* 不重复且合法([*1,4094*]之间的整数)，输入是乱序的。

示例 1

输入：

1-5

2

输出：

1,3-5

说明：

原 VLAN 资源池中有 VLAN 1、2、3、4、5，从资源池中移除 2 后，剩下 VLAN 1、3、4、5，按照题目描述格式并升序后的结果为 1,3-5。

示例 2

输入：

20-21,15,18,30,5-10

15

输出：

5-10,18,20-21,30

说明：

原 VLAN 资源池中有 VLAN 5、6、7、8、9、10、15、18、20、21、30，从资源池中移除 15 后，资源池中剩下的 VLAN 为 5、6、7、8、9、10、18、20、21、30，按照题目描述格式并升序后的结果为 5-10,18,20-21,30。

示例 3

输入：

5,1-3

10

输出：

1-3,5

说明：

原 VLAN 资源池中有 VLAN 1、2、3，5，申请的 VLAN 10 不在原资源池中，将原资源池按照题目描述格式并按升序排序后输出的结果为 1-3,5。

```

import java.io.BufferedReader;
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(new BufferedReader(System.in));
        String[] inputs = scanner.nextLine().split(",");
        int req = scanner.nextInt();

        PriorityQueue<Interval> pq = new PriorityQueue<>();
        for (String input : inputs) {
            Interval interval = convert(input);
            if (interval.contains(req))
                pq.addAll(interval.split(req));
            else
                pq.add(interval);
        }

        StringBuilder sb = new StringBuilder();
        while (!pq.isEmpty())
            sb.append(pq.poll()).append(',');

        sb.setLength(Math.max(0, sb.length() - 1));
        System.out.println(sb);
    }

    private static Interval convert(String input) {
        String[] split = input.split("-");
        int start = Integer.parseInt(split[0]);
        int end;
        if (split.length == 1)
            end = start;
        else
            end = Integer.parseInt(split[1]);

        return new Interval(start, end);
    }

    private static class Interval implements Comparable<Interval> {
        private final int start;
        private final int end;

        public Interval(int start, int end) {
            this.start = start;
        }
    }

```

```

        this.end = end;
    }

    @Override
    public int compareTo(Interval o) {
        return Integer.compare(start, o.start);
    }

    @Override
    public String toString() {
        if (start == end)
            return String.valueOf(start);
        else
            return start + "-" + end;
    }

    boolean contains(int id) {
        return id >= start && id <= end;
    }

    Collection<Interval> split(int id) {
        if (id == start && id == end) return Collections.emptyList();

        if (id == start)
            return Arrays.asList(new Interval(start + 1, end));
        if (id == end)
            return Arrays.asList(new Interval(start, end - 1));

        return Arrays.asList(new Interval(start, id - 1), new Interval(id + 1, end));
    }
}

```