题目描述：

现有两门选修课，每门选修课都有一部分学生选修，每个学生都有选修课的成绩，需要你找出同时选修了两门选修课的学生，先按照班级进行划分，班级编号小的先输出，每个班级按照两门选修课成绩和的降序排序，成绩相同时按照学生的学号升序排序。

输入描述：

第一行为第一门选修课学生的成绩，第二行为第二门选修课学生的成绩，每行数据中学生之间以英文分号分隔，每个学生的学号和成绩以英文逗号分隔，学生学号的格式为 8 位数字(2 位院系编号+入学年份后 2 位+院系内部 1 位专业编号+所在班级 3 位学号)，学生成绩的取值范围为[0,100]之间的整数，两门选修课选修学生数的取值范围为[1-2000]之间的整数。

输出描述：

同时选修了两门选修课的学生的学号，如果没有同时选修两门选修课的学生输出 NULL，否则，先按照班级划分，班级编号小的先输出，每个班级先输出班级编号(学号前五位)，然后另起一行输出这个班级同时选修两门选修课的学生学号，学号按照要求排序(按照两门选修课成绩和的降序，成绩和相同时按照学号升序)，学生之间以英文分号分隔。

示例 1

输入：

```
01202021,75;01201033,95;01202008,80;01203006,90;01203088,100

01202008,70;01203088,85;01202111,80;01202021,75;01201100,88
```

输出：

```
01202

01202008;01202021

01203

01203088
```

说明：

同时选修了两门选修课的学生 01202021、01202008、01203088，这三个学生两门选修课的成绩和分别为 150、150、185，01202021、01202008 属于 01202 班的学生，按照成绩和降序，成绩相同时按学号升序输出的结果为 01202008;01202021,01203088 属于 01203 班的学生，按照成绩和降序，成绩相同时按学号升序输出的结果为 01203088，01202 的班级编号小于 01203 的班级编号，需要先输出。

示例 2

输入：

```
01201022,75;01202033,95;01202018,80;01203006,90;01202066,100
01202008,70;01203102,85;01202111,80;01201021,75;01201100,88
```

输出：

```
NULL
```

说明：

没有同时选修了两门选修课的学生，输出 NULL。

```
import java.util.*;

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

        String[][] s1 = Arrays.stream(sc.nextLine().split(";"))
                .map(s -> Arrays.stream(s.split(",")).toArray(String[]::new))
                .toArray(String[][]::new);

        String[][] s2 = Arrays.stream(sc.nextLine().split(";"))
                .map(s -> Arrays.stream(s.split(",")).toArray(String[]::new))
                .toArray(String[][]::new);

        ArrayList<String> ans = new ArrayList<>();

        getAns(s1, s2, ans);
        for (String an : ans) {
            System.out.println(an);
        }
```

```
    }

    private static void getAns(String[][] s1, String[][] s2, ArrayList<String> ans) {
        HashMap<String, ArrayList<String>> stus = new HashMap<>();
        for (String[] s : s1) {
            String id = s[0];
            String score = s[1];
            stus.putIfAbsent(id, new ArrayList<>());
            stus.get(id).add(score);
        }

        for (String[] s : s2) {
            String id = s[0];
            String score = s[1];
            stus.putIfAbsent(id, new ArrayList<>());
            stus.get(id).add(score);
        }

        ArrayList<String[]> ids = new ArrayList<>();
        for (String id : stus.keySet()) {
            if (stus.get(id).size() == 2) {
                String class_num = id.substring(0, 5);
                int sum = Integer.parseInt(stus.get(id).get(0)) +
Integer.parseInt(stus.get(id).get(1));
                ids.add(new String[]{class_num, id, sum + ""});
            }
        }

        if (ids.isEmpty()) {
            System.out.println("NULL");
            return;
        }

        ids.sort((a, b) -> a[1].compareTo(b[1]));

        LinkedHashMap<String, ArrayList<String[]>> same_class = new LinkedHashMap<>();
        for (String[] id : ids) {
            same_class.putIfAbsent(id[0], new ArrayList<>());
            same_class.get(id[0]).add(new String[]{id[1], id[2]});
        }

        for (String key : same_class.keySet()) {
            ans.add(key);
            ArrayList<String[]> same_class_list = same_class.get(key);
```

```
                same_class_list.sort((a, b) -> !a[1].equals(b[1]) ?
            Integer.parseInt(b[1])-Integer.parseInt(a[1])
            : a[0].compareTo(b[0]));
                StringJoiner sj = new StringJoiner(";");
                for (String[] stu : same_class_list) {
                    sj.add(stu[0]);
                }
                ans.add(sj.toString());
            }
        }
}
```