题目描述

给定一组闭区间，其中部分区间存在交集。

任意两个给定区间的交集，称为公共区间(如:[1,2],[2,3]的公共区间为[2,2]，[3,5],[3,6]的公共区间为[3,5])。

公共区间之间若存在交集，则需要合并(如:[1,3],[3,5]区间存在交集[3,3]，需合并为[1,5])。

按升序排列输出合并后的区间列表。

输入描述

一组区间列表，

区间数为 N: 0<=N<=1000;

区间元素为 X: -10000<=X<=10000。

输出描述

升序排列的合并区间列表

备注

- 区间元素均为数字，不考虑字母、符号等异常输入。
- 单个区间认定为无公共区间。

用例

输入	4 0 3 1 3 3 5 3 6
输出	1 5
说明	[0,3]和[1,3]的公共区间为[1,3]， [0,3]和[3,5]的公共区间为[3,3]， [0,3]和[3,6]的公共区间为[3,3]， [1,3]和[3,5]的公共区间为[3,3]， [1,3]和[3,6]的公共区间为[3,3]， [3,5]和[3,6]的公共区间为[3,5]， 公共区间列表为[[1,3],[3,3],[3,5]]； [1,3],[3,3],[3,5]存在交集，须合并为[1,5]。

输入	4 0 3 1 4 4 7 5 8
输出	1 3 4 4 5 7
说明	无

输入	2 1 2 3 4
输出	None
说明	[1,2]和[3,4]无交集

题目解析

本题主要考察：区间交集求解、以及区间合并。

首先，我们要求解输入的多个区间中，任意两个区间的交集（公共区间）。

然后，将这些公共区间进行合并后打印。

两个区间的交集求解思路如下：

将两个区间按照开始位置进行升序，假设排序后，两个区间顺序是：[[s1, e1]，[s2, e2]]

那么必然 s1 >= s2，因此如果存在交集的话，即e1 >= s2

则交集的左边界必然是s2，而交集的右边界取值Math.min(e1, e2)

区间合并的逻辑可以参考：

Java算法源码

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;

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

    int n = sc.nextInt();

    int[][] ranges = new int[n][2];
    for (int i = 0; i < n; i++) {
      ranges[i][0] = sc.nextInt();
      ranges[i][1] = sc.nextInt();
    }

    getResult(n, ranges);
  }

  public static void getResult(int n, int[][] ranges) {
    // 区间按照开始位置升序
    Arrays.sort(ranges, (a, b) -> a[0] - b[0]);

    // combine用于保存交集
    ArrayList<int[]> combine = new ArrayList<>();

    // 求任意两个区间之间的交集
    for (int i = 0; i < n; i++) {
      int s1 = ranges[i][0], e1 = ranges[i][1];
      for (int j = i + 1; j < n; j++) {
        int s2 = ranges[j][0], e2 = ranges[j][1];
        if (s2 <= e1) {
          combine.add(new int[] {s2, Math.min(e1, e2)});
        } else {
          // 由于ranges已经升序，因此如果ranges[i]和ranges[j]没有交集的话，则也不可能和ranges[j+1]区间有交集
          break;
        }
      }
    }

```

```
    if (combine.size() == 0) {
      System.out.println("None");
      return;
    }

    // 合并公共区间
    combine.sort((a, b) -> a[0] != b[0] ? a[0] - b[0] : b[1] - a[1]);

    int[] pre = combine.get(0);
    for (int i = 1; i < combine.size(); i++) {
      int[] cur = combine.get(i);

      if (pre[1] >= cur[0]) {
        pre[1] = Math.max(cur[1], pre[1]);
      } else {
        System.out.println(pre[0] + " " + pre[1]);
        pre = cur;
      }
    }

    System.out.println(pre[0] + " " + pre[1]);
  }
}
```

JavaScript算法源码

```
/* 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 = [];
let n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    n = parseInt(lines[0]);
  }

  if (n && lines.length === n + 1) {
    lines.shift();
    const ranges = lines.map((line) => line.split(" ").map(Number));

    getResult(ranges);

    lines.length = 0;
  }
});

function getResult(ranges) {
  // 区间按照开始位置升序
  ranges.sort((a, b) => a[0] - b[0]);

  // combine用于保存交集
  const combine = [];

```

```javascript
  // 公共区间求解
  for (let i = 0; i < ranges.length; i++) {
    const [s1, e1] = ranges[i];
    for (let j = i + 1; j < ranges.length; j++) {
      const [s2, e2] = ranges[j];
      if (s2 <= e1) {
        combine.push([s2, Math.min(e1, e2)]);
      } else {
        // 由于ranges已经升序，因此如果ranges[i]和ranges[j]没有交集的话，则也不可能和ranges[j+1]区间有交集
        break;
      }
    }
  }

  if (combine.length == 0) return console.log("None");

  // 合并公共区间
  combine.sort((a, b) => (a[0] != b[0] ? a[0] - b[0] : b[1] - a[1]));

  let pre = combine[0];
  for (let i = 1; i < combine.length; i++) {
    const cur = combine[i];

    if (pre[1] >= cur[0]) {
      pre[1] = Math.max(cur[1], pre[1]);
    } else {
      console.log(pre.join(" "));
      pre = cur;
    }
  }

  console.log(pre.join(" "));
}
```

Python算法源码

```python
# 输入获取
n = int(input())
ranges = [list(map(int, input().split())) for _ in range(n)]


# 算法入口
def getResult():
    # 区间按照开始位置升序
    ranges.sort(key=lambda x: x[0])

    # combine用于保存公共区间
    combine = []

    for i in range(n):
        s1, e1 = ranges[i]
        for j in range(i + 1, n):
            s2, e2 = ranges[j]
            if s2 <= e1:
                combine.append([s2, min(e1, e2)])
            else:
                # 由于ranges已经升序，因此如果ranges[i]和ranges[j]没有交集的话，则也不可能和ranges[j+1]区间有交集
                break

    if len(combine) == 0:
        print("None")
        return

```

```
    if len(combine) == 0:
        print("None")
        return

    # 合并公共区间
    combine.sort(key=lambda x: (x[0], -x[1]))

    pre = combine[0]
    for i in range(1, len(combine)):
        cur = combine[i]

        if pre[1] >= cur[0]:
            pre[1] = max(cur[1], pre[1])
        else:
            print(" ".join(map(str, pre)))
            pre = cur

    print(" ".join(map(str, pre)))


# 算法调用
getResult()
```