

题目描述

公园园区提供小火车单向通行，从园区站点编号最小到最大通行如1~2~3~4~1，然后供员工在各个办公园区穿梭，通过对公司N个员工调研统计到每个员工的坐车区间，包含前后站点，请设计一个程序计算出小火车在哪个园区站点时人数最多。

输入描述

第1个行，为调研员工人数

第2行开始，为每个员工的上车站点和下车站点。

使用数字代替每个园区用空格分割，如3 5表示从第3个园区上车，在第5个园区下车

输出描述

人数最多时的园区站点编号，最多人数相同时返回编号最小的园区站点

用例

输入	3
	1 3
	2 4
	1 4
输出	2
说明	无

题目解析

本题其实就是求解最大重叠区间个数的变种题。

即，我们只要找到具有最大重叠部分的区间的起点就是本题题解。

关于最大重叠区间个数求解，请看[年年岁岁都容易挂的算法高频面试题，一线大厂经典面试题之堆和最大线段重叠问题_哔哩哔哩_bilibili](#)

上面视频的核心思想其实就是：

- 首先，将所有区间按开始位置升序
- 然后，遍历排序后区间，并将小顶堆中小于遍历区间起始位置的区间弹出（小顶堆实际存储区间结束位置），此操作后，小顶堆中剩余的区间个数，就是和当前遍历区间重叠数。

我们只需要在求解最大重叠数时，保留遍历的区间的起始位置即可。

对于JS而言，没有原生堆结构（即[优先队列](#)），因此我们需要手写小顶堆代码，关于优先队列，大家可以参考：

[LeetCode - 1705 吃苹果的最大数目_伏城之外的博客-CSDN博客](#)

但是本题是100分值的，可能不会存在大数量级，因此大家可以先尝试用[有序数组](#)代替小顶堆，如果不行，再实现小顶堆。

另外，本题和[华为OD机试 - 最大化控制资源成本_伏城之外的博客-CSDN博客](#)


很像，大家可以继续尝试做下上面这题。


2023.2.1根据网友指正，本题小火车是有可能走回程的，比如，坐车区间为[3,2]，即从站点3->站点4->站点1->站点2。

对于这种情况，就会破坏上面对于区间的排序。因此，我们需要将：该情况的坐车区间拆分为[3, 4]和[1, 2]

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 let n;
```

 伏城之外 [已关注](#)

 0   4  5 [专栏目录](#) [已订阅](#)

```

1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 let n;
11 rl.on("line", (line) => {
12   lines.push(line);
13
14   if (lines.length === 1) {
15     n = lines[0] - 0;
16   }
17
18   if (n && lines.length === n + 1) {
19     const ranges = lines.slice(1).map((line) => line.split(" ").map(Number));
20     console.log(getResult(ranges));
21     lines.length = 0;
22   }
23 });
24
25 function getResult(ranges) {
26   // 由于题目并未说有几个站点，因此需要计算出最后一个站点
27   const last = Math.max.apply(null, ranges.toString().split(",").map(Number));
28
29   // 如果存在 [3,2] 这种坐车区间，即从站点3 -> 站点4 -> 站点1 -> 站点2，则此时将该坐车区间拆分为 [3, 4]和[1, 2]
30   const tmp = [];
31   ranges.forEach((range) => {
32     const [s, e] = range;
33
34     if (s > e) {
35       // 拆分
36       tmp.push([s, last]);
37       tmp.push([1, e]);
38     } else {
39       tmp.push(range);
40     }
41   });
42   ranges = tmp;
43
44   // 最大重叠区间个数求解
45   ranges.sort((a, b) => a[0] - b[0]);
46   const end = new PriorityQueue((a, b) => b - a);
47
48   let max = 0;
49   let ans = ranges[0][0];
50   for (let range of ranges) {
51     const [s, e] = range;
52
53     while (end.size()) {
54       const top = end.peek();
55
56       if (top < s) {
57         end.shift();
58       } else {
59         break;
60       }
61     }
62
63     end.push(e);
64
65     if (end.size() > max) {
66       max = end.size();
67       ans = s;
68     }
69   }
70
71   return ans;
72 }
73
74 class PriorityQueue {
75   constructor(cpr) {
76     this.queue = [];
77     this.cpr = cpr;
78   }
79
80   swap(a, b) {
81     const tmp = this.queue[a];
82     this.queue[a] = this.queue[b];
83     this.queue[b] = tmp;
84   }
85
86   // 上浮
87   swim() {
88     let c = this.queue.length - 1;
89
90     while (c >= 1) {
91       const f = Math.floor((c - 1) / 2);
92

```

```

61     }
62
63     end.push(e);
64
65     if (end.size() > max) {
66         max = end.size();
67         ans = s;
68     }
69 }
70
71 return ans;
72 }
73
74 class PriorityQueue {
75     constructor(cpr) {
76         this.queue = [];
77         this.cpr = cpr;
78     }
79
80     swap(a, b) {
81         const tmp = this.queue[a];
82         this.queue[a] = this.queue[b];
83         this.queue[b] = tmp;
84     }
85
86     // 上浮
87     swim() {
88         let c = this.queue.length - 1;
89
90         while (c >= 1) {
91             const f = Math.floor((c - 1) / 2);
92
93             if (this.cpr(this.queue[c], this.queue[f]) > 0) {
94                 this.swap(c, f);
95                 c = f;
96             } else {
97                 break;
98             }
99         }
100     }
101
102     // 入队
103     push(val) {
104         this.queue.push(val);
105         this.swim();
106     }
107
108     // 下沉
109     sink() {
110         let f = 0;
111
112         while (true) {
113             let c1 = 2 * f + 1;
114             let c2 = c1 + 1;
115
116             let c;
117             let val1 = this.queue[c1];
118             let val2 = this.queue[c2];
119             if (val1 && val2) {
120                 c = this.cpr(val1, val2) > 0 ? c1 : c2;
121             } else if (val1 && !val2) {
122                 c = c1;
123             } else if (!val1 && val2) {
124                 c = c2;
125             } else {
126                 break;
127             }
128
129             if (this.cpr(this.queue[c], this.queue[f]) > 0) {
130                 this.swap(c, f);
131                 f = c;
132             } else {
133                 break;
134             }
135         }
136     }
137
138     // 出队
139     shift() {
140         this.swap(0, this.queue.length - 1);
141         const res = this.queue.pop();
142         this.sink();
143         return res;
144     }
145
146     // 查看顶
147     peek() {
148         return this.queue[0];
149     }
150
151     size() {
152         return this.queue.length;
153     }
154 }

```

Java算法源码

```
1 import java.util.ArrayList;
2 import java.util.Arrays;
3 import java.util.PriorityQueue;
4 import java.util.Scanner;
5
6 public class Main {
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9
10        int n = sc.nextInt();
11
12        int[][] ranges = new int[n][2];
13
14        for (int i = 0; i < n; i++) {
15            ranges[i][0] = sc.nextInt();
16            ranges[i][1] = sc.nextInt();
17        }
18
19        System.out.println(getResult(ranges));
20    }
21
22    public static int getResult(int[][] ranges) {
23        // 由于题目并未说有几个站点，因此需要计算出最后一个站点
24        Integer last =
25            Arrays.stream(ranges)
26                .map(range -> Math.max(range[0], range[1]))
27                .max((a, b) -> a - b)
28                .orElse(0);
29
30        // 如果存在 [3,2] 这种坐车区间，即从站点3 -> 站点4 -> 站点1 -> 站点2，则此时将该坐车区间拆分为 [3, 4]和[1, 2]
31        ArrayList<Integer[]> tmp = new ArrayList<>();
32        for (int[] range : ranges) {
33            int s = range[0];
34            int e = range[1];
35
36            if (s > e) {
37                tmp.add(new Integer[] {s, last});
38                tmp.add(new Integer[] {1, e});
39            } else {
40                tmp.add(new Integer[] {s, e});
41            }
42        }
43
44        // 最大重叠区间个数求解
45        tmp.sort((a, b) -> a[0] - b[0]);
46
47        PriorityQueue<Integer> end = new PriorityQueue<>((a, b) -> a - b);
48
49        int max = 0;
50        int ans = tmp.get(0)[0];
51        for (Integer[] range : tmp) {
52            int s = range[0];
53            int e = range[1];
54
55            while (end.size() > 0) {
56                Integer top = end.peek();
57
58                if (top < s) {
59                    end.poll();
60                } else {
61                    break;
62                }
63            }
64
65            end.offer(e);
66
67            if (end.size() > max) {
68                max = end.size();
69                ans = s;
70            }
71        }
72        return ans;
73    }
74 }
```

Python算法源码

```
1 import queue
2
3 # 输入获取
4 n = int(input())
5 ranges = [list(map(int, input().split())) for i in range(n)]
6
7
8 # 算法入口
9 def getResult(ranges):
10     # 由于题目并未说有几个站点，因此需要计算出最后一个站点
11     last = max(map(lambda x: max(x[0], x[1]), ranges))
12
13     # 如果存在 [3,2] 这种坐车区间，即从站点3 -> 站点4 -> 站点1 -> 站点2，则此时将该坐车区间拆分为 [3, 4]和[1, 2]
```



伏城之外 已关注

👍 0 🗨 4 📄 5 📁 专栏目录 已订阅

Python算法源码

```
1 import queue
2
3 # 输入获取
4 n = int(input())
5 ranges = [list(map(int, input().split())) for i in range(n)]
6
7
8 # 算法入口
9 def getResult(ranges):
10     # 由于题目并未说有几个站点, 因此需要计算出最后一个站点
11     last = max(map(lambda x: max(x[0], x[1]), ranges))
12
13     # 如果存在 [3,2] 这种坐车区间, 即从站点3 -> 站点4 -> 站点1 -> 站点2, 则此时将该坐车区间拆分为 [3, 4]和[1, 2]
14     tmp = []
15     for ran in ranges:
16         s, e = ran
17
18         if s > e:
19             tmp.append([s, last])
20             tmp.append([1, e])
21         else:
22             tmp.append(ran)
23     ranges = tmp
24
25     # 最大重叠区间个数求解
26     ranges.sort(key=lambda x: x[0])
27     end = queue.PriorityQueue()
28     maxV = 0
29     ans = ranges[0][0]
30
31     for ran in ranges:
32         s, e = ran
33
34         while end.qsize() > 0:
35             top = end.queue[0]
36
37             if top < s:
38                 end.get()
39             else:
40                 break
41
42         end.put(e)
43
44         if end.qsize() > maxV:
45             maxV = end.qsize()
46             ans = s
47
48     return ans
49
50
51 print(getResult(ranges))
```

基于差分数列求解本题（更适合本题）

关于差分数列, 请看

[算法设计 - 前缀和 & 差分数列_伏城之外的博客-CSDN博客](#)


然后可以试试leetcode下面差分数列题目

[LeetCode - 1109 - 航班预定统计_伏城之外的博客-CSDN博客](#)

本题差分数列求解的解析可以参照上面两个博客

JavaScript算法源码

```
1 /* JavaScript Node ACM模式 控制台输入获取 */
2 const readline = require("readline");
3
4 const rl = readline.createInterface({
5     input: process.stdin,
6     output: process.stdout,
7 });
8
9 const lines = [];
10 let n;
11 rl.on("line", (line) => {
12     lines.push(line);
13
14     if (lines.length === 1) {
15         n = lines[0] - 0;
16     }
17
18     if (n && lines.length === n + 1) {
19         const ranges = lines.slice(1).map((line) => line.split(" ").map(Number));
20         console.log(getResult(ranges));
21         lines.length = 0;
22     }
23 });
24
```

 伏城之外 已关注

👍 0 🗨 4 ⭐ 4



💬 5



专栏目录

已订阅

基于差分数列求解本题（更适合本题）

关于差分数列，请看

[算法设计 - 前缀和 & 差分数列_伏城之外的博客-CSDN博客](#)

然后可以试试leetcode下面差分数列题目

[LeetCode - 1109 - 航班预定统计_伏城之外的博客-CSDN博客](#)


本题差分数列求解的解析可以参照上面两个博客

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 let n;
11 rl.on("line", (line) => {
12   lines.push(line);
13
14   if (lines.length === 1) {
15     n = lines[0] - 0;
16   }
17
18   if (n && lines.length === n + 1) {
19     const ranges = lines.slice(1).map((line) => line.split(" ").map(Number));
20     console.log(getResult(ranges));
21     lines.length = 0;
22   }
23 });
24
25 function getResult(ranges) {
26   // 由于题目并未说有几个站点，因此需要计算出最后一个站点
27   const last = Math.max.apply(null, ranges.toString().split(",").map(Number));
28
29   const diff = new Array(last).fill(0);
30
31   for (let range of ranges) {
32     const [l, r] = range;
33     if (l > r) {
34       diff[l - 1] += 1;
35       diff[0] += 1;
36       diff[r] -= 1;
37     } else {
38       diff[l - 1] += 1;
39       if (r < last) diff[r] -= 1;
40     }
41   }
42
43   const preSum = [diff[0]];
44   let max = preSum[0];
45   let maxI = 0;
46   for (let i = 1; i < last; i++) {
47     preSum[i] = preSum[i - 1] + diff[i];
48     if (preSum[i] > max) {
49       max = preSum[i];
50       maxI = i;
51     }
52   }
53
54   return maxI + 1;
55 }
```

Java算法源码

```
1  import java.util.Arrays;
2  import java.util.Scanner;
3
4  public class Main {
5    public static void main(String[] args) {
6      Scanner sc = new Scanner(System.in);
7
8      int n = sc.nextInt();
9
10     int[][] ranges = new int[n][2];
11
12     for (int i = 0; i < n; i++) {
13       ranges[i][0] = sc.nextInt();
14       ranges[i][1] = sc.nextInt();
15     }
16
17     System.out.println(getResult(ranges));
18   }
19
20   public static int getResult(int[][] ranges) {
```

 伏城之外 [已关注](#)

 0   4   5  [专栏目录](#) [已订阅](#)

Java算法源码

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         int n = sc.nextInt();
9
10        int[][] ranges = new int[n][2];
11
12        for (int i = 0; i < n; i++) {
13            ranges[i][0] = sc.nextInt();
14            ranges[i][1] = sc.nextInt();
15        }
16
17        System.out.println(getResult(ranges));
18    }
19
20    public static int getResult(int[][] ranges) {
21        // 由于题目并未说有几个站点，因此需要计算出最后一个站点
22        int last =
23            Arrays.stream(ranges)
24                .map(range -> Math.max(range[0], range[1]))
25                .max((a, b) -> a - b)
26                .orElse(0);
27
28        int[] diff = new int[last];
29
30        for (int[] range : ranges) {
31            int l = range[0];
32            int r = range[1];
33
34            if (l > r) {
35                diff[l - 1] += 1;
36                diff[0] += 1;
37                diff[r] -= 1;
38            } else {
39                diff[l - 1] += 1;
40                if (r < last) diff[r] -= 1;
41            }
42        }
43
44        int[] preSum = new int[last];
45        int max = preSum[0] = diff[0];
46        int maxI = 0;
47        for (int i = 1; i < last; i++) {
48            preSum[i] = preSum[i - 1] + diff[i];
49            if (preSum[i] > max) {
50                max = preSum[i];
51                maxI = i;
52            }
53        }
54
55        return maxI + 1;
56    }
57 }
```

Python算法源码

```
1 # 输入获取
2 n = int(input())
3 ranges = [list(map(int, input().split())) for i in range(n)]
4
5
6 # 算法入口
7 def getResult(ranges):
8     # 由于题目并未说有几个站点，因此需要计算出最后一个站点
9     last = max(map(lambda x: max(x[0], x[1]), ranges))
10
11     diff = [0 for i in range(last)]
12
13     for ran in ranges:
14         l, r = ran
15         if l > r:
16             diff[l - 1] += 1
17             diff[0] += 1
18             diff[r] -= 1
19         else:
20             diff[l - 1] += 1
21             if r < last:
22                 diff[r] -= 1
23
24     preSum = [0 for i in range(last)]
25     maxV = preSum[0] = diff[0]
26     maxI = 0
27     for i in range(1, last):
28         preSum[i] = preSum[i - 1] + diff[i]
29         if preSum[i] > maxV:
30             maxV = preSum[i]
```

Python算法源码

```
1 # 输入获取
2 n = int(input())
3 ranges = [list(map(int, input().split())) for i in range(n)]
4
5
6 # 算法入口
7 def getResult(ranges):
8     # 由于题目并未说有几个站点, 因此需要计算出最后一个站点
9     last = max(map(lambda x: max(x[0], x[1]), ranges))
10
11     diff = [0 for i in range(last)]
12
13     for ran in ranges:
14         l, r = ran
15         if l > r:
16             diff[l - 1] += 1
17             diff[0] += 1
18             diff[r] -= 1
19         else:
20             diff[l - 1] += 1
21             if r < last:
22                 diff[r] -= 1
23
24     preSum = [0 for i in range(last)]
25     maxV = preSum[0] = diff[0]
26     maxI = 0
27     for i in range(1, last):
28         preSum[i] = preSum[i - 1] + diff[i]
29         if preSum[i] > maxV:
30             maxV = preSum[i]
31             maxI = i
32
33     return maxI + 1
34
35
36 # 算法调用
37 print(getResult(ranges))
```

[复制](#)