题目描述

疫情期间需要大家保证一定的社交距离，公司组织开交流会议。座位一排共 N 个座位，编号分别为[0,N-1],

要求员工一个接着一个进入会议室，并且可以在任何时候离开会议室。

满足：

每当一个员工进入时，需要坐到最大社交距离（最大化自己和其他人的距离的座位）；

如果有多个这样的座位，则坐到索引最小的那个座位。

输入描述

会议室座位总数seatNum。(1 <= seatNum <= 500)

员工的进出顺序 seatOrLeave 数组，元素值为 1，表示进场；元素值为负数，表示出场（特殊：位置 0 的员工不会离开）。

例如 -4 表示坐在位置 4 的员工离开（保证有员工坐在该座位上）

输出描述

最后进来员工，他会坐在第几个位置，如果位置已满，则输出-1。

用例

输入	10 [1,1,1,1,-4,1]
输出	5
说明	• seat -> 0,空在任何位置都行，但是要给他安排索引最小的位置，也就是座位 0 • seat -> 9,要和旁边的人距离最远，也就是座位 9 • seat -> 4,要和旁边的人距离最远，应该坐到中间，也就是座位 4 • seat -> 2,员工最后坐在 2 号座位上 • leave[4], 4 号座位的员工离开 • seat -> 5,员工最后坐在 5 号座位上

题目解析

我的解题思路如下，定义一个seat数组，用于记录已使用的座位号。

当第一个进场时，安排其坐在0号位，seat.push(0)

当第二个进场时，安排其坐在seatNum-1号位，seat.push(seatNum-1)

我们再定义一个onSeat变量，用于记录有几个座位被使用了，如果onSeat>=2时，则，下一个进场人的座位安排逻辑如下：

比如seat = [0,9]，则下一个人应该座位 0 + Math.ceil((9-0-1)/2)，即4号位

比如seat = [0,4,9]，则此时有两个区间 0~4, 4~9，我们应该尽量让人座位区间中间位置

0~4区间中间位置是2，该位置距离左右都是2

4~9区间中间位置是6或7，取较小索引的6，此时距离左边距离2，右边距离3，按照用例意思只保留最小距离2。

因此无论下一个人坐在2号位，还是6号位，距离最近的人距离哦都是2，因此我们选索引较小的2。

然后将选择的座位号记录覆盖到ans中。

最终ans就是题解。

如果下一个人找不到座位，即所有的区间内没有空位，比如2~3，3~4区间内都没有空位，则ans=-1。

如果有人离开，则找到这个人的座位号从seat中删除。

JavaScript算法源码

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

  if (lines.length === 2) {
    const n = parseInt(lines[0]);
    const arr = JSON.parse(lines[1]);

    console.log(getResult(arr, n));
    lines.length = 0;
  }
});

```

```
function getResult(seatOrLeave, seatNum) {
  // seat用于记录已使用的座位号
  const seat = [];
  // ans记录最后进来的人的座位号
  let ans = -1;

  // 遍历进出顺序
  for (let sol of seatOrLeave) {
    // onSeat记录已使用的座位的总数量
    const onSeat = seat.length;

    // 进场，即sol===1,出场，即sol!==1
    if (sol === 1) {
      if (onSeat === 0) {
        // 如果还没人坐，则进来的这个人坐0号座位
        ans = 0;
        seat.push(ans);
      } else if (onSeat === 1) {
        // 如果只有一个人在坐，那么这个人肯定坐在0号位，则下一个人肯定坐到seatNum-1号位
        ans = seatNum - 1;
        seat.push(ans);
      } else {
        // 如果有两个座位或以上被占用，则我们应该让下一个人坐在最大区间的中间位置
        // 比如已占有座位号0,2,4,9，则有区间0~2，2~4，4~9，需要注意的是0是始终占用的，9不一定，因此我们需要考虑边界9的情况
        // 我们只需要求出最大区间，然后记录该区间的起始位置start，以及区间长度一半长度maxDis，即可得出下一个人的最佳座位号
        let maxDis = 0;
        let start = -1;
        seat.reduce((p, c) => {
          // 如果p座位和c座位紧邻，则中间没有空位，因此直接下一个区间
          if (c === p + 1) return c;

          const dis = Math.ceil((c - p - 1) / 2);
          if (dis > maxDis) {
            maxDis = dis;
            start = p;
          }
          return c;
        });

        // seat的0号位的人肯定不会走（题目说的），但是seatNum-1号的人可能会走，因此我们还需要考虑 seat.at(-1) ~ seatNum -
        const brand_dis = seatNum - 1 - seat.at(-1);
        if (brand_dis > maxDis) {
          maxDis = brand_dis;
          start = seat.at(-1);
        }

        ans = start + maxDis;
        if (ans !== -1) {
          seat.push(ans);
          seat.sort((a, b) => a - b);
        }
      }
    } else {
      const idx = seat.indexOf(-sol);
      if (idx !== -1) {
        seat.splice(idx, 1);
      }
    }
  }

  return ans;
}
```

Java算法源码

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

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

    int n = sc.nextInt();
    String str = sc.next();

    Integer[] array =
        Arrays.stream(str.substring(1, str.length() - 1).split(","))
            .map(Integer::parseInt)
            .toArray(Integer[]::new);

    System.out.println(getResult(array, n));
  }

  public static int getResult(Integer[] seatOrLeave, int seatNum) {
    LinkedList<Integer> seat = new LinkedList<>();
    int ans = -1;

    for (Integer sol : seatOrLeave) {
      int onSeat = seat.size();

      // 出场，即sol!=1
      if (sol != 1) {
        // 删除出场位置
        seat.remove(new Integer(-sol));
        continue;
      }

      // 进场，即sol==1
      if (onSeat == 0) {
        // 如果还没人坐，则进来的这个人坐0号座位
        ans = 0;
        seat.add(ans);
      } else if (onSeat == 1) {
        // 如果只有一个人在坐，那么这个人肯定坐在0号位，则下一个人肯定坐到seatNum-1号位
        ans = seatNum - 1;
        seat.add(ans);
      } else {
        // 如果有两个座位或以上被占用，则我们应该让下一个人坐在最大区间的中间位置
        // 比如已占有座位号0,2,4,9，则有区间0~2，2~4，4~9，需要注意的是0是始终占用的，9不一定，因此我们需要考虑边界9的情况
        // 我们只需要求出最大区间，然后记录该区间的起始位置start，以及区间长度一半长度maxDis，即可得出下一个人的最佳座位号
```

```
        int maxDis = 0;
        int start = -1;

        for (int i = 1; i < seat.size(); i++) {
          int p = seat.get(i - 1);
          int c = seat.get(i);

          // 如果p座位和c座位紧邻，则中间没有空位，因此直接下一个区间
          if (c == p + 1) continue;

          int dis = (int) Math.ceil((c - p - 1) / 2.0);
          if (dis > maxDis) {
            maxDis = dis;
            start = p;
          }
        }

        // seat的0号位的人肯定不会走（题目说的），但是seatNum-1号的人可能会走，因此我们还需要考虑 seat.at(-1) ~ seatNum - 1
        int brand_dis = seatNum - 1 - seat.getLast();
        if (brand_dis > maxDis) {
          maxDis = brand_dis;
          start = seat.getLast();
        }

        ans = start + maxDis;
        if (ans != -1) {
          seat.push(ans);
          seat.sort((a, b) -> a - b);
        }
      }
    }
    return ans;
  }
}
```

Python算法源码

```
# 输入获取
import math

seatNum = int(input())
seatOrLeave = eval(input())


# 算法入口
def getResult():
    seat = []
    ans = -1

    for sol in seatOrLeave:
        # onSeat记录已使用的座位的总数量
        onSeat = len(seat)

        # 进场，即sol == 1
        if sol == 1:
            if onSeat == 0:
                # 如果还没人坐，则进来的这个人坐0号座位
                ans = 0
                seat.append(ans)
            elif onSeat == 1:
                # 如果只有一个人在坐，那么这个人肯定坐在0号位，则下一个人肯定坐到seatNum-1号位
                ans = seatNum - 1
                seat.append(ans)
            else:
                # 如果有两个座位或以上被占用，则我们应该让下一个人坐在最大区间的中间位置
                # 比如已占有座位号0,2,4,9，则有区间0~2，2~4，4~9，需要注意的是0是始终占用的，9不一定，因此我们需要考虑边界9的
                # 我们只需要求出最大区间，然后记录该区间的起始位置start，以及区间长度一半长度maxDis，即可得出下一个人的最佳座位
                maxDis = 0
                start = -1

```

```
                for i in range(1, len(seat)):
                    p = seat[i - 1]
                    c = seat[i]

                    # 如果p座位和c座位紧邻，则中间没有空位，因此直接下一个区间
                    if c == p + 1:
                        continue

                    dis = math.ceil((c - p - 1) / 2)
                    if dis > maxDis:
                        maxDis = dis
                        start = p

                # seat的0号位的人肯定不会走（题目说的），但是seatNum-1号的人可能会走，因此我们还需要考虑 seat.at(-1) ~ seatN
                brand_dis = seatNum - 1 - seat[-1]
                if brand_dis > maxDis:
                    maxDis = brand_dis
                    start = seat[-1]

                ans = start + maxDis
                if ans != -1:
                    seat.append(ans)
                    seat.sort()
            # 出现，即sol != 1
            else:
                if -sol in seat:
                    idx = seat.index(-sol)
                    seat.pop(idx)

    return ans


# 算法调用
print(getResult())
```