> 假设有一个考场，考场有一排共N个座位，索引分别是[0..N-1]，考生会陆续进入考场考试，并且可能在任何时候离开考场。
>
> 你作为考官，要安排考生们的座位，满足：每当一个学生进入时，你需要最大化他和最近其他人的距离；如果有多个这样的座位，安排到他到索引最小的那个座位，这很符合实际情况对吧。

分析：

如果将两个考生视作顶点，那么新安排考生就是选择最长线段，将其一份为二。leave(p)就是将相邻的两个线段合并为一个。

在遇到动态最值问题时，肯定要使用有序数据结构，常用的有堆和平衡二叉搜索树。

```
#include <map>
#include <set>
#include <utility>
#include <vector>

class ExamRoom {
public:
  // 构造函数，传入座位总数 N
  ExamRoom(int N) {
    n = N;
    addInterval(std::make_pair(-1, N));
    less::_n = n;
  }
  // 来了一名考生，返回你给他分配的座位
  int seat() {
    int seat;
    // 从有序集合中拿出最长的长度
    std::pair<int, int> longInter = *(pq.end()--);
    int                 x         = longInter.first;
    int                 y         = longInter.second;

    // 情况一
    if (-1 == x) {
      seat = 0;
    } else if (n == y) {
      // 情况二
      seat = n - 1;
    } else {
      // 情况三
      seat = (y - x) / 2 + x;
    }

    // 最长线段分为两个部分
    std::pair<int, int> left{x, seat};
    std::pair<int, int> right{seat, y};
    removeInterval(longInter);
    addInterval(left);
```

```
    addInterval(right);

    return seat;
  }
  // 坐在 p 位置的考生离开了
  // 可以认为 p 位置一定坐有考生
  void leave(int p) {
    // 找出p为端点的线段
    std::pair<int, int> left  = endMap[p];
    std::pair<int, int> right = startMap[p];
    std::pair<int, int> merge = std::make_pair(left.first, right.second);
    removeInterval(left);
    removeInterval(right);

    addInterval(merge);
  }

private:
  void removeInterval(std::pair<int, int> a) {
    pq.erase(a);
    startMap.erase(a.first);
    endMap.erase(a.second);
  }

  void addInterval(std::pair<int, int> a) {
    pq.insert(a);
    startMap[a.first] = a;
    endMap[a.second]  = a;
  }

  struct less {
    bool operator()(std::pair<int, int> a, std::pair<int, int> b) {
      if (distance(a) == distance(b)) {
        return a.first < b.first;
      }
      return distance(a) < distance(b);
    }

    int distance(std::pair<int, int> nums) {
      if (-1 == nums.first) {
        return nums.second;
      }
      if (_n == nums.second) {
        return nums.second - 1 - nums.second;
      }
      return (nums.second - nums.first) / 2;
    }

    static int _n;
  };

  int n;
  //将端点p映射到以p为左节点的线段
  std::map<int, std::pair<int, int>> startMap;
```

```
    // 将端点p映射到以p为右节点的线段
    std::map<int, std::pair<int, int>> endMap;
    // 按照线段长度从小到大存储所有长度
    std::set<std::pair<int, int>, less> pq;
};
```