Midterm Problem 3: Seating Arrangements

Sort m people based on the following rules:

- For all people, their leftmost chair should be in non-decreasing order.
- For people who have the same leftmost chair, there right most chair should be in non-increasing order.

Also, construct an array named isOccupy[ ], isOccupy[i] indicates that whether the ith chair is occupied (true = occupied, false = available). By construct this, we can know whether people can select the chair in O(1) time. And this array needs to be initialized to false.

Then just traverse all people with the order stated above, and for each person, we select the chair that as left as possible (by traversing the isOccupy[ ] array to know if a specific position is available).

The pseudocode is as follows:

$seat\_arrange(n, m, r[\,], l[\,])$
  $people[\,] = struct\ \{r[\,], l[\,]\}$
  $sort\ people[\,]\ using\ the\ previous\ rule$
  $isOccupy[\,] = false$
  $count = 0$
  $for\ i\ from\ 1\ to\ m\ do$
    $for\ j\ from\ people[i].l\ to\ people[i].r\ do$
      $if\ isOccupy(j) = false\ then$
        $count++$
        $isOccupy(j) = true$
        $break$
  $return\ count$

Correctness proof:

Let A = {p1, p2, ⋯ , pn} be the person that seat on the corresponding seat calculated by the algorithm, because the way we sort people and select seat, p1, p2, ⋯ , pn must be in the order that we sorted. Otherwise, it will break the rule that we use for sorting. Suppose B = {b1, b2, ⋯ , bn} be an optimum solution with person seat in the corresponding seat , and we sort the people in this solution using the same rule that we used before. This can be done, since no matter how l[ ] and r[ ] looks like, we can sort the using the rule. Let the sorted solution be C = {c1, c2, ⋯ , cn}.

We want to show that a1, ⋯ , a(n) is some kind equivalent to c(1), ⋯ c(n).

Because there is an possibility that the seat has no one to seat, then we allow some p or c can be ϕ.

Due to the way we sort and select, we are choosing as left as possible. So p1 must be the one that can seat at the leftmost seat that is available. This means that if c1 = p1, then our base case is equivalent. If c1 != p1, but p1 occurs in C (say at cj), then this means that even p1 can appear at the jth seat, then c1 can also appear at that seat j. then we can swap them to guarantee that position 1 is equivalent. If p1 does not occur in C, then we can simply change c1 to be p1, because p1 is legal to be put at that seat.

Similarly, we can do this step from beginning to the end. And finally, we can change C into P without reduce the number of people.

Therefore, P is an optimal solution.

The running time of this algorithm is sort $O(m \log m)$ + do the greedy $O(mn)$

So the total running time is $O(mn + m \log m)$