Code Jam 2009 - Round 2

Analysis: Crazy Rows

Reformulation of the problem

It is easy to see, for each row, only the position of the last '1' matters. We can re-formulate the problem

> *CR*: Given a list of numbers (a_0, ... a_{N-1}). You are allowed to swap adjacent numbers. Find the minimum number of swaps needed so that the i-th number is at most *i* in the final configuration.

The well known special case

Perhaps many of you know the following special case.

> *CR**: Given a permutation (x_0, ... x_{N-1}) of the numbers 0 to *N-1*. You are allowed to swap adjacent numbers. Find the minimum number of swaps needed in order to sort the list in increasing order.

Perhaps you also know the complete solution to CR*. It is very simple and elegant, and important to our problem. So we produce here.
Solution (to CR).* Define the *disorder* of the list to be the number of pairs i < j, where $x_i > x_j$. In one swap (of adjacent numbers), the total number of disorder is changed by exactly one. (Check it!) Therefore, let the disorder of initial configuration be *D*, you need at least *D* swaps. *D* swaps is also sufficient -- as long as the list is not sorted, there exist adjacent pairs in wrong order. You can swap any of such pairs and decrease the disorder by 1. ◊

In particular,

> (1) One type of optimal solutions of CR* involves first to swap the number 0 all the way to the left, then let it stay there forever.

Solution to our problem

Imagine we know which of the a_i's will finally go to position 0, and which one will go to position 1, etc., then we can simply use the algorithm for CR*. But there might be multiple candidates for a single position. For example, there might be several i's such that $a_i = 0$, and even some $a_i = -1$.

Below is the judge's C++ solution. b[i] is the "decoded" position where a[i] will be in the final configuration. The algorithm says: For those candidates for position 0, pick the leftmost one. Then in the rest, for those candidates for position 1, pick the leftmost one. And so on.

```
// -1 means no position is assigned for a[j].
for(i=0;i<N;i++) b[j]=-1;
for(i=0;i<N;i++) {
  for(j=0;j<N;j++) if(b[j]<0 && a[j]<=i) {
      b[j]=i; break;
```

```
    }
}
int r=0;
for(i=0;i<N;i++) for(j=i+1;j<N;j++)
  if(b[i]>b[j]) r++;
// output r as the answer
```

Note that, once the b[i]'s are fixed, you only need to count the disorders as in CR*; no real swapping needs to be simulated.

Proof of our solution

The key observation is, for multiple candidates for position 0, you will never need to swap any two of them. Suppose you do swap, say *u* and *v*, both are at most 0. I can simply *ignore* it, and *pretend* that they are swapped (i.e., exchange the roles of *u* and *v* thereafter). The final configuration is still a good one. Thus we proved that, for all the candidates for position 0, the leftmost one, call it *u**, will finally go to position 0.

Now, imagine we have decoded the final positions for every one. Then (1) tells us that there is one solution where we first move *u** all the way to the left, and never worry about it again. Therefore we now face the next question: Which of the rest of the numbers should go to position 1?
This is exactly the same problem, but with one number fewer. ◊