X Junior Balkan Olympiad in Informatics

Skopje, 2016

Day 1: Swaps

Short description:

One possible solution is to perform dynamic programming. Let's denote as dp[N][M] as the minimum number of swaps needed to sort the first N numbers with the last number being less than or equal to M. To solve the problem, go through all the numbers, and for each number generate all the possible numbers you can get by swapping a number of bits (this can be done in a preprocessing stage).For each of the generated numbers, calculate the minimum number of swaps needed to have the list sorted with the last number being less than or equal to than number. Since we are calculating this number for each prefix of our numbers, we can apply dynamic programming.

Let the number we are currently on be in position i, the generated number we are processing be A, and the number of swaps that are needed to get from the number in position i to A be S. For calculating dp[i][A] the following rule applies:

dp[i][A] = minX <= A { (dp[i-1][X]) + S }

As we need to check all the possible values for the number at position i-1 and we need to add the number of swaps we performed to change the i-th number. The final solution we need to print is:

minX <= M {dp[N][X] }. That is, the minimum number of swaps needed to sort the first N numbers with the last number being <= X.

***Analysis*:**

Note that in our dp equation, we are relying only on the previous number, so we can transform our state to dp[2][M] and alternate between using dp[0][M] and dp[1][M].Another observation is that we only care about the value of the previous number in the list so that we can tell if it's smaller the current number and to minimize the number of total swaps. Which means that if our generated numbers are sorted in ascending order, we can do just one pass through the possible values for the number in position i-1, always storing the minimum number of total swaps. As an example, if we can change the number at position i to be either A for a swaps or B for b swaps, where A < B, we process B normally first while saving X = (dp[i-1][y] for all y, such that 1 <= y <= B). Now when processing A, we can skip the first B numbers, and have:

dp[i][A] = min(X, min(dp[i-1][y] for all y, such that B+1 <= y <= A)) + a.

The complexity of our final solution is O(N*M), since every state in our dp table is calculated in O(1) time.