Solution to the Problem 5.4: Make It Sorted

Main Idea

Given an array $A[1..n]$ of positive integers. We can add either 1 or -1 to any element of the array. We need to find the minimum number of these operations required to make the array sorted in non-decreasing order (this means that for each $1 \le i \le n-1$ a condition $A[i] \le A[i+1]$ must be fulfilled).

Let C be $\max_{1 \le i \le n} A[i]$. According to the statement, $C \le 1000$.

We claim that it is unprofitable to make any element less than 1 or more than C. Indeed, for every possible sorted integer array $A[1..n]$ we can consider new array $A'[1..n]$ such that $A'[i] = \max(1, \min(A[i], C))$. It's not hard to notice that $A'[1..n]$ will also be sorted, and the number of operations to obtain $A'[1..n]$ from the initial array does not exceed the number of operations to obtain $A[1..n]$.

We will solve this problem using the dynamic programming technique. Define $dp[i][x]$ as the answer to this task for the array $A[1..i]$ such that $A[i] = x$. On the basis of the above, the answer to this task will be equal to $\min_{1 \le x \le C} dp[n][x]$.

It remains to describe the transitions between different states. Consider them as appending $(i+1)$-th element to the array $A[1..i]$. When we append a new element y to the sorted array, it remains sorted if and only if its last element does not exceed y. It means that we can make a transition from $dp[i][x]$ to every $dp[i+1][y]$ such that $x \le y$. Cost of this transition will be equal to $|y - A[i+1]|$, because we will need exactly this number of additions or substractions.

To sum up,

$$\begin{aligned} &dp[1][x] = |x - A[1]|; \\ &dp[i+1][y] = \min_{1 \le x \le y} \Big(dp[i][x] + |y - A[i+1]| \Big) \text{ for each } 1 \le i \le n-1, 1 \le y \le C. \end{aligned} \tag{1}$$

The only problem is that if we compute all $dp[i][x]$ using the formula above, it will take $O(n \cdot C \cdot C)$ time. It's too much. But we actually can get rid of extra C using an array of prefix minimums.

Let $prefixMin[i][x] = \min_{y=1}^{x} dp[i][y]$. Then, obviously, for each $1 \le i \le n-1$

$$prefixMin[i][x+1] = \min(prefixMin[i][x], dp[i][x+1]) \text{ for each } 1 \le x \le C-1, \tag{2}$$

and, according to the formula (1),

$$dp[i+1][y] = prefixMin[i][y] + |y - A[i+1]| \text{ for each } 1 \le y \le C. \tag{3}$$

It means that we can compute arrays *prefixMin* and *dp* in $O(n \cdot C)$ time and $O(n \cdot C)$ memory.

Implementation Details

It is most convenient to compute arrays *prefixMin* and *dp* sequentially from 1 to n. We need only i-th row of the array *dp* in order to compute $(i+1)$-th row. So this will allow, if necessary, to store only i and $(i+1)$-th rows instead of storing arrays entirely.

First we will compute the initial values:

```
C ← 0
for i from 1 to n:
  C ← max(C, A[i])
for x from 1 to C:
  dp[1][x] ← |x − A[1]|
```

Then we will compute arrays *prefixMin* and *dp*:

```
for i from 1 to n−1:
  prefixMin[i][1] ← dp[i][1]
  for x from 1 to C−1:
    prefixMin[i][x+1] ← min(prefixMin[i][x], dp[i][x+1])
  for y from 1 to C:
    dp[i+1][y] ← prefixMin[i][y] + |y − A[i+1]|
```