

The arithmetic exercise

Subgroup 1.

Let all x_i be equal to x . In the first subgroup, it is proposed to apply changes to the zero elements of the array at each step, if possible. If there are no zero cells, we will apply changes to the very last element of the array. Then the last cell will alternate between the values $x, -x, x, \dots$

Subgroup 2.

There are 2^m different ways to make a sequence of changes, as each of the m operations can be applied either to the first or the second element a . We will enumerate all possible sequences of changes and calculate the sum after their application. Among all such sums, we will take the maximum. This solution works in $O(m \cdot 2^m)$.

Subgroup 3.

We will use dynamic programming. Let $dp[pref][i][j] = true/false$ — can we process the first $pref$ changes so that $a_1 = i, a_2 = j$ holds. Note that i and j can be negative. It is clear that a_1 and a_2 cannot exceed the absolute value of the sum of the values x_i . The sum of all x_i does not exceed $10m$, so we can store values in the range from -500 to 500 .

Subgroup 4.

We will improve the dynamics from the previous group. Notice that for a fixed value of a_1 on the prefix, it makes sense to recalculate only through the minimum or maximum available a_2 . Therefore, we will store the explicit value of only one of the elements of the array. We will introduce $dp_{min}[pref][i]$ and $dp_{max}[pref][i]$, which store the minimum and maximum achievable value of a_2 on the prefix if $a_1 = i$.

Note.

Let's make a remark that will help us significantly advance towards the solution. It is clear that each element of the sequence x_i will enter the final sum either with a «+» sign or with a «-» sign. Let's divide all m elements into n sequences, depending on which position of the array a the change was applied to. Some of the sequences may be empty. In each sequence, the signs of the elements alternate, and the last element always has a "+" sign. Therefore, if we replace the signs with $+1$ and -1 respectively, the sum on each suffix of such arrays will be either 1 or 0 . We will combine all sequences back into one and look at the sum of the signs on an arbitrary suffix. It will be in the range $[0; n]$. Now we can use this as a criterion to check the correctness of the sequence of signs.

Subgroup 5.

We will iterate over the sign for each x_i and check that the described criterion holds for each suffix. The solution works in $O(m \cdot 2^m)$.

Subgroup 6 and 7.

We will again use dynamic programming. Let $dp[i][j]$ be the maximum sum that can be obtained on suffix i , if the current balance of signs equals j . Then $dp[i][j] = \max(dp[i+1][j-1] + x_i, dp[i+1][j+1] - x_i)$. This solution works in $O(nm)$.

Subgroup 8.

In this subgroup, it is proposed to act greedily, adhering to the balance criterion. For example, we can go from the end of the sequence x_i . At each step, we will try to take a new element and, if necessary, discard some element that was taken earlier. Since there are at most two different values, it makes sense to discard only the minimum. We will maintain a queue of minima that have been taken with a "+" sign and that can still be discarded.

Subgroup 9 and 10.

There are several alternative approaches. Let's consider a few of them.

1. We will improve the solution for the previous subgroup. We will go from the end, storing the current arrangement of signs for the elements of the suffix. In the segment tree, we will maintain elements for

which we can change the sign to the opposite. When we move to a new element, we try to assign it a «+» sign (if this is not possible, we change the sign of the minimum element with a «+» sign for which this can be done) and see how the sum changes as a result of this action. Similarly, we try to assign a «-» sign and see how the sum changes. Among the two options, we choose the one that yields the highest sum.

2. We will sort the sequence x_i in descending order of absolute value. We will go through the sorted values and at each step try to assign the desired sign to the element (if the number is positive — «+» else «-»), if possible. Otherwise, we will assign the sign that we are forced to assign. To understand whether we can assign a particular sign, we will maintain a segment tree, at each position of which the current balance of the corresponding suffix is stored. If we want to assign the next sign, we need to look at the minimum and maximum balance that is already achieved before it and check whether the remaining signs can be arranged so that the balance criterion is not violated.

3. Let's recall the solution for subgroups 6 and 7, which uses dynamic programming. Notice that the function is convex separately for the two parities, so we can use the slope trick.