Problem A. Natives

Author and Developer: Andrew Stankevich

Note that it is beneficial to give away least valuable treasures. Let us sort all treasures by increasing of their value and give the first $\lceil n/2 \rceil$ treasures to the natives. To find the number k of treasures that should be given away we can use the formula `k = (n + 1) / 2`, here '/' denotes integer division.

Let us give sample programs in C++:

```
sort(a.begin(), a.end());
int sum = 0;
for (int i = (n + 1) / 2; i < n; ++i){
    sum += a[i];
}
cout << sum << '\n';
```

and Python:

```
a.sort()
print(sum(a[(n + 1) // 2:]))
```

Problem B. Balanced Illumination

Suggested and developed by Andrew Stankevich

The statement developes so called balanced Gray code. It was first described in words by Adam, Bakos, Cohn, Pollac, Robinson, Tootill and others in 1950s.

Let us consider constructive algorithm that generates this code.

Any Gray code for $n = 1$ and $n = 2$ is OK. Balanced Gray codes for $n = 3$ and $n = 4$ are shown as sample test. Let us show how to convert balanced Gray code for n to a balanced Gray code for $n + 2$.

Let us first note that any Gray code is uniquely identified by a sequence of 2^n integers from 1 to n— indices of bits that are inverted when moving to the next code. For example, for $n = 3$ and Gray code in sample test the corresponding sequence is $[2, 3, 1, 3, 2, 3, 1, 3]$. Note that c_i is the number of occurrences of i in this sequence. We will now consider such sequences instead of codes.

Additional notes. Cyclic shift of the Gray code sequence is also the correct Gray code sequence with the same values of c_i. We can also apply any substitution π changing all occurrences of i in the sequence to $\pi[i]$.

Let us have a Gray code for n, let us mark l positions in it where l is odd. Let us denote the segment between adjacent marked positions up to the i-th one as α_i, an element on the i-th marked position as j_i. Let us denote β reversed as β^R.

Now if "$\alpha_1 j_1 \alpha_2 j_2 \dots \alpha_l j_l$" is the Gray code for n, then "$\alpha_1(n+2)\alpha_1^R(n+1)\alpha_1 j_1 \alpha_2(n+1)\alpha_2^R(n+2)$ $\alpha_2 j_2 \alpha_3(n+2)\alpha_3^R(n+1)\alpha_3 j_3 \dots j_l \alpha_l(n+2)\alpha_l^R(n+1)\alpha_l(n+2)j_{l-1}\alpha_{l-1}^R j_{l-2}\alpha_{l-2}^R \dots j_1\alpha_1^R(n+1)$" is the Gray code for $n + 2$.

What is left is to describe which positions to mark in order to get balanced Gray code for $n + 2$ from balanced Gray code for n. The following algorithm does it. First apply a substitution to make c_i values sorted. Then make a cyclic shift to put 1 to the last place. Choose c'_i as even integers equal to either $2\lfloor 2^{n+1}/(n+2) \rfloor$, or $2\lceil 2^{n+1}/(n+2) \rceil$ with the sum equal to 2^{n+2}, sorted in increasing order. Set $b_i = 2c_i - c'_{i+2}/2$. Decrease b_1 by one. Now mark b_i occurrences of i, including the last occurrence of one. Now apply the transformation from the previous paragraph to get a balanced Gray code for $n + 2$.

Unfortunately, this, already quite cumbersome, construction doesn't work for $n = 3$. However, there are many ways to mark 5 positions at Gray code for $n = 3$ to succeed, including, for example, marking one 1 and four 2-s.

Problem C. How Many Strings Are Less

Author: Evgeny Karpovich; developers: Evgeny Karpovich, Grigorii Shovkoplias

To solve this problem, we will use the trie data structure. Let's insert all the strings of the set D in trie

Each vertex of the trie corresponds to some prefix of at least one string of the set. We want to maintain the relevant vertex — the largest prefix of the current version of the string s, which is in the trie.

Then, we will use dynamic programming. Let's calculate the value `dp[v][c]` for each vertex v of the trie. `dp[v][c]` is vertex of the trie that will become relevant if the vertex v corresponds to some prefix of the string s and the modification with character c begins in the next character. We will also calculate the value `ans[v][c]` — how many strings in the trie are less than the prefix corresponding to vertex v if the next character is c.

How to handle the modifications? We maintain a vertex in the trie and know the next character of the current version of the string s.

- If the modification does not affect the current prefix and the character after it, then such modifications can be ignored, they will not affect the response.
- Otherwise, we will go to the vertex of the trie such that the corresponding prefix is equal to the prefix of the new version of the string s. And then we can use the value `dp[v][c]` to find the new relevant vertex, the next character will obviously be c.

The solution works in $\mathrm{O}(\alpha \cdot \sum |D_i| + q)$. Here α is the size of the alphabet, and $\sum |D_i|$ is the total length of the strings in D.

Problem D. Exam registration

Author: Sergey Kopeliovich, developer: Grigoriy Khlytin

To solve this problem, we will use a binary search for the answer.

Let ans — the current answer for which we want to check the property: is it possible to achieve that the dissatisfaction of any student does not exceed ans (note that this function is monotonic, that is, if we can achieve that the dissatisfaction of any student does not exceed ans, then it does not exceed any k, where $k > ans$).

To check this property for the answer ans, for each student we will try to shift it by the maximum number of days w to the left, such that $w \leq ans$.

This will be with the complexity $O(n)$, because it is enough to go from left to right on all the days i for which students are signed up, and maintain the position j indicating the leftmost "free" day (for which there are still free places). We need to check if j is from i no further than ans days, and at the same time we can move x students from day i to day j — then we will do this for the largest x.

If the problem has an answer other than -1, then this answer will be in the range from 0 to $n-1$ inclusive (because when a student moves from the first to the last day, his dissatisfaction will be equal to $n-1$). Based on this fact, binary search will work for $O(\log n)$.

The final asymptotic of the solution will be $O(n \cdot \log n)$.

Problem E. Fair Robbery

Problem author and developer: Daniil Oreshnikov

Let's consider any k. We can note that only the minimal and maximal amounts of money on both left and right parts of the street affect the answer. By the left side we mean the people who are not getting robbed and on the right side there are people who lose a fraction of t of their money. Also, let's note that

maximizing the amount of stolen money is equivalent to maximizing t which can be deduced from the formula for b given in the statements.

Let's introduce the following values: $\min_{\text{left}} = \min(a_1, \dots, a_{k-1})$, $\max_{\text{left}} = \max(a_1, \dots, a_{k-1})$ and, on the right side, $\min_{\text{right}} = \min(a_k, \dots, a_n)$, $\max_{\text{right}} = \max(a_k, \dots, a_n)$. Case of $k = 1$ can be considered separately since left-values are undefined. In this case the least unfair plan has $t = 1$ since all townspeople losing all their money will result in $\max(a^{\text{new}}) - \min(a^{\text{new}})$ being minimal possible and equal to 0 while the amount of stolen money is maximized.

For $k > 1$ it is enough to consider the four said values. Global maximum after the robbery equals to $\max(\max_{\text{left}}, (1-t)\max_{\text{right}})$ while minimum equals to $\min(\min_{\text{left}}, (1-t)\min_{\text{right}})$. To find an optimal t let us consider the cases of possible relations between them:

1. $\max_{\text{left}} \geq \max_{\text{right}}$ and $\min_{\text{left}} \geq \min_{\text{right}}$
   In this case $\max(a^{\text{new}}) = \max_{\text{left}}$ since right side maximum can not increase. And in the same way $\min(a^{\text{new}}) = (1-t)\min_{\text{right}}$. The difference between them is minimal when $t = 0$.

2. $\max_{\text{left}} \geq \max_{\text{right}}$ and $\min_{\text{left}} < \min_{\text{right}}$
   Same in this case, maximum always equals to $\max_{\text{left}}$, so the smallest difference is obtained with largest minimum. It's guaranteed to be less than or equal to $\min_{\text{left}}$, and maximal t for which holds $\min(a^{\text{new}}) = \min_{\text{left}}$ also implies that $\min_{\text{left}} = (1-t)\min_{\text{right}}$, so the answer is $t = 1 - \frac{\min_{\text{left}}}{\min_{\text{right}}}$.

3. $\max_{\text{left}} < \max_{\text{right}}$ and $\min_{\text{left}} \geq \min_{\text{right}}$
   Same as in the first case, $\min(a^{\text{new}})$ equals to $(1-t)\min_{\text{right}}$. And the maximum depends on t in following way: for $t \leq 1 - \frac{\max_{\text{left}}}{\max_{\text{right}}}$ the maximum equals to $(1-t)\max_{\text{right}}$ reaching its bottom value $\max_{\text{left}}$ in $t = 1 - \frac{\max_{\text{left}}}{\max_{\text{right}}}$, after which it stops decreasing. Since minimal value continues to decrease, larger t will give larger unfairness. And with t in this interval the unfairness equals to $(1-t)(\max_{\text{right}} - \min_{\text{right}})$ hitting its minimal value in maximal possible $t = 1 - \frac{\max_{\text{left}}}{\max_{\text{right}}}$.

4. $\max_{\text{left}} < \max_{\text{right}}$ and $\min_{\text{left}} < \min_{\text{right}}$
   Joining the ideas mentioned above we can deduce that before t reaches the threshold of $1 - \frac{\max_{\text{left}}}{\max_{\text{right}}}$ the maximum value keeps decreasing. In the same time either minimum equals to $\min_{\text{left}}$ in which case the unfairness keeps decreasing, or the unfairness itself equals to $(1-t)(\max_{\text{right}} - \min_{\text{right}})$ and also decreases.
   If for such t the inequality $\min_{\text{left}} < (1-t)\min_{\text{right}}$ holds, we may increase t by making it equal to $1 - \frac{\min_{\text{left}}}{\min_{\text{right}}}$, so that the resulting unfairness doesn't change while the total amount of money stolen increases. This means that the answer is $t = 1 - \min\left(\frac{\max_{\text{left}}}{\max_{\text{right}}}, \frac{\min_{\text{left}}}{\min_{\text{right}}}\right)$.

Let's note that the answer for the last case also works in all other cases. So the general case answer can be obtained as

$$t = \max\left\{0; 1 - \frac{\max_{\text{left}}}{\max_{\text{right}}}; 1 - \frac{\min_{\text{left}}}{\min_{\text{right}}}\right\}.$$

Another way to achieve the same answer is to notice that the only case in which it's disadvantageous to increase t is when $\max(a^{\text{new}}) = \max_{\text{left}}$ and $\min(a^{\text{new}}) = (1-t)\min_{\text{right}}$. Joining together all required limits for t gives the same answer as before.

So in the end, for a certain k the answer can be found at $\mathcal{O}(1)$ time complexity if the values $[\min \mid \max]_{\text{left}|\text{right}}$ are known. These values can be pre-calculated in $\mathcal{O}(n)$ time complexity with two linear iterations over a in both directions (prefix- and suffix- minimums and maximums).

Problem F. Counting Antibodies

Problem author and developer: Rita Sablina

Note, that there are $V_h \cdot D_h \cdot J_h$ potential heavy chains — for each chain one variant from V_h, D_h and J_h for fragments V, D and J is selected, respectively.

Similarly, there are $V.J_\kappa$ potential type κ light chains and $V_\lambda \cdot J_\lambda$ potential type λ light chains.

Each heavy chain is bonded with κ or λ light chain. In the immunoglobulin molecule one pair of heavy and light chain, but chains in the both pairs are similar. So, total count of potential immunoglobulin molecules can be counted with the formula $(V_\kappa \cdot J_\kappa + V_\lambda \cdot J_\lambda) \cdot V_h \cdot D_h \cdot J_h$

This problem demonstrates that it is sometimes quite difficult for a programmer to understand some narrow subject area.

Problem G. The Math of Sailing

Problem author and developer: Daniil Oreshnikov

Let's look at the equality between the maneuverability and stability: $a_1a_4 + a_2 + a_3 = a_1 + a_4 + a_2a_3$. Note that xy can be rewritten as $(x-1)(y-1)+x+y-1$ which means that if we denote the sum $a_1 + a_2 + a_3 + a_4$ as A, we can rewrite maneuverability as

$$m = (a_1 - 1)(a_4 - 1) + a_1 + a_4 - 1 + a_2 + a_3 = A - 1 + (a_1 - 1)(a_4 - 1)$$

and stability in the same way as

$$s = A - 1 + (a_2 - 1)(a_3 - 1).$$

This means that we have to achieve an equality of

$$m - A + 1 = \Big[(a_1 - 1)(a_4 - 1) = (a_2 - 1)(a_3 - 1)\Big] = s - A + 1.$$

Since the order of the sails in these equations is unambiguous we can denote the minimum of all sails sizes as a_1 and the minimum of middle mast sails sizes as a_2. In that case it's sufficient to find an answer for which the inequality $a_1 \le a_2 \le a_3 \le a_4$ holds.

Let's consider initial sizes of pieces of fabric in non-decreasing order: $t_{p_1} \le t_{p_2} \le t_{p_3} \le t_{p_4}$. This permutation p will later form the first line of an output. Let's denote by q_i the value $t_{p_i} - 1$, the i-th size of piece of fabric in non-decreasing order, decreased by 1. We can note then that $(a_1 - 1)(a_4 - 1) = (a_2 - 1)(a_3 - 1)$ both have an upper limit of $\min(q_1 \cdot q_4, q_2 \cdot q_3)$. Indeed, if these products are equal and each factor does not exceed a certain q_i due to the problem statement then both products do not exceed the minimal product of two values q_i forming them. Let's consider all cases of which q is "paired" with q_4. We get that if q_1 is paired with q_4 then each of those products can't exceed the desired limit of $q_1 \cdot q_4$ and $q_2 \cdot q_3$ respectively. And if q_4 is paired with q_2 or q_3 then the other product will not exceed $q_1 \cdot q_2$ or $q_1 \cdot q_3$ respectively, both of which do not exceed neither $q_2 \cdot q_3$ nor $q_1 \cdot q_4$.

With this we have estimated that $m - A + 1 = s - A + 1$ do not exceed $\min(q_1 \cdot q_4, q_2 \cdot q_3)$. Now we will prove that this estimation is exact. More than that, we'll prove that to reach this limit only one of the pieces of fabric should be cut, which allows us to maximize the sum of sizes A alongside their products. And if both of those values are maximized then m and s are maximized as well.

So if $q_1 \cdot q_4 < q_2 \cdot q_3$ let's decrease the value of $q_2 \cdot q_3$ down to $q_1 \cdot q_4$ without changing the values of q_1 and q_4 and vice versa. We only have to determine how to decrease the higher product. It's sufficient to consider the case of $q_1 \cdot q_4 < q_2 \cdot q_3$ since the other one is similar. In this case we have to choose $a_1 = q_1 + 1$ and $a_4 = q_4 + 1$. To understand which a_2 and a_3 give the best result we can recall that $s = A - 1 + (a_2 - 1)(a_3 - 1)$ where A equals to $a_1 + a_2 + a_3 + a_4$. So basically we have to maximize the sum $a_1 + a_2 + a_3 + a_4$ for a fixed product of $(a_2 - 1)(a_3 - 1)$. And since both a_1 and a_4 are also already selected and fixed, we just need to maximize the sum of $(a_2 - 1)$ and $(a_3 - 1)$ with their product being a fixed number. It's a common fact that the sum of such factors increases with them moving one from another, so such sum is maximized when only the smaller factor gets decreased.

The resulting sails sizes will be equal to $q_1 + 1, \frac{q_1 q_4}{q_3} + 1, q_3 + 1, q_4 + 1$. Similar for the case when $q_1 \cdot q_4 \ge q_2 \cdot q_3$, the optimal answer is $\frac{q_2 q_3}{q_4} + 1, q_2 + 1, q_3 + 1, q_4 + 1$. As for the permutation p, we choose the permutation used to sort t_i in non-decreasing order mentioned above. The time complexity of this solution is $\mathcal{O}(1)$.

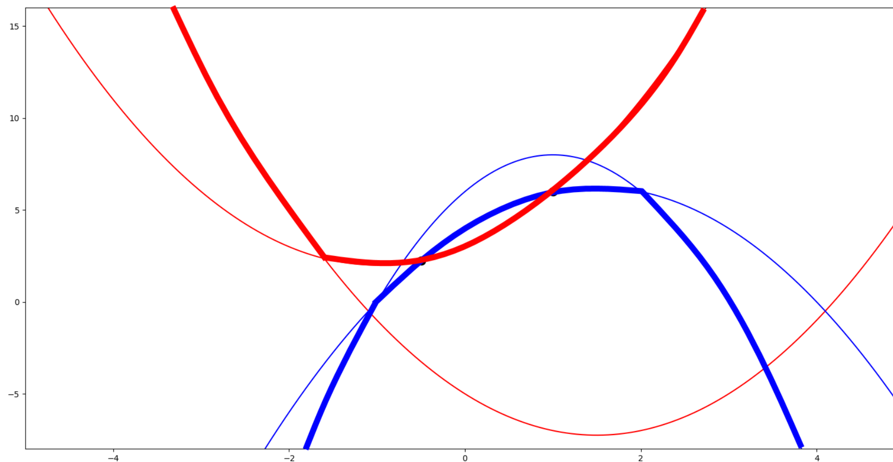
Problem H. Lots of Parabolas

Author: Mikhail Pyaderkin, developer: Sergey Melnikov, tutorial: Andrew Stankevich

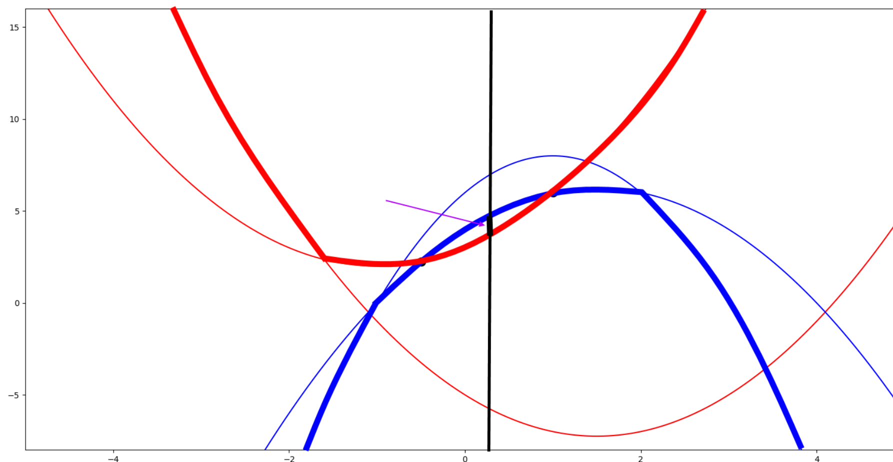
It turns out that it is possible to use half-plane intersection algorithm for this problem. It works for any convex functions.

Consider all parabolas with $a > 0$, that is, ones for which the point must be above the parabola. For each possible x find the topmost point for these parabolas. The resulting function $low(x)$ is concave.

Similarly consider all parabolas with $a < 0$, that is, ones for which the point must be below the parabola. For each possible coordinate x find the bottom point for these parabolas. The resulting function $up(x)$ is convex.



Now consider the difference $f(x) = up(x) - low(x)$. This function is convex. Its maximum can therefore be found using ternary search. To calculate the value of f at x we evaluate all quadratic functions at x and choose maximum from among those with $a > 0$ and minimum from among those with $a < 0$.



The maximum value of f is positive, we use the corresponding x as the answer. Find maximum from parabola coordinates with $a > 0$ and minimum from among those with $a < 0$, the take their average as y.

Problem I. Wheel of Fortune

Author: Gennady Korotkevich, developer: Nikolay Vedernikov, translation: Rita Sablina

If there is only one word in the stolen list, Katya will win.

Otherwise if there is a letter that doesn't occur in all words in the list, Katya couldn't win with a guarantee: for any letter she names there will be a word without this letter. So if this word is hidden, she will lose.

Katya names a letter that occurs in all words in the list. If there are several such letters, Katya can name any of them. Naming a letter splits the list in several groups. In one group there will be a words with the same position of named letter.

For each group algorithm without already named letter will be repeated.

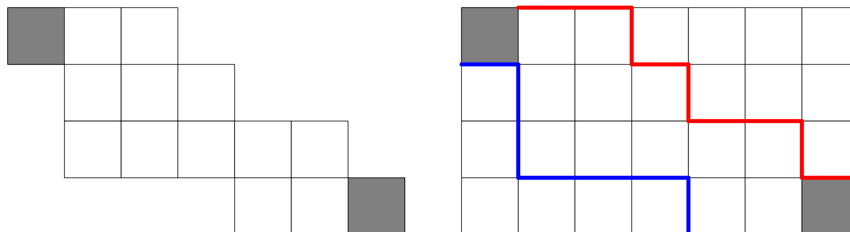
In other words, letter splits words set in subsets, for each subset the algorithm runs recursively. There are 26 letter, so maximum recursion depth will be 26. Each word on each depth will occur only once. The solution works in $O(s \cdot N \cdot L)$, where s is the size of the alphabet.

Problem J. Yurik and Woodwork Lesson

Author: Andrew Stankevich, developer: Mike Perveev

To begin with, let's understand that the last two conditions that a nice board must satisfy, mean that in each row some, possibly zero, number of cells on the left and on the right should be cut away. What is more, the number of cells that were cut away on the left in row i should be not less that the number of such cells in row $i-1$. Similarly, the number of cells that were cut away on the right in row i should be not greater than the number of such cells in row $i-1$. If these conditions are not satisfied, there will exist a column that has the fifth condition not satisfied.

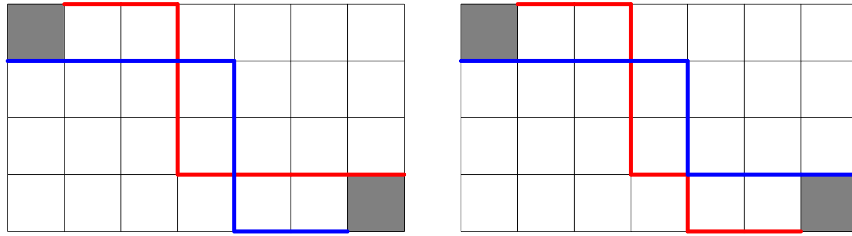
Such reasonings lead to the fact that any correct way to cut away some cells is fixed in two paths of the following form:



To satisfy the third condition, paths should not intersect each other. So, the problem now is to calculate the number of pairs of not intersecting paths that begin and end in points shown in the picture.

Let's calculate the number of pairs of all paths. Each path consists of $N+M-2$ segments of length 1. Each of them goes down or right. So, the number of all paths is $\binom{N+M-2}{M-1}$ because we need to select $M-1$ horizontal segments among $N+M-2$ segments. So the number of pairs of paths is $\binom{N+M-2}{M-1}^2$.

Now we need to subtract the number of pairs of intersecting paths from the answer. Consider a picture of a pair of intersecting paths and note that each pair of such paths unambiguously corresponds to another pair and vise versa. Consider the last intersection point of two paths. It exists because we consider only pairs on intersecting paths. After this point paths never intersect each other. Let's take ends of two paths and swap it with each other: the end of the first path will be the end of the second one and vise versa. Example of such transformation is illustrated on the picture below:



The number of pairs of such paths equals to $\binom{N+M-2}{M-2} \cdot \binom{N+M-2}{N-2}$. So the answer is: $\binom{N+M-2}{M-1}^2 - \binom{N+M-2}{M-2} \cdot \binom{N+M-2}{N-2}$. We can transform it into a more simple form: $\frac{\binom{N+M-1}{M} \cdot \binom{N+M-1}{M-1}}{N+M-1}$.

Problem K. Railroad sorting

Author and developer: Andrew Stankevich

We will send the cars to the exit track one at a time in ascending order.

If the i-th car is still on the input track by the time when the first $i-1$ cars is already on the output track, then we send all cars in front of the i car on the input track and itself to the first dead end, after which let's send it from the first dead end to the output track. This requires no more than $n-1$ commands to move the previous cars, and two more commands to move the i-th car to and from the first dead end.

If the i-th car is in one of the dead ends, then we send all the cars in front of the same dead end to another dead end, and this car itself is sent to the output track. This requires no more than $n-1$ commands to move the previous cars, and one more command to move the i-th car out of the dead end.

As a result, no more than $n+1$ commands are used for each of the n cars, which means that in total no more than n^2+n commands are required, which is less than $2\ cdot 10^6$.

Problem L. Birthday

Author, developer: Evgeny Karpovich, translation: Mikhail Dvorkin

For simplicity, let's consider $a_i \le b_i$, otherwise just swap them.

Let's first solve the problem on the entire array. Note that if the sum of all b_i is not divisible by k, it is the desired answer, otherwise, we need to select exactly one card and take its a_i instead of b_i, but having $a_i \bmod k \ne b_i \mod k$, and minimizing the value $b_i - a_i$.

Let's make two arrays c and $pref$. If $a_i \bmod k \ne b_i \bmod k$, then $c_i = b_i - a_i$, otherwise $c_i = +\infty$. A $pref$ is the array of prefix sums, $pref_i = pref_{i-1} + b_i$. Define $min(l, r)$ as the minimum on the subsegment of the array c with indices $l \dots r$. Now the answer for each subsegment is either $pref_r - pref_{l-1}$, if this value is not divisible by k, or $pref_r - pref_{l-1} - min(l, r)$.

In both cases we have a term $pref_r - pref_{l-1}$, so let's count the sum of such terms over all subsegments. The value of $pref_i$ will be taken with coefficient -1 (i. e. as a left border) for $(n-i)$ subsegments, and with coefficient $+1$ (i. e. as a right border) for i subsegments.

Now we need to count separately the subsegments in which $pref_r \bmod k = pref_{l-1} \bmod k$. Let's for each remainder x find all positions i in which $pref_i \mod k = x$. Let's process each remainder separately.

Consider that we have the list of positions for some fixed remainder: $pos_1, pos_2, \dots, pos_m$. Note that we need to substract from the overall answer the following value: $\sum_{l=1}^{m} \sum_{r=l+1}^{m} min(pos_l + 1, pos_r)$. To do this, let's make array d, where $d_i = min(pos_i + 1, pos_{i+1})$ (minimum on a subsegment of array c can be found using a data structure like Segments tree). In the array d we now need to calculate the sum of minimums over all subsegments. That can be done using a linear-time algorithm or using a Segments tree. To do this, for each element i let's find: in how many subsegments the element i is the leftmost minimum. Let

$left$ be the closest element to the left which is less than or equal, and let $right$ be the closest element to the right which is strictly less. Then the number of subsegments in which element i is the leftmost minimum equals $(i - left) \cdot (right - i)$.

Now we need to process those segments in which it is impossible to select sides to receive a sum that is indivisible by k. On such segments the smallest element equals $+\infty$, so for each such segment we need to subtract $pref_r - pref_{l-1}$ from the total answer.

The solution works in $O(n \log n)$.