BERBA

In this description we will refer to the nominators as A_1, B_1, C_1, D_1 and E_1, while the denominators are A_2, B_2, C_2, D_2 and E_2.

Let M be the largest nominator that can appear in the inequality. It is important to note that, even though numbers in the input are at most one thousand, M can be up to one million; for example, when A_2 is large and E_2 small so there are many fractions in between.

Obviously, a brute-force solution trying all $O(M^3)$ possibilities for numerators will not be efficient enough.

An improvement over this solution is to try all ways of choosing B_1 and C_1, then for each of them quickly calculate the number of appropriate values for D_1. B_1 does not influence this number so it suffices to calculate all D_1 satisfying:

$$\frac{C_1}{C_2} < \frac{?}{D_2} < \frac{E_1}{E_2}$$

For D_1 to satisfy this compound inequality it needs to satisfy both simple inequalities – the solution is the intersection of solutions for the elementary inequalities. This intersection can be easily (but carefully) calculated in O(1), for total complexity $O(M^2)$, which is still not good enough.

The model solution builds on this idea: try all values of C_1 and, for each, separately calculate the numbers of possible B_1 and D_1, multiplying the numbers of possibilities. Multiplication is possible because for a fixed C_1, the choice of B_1 does not influence the choice of D_1 and vice versa. The overall complexity is O(M).

CENZURA

We use two stacks – a "left" and a "right" stack. The algorithm works in two phases.

We start the phase one by putting letters from the text onto the left stack, checking if the top segment on the stack corresponds to the word A we are to remove. After we encounter the word A, we remove all its letters off the stack and switch to the right stack. Now we put letters from the end of the text in order onto the right stack until the top of the stack matches word A etc.

Phase one ends when we have exhausted all letters from the text. If we were to merge the two stacks at this point, the result would be the text with almost all occurrences of A removed; the only possible remaining occurrences of A are on the border between stacks. Those occurrences still need to be removed. An elegant way to do that is to move letters one by one from the left stack to the right one, again removing the word A upon encounter.

If we also use A and T for the lengths of the given strings, the complexity of this algorithm is $O(T \cdot A)$.

In phase one we put every letter from T exactly once onto one of the stacks and do an O(A) string comparison immediately after. In phase two we move every letter between stacks at most once, again followed by an O(A) comparison every time.

This is not the only approach with this complexity; any approach that does not naively remove the substring from the text should do. Doing so would raise the complexity to $O(T^2)$.

MISOLOVKE

Imagine going through the grid top to bottom. In the first row we can choose any set of K consecutive cells. In the second row and all after it we need take care so our choice does not allow mice get from the left to the right side of the basement. In the last row we need to take care not to allow mice get from the top to the bottom.

A brute-force solution tries all ways of selecting groups of cells in each row, which is too slow. A smarter approach is to notice that, to determine which choices are safe in a row, we do not need to know the exact choices that were made before. Instead, we need only the following information:

- Which group of K consecutive cells was chosen in the previous row;
- Is it possible to reach the group in the previous row from the left side of the basement;
- Is it possible to reach the group in the previous row from the right side of the basement;
- Is it possible to reach the group in the previous row from the top of the basement;

Using the index of the current row and these four pieces of information as the state, it is possible to achieve a dynamic programming solution of time complexity $O(N^3)$ – the number of states is about $8 \cdot N \cdot K$ and the best placement for a state can be calculated in $O(N-K)$. For implementation details see the official source code.