Analysis

This is a difficult problem, even though the implementation of the solution is not very hard.

All solutions solve each test case inside the input file separately and hence the complexities given are for a single test case.

For L > B it is obvious that Lora wins the game. Solutions in this analysis presume that L ≤ B.

Subtask 1

Any kind of correct bruteforce solution suffices for this subtask. The only difficulty is that the game might theoretically go on forever. This can be ignored by assuming that if a game goes on for too many moves, it can be considered a draw (in analysis of subsequent subtasks we will see that this is indeed a correct assumption).

The subtask allows precomputation and hardcoding of the answers as well.

Subtask 2 – $O(N^4)$

It is a common approach for this type of problems to define winning, losing and drawn positions. The logic is not too different from the standard winning/losing positions. Formally:

- A position is winning for the player on the move if there is a move that will lead to a position that is losing for the other player.
- A position is losing for the player on the move if every possible move leads to a winning position for the other player.
- A position is drawn if it is neither winning nor losing.

These definitions work fine for an acyclic game and so we want to transform our game into an acyclic one. The proposed solutions deal with this problem in different ways.

Let us define a position in the game. Since Lora and Bobi don't use the exact same rules we need to keep information about them both. We can describe a position with the following information:

- Position of Lora
- Position of Bobi
- Player to move

This leads to state of size $O(N^2 * 2) = O(N^2)$. The problem, however, is that starting from a valid position it is possible to make a few moves and go back to the same position, i.e. our game is cyclic. One way to fix this is to add additional state "current move number". Clearly, this makes the game acyclic, but it is not clear how many moves might a game last. Intuitively, a decisive game shouldn't last too long. It turns out that it is enough to assume that every game lasting over 3N moves is a draw. This is not a strict bound, but it is important to note that setting the limit to N moves is wrong. For a game with L=1, B=2, K=1, the winner is Bobi, but Lora can "stall" the game for up to 2N moves.

We can thus describe a position of the game in $O(N^3)$ information. To compute the result of a position we need to iterate over all possible moves. This takes either O(L) or O(B), but in both cases this is bounded by O(N). We get a complexity of $O(N^4)$.

Subtask 3 – O(N^3)

The insight for this subtask is that even though our game is cyclic, it seems to be **almost** acyclic.

Let the position of Lora at a given moment be P_L and the position of Bobi be P_B. Consider how the value of $MAX(P_L, P_B)$ changes in one move. If neither of the players pushes the other, then one of the two values simply increases and hence the value of $MAX(P_L, P_B)$ either remains unchanged or increases as well. On the other hand, if one player pushes the other, it is then clear that the pushed player was setting the value of $MAX(P_L, P_B)$ before the move, but now the other player is occupying that space and hence the value hasn't changed. We therefore conclude that this value is **non-decreasing** throughout the game. The cyclic part of the game comes from the fact that the value might remain unchanged.

Consider a set of three consecutive moves. If the value of $MAX(P_L, P_B)$ does not change for the duration of the three moves, this means that all three moves consisted of a push. On the other hand, we can note that if three consecutive moves are pushes, then we have a repeated position that both players had a chance to avoid. If either of the players had a winning strategy then they wouldn't ever need to repeat a position. From this follows that instead of keeping count of the number of moves made, we can simply keep track of the amount of consecutive pushes. All positions with 3 consecutive pushes can immediately be declared a draw. Note that positions with 2 consecutive pushes need not be a draw. This observation also shows the correctness of the solution in subtask 2.

This reduces the state of a position to O(2*3*N^2) = O(N^2).

The state computation is still linear, hence we get a total complexity of O(N^3).

Subtask 4 – O(N^2)

This subtask was the hard step in the solution.

Since working with positions that have three possible outcomes (win for Lora, win for Bobi and draw) are harder to analyse, we will simplify the game so that during computation we only consider winning and losing positions. We do this by introducing two conjectures. The first conjecture is that Lora can win. Considering this conjecture the game remains exactly the same but a drawn game is considered to be the same as a win for Bobi. Similarly, the second conjecture is that Bobi can win and in considering it we treat a draw as if it's a win for Lora. It is clear that if either conjecture turns out to be true then the corresponding player is the winner for the position. If both conjectures are false then the position is a draw.

Imagine that the players could move to the same space and couldn't push each other. In such case the game would be elementary – the optimal strategy would be to always make a maximal move until eventually somebody reaches the final. Intuitively, pushes shouldn't create an entirely different game.

Let us make a few observations on how a game proceeds and what interesting moves the players might make:

- After every move Lora makes she wants to be ahead of Bobi. Since $L \leq B$ if at any point it is Bobi to move and he is ahead of her then he can win by simply making maximal moves.
- If Bobi can make a maximal move that moves him at a distance above L in front of Lora, then he makes that move since Lora will never be able to catch up (on her next move she'd end up behind him). Such a move for Bobi will be called "***a runaway***" and respectively if Bobi can make such a move we will say that

he can "***run away***".

- If after his move Bobi wants to end up behind Lora then it only makes sense to move exactly one space behind her. Any other space behind Lora would just give him fewer opportunities in the future. We will call a move in which Bobi moves exactly one space behind Lora a "***passive move***". On first glance it is not clear whether it ever makes sense for Bobi to make a passive move. It turns out, however, that there are games in which Bobi has to make a passive move in order to win. The smallest such example is the game "10 4 6 4". If Lora moves from space 1 to space 5 on her first move, then the only way for Bobi to subsequently win is to move from space 1 to space 4, i.e. to make a passive move.

The interesting positions in the game are those directly after some player is pushed, as well as directly after a passive move. These positions are interesting because they involve strategical moves that are not based on moving as much as possible towards the finish. We define the following values, which are simply outcomes of interesting states:

$pushL_i$ = the outcome of the position in which Lora is on the move and is at space $MAX(1, i-K)$, while Bobi is at space i

$pushB_i$ = the outcome of the position in which Bobi is on the move and is at space $MAX(1, i-K)$, while Lora is at space i

$passive_i$ = the outcome of the position in which Lora is on the move and is at position i, while Bobi is on space $i-1$

The outcome of a position is the result starting from the position and assuming optimal play. An outcome is thus a win for Lora, a win for Bobi or a draw. Clearly the values described correspond to the outcomes of positions immediately after a push or a passive move.

Imagine, however, that these positions can be precomputed and we would like to test the conjecture that Lora can win from a given position. We can then simply simulate a game "without pushing" similarly to the greedy approach:

- If Bobi can run away, then he does that. Otherwise he goes to the furthest space i for which $pushB_i$ is not a win for Lora.
- On her move Lora goes to the furthest space i for which $pushL_i$ is a win for Lora and $passive_i$ is a win for Lora as well.

Using this simulation we don't have to consider pushing or passive moves, since neither player is going to put themselves in a position in which a push or a passive move will ruin his goal. The simulation to check whether the conjecture that Bobi can win is very similar.

Notice that computing these values solves the whole problem itself, since what we are looking for is $pushL_1$. All that's left is to somehow compute these values.

Suppose we have already computed $pushL_j$, $pushB_j$ and $passive_j$ for all $j > i$ and now we want to compute $pushL_i$, $pushB_i$ и $passive_i$.

To compute the outcome of a certain position we use the two conjectures approach described above. The computation of each of the values $pushL_i$, $pushB_i$, $passive_i$ is done simply by testing both conjectures starting

from the corresponding position and using the simulations without pushes described above.

A small problem is that in reality we can have transitions between there three positions, while it is clear that we cannot know their values at the time of their computation. For example from the position corresponding to $pushB_i$ for $K \neq 1$ we can go to the position of $passive_i$. Similarly we can go from the position of $pushB_i$ to that of $pushL_i$ and vice versa. This small cyclic dependency can be fixed manually:

- During the simulations we presume that no transitions to uncomputed states may occur. This means that when simulating for $pushB_i$ the very first move is never a push or a passive move, and when simulating for $pushL_i$ the very first move is never a push.
- After calculating the three values it is necessary to apply manual corrections due to the transitions between them. We need two corrections:
  - Since from the position corresponding to $pushB_i$ Bobi can move to the position of $passive_i$, then if the latter is better for him we set the former's value to the latter's. For example if the simulations conclude that $pushB_i = draw$, but $passive_i = win\_Bobi$, then we manually assign $pushB_i = win\_Bobi$.
  - If the outcome $pushL_i$ is better for Bobi than $pushB_i$, then it follows that surely $pushB_i$ is better for Lora than $pushL_i$. In such case starting from either position both players will want to move to the other. This leads to an endless loop between those two positions and hence a draw. In any other case we retain the values that were computed using the simulations.

Note that neither correction is applied for $i = 1$ as the transitions are not possible!

The simulation can be implemented in O(N) time. Computing the required values starting from the end and moving backwards yields a total complexity of $O(N^2)$.

Subtask 5 – O(N)

The step towards a linear solution is to use the fact that in the simulation described above a player always benefits from being closer to the final.

Suppose that Lora is on the move and she is on space P_L. Suppose we vary Bobi's position from 1 to N and check the result of the simulation for the conjecture that Lora can win for each of Bobi's positions. From the way the simulation works, it is clear that being further ahead is never a bad thing. Therefore the result would be that for some prefix of Bobi's positions Lora would be the first to reach the final in the simulation, while for all other positions Bobi will be the one to reach the final first. We can then define the following function:

$F_i =$ the smallest numbered space on which Bobi can start so that the simulation for the conjecture that Lora can win, starting with Lora on the move and on position i, gives that Bobi reaches the final first.

We can also define a similar function G for the conjecture that Bobi can win:

$G_i =$ the smallest numbered space on which Bobi can start so that the simulation for the conjecture that Bobi can win, starting with Lora on the move and on position i, gives that Bobi reaches the final first.

Observe that $F_i \leq F_{i+1}$. To see why this is so imagine that Lora is on the move and is on space P_L, while Bobi is on space P_B. Let the result of the simulation of the conjecture that Lora wins be that Bobi reaches the final first. It is then clear that if we repeat the simulation, but this time let Lora start from space $P_L - 1$ instead, then Bobi will

still be the winner. From this follows $F_i \leq F_{i+1}$. Similarly we can show that $G_i \leq G_{i+1}$.

If we could compute these functions then it would be easy to predict the result of a given simulation quickly. For example if Lora is on space P_L and is on the move, while Bobi is on space P_B, and we want to test the conjecture that Lora can win, it is necessary to simply check whether $F_{P_L} > P_B$. If it was Bobi's move instead, we could just simulate his first move and thus reach a position of the previous type.

In the process of computing values from the end moving backwards, we can also compute the first few moves of each simulation. Suppose we want to calculate F_i. We want to be able to check whether $F_i \leq x$. We thus want to check the outcome of the simulation of the conjecture that Lora can win, with Lora on the move and on space i, and Bobi on space x. We can do this simply by simulating one move for each player which will lead to a position for which the F value is computed and we can check the outcome in constant time. If it turns out that Bobi reaches the final first, then we know that $F_i \leq x$, otherwise $F_i > x$. Since we know that $F_i \leq F_{i+1}$, we can apply linear search for F_i starting from F_{i+1} and reducing the value we are testing for. This gives us amortized constant time for computing F_i. The function G is calculated in a similar way, just using the other conjecture. Since during computation we can only use values which are already computed, we need to presume that $F_i \geq i - 1$. This is not a problem since with optimal play Bobi will never be more than 1 space behind Lora whenever she is on the move.

Thus the whole solution is implemented as one loop from N to 1, keeping *several* values for each space.

Author: Encho Mishinev