15295 Fall 2019 #1 Problem Discussion
November 13, 2019

This is where we collectively describe algorithms for these problems. To see the problem statements follow this link. To see the scoreboard, go to this page and select this contest.

A. Little Girl and Game

We first look at the conditions for which a string can be rearranged into a palindrome. We see if a specific character, say ‘a’, shows up twice, we can ignore both of them, palindromize the remaining string and then add a ‘a’ to both sides. So if each character in the string has even frequency then it’s palindromizable. We also note if only one character has an odd frequency, we can still palindromize the string.

So we attempt to calculate the number of characters that have an odd frequency throughout the game. We note that this number changes by 1 every turn, since either we removed a character of odd frequency, and it decreases by 1, or we remove a character of even frequency, and it increases by 1. So the parity of the number of characters with odd frequency is invariant for each player.

So if you start with an odd number of characters with odd frequency, eventually on the start of your turn, there will be only 1 character of odd frequency, and you will win.
-Zack

B. Yet Another Number Game

In the case where n = 1, this problem is obvious (just check if the only input is nonzero). If n = 2, we can use a dynamic programming to compute the answer in time O(m^3), where m is the maximum number in the array (<= 300). To do that, we can just simulate all moves from a given state (i, j) (where we have i elements in the first pile and j in the second) to compute P and N positions. This is fast enough.

For n = 3, there is a nice trick we can use. Supposing that there were no move that decrements every number by some constant, we can recall that the grundy number of the game is the XOR of the sizes of the piles, and checking if this is nonzero indicates whether or not we have a P (losing) or N (winning) position. It turns out that this remains valid in the n = 3 case.

To prove the correctness of this we need only note that if (a xor b xor c) = 0 (so we are on a losing position), then ((a - k) xor (b - k) xor (c - k)) cannot be 0, where 1 <= k <= min{a ,b, c} (that is to say, every transition leads to a winning position). We can see this as follows: Let i be the least significant bit in k that is a 1. Every bit in a, b, and c at bit i will be forced to toggle. Since a xor b xor c = 0, we must have had that an even number of a, b, and c (so either 2 or 0) had a 1 bit in this location. Upon toggling, we have an odd number of 1s in this location and so the xor becomes nonzero, as desired.

-- Wassim

Alternatively, we can also do the n = 3 case with an O(m^3) dp, by computing the answer for "lower" triples like (a, b, c) = (0, 0, 0), and working up to "higher" triples like (a, b, c) = (300, 300, 300). Initially, set all states as losing. Once we find a losing state - the first one being (0, 0, 0) - we update all states which can reach it to become winning states.

Since (a, b, c1) and (a, b, c2) cannot both be losing positions (otherwise you could reach a losing position from another losing position), there are at most m^2 losing positions, so the whole algorithm is O(m^3).

-- Howard

C. Playing With String

The condition that we can only choose positions which are the center of some palindrome is equivalent to only choosing a position i such that s[i-1] = s[i+1]. Call index i *good* if s[i-1] = s[i+1], and i is not at the start or end of the string. Instead of splitting the string after every turn, we can state the game as follows:

Two players alternate turns, choosing a good index which is not adjacent to any previously chosen index. The player who cannot move loses.

The set of good indices can be split into several contiguous segments. Each segment can be treated as its own subgame. Now we can apply the Sprague-Grundy Theorem, so it suffices to calculate the Grundy value of each segment.

If we choose the i-th position in a segment of n contiguous good indices, it splits the segment into left and right segments of length L[i] and R[i], which are strictly less than n. This means we can use dynamic programming. Let dp[n] is the Grundy value of the game played on a segment of length n. Then dp[n] = mex(S), where mex denotes the minimum excluded value, and S is the set of dp[L[i]] XOR dp[R[i]] over all i from 1 to n.

This dp computation takes O(n^2) time. After doing this precomputation, we can find the lengths a_1, a_2, …, a_k of contiguous good segments in the given string. Then, we compute G = dp[a_1] XOR dp[a_2] XOR … XOR dp[a_k], the Grundy value of the whole game. If G > 0, the first player wins. If G = 0, the second player wins.

In the case of the first player winning, we try all possible initial moves to find the first one that leads to a winning position. This can be done in O(n) by making small modifications to the value of G.

-- Howard

D. 1-2-K Game

The k will only make a difference if it is a multiple of 3, so that it messes up the regular fail pass pass pattern formed by just the 1 and 2. If it’s not a multiple of 3, then the answer is simply n%3==0 means that the first player loses, or wins otherwise. If k is a multiple of 3, then we have the regular fail pass pass pattern until we get to k, then we continue the fail pass pass pattern starting at k+1, and this pattern repeats every k+1. So just take n%(k+1). If it happens to be k, then we know we pass. Otherwise, we have now reduced to the fail pass pass pattern, so just take mod 3 as before.

E. Tokitsukaze, CSL and Stone Game

F. Game With String

The input string consists of a bunch of sequences of dots, separated by crosses. If one of the sequences of dots has length greater than or equal to b but strictly smaller than a, Bob will always win. This is because, whenever Alice has a move, Bob also has a move (he can make a move wherever Alice can). So if Alice has no moves left and it's Alice's turn, she loses and if Alice runs out of moves but it's Bob's turn, he can take the sequence of dots of length at least b but less than a, and Alice still loses on the next turn. So, if Bob gets the opportunity to create such a sequence, he immediately wins.

If, on Bob's turn, there is a sequence of dots of length x, where x is greater than or equal to 2b, Bob can take the subsequence of dots from indices b+1 to 2b away from this sequence (letting the leftmost dot be at index 1 and the rightmost one be at index x). Then, he will have created a new sequence of length b (the sequence of dots to the left of the ones he took away), which means he can win the game. So there not being a sequence of dots of length at least 2b on Bob's turn is a necessary condition for Alice to win.

Now, we can case on the starting position:

If, at the start, there is already a sequence that has length greater than or equal to b but less than a, then Bob will win.

If, at the start, there are 2 sequences with length greater than or equal to 2b, no matter what Alice does on the first turn, there will be at least one remaining when it's Bob's turn; so here, Bob will win (because at the end of his turn, he can create the first type of sequence).

Otherwise, all sequences of length greater than or equal to b (which are the only ones that matter because otherwise neither player can pick them up) also have length greater than or equal to a; furthermore, there is at most one sequence with length greater than or equal to 2b. In the case that there is no sequence with length greater than or equal to 2b, then every turn, one sequence gets destroyed, so if there are an even number of such sequences then Bob wins, otherwise Alice wins.

The last case is that there is exactly one sequence with length greater than or equal to 2b (and all sequences (with length at least b) have length greater than or equal to a). Alice must split up this sequence on her first turn. One subcase is that the special sequence has length

at most a + b - 1, in which case the result depends on parity like before. Another subcase is that the special sequence has length greater than or equal to a + 4b - 1, in which case there's no way for Alice to break up the sequence without there being a sequence of length at least 2b on Bob's turn - so then Bob wins. The remaining possibility is that the special sequence has length less than or equal to a + 4b - 2, and length greater than or equal to a + b. We can then again case on whether it is possible for Alice to do one of the following during her first turn: split the sequence into two sequences of length greater than or equal to a but smaller than 2b (maintaining the parity of the number of valid sequences), split the sequence into two sequences with length smaller than b including a possibly empty sequence (also maintaining the parity), or split the sequence into one sequence of length greater than or equal to a but smaller than 2b, and one sequence with length smaller than b (changing the parity). If she can both maintain and change the parity, she automatically wins. Otherwise, if she can only maintain the parity, she wins iff there are an odd number of valid sequences (of length >= b) originally and if she can only change the parity, she wins if there are an even number of sequences originally.
-- Ram