

NOI.PH Elimination Round Solutions

Isaac Villamin

August 2022

A. Bruno Gets Disqualified

Solution: If $x + y > 0$, then Bruno is disqualified, so print "BruYesYesYes". Otherwise, Bruno is not disqualified, so print "BruNoNoNo".

B. Paru-Paro G

Solution: Determine the pattern from top to bottom. There are three intervals ($0 \leq i < n$, $i = n$, $n < i < 2n$) for which the i th row has a specific pattern. For the first interval ($0 \leq i < n$), there are $i - 1$ '.' that appear on the left and the right. Then, there is a '\ ' after the left dots and '/' after the right dots. Then, there are $2(n - i) + 1$ ' ' in between. For the second interval ($i = n$), the pattern is the same with the previous one except that the middle is a 'G'. For the third interval ($n < i < 2n$), there are $2n - i$ '.' that appear on the left and the right. Then, there is a '/' after the left dots and '\ ' after the right dots. Then, there are $i - n - 1$ ' ' in between.

C. Cryptographic Hashdle

Solution: Create a hash map that determines whether a character exists in string s so that we can determine in logarithmic time if a character in string t exists in string s . Iterate for each character in t from left to right. If $t_i = s_i$, then print 'G'. If $t_i \neq s_i$, but t_i exists in s , then print 'Y'. Otherwise, print '.'.

D. Slime King

Solution: Claim: For all possible ways to select the order of absorptions, the total cost is

$$\sum_{1 \leq i < j \leq n} a_i a_j.$$

Hence, the average cost is the same as the total cost for each way.

Proof: Since all initial terms of a is combined in the final operation, each initial term is multiplied to the other initial terms at least once. After k operations, we pick some i ($1 \leq i < n - k$), $a_i := a_i + a_{i+1}$, delete a_{i+1} , and for all the slimes after i , move them to the left. This means that a_i can never be multiplied to a_{i+1} after the operation. Hence, the initial terms of a , can only be multiplied to the other initial terms once. Therefore, the total cost is the sum of all $a_i a_j$ for all possible pairs (i, j) such that $i < j$.

Since all the possible ways of ordering the absorption gives the same cost, then we can pick a convenient order to calculate the total cost. We can do this by iterating from $i = 1$ to $i = n$, and increasing the total cost by

$$a_i \cdot \sum_{1 \leq j < i} a_j.$$

We can compute this in $O(n)$ by keeping track of the current prefix sum $\sum_{1 \leq j < i} a_j$.

E. Nice Numbers (Decision)

Solution: If a number N only has 2, 3, 5, 7 as its prime factors, then N is a nice number because we can always express N as its prime factorization. If N has a prime factor with two digits (i.e. 11, 13, ...), then there will always be a number with two digits when we express N as the product of numbers. We can check if N only has 2, 3, 5, 7 as its prime factors in $O(\log N)$. While $2|N$, divide N by 2. Do the same for 3, 5, 7. After the loop, if $N = 1$, then N is nice. Otherwise, N is not nice.

F. Nice Numbers (Counting)

Solution: We use the same observation we made in problem E. Since L and R can be very large ($1 \leq L, R \leq 10^{18}$), we cannot check for all N ($L \leq N \leq R$) because this is way above the time limit. An idea is to generate a list of nice numbers.

Claim: The number of nice numbers is relatively small (66,061).

Proof: A nice number N is of the form $N = 2^a 3^b 5^c 7^d$. Also, $\log_2 N < 60$, $\log_3 N < 38$, $\log_5 N < 26$, $\log_7 N < 22$. This means that there are no more than $60 \cdot 38 \cdot 26 \cdot 22 = 1,304,160$ possible nice numbers. Also, a, b, c, d cannot all be too large so the number of possible nice numbers (66,061) is much less than this number.

We can iterate for all the possible values a, b, c, d can take. Make sure to reset the iteration as soon as the product is greater than 10^{18} if you use the 64-bit integer data type to avoid overflows. A long long int is at most $2^{63} = 9,223,372,036,854,775,807 < 10^{19}$. Since we are only multiplying 2, 3, 5, 7, we don't have to

worry about overflowing when the product is close to 10^{18} because $7 \cdot 10^{18} < 10^{19}$. Store all of the possible values in a sorted array.

Use binary search to compute the number of nice numbers between L and R , inclusive, in $O(\log n)$. Take the index of the lower bound of L (idx_l) and the index of the upper bound of R (idx_r). The answer is $idx_r - idx_l$. The time complexity is $O((T \log LR) + M)$, where M is the number of possible nice numbers N ($1 \leq N \leq 10^{18}$).

I. Nonstop Fitness Training

Solution: Let pos be the current floor of Astrology Girl. If the operation is UP , then $pos := 0$ because floor 0 always becomes free. If the operation is FL , then pos is nondecreasing. If $y = pos$, then pos keeps increasing until floor pos does not have a follower. Since x, y can be very large ($0 \leq x, y \leq 10^9$), we cannot increment pos until floor pos does not have a follower. We can solve this in logarithmic time using the map data structure since we cannot store all floors i since i can be very large.

If the operation is UP , we can maintain the total of x 's tot . To avoid incrementing each floor by x , we can just decrease the lowest floor by x . Hence, instead of storing $pos = 0$ after UP , store it as $pos = -tot$. The answer for each UP query is $pos + tot$.

Notice that if $y = pos$, pos increases by the number of consecutive occupied floors directly above it plus 1. Let us mark each floor y . We say that floors are connected with each other if the floors are marked and they are consecutive floors. For a marked floor i , let bot_i be the bottommost floor that is connected with i and top_i the topmost floor that is connected with i . If the operation is FL , we initialize $bot_y = y$ and $top_y = y$. If floor $y - 1$ is marked, then $bot_y := bot_{y-1}$. If floor $y + 1$ is marked, then $top_y := top_{y+1}$. Then, $top_{bot_y} := top_y$ and $bot_{top_y} := bot_y$. This combines adjacent chunks of 1's into one chunk of 1's. Make sure to decrease each y by tot first. The time complexity is $O(n \log n)$ because there can be at most n FL operations, and there are at most n new floors. We use the map data structure for each FL operations which is logarithmic.