BdOI Monthly Contest Round1 Solutions: Problem Blackout Math

Problem Information

Problem Author: Nabeul Islam

Problem Preparation: Nabeul Islam

Editorial: Nabeul Islam

Subtasks hinting key observations

Preliminary Observation:

The sequence is defined as

$$F_1 = A$$

$$F_2 = B$$

$$F_i = F_{i-1} \oplus F_{i-2} \quad \text{for all } i \geq 3$$

Since N can be as large as 10^{18}, we clearly cannot simulate the recurrence term by term for the full solution within the time constraints. We need to find a efficient solution to get a full score. However a basic simution will award you the partial points.

Subtask 1: $N \leq 3$

This is a very easy subtask. For $N \leq 2$ we just need need print the first and second number in the sequence depending on the input of N. For $N = 3$ we just need to print $A \oplus B$.

Subtask 2: $A = 1$ and $B = 1$ and $N \leq 10$

In this subtask , we can simulate N^{th} term in the problem by writing a recursive function.

Subtask 3: $N \leq 20$

same as subtask 2.

Subtask 4: $N \leq 1000$

In this subtask , we can simulate N^{th} term in the problem by writing a recursive function. However, it is not efficient always. It can cause stack overflow and give you error. A simple approach to use a loop to simulate the sequence.

Subtask 5: $N \leq 10^5$

same as subtask 4.

Subtask 6(Full solution)

Here N is far too large to simulate directly. If we do some dry runs we will find a pattern. The key property to notice here the functionality and property of the XOR($\oplus$) operator. Two of its important properties we need to notice are $x \oplus x = 0$ and $x \oplus 0 = x$. Let's use this to compute a few terms and see what happens:

$$F_3 = F_2 \oplus F_1 = A \oplus B$$

$$F_4 = F_3 \oplus F_2 = (A \oplus B) \oplus B = A \oplus (B \oplus B) = A$$

$$F_5 = F_4 \oplus F_3 = A \oplus (A \oplus B) = (A \oplus A) \oplus B = B$$

$$F_6 = F_5 \oplus F_4 = B \oplus A = A \oplus B$$

$$F_7 = F_6 \oplus F_5 = (A \oplus B) \oplus B = A \oplus (B \oplus B) = A$$

Turns out the There is a repeated sequence throughout the terms, we can take this advantange to Optimize our solution.
This pattern repeats in every 3 sequence!
The sequence is periodic with period 3, cycling through $A, B, A \oplus B, A, B, A \oplus B, \ldots$ forever.
With this simple oveservation our problem simplifies to the idea below:

$$F_N = \begin{cases} A & \text{if } N \bmod 3 = 1 \\ B & \text{if } N \bmod 3 = 2 \\ A \oplus B & \text{if } N \bmod 3 = 0 \end{cases}$$

This reduces the time complexity and gives a solution that score full points.

BdOI Monthly Round 1 Solutions: Problem Groupchat

Problem Information

Problem Author: Aritro Sarkar

Problem Preparation: Aritro Sarkar, Nabeul Islam

Editorial: Aritro Sarkar, Md Sazid Hasan

Subtask by subtask solutions

Preliminary Observation:

Let S_i be the last message seen by the i-th person. Let m be the current total number of messages sent.
Let a simple simulation of events be:
For every query of type 1, the number of sent messages increase by 1. Formally, $m := m + 1$.
For every query of type 2, friend f reads till the very last message m. Formally, $S_f := m$.
For every query of type 3, we are tasked with finding the number of indices i, such that, $S_i \geq s$.
Since $N \cdot Q \leq 2.5 \cdot 10^{11}$, it is clear that a standard simulation of the events will result in a Time Limit Exceeded error. However, it will yield partial scores.

Subtask 1: $N = 1$

This subtask is quite easy. A standard simulation of events will suffice. However, there is an even easier solution. For each query of type 1, m, the total number of messages sent so far will increase by 1. Formally, $m := m + 1$. For every query of type 2, we will set S_1 to m. Formally, $S_1 := m$. For every query of type 3,

$$\textbf{The answer } \mathbf{A} = \begin{cases} 1 & \text{if } S_1 \geq s, \\ 0 & \text{otherwise} \end{cases}$$

Maintaining this simple simulation with a loop will suffice.

Subtask 2: $N, Q \leq 5000$

For this subtask, a simulation of the events that occur will yield points. The simulation goes as follows:

Type 1: $m := m + 1$.
Type 2: $S_f := m$
Type 3:

$$\textbf{The answer A} = \sum_{i=1}^{n} \begin{cases} 1 & \text{if } S_1 \geq s, \\ 0 & \text{otherwise} \end{cases}$$

Subtask 3: All type 3 queries appear after all other queries

This subtask is special. Once all the type 1 and type 2 queries have taken place, only after, we will have to answer type 3 queries and S will not change again. Each type 3 queries ask us how many elements of S are larger or equal to the input s. We can solve this sub-problem using an ordered set, but an even easier solution is using difference array and prefix sum techniques. Here's how: We will do everything as the same for types 1 and 2 queries. Let D be the difference array. D_i is going to help us calculate how many people have seen the i-th message. Initially, $D_i = 0$ for all indices. For every person from 1 to n, D_{1,S_i} should be increased by 1. For every person i, $D_i := D_i + 1$ and $D_{S_i+1} := D_{S_i+1} - 1$.
Afterwards, we will perform prefix sum on D.
Formally, let $P_i = \sum_{j=1}^{i} D_j$. Now, P_i is the number of people who have read the i-th message. Our answer for a type 3 query is P_s.

Full solution description

Looking back at the standard simulation, the only reason it won't give full scores is that because it's time complexity is $O(N \cdot Q)$. Our simulation does type 1 and type 2 queries in $O(1)$, but the type 3 queries take $O(N)$ complexity. An ordered set or a segment tree can fully solve the problem. details are as follows:

Ordered Set: For every type 1 query, we assign it a timestamp which is equal to the number of current queries and insert it to a vector which contains the assigned timestamp of every sent message, let it be M. For every type 2 query, if $S_f \neq 0$, then we erase S_f from the ordered set. After, we update S_f. Now, we insert S_f to the ordered set.

For every type 3 query, the answer is **os.size()** $-$ **os.order_of_key**(M_{s-1}).

Segment Tree or Fenwick Tree: Let each of the tree indices denote the timestamp. Now, similarly to the ordered set solution, when we are given a type 2 query if $S_f \neq 0$, we decrease 1 from the $S_f th$ index of the tree, and after updating S_f, we increase the new $S_f th$ index by 1.
Now, for query of type 3, the answer is the sum of the range [**M, R**], where R denotes the last index of the tree.

Both of these solution achieves a Time Complexity of $O(Q \cdot \log_2 n)$, which will suffice for this problem's constraints.

BdOI Monthly Contest Round1 Solutions: Problem Summand

Problem Information

Problem Author: Jahin Ahnaf

Problem Preparation: Jahin Ahnaf,Nabeul Islam

Editorial: Jahin Ahnaf,Sazid Hasan,Nabeul Islam

Subtasks hinting key observations

Preliminary Observation:

The problem asks us to find the minimum number m of non-negative integers $a_1, a_2, \ldots, a_m$ such that:

$$n = a_1^x + a_2^x + \cdots + a_m^x$$

Since $1^x = 1$ is true for any positive integer x, a valid representation always exists for any $n \geq 1$ by taking $m = n$ copies of 1. Therefore, an impossible output $(-1 - 1)$ is never triggered for valid positive inputs.

Subtask 1: $x = 1$

When the exponent is 1, the equation simplifies to $n = a_1 + a_2 + \cdots + a_m$. Therefore, We can say $m = 1$ always for this case. We can achieve $m = 1$ by setting $a_1 = n$. Then can just print 1 and a_1. It will give us full score in this subtask.

Subtask 2: $2^x > n$

In this subtask we can use our preliminary observation. As defined in the subtask $2^x > n$. Therefore, no integer greater than or equal to 2 can be used as a summand. So we can use this property for any x^{th} power

$$1^x = 1$$

. So, the only way to form the sum n is by using n copies of 1. So the answer is $m = n$. The sequence will be $[1, 1, \ldots, 1]$ which is n copies of 1. This will give us the full score in this subtask.

Subtask 3: $m = 2$

This subtask gives us an important distinction. The answer m is always 2. If we submit $m - 1$ it will reward us free partial points. For the sequence we need to test if a 2-summand partition exists. For this we can iterate from 1 upto $\lfloor \sqrt[x]{n} \rfloor$. Let's say a possible candidate is a_1. For each possible candidate a_1, we compute the remainder $rem = n - a_1^x$ and check if rem is a perfect x^{th} power by taking $a_2 = \lfloor \sqrt[x]{rem} \rfloor$ and verifying if $a_1^x + a_2^x = n$.

Subtask 4 & 5:

For smaller values of n, this problem can be modeled as the classic coin change problem in dynamic programming. Let's define a Dynamic Programming (DP) table:

$$\text{dp}[i] = \text{minimum number of } x^{th} \text{ powers needed to sum to } i$$

In base case $\text{dp}[0] = 0$, and $\text{dp}[i] = \infty$ for all $i > 0$. In simple terms this means we need infinite amount of summands(possibly n) to construct our sum. However, we update the dp state if we find a better solution. For each state i from 1 to n, we iterate over all valid bases $j \geq 1$ such that $j^x \leq i$:

$$\text{dp}[i] = \min_{1 \leq j^x \leq i} (\text{dp}[i - j^x] + 1)$$

This will yield us the value of m which will reward us in partial points. In order to construct the sequence, we maintain a parent[i] array that records the optimal base j used to transition into state i. By backtracking from n to 0, we collect the summands and sort them in non-decreasing order.

Subtask 6 & 7(Full Solution)

If we compute the dp table from scratch for every test case, it will result in Time Limit Exceeded (TLE). So, it's not possible to compute dp table in every test cases as test cases upto be $T \leq 1000$. Computing dp every time for a large number of test cases will reward us in TLE. However, one noticeable thing is x is very small ($1 \leq x \leq 10$). There are only 9 non-trivial exponents ($x \in [2, 10]$) we need to worry about. We can precompute the dp tables globally once up to $N_{\max} = 10^5$ before answering any queries! We can make a slight adjustment to our previous dp table. Let, $\text{dp}[x][i]$ be the minimum number of x^{th} powers needed to sum to the integer i. For each exponent $x \in [2, 10]$ the base case is $\text{dp}[x][0] = 0$ (a sum of 0 requires zero summands) and $\text{dp}[x][i] = \infty$ for all $1 \leq i \leq N_{\max}$. For each $x \in [2, 10]$ and for each target sum i from 1 to $N_{\max}$ we compute the following dp table:

$$\text{dp}[x][i] = \min_{1 \leq j^x \leq i} (\text{dp}[x][i - j^x] + 1)$$

This will yield us the minimum number of summands, which will reward us in partial points. In order to construct the sequence, we maintain a $\text{parent}[x][i] = j$ array that records the optimal base j used to transition into state i. By backtracking from n to 0, we collect the summands and sort them in non-decreasing order.