Inception 5.0

Coding Competition

ACM NIT Surat

Editorial

Crystals in Making

Author: Shivangi Dubey

Key Idea: Binary Representation of Numbers

Solution:

What is one greedy way to represent a number such that the number of elements from the sequence of power of 2 used is minimum? It will be the number of ones in the binary representation of that number itself.

How?

One way to prove it is: for having minimum elements we won't be using any number twice because $sequence[i+1] = 2 \times sequence[i]$ right? So why will we use $sequence[i]$ twice when we can replace it with a single element $sequence[i+1]$.

So the greedy approach we end up with is recursively picking the largest power of 2 that is smaller than our desired sum (*k* in this case). That is exactly how we obtain binary representation of a number.

Let's talk about its implementation, there are 3 ways to implement this:

1. Using inbuilt function: `__builtin_popcount(n)` is an inbuilt function that returns the number of set bits of an integer n, and that is our answer in this case. However the upper range of n in our case exceeds the limit of "int" hence we will be needing `__builtin_popcountll(n)` which is the same as the previous one but for "long long int".
2. Simple division: We can write a program which does exactly what we do when we find the binary representation of a number manually, that is recursively dividing the number by two and each time adding the remainder/modulo of the number by 2 to the answer.
3. Using bit properties: The exact bit property we are going to use is :- *n&(n-1)* has 1 set bit lesser than n, It can be proved by
   If *n* = ****10000 then *n-1* = ****01111 so n&(n-1) = ****00000
   so if we keep replacing n with $n\&(n-1)$ we will know the number of set bits in the given n.

The time complexity of these solutions will be around $O(logN)$ and $O(1)$ space complexity.

Rotating Balls

Author: Mihir Khambhati

Key Idea: Basic Maths

Solution:
If each person has the Balls numbered 1 and 2, we see that the pattern is periodic, each person just passes the number 1 to her neighbour, and each person always holds numbers 1 and 2.
So the conclusion is not true for two people. Does parity (evenness or oddness) matter, then? Consider four people, initially holding

person 1	person 2	person 3	person 4
1	1	2	2
3	3	4	4

We see that at each turn, everyone will hold a 1 or 2 paired with a 3 or 4. So again, the conclusion is not true, and for any even number of people, we can make sure that it never is true. For example, if there are $10 = 2 \times 5$ people, just start by giving each person one card chosen from {1, 2, . . . , 5} and one card chosen from {6 , 7 , . . , 10} .
Then, at every turn, each person holds a card numbered in the range 1-5 paired with one in the range 6-10, so no one ever holds two cards with the same numbers. Now In the case where there is an odd number of people.
Here is an example involving 5 people.

person 1	person 2	person 3	person 4	person 5
1	2	3	1	4
4	3	5	2	5

We arranged the table so that the top row cards are smaller than the bottom row, so we know that these are the cards to be passed on the next turn. After that, we sort them again so that the top-level contains the smaller numbers:

person 1	person 2	person 3	person 4	person 5
2	2	1	3	1
4	4	5	3	5

So, what actually happened? We were able to get the two 3's to coincide because the top and bottom rows stopped "mixing," and there was a 3 in the top and a 3 in the bottom.
The time complexity for the solution is $O(1)$.

Minimized Way

Author: Jignesh Jinjala

Key Idea: The problem can be solved using stack and dp.

Solution:
Here, there are two choices for any position i. Also, you have to minimize the total energy absorption. From these two observations- choice and optimization, we can figure out that the problem can be solved using dynamic programming.

Now, think about 2 choices given in the problem statement.

1) $A[j] \geq A[i]$, j is the smallest possible $(i < j \leq N)$ and there is no k such that $A[k] \geq A[i]$, $i < k < j$.
- This definition simply suggests that Jonas can move from i^{th} position to nearest greater or equal element position in the right side.
2) $A[j] < A[i]$, j is the smallest possible $(i < j \leq N)$ and there is no k such that $A[k] < A[i]$, $i < k < j$.
- This definition simply suggests that Jonas can move from i^{th} position to the nearest smallest element position on the right side.

Now, it is easy to implement a solution.

We can find the nearest greater or equal and nearest smallest element using stack.

Time Complexity: $O(N)$

Grand Line

Author: Krunal Rank

Key Idea: Divisors of numbers

Solution:
For any given Island N, we can reach N through all the islands whose values divide N.

Now, if one thinks clearly, Island N can be reached using all the divisors of N. Thinking recursively, all the divisors of N can be reached only through their divisors and so on.
This might give you an idea that memoization of a number of paths to reach any particular divisor of N is required.

Also, since the question is a query-type question with at most 10^5 queries, the most optimal way is to store all the answers before hand into a 1-Dimensional Array, say dp, where $dp[i]$ stores the number of ways to reach to Island i.

Now for the toughest part to think of, how to calculate the number of ways for reaching Island I, that is, the value of $dp[i]$?

As we already discussed a small part of the recursive relation, we can formulate it as follows:-

$$dp[i] = \sum_{j=1}^{j<i} dp[j] \; where \; j \; is \; the \; divisor \; of \; i$$

For the base cases as specified in the question and using one's own observation, we need to initialize the base values of dp as follows:-

$$dp[i] = 0 \; when \; i = 1$$
$$dp[i] = 1 \; when \; 1 \leq i \leq N$$

The first expression defines the condition given in the question (No. of ways to reach Island 1 is 0).
The second expression defines the one path to reach Island i from Island 1.

The values for dp can be calculated in $O(N \times \sqrt{N})$. However, this will result in a Time Limit Exceeded Verdict, since N has a maximum value of 10^6.

A more optimised method allows you to calculate dp in $O(N \times logN)$.
Hence, the answer will have the time complexity $O(max(N \times logN, T))$.

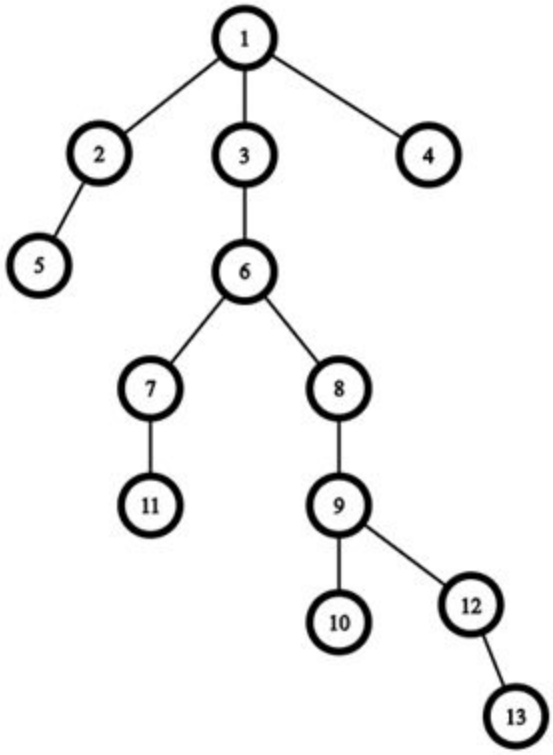
Please check the solution code for more information regarding this.

Too Much Dark

Author: Jitendra Jat

Key Idea: -

Solution:
As Jonas wants to catch Martha, it is good for Jonas if he moves towards Martha instead of either staying at the same node or moving away from Martha.



Now, consider the above graph and assume that the initial position of Jonas is node 1.

CASE 1: Martha moves only away from Jonas.

Consider the starting position of Martha as node 8.

In the worst-case Martha will choose a path such that she will reach a maximum possible depth node in subtree starting from 8.

So Martha will follow 8->9->12->13 and then stay at node 13.

Moves required by Jonas = depth of node 13 in the tree rooted at node 1 = 6

CASE 2: Martha first moves towards Jonas then move away from Jonas

Consider the starting position of Martha as node 7.

If we apply case 1, she will rest at node 11 and Jonas will catch her in 4 moves.

But here the worst case is possible if Martha goes to node 6 first then apply **CASE 1**. i.e. 7-6-8-9-12-13.

In this case, Jonas will catch her in 6 moves.

Approach

Run a DFS from the initial position of Jonas and find maximum possible depth from every node.

Say a node p 'safe' if moves required by Martha to reach node 'p' are less than moves required by Jonas to reach node 'p'. In other words, Martha will not be caught at node 'p'.

Now move from the initial position of Martha to the position of Jonas. (We can do it by simply keeping track of parent nodes).

Initialize ans = 0;

If current node is safe ans = max(ans, max_depth_from_p + level[s])

AI Apocalypse

Author: Mihir Khambhati, Krunal Rank

Key Idea: Parities of Numbers

Solution:

First of All, let's have a more approaching look at the function, its trivial for the case when the position of AI is 1 or N. But in other cases, the position in which the AI can move must be multiple of $(pos^2 - 1)$ and the distance between them must be minimum.

So,

$$pos^2 - 1 = (pos - 1) * (pos + 1)$$

So, we can't get a difference better than 2.

So, Let's modify for the problem, "After every query ask by us the AI will move to the left or to right by one step"

This is a famous "Fox Hole Problem".

Consider what happens if you inspect the position (2, 3, 4, ..., n−1) on successive queries. Prove that if the AI starts at an even-numbered position, then it will not be able to get past your sweep and that you will therefore catch it.

If you did not catch it, then it must have started in an odd-numbered position. Now number the position starting from the other end. Prove that the AI now (i.e. after that first sweep) must be in an even-numbered position using that new numbering.

This means that you can do a second sweep in the opposite direction and must catch it.

Therefore the sequence (2, 3, 4, ..., n−1) followed by (n−1, n−2, ..., 3, 2) will catch the AI.