code jam
hello, world!

Practice Mode

Round 1A 2010

Contest Analysis

Questions asked 1

Submissions

Rotate

11pt	Not attempted **2076/2436 users** correct (85%)
12pt	Not attempted **1855/2071 users** correct (90%)

Make it Smooth

12pt	Not attempted **509/954 users** correct (53%)
24pt	Not attempted **319/482 users** correct (66%)

Number Game

16pt	Not attempted **680/1091 users** correct (62%)
25pt	Not attempted **244/450 users** correct (54%)

Top Scores

rng..58	100
Pipi	100
cgy4ever	100
rem	100
XiaoZiqian	100
qizichao	100
exod40	100
GarnetCrow	100
hos.lyric	100

Contest Analysis

Overview | Problem A | Problem B | Problem C

Let us begin by focusing on a single game. Given **A** and **B**, we can first assume without loss of generality that `A ≥ B`. Now, how can we decide if it is a winning position or not? (It is impossible to have a tie game since A+B is always decreasing.) Well, a position is winning if and only if, in one step, you can reach a losing position for your opponent. This is an important fact of combinatorial games, so make sure you understand why it's true!

Observations, easy and not so easy

One trivial observation is that `(A, A)` is a losing position.

Another observation is much trickier unless you have already been exposed to combinatorial games before:

> If `A ≥ 2B`, then `(A, B)` is a winning position.

To justify this: in such a position, suppose k is the largest number of B's we can subtract from A, i.e., `A - kB ≥ 0` and `A - (k+1)B < 0`. We do not know yet whether `(A-kB, B)` is a winning position or not. But there are just two possibilities. If it is losing, great, we can subtract kB from A, and hand the bad position to our opponent. On the other hand, if it is winning, we can subtract `(k-1)B` from A, and our opponent has no choice but to subtract another B from the result, giving us the winning position `(A-kB, B)`. Therefore, `(A, B)` is a winning position either way!

Expand further

The observation above gives us a fairly quick algorithm to figure out who wins a single game `(A, B)`. Instead of using dynamic programming to solve the subproblem for all `(A', B')` with `A' ≤ A` and `B' ≤ B`, which is the most common way of analyzing this kind of game, we can do the following:

- In round 1: If `A ≥ 2B`, it's a winning position and we're done. Otherwise, we have only one choice: subtract B from A, and give our opponent `(B, A-B)`.
- In round 2: If `B ≥ 2(A-B)`, it is a winning position for our opponent. Otherwise, the only choice he has is to subtract `A-B` from B, and hand us `(A-B, 2B-A)`.
- In round 3: If `A-B ≥ 2(2B-A)`, it is a winning position for us again. Otherwise, we are going to make it `(2B-A, 2A-3B)`.

And so on. This leads to the following algorithm for efficiently solving a single game `(A, B)`, assuming `A ≥ B`:

ACRush	100

```
bool winning(int A, int B) {
  if (B == 0) return true;
  if (A >= 2*B) return true;
  return !winning(B, A-B);
}
```

Does this sound familiar? One connection you might see is that the Number Game closely resembles Euclid's algorithm for greatest common divisor. It is not hard to see that this algorithm, like Euclid's, will need to recurse at most `O(log A)` times.

Unfortunately, we still cannot afford to run the algorithm for every possible `(A, B)`! To solve the problem, we need to work with many positions at once. Let us go through the same rounds, but imagine having a fixed B and consider all possible A's at the same time:

- Round 1: `(A, B)`. If `A ≥ 2B`, i.e., `A/B ≥ 2`, then `(A, B)` is winning.
- Round 2: `(B, A-B)`. If `B ≥ 2(A-B)`, i.e., `A/B ≤ 3/2`, then `(A, B)` is losing.
- Round 3: `(A-B, 2B-A)`. If `A-B ≥ 2(2B-A)`, i.e., `A/B ≥ 5/3`, then `(A, B)` is wining.
- Round 4: `(2B-A, 2A-3B)`. If `2B-A ≥ 2(2A-3B)`, i.e., `A/B ≤ 8/5`, then `(A, B)` is losing.
- And so on.

This gives a fast enough solution to our problem. For each B, we consider all $_A$'s in the above manner, and in $O(\log 10^6)$ rounds, we can classify all the A's.

Simplify

The above method is perfectly correct, but it can be made simpler. First of all, does anything in the above list look familiar? There are Fibonacci numbers all over the place! Let `F(i)` be the `i`-th Fibonacci number. One can show by induction that the previous analysis is actually saying the following:

- Round 2t-1: If `A/B ≥ F(2t+1)/F(2t)`, then `(A,B)` is a winning position.
- Round 2t: If `A/B ≤ F(2t+2)/F(2t+1)`, then `(A,B)` is a losing position.

It turns out that both `F(2t+1)/F(2t)` and `F(2t+2)/F(2t+1)` approach the golden ratio `φ = (1 + √5) / 2` as t gets large. This means there is a very simple characterization of all winning positions!

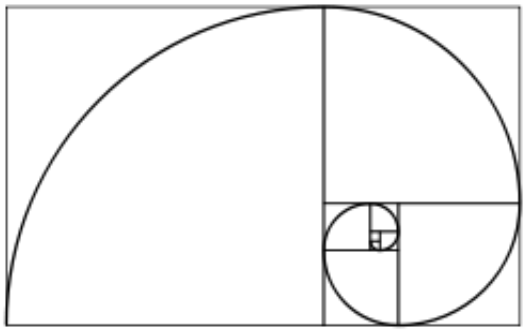
> **Theorem:** `(A, B)` is a winning position if and only if `A ≥ φ B`.

Using this theorem, it is easy to solve the problem. Loop through each value for B, and count how many A values satisfy `A ≥ φ B`.

Why it is Golden?

Once we have stumbled upon the statement of this theorem, it is actually pretty easy to prove. Here is one method: Using mathematical induction, assume we proved the theorem for all smaller A's and B's. If $A \geq 2B$, then (A, B) is a winning position as we discussed earlier. Otherwise we will leave our opponent with $(B, A-B)$. Then (A, B) is winning if and only if $(B, A-B)$ is losing. By our inductive hypothesis, this is equivalent to $B \leq \varphi (A - B)$, or $A \geq ((1 + \varphi) / \varphi) * B$. Since $\varphi = (1 + \varphi) / \varphi$, this proves (A, B) is winning if and only if $A \geq \varphi B$, as required.

Here is another, more geometric viewpoint. You start with a piece of paper which is an A by B rectangle ($A \geq B$), and cut out a B by B square from it. If the rectangle is *golden*, where $A = \varphi B$, then the resulting rectangle will also be golden. In our game, A and B are integers, so the rectangle is never golden. We call it *thin* if $A > \varphi B$, otherwise we call it *fat*. From a thin rectangle you can always cut it to a fat rectangle, and from a fat one you can only cut it to a thin one. They correspond to the winning positions and losing positions, respectively.



A picture of golden rectangles from Wikipedia.

Other Approaches

There are many ways of arriving at the solution to this problem -- our analysis focuses on only one of these ways. Another approach would be to start with a slower method and compute which (A, B) are winning positions for small A, B. From looking at these results, you could easily guess that (A, B) is winning if and only if $A \geq x B$ for some x, and all that remains would be to figure out what x is!

More Information

Euclid's Algorithm - Fibonacci Numbers - Golden Ratio