Code Jam 2008 - Round 1A

Analysis: Numbers

This problem with a simple statement was in fact one of the hardest in Round 1. The input restrictions for the small input were chosen so that straightforward solutions in Java or Python, which have arbitrary-precision numbers, would fail. The trick was calculating √5 to a large enough precision and not using the default double value. It turned out that using the Windows calculator or the UNIX `bc` tool was enough to solve the small tests as well.

Solving the large tests was a very different problem. The difficulty comes from the fact that √5 is irrational and for n close to 2000000000 you would need a lot of precision and a lot of time if you wanted to use the naive solution.

The key in solving the problem is a mathematical concept called conjugation. In our problem, we simply note that (3 - √5) is a nice conjugate for (3 + √5). Let us define

(1) $\alpha := 3 + \sqrt{5}$, $\beta := 3 - \sqrt{5}$, and $X_n := \alpha^n + \beta^n$.

We first note that X_n is an integer. This can be proved by using the binomial expansion. If you write everything down you'll notice that the irrational terms of the sums cancel each other out.

(2) $$X_n = \alpha^n + \beta^n = 2\left(\sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{2i} 3^{n-2i} 5^i\right)$$

Another observation is that $\beta^n < 1$, so X_n is actually the first integer greater than α^n. Thus we may just focus on computing the last three digits of X.

A side note. In fact, β^n tends to 0 so quickly that that our problem would be trivial if we asked for the three digits *after* the decimal point. For all large values of n they are always 999.

Based on (1) and (2), there are many different solutions for finding the last three digits of X_n.

Solution A. [the interleave of rational and irrational]

One solution goes like this: α^n can be written as $(a_n + b_n\sqrt{5})$, where a_n and b_n are integers. At the same time, β^n is exactly $(a_n - b_n\sqrt{5})$ and $X_n = 2a_n$. Observe that

(3) $\alpha^{(n+1)} = (3 + \sqrt{5})(a_n + b_n\sqrt{5}) = (3a_n + 5b_n) + (3b_n + a_n)\sqrt{5}$.

So $a_{n+1} = 3a_n + 5b_n$ and $b_{n+1} = 3b_n + a_n$. This can be written in matrix form as

(4) $$\begin{pmatrix} a_n \\ b_n \end{pmatrix} = A \begin{pmatrix} a_{n-1} \\ b_{n-1} \end{pmatrix} = A^n \begin{pmatrix} a_0 \\ b_0 \end{pmatrix}, \quad A = \begin{pmatrix} 3 & 5 \\ 1 & 3 \end{pmatrix}$$

Since $\alpha^0 = 1$, we have $(a_0, b_0) = (1, 0)$.

Now we use the standard fast exponentiation to get A^n in O(log n) time. Note that we do all operations modulo 1000 because we just need to return the last three digits of a_n.

Here's some Python code that implements this solution:

```
def matrix_mult(A, B):
  C = [[0, 0], [0, 0]]
  for i in range(2):
    for j in range(2):
      for k in range(2):
        C[i][k] = (C[i][k] + A[i][j] * B[j][k]) % 1000
  return C

def fast_exponentiation(A, n):
  if n == 1:
    return A
  else:
    if n % 2 == 0:
      A1 = fast_exponentiation(A, n/2)
      return matrix_mult(A1, A1)
    else:
      return matrix_mult(A, fast_exponentiation(A, n - 1))

def solve(n):
  A = [[3, 5], [1, 3]]
  A_n = fast_exponentiation(A, n)
  return (2 * M_n[0][0] + 999) % 1000
```

Solution B. [the quadratic equation and linear recurrence]

Experienced contestants may notice there is a linear recurrence on the X_i's. Indeed, this is not hard to find -- the conjugation enters the picture again.

Notice that

(5) $\alpha + \beta = 6$, and $\alpha\,\beta = 4$.

So α and β are the two roots of the quadratic equation $x^2 - 6x + 4 = 0$. i.e.,

(6) $\alpha^2 = 6\alpha - 4$, and $\beta^2 = 6\beta - 4$.

Looking at (1) and (6) together, we happily get

(7) $X_{n+2} = 6X_{n+1} - 4X_n$.

Such recurrence can always be written in matrix form. It is somewhat redundant, but it is useful:

$$\begin{pmatrix} X_{n+1} \\ X_n \end{pmatrix} = B \begin{pmatrix} X_n \\ X_{n-1} \end{pmatrix} = B^n \begin{pmatrix} X_1 \\ X_0 \end{pmatrix}, \quad B = \begin{pmatrix} 6 & -4 \\ 1 & 0 \end{pmatrix}$$

From here it is another fast matrix exponentiation. Let us see **radeye**'s perl code that implements this approach here:

```
sub mul {
    my $a = shift ;
    my $b = shift ;
    my @a = @{$a} ;
    my @b = @{$b} ;
    my @c = ($a[0]*$b[0] + $a[1]*$b[2],
             $a[0]*$b[1] + $a[1]*$b[3],
             $a[2]*$b[0] + $a[3]*$b[2],
             $a[2]*$b[1] + $a[3]*$b[3]) ;
    @c = map { $_ % 1000 } @c ;
    return @c ;
}
sub f {
    my $n = shift ;
    return 2 if $n == 0 ;
    return 6 if $n == 1 ;
    return 28 if $n == 2 ;
    $n -= 2 ;
    my @mat = (0, 1, 996, 6) ;
    my @smat = @mat ;
    while ($n > 0) {
        if ($n & 1) {
            @mat = mul([@mat], [@smat]) ;
        }
        @smat = mul([@smat], [@smat]) ;
        $n >>= 1 ;
    }
    return ($mat[0] * 6 + $mat[1] * 28) % 1000 ;
}
sub ff {
   my $r = shift ;
   $r = ($r + 999) % 1000 ;
   $r = "0" . $r while length($r) < 3 ;
   return $r ;
}
for $c (1..<>) {
    $n = <> ;
    print "Case #$c: ", ff(f($n)), "\n" ;
}
```

Solution C. [the periodicity of 3 digits]

For this problem, we have another approach based on the recurrence (7). Notice that we only need to focus on the last 3 digits of X_n, which only depends on the last 3 digits of the previous two terms. The numbers eventually become periodic as soon as we have (X_i, X_{i+1}) and (X_j, X_{j+1}) with the same last 3 digits, where $i < j$. It is clear that we will enter a cycle no later than 10^6 steps. In fact, for this problem, you can write some code and find out that the cycle has the size 100 and starts at the 3rd element in the sequence. So to solve the problem we can just brute force the results for the first 103 numbers and if n is bigger than 103 return the result computed for the number (n - 3) % 100 + 3.

Solution D. [the pure quest of numbers and combinatorics]

Let us see one more solution of different flavor. Here is a solution not as general as the others, but tailored to this problem, and makes one feel we are almost solving this problem by hand.

Let us look again at (2). We want to know X_n mod 1000. We know from the chinese remander theorem that if we can find X_n mod 8 and X_n mod 125, then X_n mod 1000 is uniquely determined.

(a) For $n > 2$, X_n mod 8 is always 0. Since $5^i \equiv 1 \pmod 4$, $3^{n-2i} \equiv 1$ or $-1 \pmod 4$ depending on n, so, for $n > 2$,

$$\pm X_n \equiv 2 \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{2i} \equiv 2 \cdot 2^{n-1} \equiv 0 \mod 8.$$

(b) To compute X_n mod 125, we only need to worry about i=0,1,2. All the rest are 0 mod 125. In other words, all we need to compute is

$$2\left(\binom{n}{0}3^n + \binom{n}{2}3^{n-2}5 + \binom{n}{4}3^{n-4}25\right) \mod 125.$$

There are various ways to compute the elements in the above quantity. The exponents can be computed by fast exponentiation, or using the fact that 3^n mod 125 is periodic with cycle length at most 124. The binomial numbers can be computed using arbitrary precision integers in languages like Java and Python, or with a bit careful programming in languages like C++.

More information:

Binomial numbers - Linear recurrence - Exponentiation - Chinese remainder theorem