Power Fib

codility
WE TEST CODERS

It's time to show you how the Codility challenge code-named Omicron can be solved. You can still give it a try, but no certificate will be granted.

In the Omicron certificate problem, the task was to compute $Fib_{N^M} \mod 10\,000\,103$, where Fib_i is the i-th Fibonacci number. Fibonacci numbers grow exponentially quickly, so without performing calculations modulo 10 000 103, one would have to use arbitrarily large integers. Note that 10 000 103 is a prime number. Unsurprisingly, you may have seen this trick in various Codility tasks before; it simplifies computations. As long as we use only addition, subtraction and multiplication, it suffices to perform all the operations modulo 10 000 103, and there is no need to implement arithmetic for arbitrary large integers. Even division can be used, though its implementation is less straightforward.

So, we can operate on *ordinary* integers and avoid arithmetical overflows. But how quickly can we compute Fibonacci numbers? There is a closed analytic formula: $Fib_i = \frac{\varphi^i - (1-\varphi)^i}{\sqrt{5}}$, where φ is the golden ratio $\varphi = \frac{1+\sqrt{5}}{2}$. Unfortunately, for $i = N^M$, use of this formula is not feasible due to the limited precision of floating point operations.

The second fastest method of computing Fibonacci numbers is based on matrix multiplication and uses only operations on integers. It is based on the following equation, where:

$$F = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$F \times \begin{pmatrix} Fib_n \\ Fib_{n-1} \end{pmatrix} = \begin{pmatrix} Fib_{n+1} \\ Fib_n \end{pmatrix}$$

Hence:

$$F^k \times \begin{pmatrix} Fib_n \\ Fib_{n-1} \end{pmatrix} = \begin{pmatrix} Fib_{n+k} \\ Fib_{n+k-1} \end{pmatrix}$$

Taking $n = 1$, we obtain:

$$F^k \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} Fib_{k+1} \\ Fib_k \end{pmatrix}$$

Generally, one can prove by simple induction that:

$$F^k = \begin{pmatrix} Fib_{k+1} & Fib_k \\ Fib_k & Fib_{k-1} \end{pmatrix}$$

Powers of matrices can be computed in a similar way to powers of integers: by squaring and multiplication. Elements of A^k (modulo 10 000 103) can be computed in $O(\log k)$ time. Here are procedures in Pascal to implement multiplication and exponentiation of 2×2 matrices, modulo 10 000 103, and to compute Fibonacci numbers:

1: Procedures in Pascal that compute Fibonacci numbers.

```
const
  P = 10000103;

type
   matrix = array[1..2, 1..2] of int64;

const
   F  : matrix = ((1, 1), (1, 0));
   id : matrix = ((1, 0), (0, 1));

function mul(A, B : matrix) : matrix;
var
  C :  matrix;
begin
   C[1,1] := (A[1,1] * B[1,1] + A[1,2] * B[2,1]) mod P;
   C[1,2] := (A[1,1] * B[1,2] + A[1,2] * B[2,2]) mod P;
   C[2,1] := (A[2,1] * B[1,1] + A[2,2] * B[2,1]) mod P;
   C[2,2] := (A[2,1] * B[1,2] + A[2,2] * B[2,2]) mod P;
   mul := C
end;

function pow(A : matrix;  K : longint) :  matrix;
begin
   if K = 0 then
      pow := id
   else if K mod 2 = 0 then
      pow := pow(mul(A,A), K div 2)
   else
      pow := mul(pow(A, K-1), A)
end;

function Fib(K :  longint) : longint;
var
   A : matrix;
begin
   A := pow(F, K);
   Fib := A[1,2]
end;
```

Unfortunately, such a solution doesn't work for larger values of N^M. Even if we modified it to work for an arbitrarily large $K = N^M$, it would be too slow. Computing F^{N^M} (modulo 10 000 103) requires $O(\log(N^M)) = O(M \log N)$ time.

Use of modulo 10 000 103 not only simplifies arithmetical operations: for any $P > 0$, the sequence $(Fib_i \mod P)$ is periodic. It follows from the fact that values of $Fib_i \mod P$ and $Fib_{i+1} \mod P$ fully determine the following elements of this sequence, and there are at most P^2 different pairs of such values. In particular, for $P = 10\,000\,103$ the sequence $(Fib_i \mod P)$ has period $Q = 20\,000\,208$. Its value can be established by the use of a simple program.

2: Function in Pascal that computes the period for a given P.

```
function count_period(P : longint) : longint;
var
   a, b, c, i : longint;
begin
   a := 1;
   b := 2;
   i := 1;
   while (a <> b) or (a <> 1) do begin
      c := (a+b) mod P;
      a := b;
      b := c;
      inc(i);
   end;
   count_period := i
end;
```

So, the task reduces to computing $(Fib_{N^M \mod Q}) \mod P$.

3: Procedure in Pascal that computes $N^M \mod Q$

```
function exp(N : int64; M : int64) : longint;
begin
   if M = 0 then
      exp := 1
   else if M mod 2 = 0 then
      exp := exp((N*N) mod Q, M div 2)
   else
      exp := (N*exp(N, M-1)) mod Q
end;
```

Finally, here is a solution to this task:

4: Final solution.

```
function power_fib(N: longint; M : longint) : longint;
begin
   power_fib := Fib(exp(N,M));
end;
```