Last update: April 22, 2025 Translated From: e-maxx.ru

Fibonacci Numbers

The Fibonacci sequence is defined as follows:

$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$$

The first elements of the sequence (OEIS A000045) are:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

Properties

Fibonacci numbers possess a lot of interesting properties. Here are a few of them:

- Cassini's identity:

$$F_{n-1}F_{n+1} - F_n^2 = (-1)^n$$

> This can be proved by induction. A one-line proof by Knuth comes from taking the determinant of the 2x2 matrix form below.

- The "addition" rule:

$$F_{n+k} = F_k F_{n+1} + F_{k-1} F_n$$

- Applying the previous identity to the case $k = n$, we get:

$$F_{2n} = F_n (F_{n+1} + F_{n-1})$$

- From this we can prove by induction that for any positive integer k, F_{nk} is multiple of F_n.
- The inverse is also true: if F_m is multiple of F_n, then m is multiple of n.
- GCD identity:

$$GCD(F_m, F_n) = F_{GCD(m,n)}$$

- Fibonacci numbers are the worst possible inputs for Euclidean algorithm (see Lame's theorem in Euclidean algorithm)

Fibonacci Coding

We can use the sequence to encode positive integers into binary code words. According to Zeckendorf's theorem, any natural number n can be uniquely represented as a sum of Fibonacci numbers:

$$N = F_{k_1} + F_{k_2} + \ldots + F_{k_r}$$

such that $k_1 \geq k_2 + 2,\ k_2 \geq k_3 + 2,\ \ldots,\ k_r \geq 2$ (i.e.: the representation cannot use two consecutive Fibonacci numbers).

It follows that any number can be uniquely encoded in the Fibonacci coding. And we can describe this representation with binary codes $d_0 d_1 d_2 \ldots d_s 1$, where d_i is 1 if F_{i+2} is used in the representation. The code will be appended by a 1 to indicate the end of the code word. Notice that this is the only occurrence where two consecutive 1-bits appear.

$$\begin{aligned}
1 = 1 &= F_2 &= (11)_F \\
2 = 2 &= F_3 &= (011)_F \\
6 = 5 + 1 &= F_5 + F_2 &= (10011)_F \\
8 = 8 &= F_6 &= (000011)_F \\
9 = 8 + 1 &= F_6 + F_2 &= (100011)_F \\
19 = 13 + 5 + 1 &= F_7 + F_5 + F_2 &= (1001011)_F
\end{aligned}$$

The encoding of an integer n can be done with a simple greedy algorithm:

1. Iterate through the Fibonacci numbers from the largest to the smallest until you find one less than or equal to n.
2. Suppose this number was F_i. Subtract F_i from n and put a 1 in the $i-2$ position of the code word (indexing from 0 from the leftmost to the rightmost bit).
3. Repeat until there is no remainder.
4. Add a final 1 to the codeword to indicate its end.

To decode a code word, first remove the final 1. Then, if the i-th bit is set (indexing from 0 from the leftmost to the rightmost bit), sum F_{i+2} to the number.

Formulas for the n^{th} Fibonacci number

Closed-form expression

There is a formula known as "Binet's formula", even though it was already known by Moivre:

$$F_n = \frac{\left(\frac{1 + \sqrt{5}}{2}\right)^n - \left(\frac{1 - \sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

This formula is easy to prove by induction, but it can be deduced with the help of the concept of generating functions or by solving a functional equation.

You can immediately notice that the second term's absolute value is always less than 1, and it also decreases very rapidly (exponentially). Hence the value of the first term alone is "almost" F_n. This can be written strictly as:

$$F_n = \left[\frac{\left(\frac{1+\sqrt{5}}{2}\right)^n}{\sqrt{5}}\right]$$

where the square brackets denote rounding to the nearest integer.

As these two formulas would require very high accuracy when working with fractional numbers, they are of little use in practical calculations.

Fibonacci in linear time

The n-th Fibonacci number can be easily found in $O(n)$ by computing the numbers one by one up to n. However, there are also faster ways, as we will see.

We can start from an iterative approach, to take advantage of the use of the formula $F_n = F_{n-1} + F_{n-2}$, therefore, we will simply precalculate those values in an array. Taking into account the base cases for F_0 and F_1.

```
int fib(int n) {
    int a = 0;
    int b = 1;
    for (int i = 0; i < n; i++) {
        int tmp = a + b;
        a = b;
        b = tmp;
    }
    return a;
}
```

In this way, we obtain a linear solution, $O(n)$ time, saving all the values prior to n in the sequence.

Matrix form

To go from (F_n, F_{n-1}) to (F_{n+1}, F_n), we can express the linear recurrence as a 2x2 matrix multiplication:

$$\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}F_n\\ F_{n-1}\end{pmatrix} = \begin{pmatrix}F_n + F_{n-1}\\ F_n\end{pmatrix} = \begin{pmatrix}F_{n+1}\\ F_n\end{pmatrix}$$

This lets us treat iterating the recurrence as repeated matrix multiplication, which has nice properties. In particular,

$$\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^n\begin{pmatrix}F_1\\ F_0\end{pmatrix} = \begin{pmatrix}F_{n+1}\\ F_n\end{pmatrix}$$

where $F_1 = 1, F_0 = 0$. In fact, since

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix}$$

we can use the matrix directly:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

Thus, in order to find F_n in $O(\log n)$ time, we must raise the matrix to n. (See Binary exponentiation)

```
struct matrix {
    long long mat[2][2];
    matrix friend operator *(const matrix &a, const matrix &b){
        matrix c;
        for (int i = 0; i < 2; i++) {
          for (int j = 0; j < 2; j++) {
              c.mat[i][j] = 0;
              for (int k = 0; k < 2; k++) {
                  c.mat[i][j] += a.mat[i][k] * b.mat[k][j];
              }
          }
        }
        return c;
    }
};

matrix matpow(matrix base, long long n) {
    matrix ans{ {
      {1, 0},
      {0, 1}
    } };
    while (n) {
        if(n&1)
            ans = ans*base;
        base = base*base;
        n >>= 1;
    }
    return ans;
}

long long fib(int n) {
    matrix base{ {
      {1, 1},
      {1, 0}
    } };
    return matpow(base, n).mat[0][1];
}
```

Fast Doubling Method

By expanding the above matrix expression for $n = 2 \cdot k$

$$\begin{pmatrix} F_{2k+1} & F_{2k} \\ F_{2k} & F_{2k-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{2k} = \begin{pmatrix} F_{k+1} & F_{k} \\ F_{k} & F_{k-1} \end{pmatrix}^{2}$$

we can find these simpler equations:

$$\begin{aligned} F_{2k+1} &= F_{k+1}^2 + F_k^2 \\ F_{2k} &= F_k(F_{k+1} + F_{k-1}) = F_k(2F_{k+1} - F_k) \end{aligned}.$$

Thus using above two equations Fibonacci numbers can be calculated easily by the following code:

```
pair<int, int> fib (int n) {
    if (n == 0)
        return {0, 1};

    auto p = fib(n >> 1);
    int c = p.first * (2 * p.second - p.first);
    int d = p.first * p.first + p.second * p.second;
    if (n & 1)
        return {d, c + d};
    else
        return {c, d};
}
```

The above code returns F_n and F_{n+1} as a pair.

Periodicity modulo p

Consider the Fibonacci sequence modulo p. We will prove the sequence is periodic.

Let us prove this by contradiction. Consider the first $p^2 + 1$ pairs of Fibonacci numbers taken modulo p:

$$(F_0,\ F_1),\ (F_1,\ F_2),\ \ldots,\ (F_{p^2},\ F_{p^2+1})$$

There can only be p different remainders modulo p, and at most p^2 different pairs of remainders, so there are at least two identical pairs among them. This is sufficient to prove the sequence is periodic, as a Fibonacci number is only determined by its two predecessors. Hence if two pairs of consecutive numbers repeat, that would also mean the numbers after the pair will repeat in the same fashion.

We now choose two pairs of identical remainders with the smallest indices in the sequence. Let the pairs be $(F_a,\ F_{a+1})$ and $(F_b,\ F_{b+1})$. We will prove that $a = 0$. If this was false, there would be two previous pairs $(F_{a-1},\ F_a)$ and $(F_{b-1},\ F_b)$, which, by the property of Fibonacci numbers, would also be equal. However, this contradicts the fact that we had chosen pairs with the smallest indices, completing our proof that there is no pre-period (i.e the numbers are periodic starting from F_0).

Practice Problems

- SPOJ - Euclid Algorithm Revisited
- SPOJ - Fibonacci Sum

- HackerRank - Is Fibo
- Project Euler - Even Fibonacci numbers
- DMOJ - Fibonacci Sequence
- DMOJ - Fibonacci Sequence (Harder)
- DMOJ UCLV - Numbered sequence of pencils
- DMOJ UCLV - Fibonacci 2D
- DMOJ UCLV - fibonacci calculation
- LightOJ - Number Sequence
- Codeforces - C. Fibonacci
- Codeforces - A. Hexadecimal's theorem
- Codeforces - B. Blackboard Fibonacci
- Codeforces - E. Fibonacci Number

Contributors:

gabrielsimoes (23.55%) jakobkogler (20.48%) jxu (18.77%) Carlos Javier Blanco (15.7%) madhur4127 (6.14%) tcNickolas (5.12%) adamant-pwn (2.73%) Morass (2.05%) md-shamim-ahmad (1.71%) boxlesscat (1.02%) yeetholmes619 (1.02%) pasthec (0.68%) mhayter (0.34%) conlacda (0.34%) ankit-kumar-dwivedi (0.34%)