Fibonacci EC report

December 8, 2022

1 Find i-th Fibonacci number efficiently

Suppose i is any integer. We describe an algorithm for finding i-th Fibonacci number in O(log(i)) time.
We define i-th Fibonacci term F_i as following:

$$F_i = \begin{cases} F_{i-2} + F_{i-1} & \text{if } i >= 1 \\ 1 & \text{if i = 0} \\ 0 & \text{if } i < 0 \end{cases}$$

1.1 theorem 1:

For all integers j such that $j >= 0$,

$$\begin{bmatrix} F_{j+1} \\ F_j \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} F_j \\ F_{j-1} \end{bmatrix}$$

1.1.1 Proof for theorem 1:

Since $j >= 0 \implies j + 1 >= 1$, by definition of Fibonacci sequence, we have

$$F_{j+1} = F_{j-1} + F_j$$

$$F_j = F_j$$

which is equivalent to the matrix multiplication above.

1.2 Theorem 2:

Let T = $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ which we call as a transition matrix. For all non-negative integers n,

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = T^n \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{1}$$

1.2.1 Proof by induction for theorem 2:

Suppose P(n) is a statement:

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = T^n \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2}$$

Show P(0) is true:
To establish P(0), we must show

$$\begin{bmatrix} F_0 \\ F_{-1} \end{bmatrix} = T^0 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{3}$$

Notice

$$LHS = \begin{bmatrix} F_0 \\ F_{-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

and

$$RHS = T^0 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = I \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Thus, the left hand side and right hand side of P(0) is the same. Hence, P(0) is true.
Show if P(n) is true, then P(n+1) must be true:
Suppose n is any non-negative integer such that

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = T^n \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{4}$$

We wish to show

$$\begin{bmatrix} F_{n+1} \\ F_n \end{bmatrix} = T^{n+1} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{5}$$

Notice

$$\begin{aligned} & T^{n+1} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ &= T \cdot (T^n \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}) \text{ (by Alg)} \\ &= T \cdot \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} \text{ (by Inductive Hypothesis)} \\ &= \begin{bmatrix} F_{n+1} \\ F_n \end{bmatrix} \text{ (by Theorem 1)} \end{aligned} \tag{6}$$

This is what we wished to show.
Since we have shown basis step and inductive step, the theorem must be true.

1.3 Algorithm 1:

By theorem 2, we know that:

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = T^n \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{7}$$

We can accurately calculate T^n within O(log n) time complexity using an algorithm called binary exponentiation.

Notice

$$T^n = \begin{cases} I & \text{if } n == 0 \\ (T^{\frac{n}{2}})^2 & \text{if n is even and } n > 0 \\ (T^{\frac{n-1}{2}})^2 \cdot T & \text{if n is odd and } n > 0 \end{cases}$$

Using this relation, we can consider T^n as a recursive function with argument n.

Pseudocode:

```
function power_matrix(T, n) is
    if n == 0 do
        return I
    if n is even:
        P := power_matrix(T, n/2)
        return P * P
    else:
        P := power_matrix(T, (n-1)/2)
        return P * P * T
```

We can see the time complexity of the function is O(log(n)) since the n is divided by 2 every invocation of the function. So, the entire calculation of F_n becomes O(log(n)) although the cost of matrix multiplication is now not a constant time calculation once number gets larger.

2 Determine which term in the Fibonacci sequence is n efficiently

Since we can calculate F_i within a O(log(i)) time complexity, we can find the index of n in the Fibonacci sequence within O(log(i)*log(i)) time complexity using binary search where i is the index of n in Fibonacci sequence.

We are binary searching the Fibonacci sequence to find the index of the term, only calculating the terms that are needed. But, since the size of Fibonacci sequence is infinite, we want to find an possible "upper bound" for the index of the term.

We claim that the upper bound $UB = 1 + 2 * log_2(n)$

(We found this by looking general formula of Fibonacci sequence ($F_n = \frac{1}{\sqrt{5}} \cdot (\frac{1+\sqrt{5}}{2})^n$) using real number multiplications. We realized we can't use this formula in computer reliably since real number is represented as floating point number which is destined to have inaccuracy even if we increase the bits for the number, but it was useful for finding this bound.)

2.1 Theorem 3:

For all integers i such that $i >= 0, i < 1 + 2 * log_2(F_i)$

2.1.1 Proof by induction for theorem 3:

Suppose P(i) is an inequality

$$i < 1 + 2 * log_2(F_i)$$

Show P(0) is true:
To establish P(0), we must show

$$0 < 1 + 2 * log_2(F_0)$$

Notice $1 + 2 * log_2(F_0) = 1 + 2 * log_2(1) = 1$ and $0 < 1$
Hence, P(0) is true.
Show P(1) is true:
To establish P(1), we must show

$$0 < 1 + 2 * log_2(F_1)$$

Notice $1 + 2 * log_2(F_1) = 1 + 2 * log_2(1) = 1$ and $0 < 1$
Hence, P(1) is true.
Show if p(i-1) and P(i) is true, then P(i+1) must be true:
Suppose i is any integer such that $i >= 2$ and

$$i - 1 < 1 + 2 * log_2(F_{i-1})$$
$$i < 1 + 2 * log_2(F_i)$$

Since a exponential function is monotonically increasing function, these are equivalent to

$$2^{i-1} < 2^{1+2*log_2(F_{i-1})} = 2 * F_{i-1}^2$$
$$2^i < 2^{1+2*log_2(F_i)} = 2 * F_i^2$$

We wish to show

$$i + 1 < 1 + 2 * log_2(F_{i+1})$$

Since a exponential function is monotonically increasing function, this is equivalent to

$$2^{i+1} < 2^{1+2*log_2(F_{i+1})} = 2 * F_{i+1}^2$$

Then,

$$2^{i-1} < 2 * F_{i-1}^2$$
$$< 2 * F_{i-1} * F_i \text{ (since } F_{i-1} < F_i \text{ and both are positive)}$$
$$\implies 2^i < 4 * F_{i-1} * F_i \text{ (by Alg)}$$

Notice

$$2^{i+1} = 2^i + 2^i < 2 * F_i^2 + 4 * F_{i-1} * F_i$$

(since $2^i < 2 * F_i^2$ by inductive hypothesis and $2^i < 4 * F_{i-1} * F_i$ as we've shown above)

$$< 2 * F_i^2 + 4 * F_{i-1} * F_i + 2 * F_{i-1}^2 \text{ (since } 2 * F_{i-1}^2 >= 0)$$
$$= 2 * (F_{i-1} + F_i)^2 \text{ (by Alg)}$$
$$= 2 * F_{i+1}^2 \text{ (by def of Fibonacci term)}$$

This is what we wished to show.
Since we have shown basis step and inductive step, the theorem must be true.

2.2 Algorithm 2:

By theorem 3, we can binary search the range [0, $1+2*log_2(n)$] of the Fibonacci sequence to always find the right term if n is Fibonacci number. This will look at O(log(2*$log_2(n)$)) terms of Fibonacci sequence and querying each term F_i takes O(log(i)) time by using Algorithm 1. In total, this process takes O(log log(n)*log log log(n)).