

Analisis y Diseño de Algoritmos

Marco Antonio Bastida Flores

06 Junio 2023

En el presente reporte se muestra el desarrollo del algoritmo para obtener el n-esimo numero de Fibonacci utilizando un algoritmo logaritmico.

1 Fibonacci

La serie de fibonacci se puede obtener de manera matricial utilizando la siguiente formula:

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n = \begin{bmatrix} f_n - 1 & f_n \\ f_n & f_n + 1 \end{bmatrix}$$

Se puede observar que al multiplicar $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ por si misma n veces, se obtiene f_n que representa el n-esimo numero de la serie de Fibonacci. Y $f_n + 1$ que representa el siguiente número en la serie de Fibonacci.

El código para la multiplicación de dos matrices cuadradas se muestra a continuación.

```
def mul_square_matrix(A, B):  
    C = [ [0,0], [0,0] ]  
    C[0][0] = A[0][0]*B[0][0] + A[0][1]*B[1][0]  
    C[0][1] = A[0][0]*B[0][1] + A[0][1]*B[1][1]  
    C[1][0] = A[1][0]*B[0][0] + A[1][1]*B[1][0]  
    C[1][1] = A[1][0]*B[0][1] + A[1][1]*B[1][1]  
    return C
```

Dado que sabemos de antemano que nuestro algoritmo ocupará únicamente matrices cuadradas de 2×2 , podemos hacer uso de las ecuaciones de Strassen para resolver la multiplicación de una manera sencilla. La función `mul_square_matrix(A, B)` Toma dos matrices cuadradas A y B y retorna el resultado de la multiplicación $A * B$.

2 Exponenciación

Para multiplicar n veces la matriz base, utilizaremos el algoritmo de "squaring by exponentiation" que se representa de la siguiente manera:

$$x^n = \begin{cases} x(x^2)^{(n-1)/2} & \text{Si } n \text{ es impar} \\ (x^2)^{n/2} & \text{Si } n \text{ es par} \end{cases}$$

El código para obtener la potencia de un número usando exponenciación se muestra a continuación:

```
def exp_squaring(base, exp):
    if exp == 0:
        return 1
    while exp > 0:
        if exp % 2 == 1:
            return base * exp_squaring( base * base, (exp - 1) / 2 )
        else:
            return exp_squaring( base * base, exp / 2 )
```

Utilizando las condiciones de la definición se parte el problema por la mitad, utilizando un algoritmo recursivo donde el caso base es cuando el exponente es 0 y el resultado es 1.

3 Integración de algoritmos para calcular en n-esimo numero de Fibonacci

Ahora, se integran ambos módulos para obtener la n-ésima potencia de la matriz base $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ para obtener la serie de Fibonacci.

El código se muestra a continuación.

```
base_matrix = [ [0,1], [1,1] ]
identity_matrix = [ [1,0], [0,1] ]

def mul_square_matrix(A, B):
    C = [ [0,0], [0,0] ]
    C[0][0] = A[0][0]*B[0][0] + A[0][1]*B[1][0]
    C[0][1] = A[0][0]*B[0][1] + A[0][1]*B[1][1]
    C[1][0] = A[1][0]*B[0][0] + A[1][1]*B[1][0]
    C[1][1] = A[1][0]*B[0][1] + A[1][1]*B[1][1]
    return C

def exp_square(base, n):
    if n == 0 :
        return identity_matrix
    while n > 0 :
        if n % 2 == 1:
            return mul_square_matrix(base, exp_square(\
                mul_square_matrix(base, base), (n - 1)/2 ) )
        else:
            return exp_square( mul_square_matrix(base, base), n / 2 )

def fibonacci(n):
    return exp_square(base_matrix, n)[1][0]
```

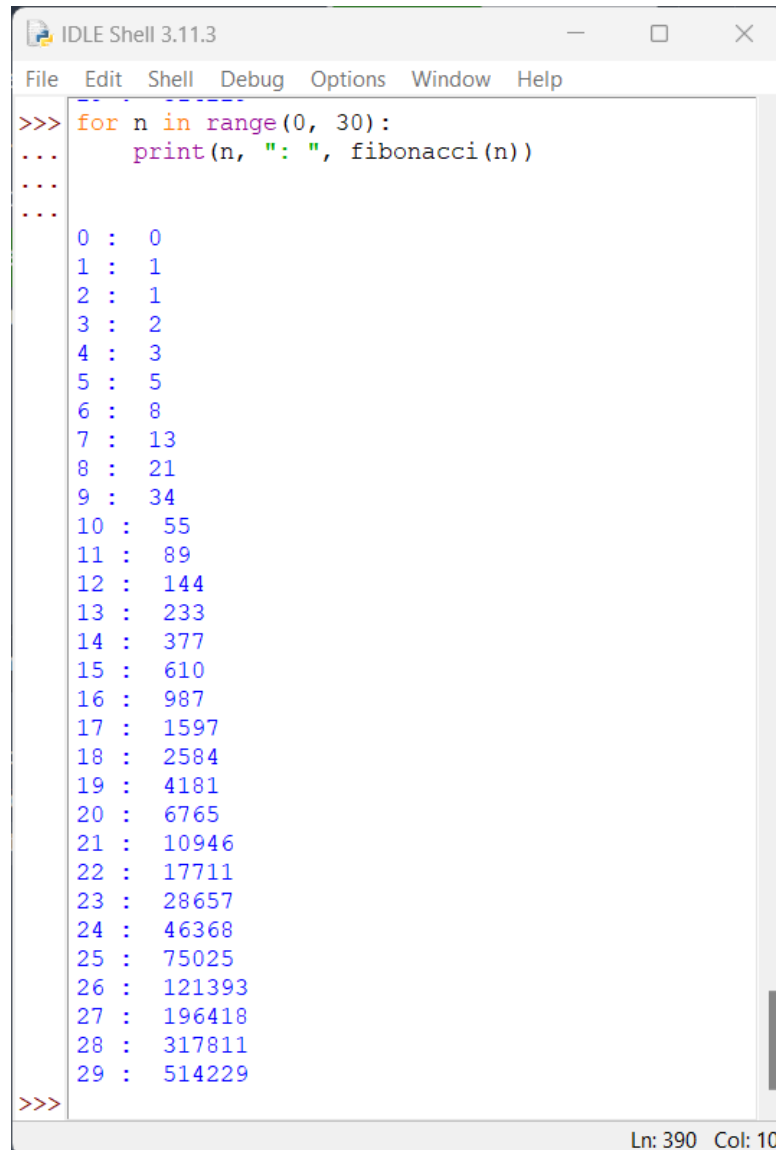
Se sustituye en el algoritmo de exponenciación la multiplicación de enteros por la multiplicación de matrices usando la función `mul_square_matrix(A, B)`. El caso base será cuando el exponente n sea igual a 0 y en este caso se hace uso de la matriz identidad $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ para sustituir el caso base de multiplicar por 1, para ser consistentes con el algoritmo que exponentia números enteros y que se toma como base.

3.1 Resultados

Para validar que se obtiene la serie de Fibonacci con el algoritmo se hace uso del siguiente snippet:

```
for n in range(0, 30):  
    print(n, ": ", fibonacci(n))
```

En la Figura 1 se puede observar que la secuencia de los números de la serie de Fibonacci se siguen correctamente.



The screenshot shows a window titled 'IDLE Shell 3.11.3' with a menu bar (File, Edit, Shell, Debug, Options, Window, Help). The shell contains the following code and output:

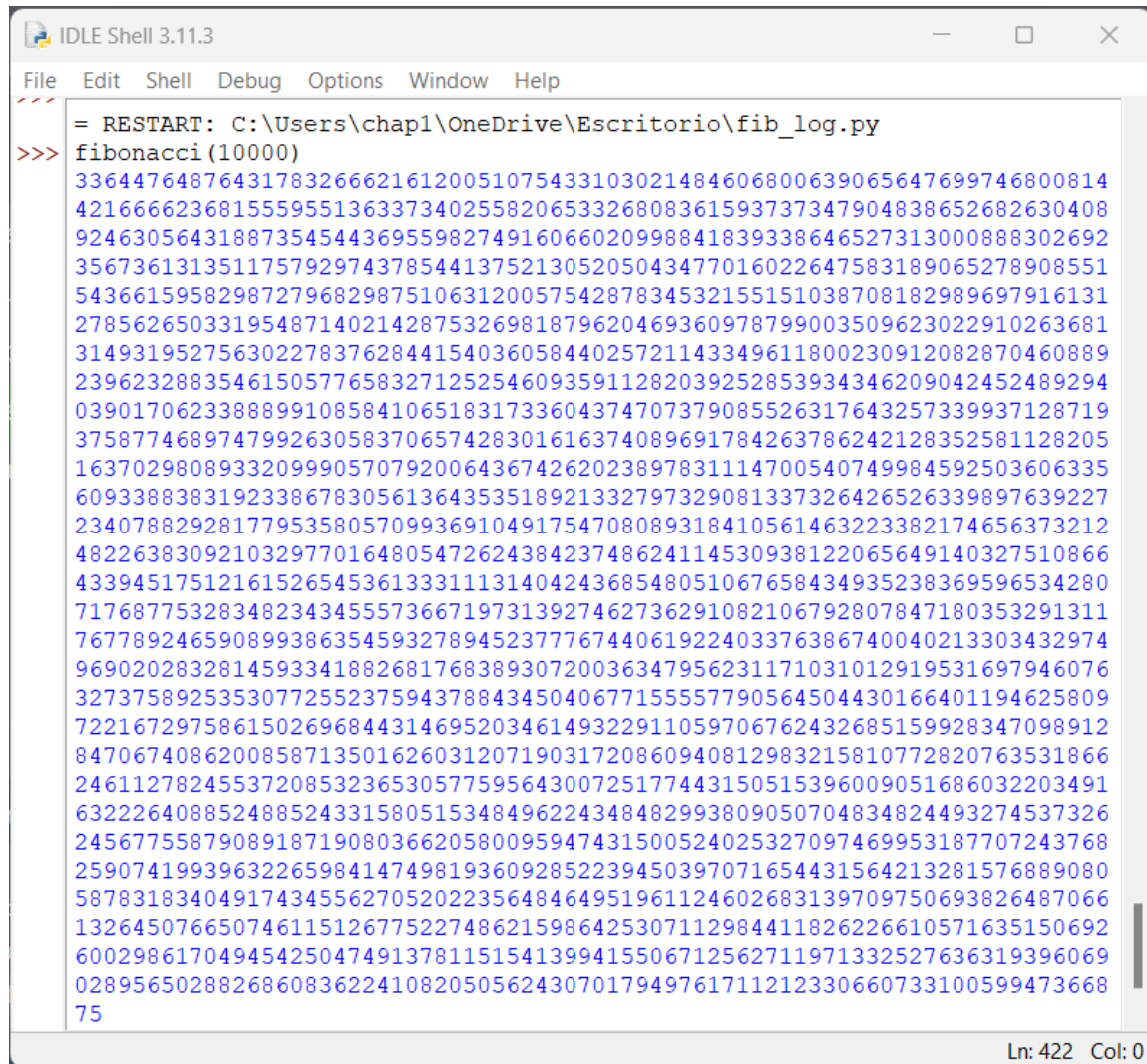
```
>>> for n in range(0, 30):  
...     print(n, ": ", fibonacci(n))  
...  
...  
0 : 0  
1 : 1  
2 : 1  
3 : 2  
4 : 3  
5 : 5  
6 : 8  
7 : 13  
8 : 21  
9 : 34  
10 : 55  
11 : 89  
12 : 144  
13 : 233  
14 : 377  
15 : 610  
16 : 987  
17 : 1597  
18 : 2584  
19 : 4181  
20 : 6765  
21 : 10946  
22 : 17711  
23 : 28657  
24 : 46368  
25 : 75025  
26 : 121393  
27 : 196418  
28 : 317811  
29 : 514229  
>>>
```

The status bar at the bottom right indicates 'Ln: 390 Col: 10'.

Figure 1: Validación del algoritmo de potencia por exponenciación

En la figura 2 se hace una prueba con un número de Fibonacci igual a 10,000 y se puede observar que

el resultado se pinta en consola casi inmediatamente.



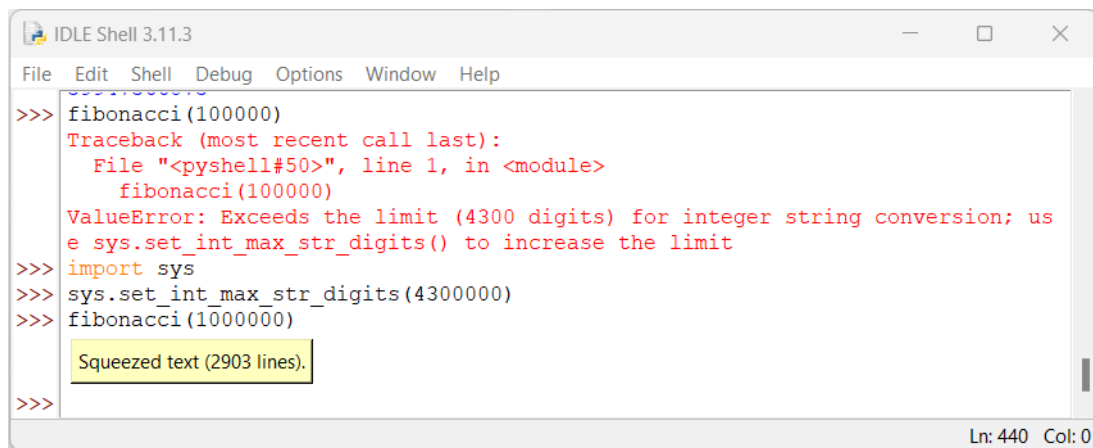
```

IDLE Shell 3.11.3
File Edit Shell Debug Options Window Help
= RESTART: C:\Users\chap1\OneDrive\Escritorio\fib_log.py
>>> fibonacci(10000)
336447648764317832666216120051075433103021484606800639065647699746800814
421666623681555955136337340255820653326808361593737347904838652682630408
924630564318873545443695598274916066020998841839338646527313000888302692
356736131351175792974378544137521305205043477016022647583189065278908551
543661595829872796829875106312005754287834532155151038708182989697916131
278562650331954871402142875326981879620469360978799003509623022910263681
314931952756302278376284415403605844025721143349611800230912082870460889
239623288354615057765832712525460935911282039252853934346209042452489294
039017062338889910858410651831733604374707379085526317643257339937128719
375877468974799263058370657428301616374089691784263786242128352581128205
163702980893320999057079200643674262023897831114700540749984592503606335
609338838319233867830561364353518921332797329081337326426526339897639227
234078829281779535805709936910491754708089318410561463223382174656373212
482263830921032977016480547262438423748624114530938122065649140327510866
433945175121615265453613331113140424368548051067658434935238369596534280
717687753283482343455573667197313927462736291082106792807847180353291311
767789246590899386354593278945237776744061922403376386740040213303432974
969020283281459334188268176838930720036347956231171031012919531697946076
327375892535307725523759437884345040677155557790564504430166401194625809
722167297586150269684431469520346149322911059706762432685159928347098912
847067408620085871350162603120719031720860940812983215810772820763531866
246112782455372085323653057759564300725177443150515396009051686032203491
632226408852488524331580515348496224348482993809050704834824493274537326
245677558790891871908036620580095947431500524025327097469953187707243768
259074199396322659841474981936092852239450397071654431564213281576889080
587831834049174345562705202235648464951961124602683139709750693826487066
132645076650746115126775227486215986425307112984411826226610571635150692
600298617049454250474913781151541399415506712562711971332527636319396069
028956502882686083622410820505624307017949761711212330660733100599473668
75
Ln: 422 Col: 0

```

Figure 2: Algoritmo exponencial para obtener el 10,000 número de Fibonacci

Por ultimo se muestra en la figura 3 el calculo de el millonesimo numero de Fibonacci pudiendo observar que el tiempo que tomo para hacer el calculo fue de unos cuantos segundos.



```
File Edit Shell Debug Options Window Help
>>> fibonacci(1000000)
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    fibonacci(1000000)
ValueError: Exceeds the limit (4300 digits) for integer string conversion; use sys.set_int_max_str_digits() to increase the limit
>>> import sys
>>> sys.set_int_max_str_digits(4300000)
>>> fibonacci(1000000)
Squeezed text (2903 lines).
>>>
Ln: 440 Col: 0
```

Figure 3: Algoritmo exponencial para obtener el 1,000,000 número de Fibonacci



[View the Code on GitHub](#)

[Click to browse repository](#)