Algoritmos para calcular la sucesión de FIBONNACI usando latexify para generar el código LaTeX

Resumen

En este artículo uso como excusa la sucesión de FIBONACCI y algunos algoritmos que permiten su cálculo para analizar la forma en que puedo usar el paquete `latexify` de `python` para obtener, a partir del código `python` y de forma directa, los algoritmos de las funciones así como las expresiones de las funciones matemáticas en código LaTeX. Además uso `python` para realizar los cálculos, `pandas` para realizar las tablas y el paquete `pythontex` para crear de forma dinámica el pdf resultado de compilar el fichero LyX. Los algoritmos usados se pueden encontrar por internet en las páginas referenciadas. Termino el artículo con el cálculo del término 10000 de la sucesión.

La sucesión de FIBONACCI es la sucesión infinita de números naturales

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

donde el primer elemento es 0, el segundo es 1 y cada elemento restante es la suma de los dos anteriores. A cada elemento de esta sucesión se le llama número de FIBONACCI. Se debe a Leonardo de Pisa (1170-1250), matemático italiano del siglo XIII también conocido como FIBONACCI (hijo de Bonacci[1]). La ideó como la solución a un problema de la cría de conejos:

Cierto hombre tenía una pareja de conejos juntos en un lugar cerrado y uno desea saber cuántos son creados a partir de este par en un año cuando es su naturaleza parir otro par en un simple mes, y en el segundo mes los nacidos parir también.

Para obtener sus términos, se cumple la relación de recurrencia siguiente:

$$\mathrm{fib}(x)=\begin{cases}0, & \mathrm{if}\ x=0\\ 1, & \mathrm{if}\ x=1\\ \mathrm{fib}(x-1)+\mathrm{fib}(x-2), & \mathrm{otherwise}\end{cases}$$

Podemos programar fácilmente el algoritmo que se deduce de la fórmula anterior a partir de una función recursiva:

```
function FIB(x)
    if x = 0 then
        return 0
    else
        if x = 1 then
            return 1
        else
            return fib(x − 1) + fib(x − 2)
        end if
    end if
end function
```

[1] figlio de Bonacci

Las dos expresiones anteriores se han obtenido usando latexify a partir de las funciones en código python se han escrito en código LaTeX:

```
$\py{latexify.get_latex(fib,use_signature=True)}$
\py{latexify.get_latex(fib,style=latexify.Style.ALGORITHMIC)}
```

El mismo procedimiento se ha seguido para obtener el resto de algoritmos del documento ajustando el nombre de las funciones.
Para poder usar el código devuelto por latexify en el estilo ALGORITHMIC es necesario tener cargado el paquete de LaTeX
`\usepackage{algpseudocode}`

En `python` quedaría:

```
def fib(x):
  if x == 0:
    return 0
  elif x == 1:
    return 1
  else:
    return fib(x-1) + fib(x-2)
```

con la función anterior, por ejemplo, podemos obtener que $fib(20) = 6765$. Pero su eficiencia es muy pobre. La programación recursiva de la función de FIBONACCI tiene una complejidad, como mínimo exponencial (véase, por ejemplo https://www.ugr.es/ eaznar/fibo.htm). De hecho, calcular el término 100 con la función anterior ya no es factible.

Existen algoritmos mucho más eficientes (por ejemplo la solución iterativa aunque la forma directa tampoco es buena) y me voy a referir a dos de ellos, el primero tomado de la Wikipedia

https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci. Para construirlo debemos usar dos cuestiones:

1. Usar un algoritmo óptimo para calcular potencias:

$$x^n = \begin{cases} 1 & n = 0 \\ \left(x^{n/2}\right)^2 & n\ par \\ x \cdot x^{n-1} & n\ impar \end{cases}$$

o equivalentemente:

$$\mathrm{potencia}(x, n) = \left\{ \begin{array}{ll} x, & \mathrm{if} \ n = 1 \\ \left(\mathrm{potencia} \mathopen{}\left(x, \left\lfloor \frac{n}{2} \right\rfloor \mathclose{}\right) \right)^{2}, & \mathrm{if} \ n \mathbin{\%} 2 = 0 \\ x \cdot \mathrm{potencia} \mathopen{}\left(x, n - 1 \mathclose{}\right), & \mathrm{otherwise} \end{array} \right.$$

cuyo algoritmo es:

function POTENCIA(x, n)
  if $n = 1$ **then**
    return x
  else
    if $n \% 2 = 0$ **then**
      return $\left(\mathrm{potencia}\left(x, \left\lfloor \frac{n}{2} \right\rfloor\right)\right)^2$
    else
      return $x \cdot \mathrm{potencia}(x, n-1)$

end if
end if
end function

y en python podría ser:

```
# Potencia
def potencia(x,n):
    if n==1:
        return x
    elif n % 2 == 0:
        return (potencia(x,n // 2))**2
    else:
        return x*potencia(x,n-1)
```

Pero no vamos a usar el algoritmo anterior, sino otro (véase, por ejemplo, https://www.geeksforgeeks.org/fast-exponentiation-in-python/) para calcular la potencia de un número de forma iterativa que consiste en

Algoritmo 1 Algoritmo de la potencia óptima iterativa

function POT_ITE(x,n)
  $i \leftarrow n$
  $a \leftarrow 1$
  $c \leftarrow x$
  while $i > 0$ **do**
    if $i \,\%\, 2 \neq 0$ **then**
      $a \leftarrow ac$
    end if
    $c \leftarrow c^2$
    $i \leftarrow \left\lfloor \frac{i}{2} \right\rfloor$
  end while
  return a
end function

y en python podría ser:

Algoritmo 2 Potencia iterativa en python

```
# Potencia
def pot_ite(x,n):
    i=n
    a = 1
    c = x
    while i > 0:
        if i % 2 != 0:
            a = a * c
        c = c**2
        i = i // 2
    return a
```

Con él podemos obtener, por ejemplo, que `pot_ite(2,10)` $= 2^{10} = 1024$

2. Además, podemos usar matrices para determinar los términos de la sucesión de FIBONACCI

Proposición.

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix}$$

Demostración.

Demostración. Se demuestra por inducción:

- $n = 1$
  $$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 = \begin{bmatrix} F(2) & F(1) \\ F(1) & F(0) \end{bmatrix} \text{ que es cierto}$$
- Suponemos cierto para $n \longmapsto \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix}$
- Veamos que es cierto para $n+1$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n+1} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} =$$
$$= \begin{bmatrix} F(n+1)+F(n) & F(n+1)+0 \\ F(n)+F(n-1) & F(n)+0 \end{bmatrix} = \begin{bmatrix} F(n+2) & F(n+1) \\ F(n+1) & F(n) \end{bmatrix}$$

□

□

Observar que las matrices
$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix}$$
quedan completamente determinada solo por dos valores, si $F(n-1) = a$ y $F(n) = b \Rightarrow F(n+1) = a+b$, es decir, todas son de la forma $\begin{bmatrix} a+b & b \\ b & a \end{bmatrix}$

Si llamamos $c = F(k-1)$ y $d = F(k) \Rightarrow F(k+1) = c+d$, por tanto

$$\begin{bmatrix} F(2k+1) & F(2k) \\ F(2k) & F(2k-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{2k} = \left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^k \right)^2 = \begin{bmatrix} F(2k+1) & F(2k) \\ F(2k) & F(2k-1) \end{bmatrix}^2 =$$
$$= \begin{bmatrix} c+d & d \\ d & c \end{bmatrix}^2 = \begin{bmatrix} c+d & d \\ d & c \end{bmatrix} \cdot \begin{bmatrix} c+d & d \\ d & c \end{bmatrix} =$$
$$\begin{bmatrix} d^2+(c+d)^2 & cd+d(c+d) \\ cd+d(c+d) & c^2+d^2 \end{bmatrix} = \begin{bmatrix} d^2+(c+d)^2 & d(2c+d) \\ d(2c+d) & c^2+d^2 \end{bmatrix}$$

Es decir,

- si $n = 2 \cdot k$ (n par) y conozco $F(k-1) = c$ y $F(k) = d$ puedo calcular, a partir de los valores anteriores, tanto $F(n-1) = F(2k-1) = c^2+d^2$ como $F(n) = F(2k) = d \cdot (2 \cdot c + d)$
- si $n = 2 \cdot k + 1$ (n es impar) y aplico el algoritmo 2 en la página anterior tendría que multiplicar las matrices
  $$A * C = \begin{bmatrix} a+b & b \\ b & a \end{bmatrix} \cdot \begin{bmatrix} c+d & d \\ d & c \end{bmatrix} = \begin{bmatrix} bd+(a+b)(c+d) & bc+d(a+b) \\ ad+b(c+d) & ac+bd \end{bmatrix}$$
  y en consecuencia $F(n) = b \cdot c + d \cdot (a+b)$ y $F(n-1) = a \cdot c + b \cdot d$

A partir de lo anterior y el algoritmo para calcular la potencia tenemos que:

function FIB_OPTIMA(n)
  if $n \leq 0$ **then**
    return 0
  end if
  $i \leftarrow n-1$

$(a,b) \leftarrow (1,0)$
$(c,d) \leftarrow (0,1)$
while $i > 0$ **do**
    if $i \,\%\, 2 \neq 0$ **then**
        $(a,b) \leftarrow (ac+bd, bc+d\cdot(a+b))$
    end if
    $(c,d) \leftarrow \left(c^2+d^2, d\cdot(2c+d)\right)$
    $i \leftarrow \left\lfloor \frac{i}{2} \right\rfloor$
end while
return $a+b$
end function

y en python:

```
# Solución de wikipedia
def fib_optima(n):
    if n<=0:
        return 0
    # Llegamos hasta el n-1 ya que el último (F(n)) se obtiene sumando a+b
    i=n-1
    a, b = 1, 0
    c, d = 0, 1
    while i > 0:
        if i % 2 != 0:
            a, b = a*c+b*d, b*c+d*(a+b)
        c, d = c**2+d**2, d*(2*c+d)
        i = i // 2
    return a+b
```

con él, por ejemplo, ya sí podemos calcular:

$$fib(100) = 354224848179261915075$$

o:

$$fib(1000) = 434665576869374564356885276750406258025646605173717804024817290895365590403879840079255169295922593080322634775209689623239873322471161642996440906533179382989696499285160037044761377951668492288 75$$

En Project Nayuki tenemos otra forma de calcular los términos de la sucesión de forma incluso más eficiente, en este caso nos basamos en que:

$$\begin{bmatrix} F(2n+1) & F(2n) \\ F(2n) & F(2n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{2n} = \left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n} \right)^2 = \begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix}^2 =$$
$$\begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix} \cdot \begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix} = \begin{bmatrix} F(n+1)^2+F(n)^2 & F(n+1)F(n)+F(n)F(n-1) \\ F(n)F(n+1)+F(n-1)F(n) & F(n)^2+F(n-1)^2 \end{bmatrix}$$

Y en consecuencia obtenemos que:

$$\begin{aligned} F(2n+1) &= F(n+1)^2+F(n)^2 \\ F(2n) &= F(n)\left[F(n+1)+F(n-1)\right] \\ &= F(n)\left[F(n+1)+(F(n+1)-F(n))\right] \\ &= F(n)\left[2F(n+1)-F(n)\right] \\ F(2n-1) &= F(n)^2+F(n-1)^2 \end{aligned}$$

Es decir, si conozco $F(k) = a$ y $F(k+1) = b$ puedo calcular $c = F(2k) = a\cdot(2\cdot b - a)$ y $d = F(2k+1) = a^2 + b^2$, por tanto:

- Si $n = 2\cdot k$ es par $\Rightarrow F(n) = F(2\cdot k) = c = a\cdot(2\cdot b - a)$ y $F(n+1) = F(2k+1) = d = a^2 + b^2$
- Si $n = 2\cdot k + 1$ es impar $F(n) = F(2\cdot k+1) = d$ y $F(n+1) = F(2k+2) = F(2k) + F(2k+1) = c + d$

y obtenemos los algoritmos

function FIBONACCI(n)
  if $n < 0$ **then**
    return 0
  end if
  return _fibo$(n)_0$
end function

function _FIBO(n)
  if $n = 0$ **then**
    return $(0,1)$
  else
    $(a,b) \leftarrow$ _fibo$\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$
    $c \leftarrow a\cdot(2b - a)$
    $d \leftarrow a^2 + b^2$
    if $n \,\%\, 2 = 0$ **then**
      return (c,d)
    else
      return $(d, c+d)$
    end if
  end if
end function

y sus correspondientes funciones en `python`:

```python
# Algoritmo de https://www.nayuki.io/page/fast-fibonacci-algorithms
# (Pública) Retorna F(n).
def fibonacci(n):
        if n < 0:
                return 0
        return _fibo(n)[0]


# (Privada) Retorna la tupla (F(n), F(n+1)).
def _fibo(n):
        if n == 0:
                return (0, 1)
        else:
                a, b = _fibo(n // 2)
                c = a * (2 * b - a)
                d = a**2 + b**2
                if n % 2 == 0:
                        return (c, d)
                else:
                        return (d, c + d)
```

A partir de esta última, por ejemplo, podemos plantear y comprobar las cuestiones:

1. Halla el cociente $\frac{\text{fi}(n+1)}{\text{fib}(n)}$ para los 45 primeros términos de la sucesión (salvo el primero). ¿A qué número famoso se parece cada vez más?

n	F(n)	F(n+1)	Cociente
1	1	1	1.0000000000000000
2	1	2	2.0000000000000000
3	2	3	1.5000000000000000
4	3	5	1.6666666666666667
5	5	8	1.6000000000000001
6	8	13	1.6250000000000000
7	13	21	1.6153846153846154
8	21	34	1.6190476190476191
9	34	55	1.6176470588235294
10	55	89	1.6181818181818182
11	89	144	1.6179775280898876
12	144	233	1.6180555555555556
13	233	377	1.6180257510729614
14	377	610	1.6180371352785146
15	610	987	1.6180327868852460
16	987	1597	1.6180344478216819
17	1597	2584	1.6180338134001253
18	2584	4181	1.6180340557275541
19	4181	6765	1.6180339631667064
20	6765	10946	1.6180339985218033
21	10946	17711	1.6180339850173580
22	17711	28657	1.6180339901755971
23	28657	46368	1.6180339882053250
24	46368	75025	1.6180339889579021
25	75025	121393	1.6180339886704431
26	121393	196418	1.6180339887802426
27	196418	317811	1.6180339887383031
28	317811	514229	1.6180339887543225
29	514229	832040	1.6180339887482036
30	832040	1346269	1.6180339887505408
31	1346269	2178309	1.6180339887496482
32	2178309	3524578	1.6180339887499890
33	3524578	5702887	1.6180339887498589
34	5702887	9227465	1.6180339887499087
35	9227465	14930352	1.6180339887498896
36	14930352	24157817	1.6180339887498969
37	24157817	39088169	1.6180339887498940
38	39088169	63245986	1.6180339887498951
39	63245986	102334155	1.6180339887498947
40	102334155	165580141	1.6180339887498949
41	165580141	267914296	1.6180339887498949
42	267914296	433494437	1.6180339887498949
43	433494437	701408733	1.6180339887498949
44	701408733	1134903170	1.6180339887498949
45	1134903170	1836311903	1.6180339887498949

es decir: $\lim\limits_{n\to\infty} \dfrac{\text{fib}(n+1)}{\text{fib}(n)} = \varphi = \dfrac{1+\sqrt{5}}{2} \approx 1{,}6180339887498948482045\ldots$

2. Halla el máximo común divisor de dos números de FIBONACCI ¿qué se obtiene? ¿existe alguna relación entre ellos?

n	m	F(n)	F(m)	MCD(n,m)	F(MCD(n,m))	MCD(F(n),F(m))
10	39	55	63245986	1	1	1
15	43	610	433494437	1	1	1
20	32	6765	2178309	4	3	3
25	29	75025	514229	1	1	1
30	48	832040	4807526976	6	8	8
35	57	9227465	365435296162	1	1	1
40	70	102334155	190392490709135	10	55	55
45	56	1134903170	225851433717	1	1	1
50	61	12586269025	2504730781961	1	1	1
55	84	139583862445	160500643816367088	1	1	1
60	62	1548008755920	4052739537881	2	1	1
65	94	17167680177565	19740274219868223167	1	1	1
70	72	190392490709135	498454011879264	2	1	1
75	85	2111485077978050	259695496911122585	5	5	5

es decir, el máximo común divisor de dos números de FIBONACCI es otro número de FIBONACCI y además $MCD(F(n),F(m)) = F(MCD(n,m))$.

De lo anterior se deduce que $F(n)$ y $F(n+1)$ son primos relativos y que $F(k)$ divide exactamente a $F(n\cdot k)$

Para terminar, calculemos con el último algoritmo el término 10000 de la sucesión de FIBONACCI:

$fib(10000) =$ 33644764876431783266621612005107543310302148460680063906564769974680081
6815559551363373402558206533268083615937373479048386526826304089246305643188735454
4369559827491606602099884183933864652731300088830269235673613135117579297437854413
7521305205043477016022647583189065278908551543661595829872796829875106312005754
2878345321551510387081829896979161312785626503319548714021428753269818796204693609
7879900350962302291026368131493195275630227837628441540360584402572114334961180023
0912082870460889239623288354615057765832712525460935911282039252853934346209042
4524892940390170623388899108584106518317336043747073790855263176432573399371287193
7587746897479926305837065742830161637408969178426378624212835258112820516370298089
3320999057079200643674262023897831114700540749984592503606335609338838319233867830
5613643535189213327973290813373264265263398976392272340788292817795358057099369104
9175470808931841056146322338217465637321248226383092103297701648054726243842374862
4114530938122065649140327510866433945175121615265453613331113140424368548051067658
4349352383695965342807176877532834823434555736671973139274627362910821067928078471
8035329131176778924659089938635459327894523777674406192240337638674004021330343297
4969020283281459334188268176838930720036347956231171031012919531697946076327375892
5353077255237594378843450406771555577905645044301664011946258097221672975861502696
8443146952034614932291105970676243268515992834709891284706740862008587135016260312
0719031720860940812983215810772820763531866246112782455372085323653057759564300725
1774431505153960090516860322034916322264088524885243315805153484962243484829938090
5070483482449327453732624567755879089187190803662058009594743150052402532709746995
3187707243768259074199396322659841474981936092852239450397071654431

564213281576889080587831834049174345562705202235648464951961124602683139709750693
826487066132645076650746115126775227486215986425307112984411826226610571635150692
600298617049454250474913781151541399415506712562711971332527636319396069028956502
8268608362241082050562430701794976171121233066073310059947366875

con un tiempo de ejecución para calcularlo de 3.0517578125e-05 segundos.

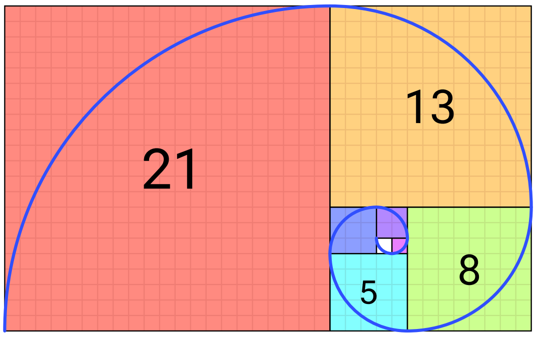

Figura 1: Espiral de Fibonacci

www.picasa.org