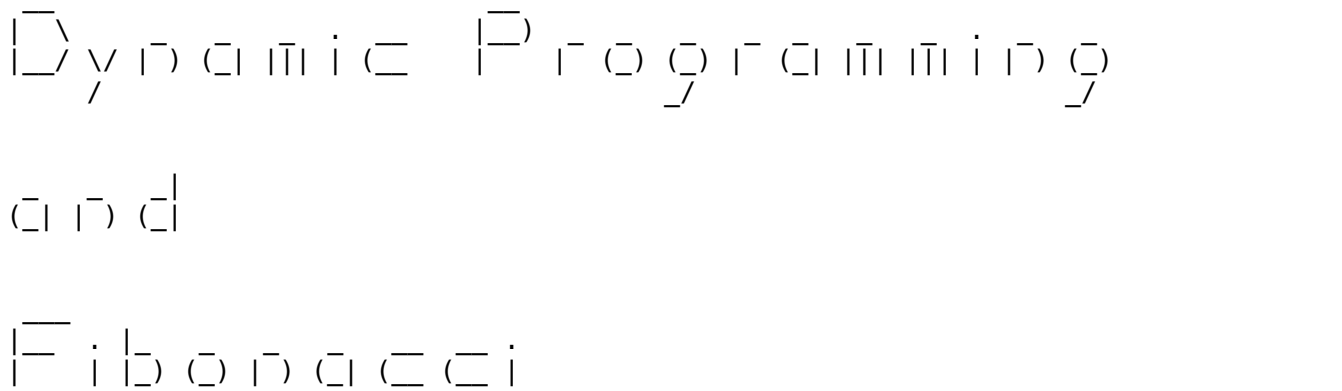
"Análisis de Algoritmos"
José Helo Guzmán



Contenido

▬▬▬▬ 1. Introducción.

En matemática, los números de Fibonacci son una secuencia numérica producida por la siguiente relación de recurrencia:

fibo(0) = 0

fibo(1) = 1

fibo(n) = fibo(n-1) + fibo(n-2)

Existe toda una historia sobre su descubrimiento, y cómo ha surgido esta secuencia en múltiples momentos gracias a los cálculos de diferentes científicos.

Esta fórmula se relaciona directamente con el “golden ratio” y la fórmula de Binet.

En este capítulo vamos a tratar de construir unos algoritmos para construir esta serie.

▬▬▬ 2. Secuencia de Fibonacci.

La secuencia de Fibonacci está definida como:

fibo(0) = 0

fibo(1) = 1

fibo(n) = f(n-1) + f(n-2)

Al desarrolla esta secuencia encontramos los siguientes valores:

fibo(0) = 0

fibo(1) = 1

fibo(2) = fibo(1) + fibo(0) = 0 + 1 = 1

fibo(3) = fibo(2) + fibo(1) = 1 + 1 = 2

fibo(4) = fibo(3) + fibo(2) = 2 + 1 = 3

fibo(5) = fibo(4) + fibo(3) = 3 + 2 = 5

fibo(6) = fibo(5) + fibo(4) = 5 + 3 = 8

Se construye entonces la siguiente sucesión de números:

fib(0) = 0

fib(1) = 1

fib(2) = 1

fib(3) = 2

fib(4) = 3

fib(5) = 5

fib(5) = 8

fib(6) = 13

fib(7) = 21

fib(9) = 34

fib(10)= 55

fib(11)= 89

fib(12)= 144

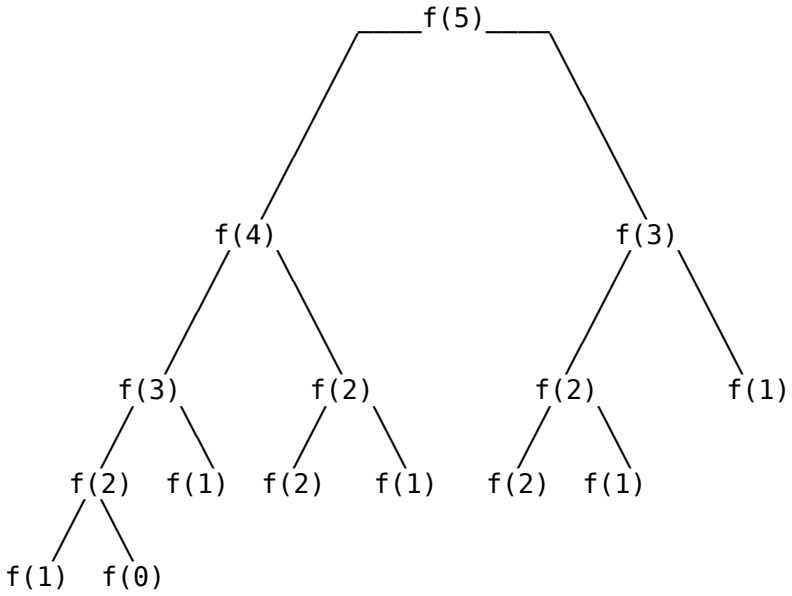
Deseamos construir un algoritmo que al recibir un valor de “n”, con n ≥ 0 nos devuelva el respectivo valor de fibo(n).

▬▬▬ 3. Recursión de Pila.

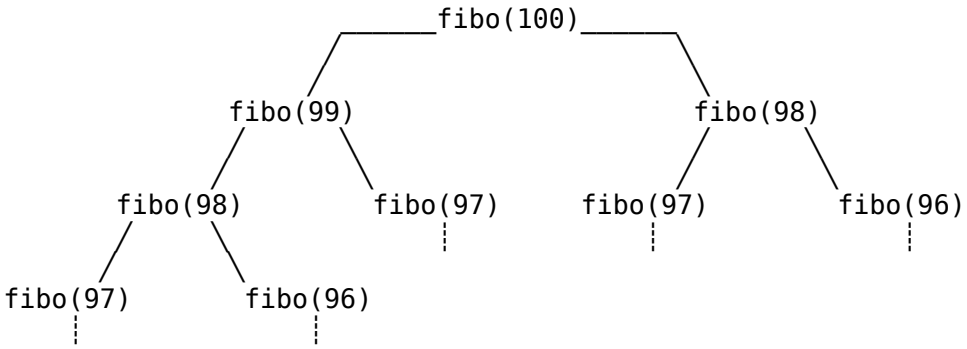
Tenemos el siguiente algoritmo que produce una recursión de pila.

```
fibo(n):=block
(
   if (n <= 1) then
      n
   else
      fibo(n-1) + fibo(n-2)
);
```

Veamos que ocurre con un problema sencillo:



Ocurre lo mismo con un problema de mayor tamaño:



▬▬▬ 4. Análisis del BigOh.

Analicemos la complejidad del código de recursión de pila. A este código usualmente se le conoce como “recursión naive”, o sea, recursión ingenua.

Observemos el número de operaciones:

```
fibo(n):=block
(        ▼
   if (n<=1) then
      n
   else     ▼   ▼       ▼
      fibo(n-1) + fibo(n-2)
);
```

Hagamos la fórmula exacta:

$T(0) = 1$

$T(1) = 1$

$T(n) = T(n-1) + T(n-2) + 4$

Esta relación de recurrencia no se puede resolver.
Por lo tanto para resolverla, se debe simplificar de la siguiente manera.

Supongamos que $T(n-2) \approx T(n-1)$

Para construir el peor caso, se utilizará la siguiente fórmula:

$T(0) = 1$

$T(n) = T(n-1) + T(n-1) + 4$

T(n) = 2*T(n-1) + 4

Al resolver la relación tenemos:

$T(n) = 5*2^n - 4$

Por lo tanto es $O(2^n)$

▬▬▬ 5. Utilizando Memoization.

Se puede utilizar una lista que vaya guardando cada uno de los resultados que se calculan. Por ejemplo:

```
fiboMem(n):=block
(  [memo],
   memo: [ [0,0],
           [1,1]
         ],
   fiboMemAux(n)
);
```

```
fiboMemAux(n):=block
(  [res],

   res: assoc(n,memo),

   if (res#false) then
      res
   else
      (
         res: fiboMemAux(n-1) + fiboMemAux(n-2),
         memo: endcons([n,res],memo),
         res
      )
);
```

Cuando se invoca:

```
(%i) fiboMem(5)
5
```

En la memoria se guardan los valores que se calcularon:

```
(%i) memo;
(%o) [[0,0],[1,1],[2,1],[3,2],[4,3],[5,5]]
```

El mecanismo de memorization es muy usado. Por esta razón el programa de Maxima tiene una forma de hacerlo automáticamente. Aquí está el programa:

```
fibo[n]:=block
(
   if (n<=1) then
      n
   else
      fibo[n-1] + fibo[n-2]

);


(%i) fibo[100];
(%o) 354224848179261915075
```

▬▬▬ 6. Análisis del Big Oh.

Este procedimiento, hace un intercambio de memoria por velocidad.

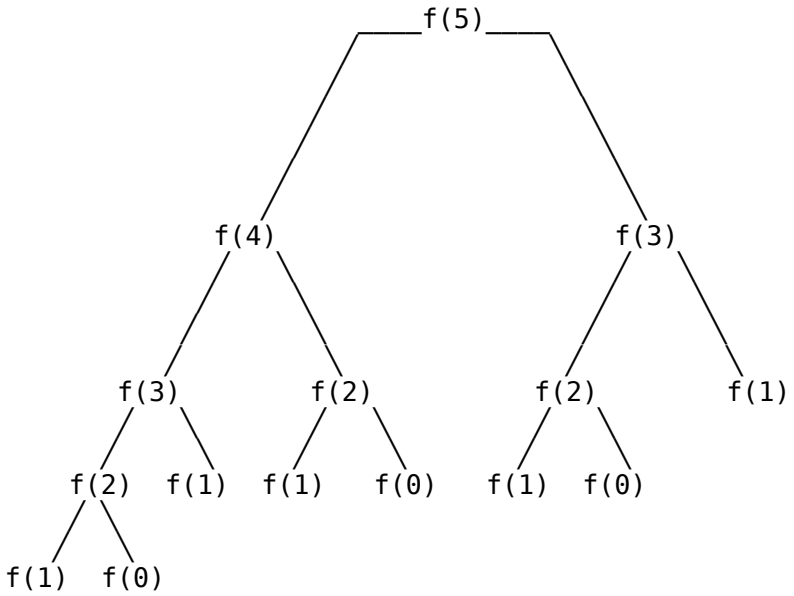
La complejidad del algoritmo es equivalente al tiempo de búsqueda en la memoria. Si suponemos que se hace una búsqueda lineal se tiene que su tiempo de ejecución es de:

O(n^2)

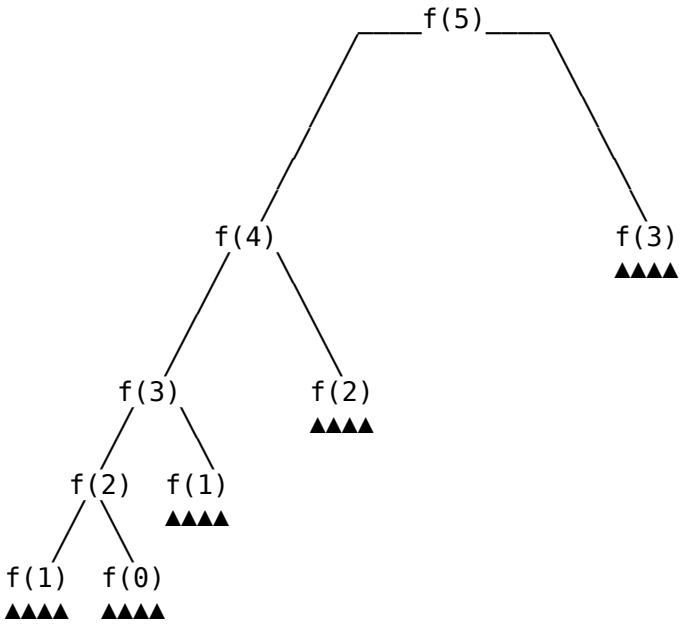
Pero como consecuencia, se necesita un vector de tamaño “n” para guardar cada uno de los resultados.

Este proceso se puede observar en la implementación recursiva del algoritmo.

Veamos un árbol con la recursión de pila para fib(5):



Ahora comparemos ese árbol con la memorización:



Podemos observar que para fibo(5), los únicos número que se calculan son:
fibo(2)
fibo(3)
fibo(4)
fibo(5)

No es necesario calcular fibo(n-2).

De esta manera la relación de recurrencia se establece como:

$T(0) = 1$

$T(n) = T(n-1) + T(n-2) + n$

$T(n) = T(n-1) + 0 + n$

$T(n) = T(n-1) + n$

Tenemos la relación de recurrencia dada por:

$T(0) = 1$

$T(n) = T(n-1) + n$

Al resolverla se obtiene:

$T(n) = n + 1$

Que es $O(n^2)$

▬▬▬ 7. Utilizando Dynamic Programming.

La programación dinámica es un técnica muy interesante para resolver problemas. Consiste en un conjunto de etapas que se deben resolver. Usualmente los problemas se resuelven iniciando en el lado opuesto de la pregunta. Por ejemplo, en este caso cuando se nos solicita el fibonacci de un número “n”. Debemos iniciar el cálculo desde el fibonacci más lejano. Es decir desde el fibonacci de “0”.

Es usual representar los cálculos de la programación dinámica mediante una tabla. Vemos los cálculos para el fibonacci de 10. La tabla se construye desde “0” hacia el número “10”.

Observemos la relación que se da en la tabla. Al sumar los elementos fibo(i-1) y fibo(i-2) de la fila “i”, se obtiene el valor de la fila siguiente. Además fibo(i-1) de la fila “i”, será el elemento fibo(i-2) de la siguiente fila.

i	fibo(i)	fibo(i-1)	fibo(i-2)
0	0	.	.
1	1	.	.
2	1	1	0
3	2	1	1
4	3	2	1
5	5	3	2
6	8	5	3
7	13	8	5
8	21	13	8
9	34	21	13
10	55	34	21

●● Recursión de Cola.

Veamos como debe funcionar el programa:

```
(%i) fibo(7)
     fiboAux(2, 7, 1, 0)
     fiboAux(3, 7, 1, 1)
     fiboAux(4, 7, 2, 1)
     fiboAux(5, 7, 3, 2)
     fiboAux(6, 7, 5, 3)
     fiboAux(7, 7, 8, 5)
     8+5
(%o) 13
```

Veamos este código en recursión de cola.

```
fibo(n):=block
(
   if (n<=1) then
      n
   else
      fiboAux(2,n,1,0)
```

```
);


fiboAux(ini,n,fm1,fm2):=block
(
   if (ini=n) then
      (fm1 + fm2)
   else
      fiboAux(
                ini + 1,
                n,
                fm1 + fm2,
                fm1
              )

);
```

●● En forma iterativa

También podemos programar este algoritmo de forma iterativa.

```
fibo(n):=block
(   [fm1,fm2,res,temp],
    if (n<=1) then
       n
    else
    (
       fm1:1,
       fm2:0,
       for i:2 thru n do
       (
           res: fm1 + fm2,
           temp:fm1,

           fm1: fm1 + fm2,
           fm2: temp
       ),
       res
     )
);
```

Observemos que sus respuestas son correctas, pues:

```
(%i) fibo(100)
354224848179261915075
```

Que es la respuesta correcta.

▬▬▬▬ 8. Análisis del BigOh.

Cuando programamos fibonacci con programación dinámica, podemos ver como se necesita un “for” de 2 hasta “n”. Por lo tanto su tiempo de ejecución es:

O(n)

▬▬▬ 9. Solución Matemática.

Podemos ver como este algoritmo genera la siguiente relación de recurrencia:

fibo(0) = 0

fibo(1) = 1

fibo(n) = fibo(n-1) + fibo(n-2)

●● Análisis de Complejidad.

Al resolver esta relación de recurrencia encontramos:

Que es básicamene una relación de recurrencia.

fibo(n) = fibo(n-1) + fibo(n-2)

fibo(n) - fibo(n-1) - fibo(n-2) = 0

de la forma:

x^2 - x - 1 = 0

Al resolver este polinomio se
tienen las raíces de:

$$x1 = \left(\frac{1 + \sqrt{5}}{2} \right)$$

$$x2 = \left(\frac{1 - \sqrt{5}}{2} \right)$$

Por lo que fibonacci se puede escribir como:

$$fibo(n) = A*\left(\frac{1 + \sqrt{5}}{2}\right)^n + B*\left(\frac{1 - \sqrt{5}}{2}\right)^n$$

Para encontrar los valores de A y B se utilizan las condiciones iniciales:

fibo(0) = 0

fibo(1) = 1

Por lo que:

$$fibo(0) = 0 = A*\left(\frac{1 + \sqrt{5}}{2}\right)^0 + B*\left(\frac{1 - \sqrt{5}}{2}\right)^0$$

$$fibo(1) = 1 = A*\left(\frac{1 + \sqrt{5}}{2}\right)^1 + B*\left(\frac{1 - \sqrt{5}}{2}\right)^1$$

Se obtiene el sistema de ecuaciones:

$0 = A + B$

$$1 = A*\left(\frac{1 + \sqrt{5}}{2}\right) + B*\left(\frac{1 - \sqrt{5}}{2}\right)$$

Al resolver el sistema de ecuaciones se obtiene:

$$A = \frac{1}{\sqrt{5}}$$

$$B = \frac{-1}{\sqrt{5}}$$

Por lo que la fórmula final se puede escribir como:

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

Que se puede expresar en punto flotante como:

$$fibo(n) = \left(\frac{1}{\sqrt{5}}\right)*\left(1.6180...\right)^n + \left(\frac{-1}{\sqrt{5}}\right)*\left(-0.6180...\right)^n$$

Lo que establece que su tiempo de ejecución está dado aproximadamente por:

$O(1.62^n)$

Que se puede programar como:

```
fibo(n):=block
(
   a: 1/sqrt(5),
   r1:((1+sqrt(5))/2)^n,
   b: -1/sqrt(5),
   r2:((1-sqrt(5))/2)^n,

   res:a*r1 + b*r2,
   printf(true,"Fibonacci: ~h", res)
);


> fibo(100)
354224848179261900000.0 -> Error!
354224848179261915075   -> Valor real
```

▬▬▬ 10. Tratando de Mejorar la Fórmula.

Analicemos más de cerca esta fórmula:

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

$$\text{Sea } G(n) = \left(\frac{1}{\sqrt{5}}\right)*\left(\frac{1+\sqrt{5}}{2}\right)^n$$

$$\text{Sea } E(n) = \left(\frac{-1}{\sqrt{5}}\right)*\left(\frac{1-\sqrt{5}}{2}\right)^n$$

$fibo(n) = G(n) + E(n)$

Observe como G(n) es la fuerza principal del resultado:

n	G(n)	E(n)	fibo(n)
1	0.723607	0.276393	1
2	1.17082	-0.17082	1
3	1.894427190	0.105572809	2
4	3.065247584	-0.65247584	3
5	4.95967477	0.040325224	5
6	8.02492225	-0.024922359	8
7	12.9845971	0.015402865	13

Observe que

fiboG(n) = round(G(n))

Por lo tanto se podría escribir el siguiente programa:

```
fiboG(n):=block
(
   a: 1/sqrt(5),
   b: ((1 + sqrt(5))/2)^n,
   round(float(a*b))
);


(%i) fiboG(10);
(%o) 55

(%i) fiboG(100);
(%o) 354224848179263111168
```

El valor correcto es:

```
(%i) fibo(100);
(%o) 354224848179261915075
```

Como se tiene este error numérico, se tienen otras
formas equivalentes de presentar la fórmula anterior.

▬▬▬ 11. La Fórmula de Binet.

Usualmente la relación de recurrencia anterior suele escribirse como la fórmula de Binet, que se expresa de la siguiente manera.

Sea:

$$\varphi = \left(\frac{1 + \sqrt{5}}{2}\right) \approx 1.6180339887...$$

Podemos escribir la solución de Fibonacci como:

$$fibo(n) = \left(\frac{1}{\sqrt{5}}\right)*\left(1.6180...\right)^{n} + \left(\frac{-1}{\sqrt{5}}\right)*\left(-0.6180...\right)^{n}$$

$$fibo(n) = \left(\frac{1}{\sqrt{5}}\right)*\left(\varphi\right)^{n} - \left(\frac{1}{\sqrt{5}}\right)*\left(1 - \varphi\right)^{n}$$

$$fibo(n) = \left(\frac{1}{\sqrt{5}}\right)*\left(\varphi^{n} - (1 - \varphi)^{n}\right)$$

$$fibo(n) = \left(\frac{\varphi^{n} - (1 - \varphi)^{n}}{\sqrt{5}}\right)$$

Adicionalmente se puede probar que:

$2*\varphi - 1 = \sqrt{5}$

Con lo que se escribe la fórmula como:

$$fibo(n) = \left(\frac{\varphi^{n} - (1 - \varphi)^{n}}{2*\varphi - 1}\right)$$

En ese caso, si se tuviera un cálculo para “φ”, el tiempo de ejecución de fibonacci sería de O(1)!!!!

Esta fórmula de Binet, se puede programar como:

```
fibo(n):=block
(
   (%phi^n-(1-%phi)^n)
    /
   (2*%phi-1)
);
```

Para ver el resultado como un punto flotante hacemos:

```
fibo(n):=block
(
   float( (%phi^n-(1-%phi)^n)
           /
          (2*%phi-1))
);
```

Sin embargo, en este caso se ha convertido un problema entero, a un problema con punto flotante, lo que produce errores de cálculo. Por ejemplo

```
(%i) fibo(5)
(%o) 5.000000000000001


(%i) fibo(10);
(%o) 55.00000000000001
```

Para evitar esto hacemos:

```
fibo(n):=block
(
   round(float( (%phi^n-(1-%phi)^n)
                /
                (2*%phi-1)))
);
```

Con lo que obtenemos:

```
(%i) fibo(5);
(%o) 5


(%i) fibo(10);
(%o) 55
```

Pero tenemos un nuevo problema:

```
(%i) fibo(100);
(%o) 354224848179263111168
```

Y este resultado no es correcto, pues fibonacci de 100 es:
354224848179261915075

Y nuestro algoritmo fue:
354224848179263100000

Existen muchos otros algoritmos que utilizan la fórmula de Binet y tratan de arreglar el error que produce.

De igual manera, hay muchos más algoritmos que tratan de solucionar el problema de fibonacci, de hecho hay cursos completos al respecto.

▬▬▬ 12. Resumen.

• Fibonacci se puede resolver utilizando programación recursiva “naive”.

• Se pueden utilzar también técnicas de “memoization”.

• Se pueden utilzar programación dinámica, esta va de 1 hacia “n” y es similar en su comportamiento al proceso de memoization.

• Al utilizar la fórmula de Binet, se presentan problemas pues el punto flotante produce pequeños errores que se van acumulando.

▬▬▬ 13. Ejercicios.

●● Ejercicio 1.

Por lo que la fórmula final, e indique hasta qué número da resultados correctos.

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

●● Ejercicio 2.

Analicemos más de cerca esta fórmula:

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

$$\text{Sea } G(n) = \left(\frac{1}{\sqrt{5}}\right)*\left(\frac{1+\sqrt{5}}{2}\right)^n$$

$$\text{Sea } E(n) = \left(\frac{-1}{\sqrt{5}}\right)*\left(\frac{1-\sqrt{5}}{2}\right)^n$$

$$\text{fibo}(n) = G(n) + E(n)$$

Observe como G(n) es el valor del resultado:

n	G(n)	E(n)	fibo(n)

```
1    0.723607       0.276393      1
2    1.17082       -0.17082       1
3    1.894427190    0.105572809   2
4    3.065247584   -0.65247584    3
5    4.95967477     0.040325224   5
6    8.02492225    -0.024922359   8
7   12.9845971      0.015402865   13
```

Observe que

fibo(n) = round(G(n))

Intente programar este resultado.

●● Ejercicio 3.

Resuelva la relación de recurrencia de fibonacci utilizando wxmaxima

```
load("solve_rec");

rec: f[n] = f[n-1] + f[n-2];

solve_rec(rec, f[n], f[0]=1, f[1]=1);
```

●● Ejercicio 4.

Visite este sitio, que es un publicación dedicada únicamente a la sucesión de Fibonacci. En ella puede encontrar nuevos algoritmos y el estado actual de la investigación en esta área.

https://www.fq.math.ca/