Universidad Autónoma de Nuevo León

FCFM

Matemáticas computacionales

Ana Cecilia García Esquivel Matrícula: 1749760

Fecha: 13/10/2017

Resumen

En el siguiente reporte se muestran algunos códigos hechos en clase: Primos, Fibonacci, y Fibonacci recursivo con memoria. Estos últimos 3 son basados en la serie de Fibonacci sólo que hay una diferencia entre cada uno de ellos, lo cual veremos a continuación.

Primos

Este es un programa que elaboramos con la intención de decir si el número ingresado era primo o no lo era. Realmente fue muy simple entenderlo y programarlo, la única dificultad que tuve fue en identificar el rango que usaríamos para saber hasta dónde revisaríamos el comportamiento de los divisores en cada número.

En seguida se muestra el programa realizado:

```
def primo(n):
        cnt=0
        for i in range(2, round(n**0.5)):
                cnt=cnt+1
                if ((n % 1)== 0):
                        return (“no es primo”)
        return  (“es primo”)
```

Fibonacci

La serie de Fibonacci es una serie muy conocida dentro del área de las matemáticas la cual fue creada o descubierta por el italiano Leonardo de Pisa, también conocido como Fibonacci. Esta serie consta en sumar los dos primero términos que están antes del término en cuestión, es decir:

xn = xn-1 + xn-2

donde: xn es el término en posición "n, xn-1 es el término anterior (n-1), xn-2 es el anterior a ese

(n-2). Así:

0,1,(1+0)=1, (1+1)=2, (1+2)=3, (2+3)=5,…

Entonces eso es lo que utilizamos para hacer el siguiente programa, el cual realiza el cálculo de cualquier posición 'n' que quisiésemos encontrar.

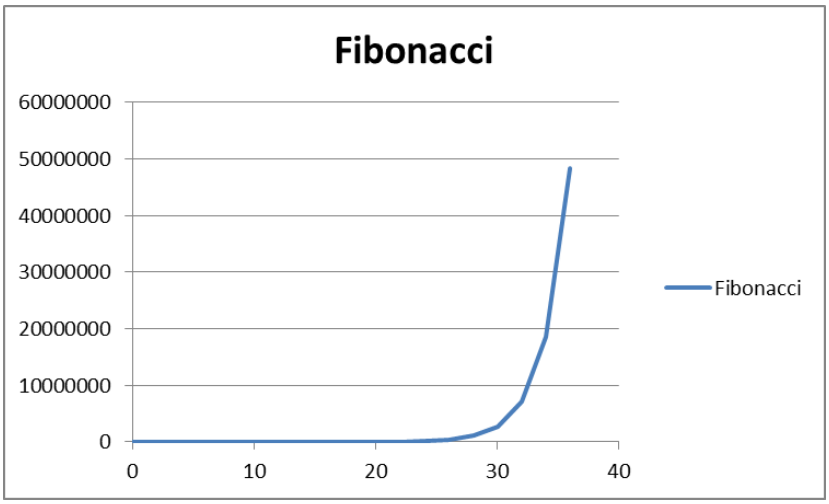
Primero tuvimos que observar que para el '0' y'1' su valor en la serie es de 1 por eso si se ingresan esos dos valores lo que regresará la función es 1, sin embargo para los demás números naturales que se ingresen se regresará el valor en la serie de n que es igual a [(n-2)+(n-1)].

En seguida se muestra el programa:

```
>>> contador=0
>>> def fibonacci(n):
        global contador
        contador+=1
        if n==0 or n==1:
                return(1)
        else:
                return fibonacci(n-2) + fibonacci(n-1)
```

Ahora, se mostrará una gráfica correspondiente al número de operaciones que realiza el programa dependiendo de la posición que tomemos.

Cabe mencionar que yo sólo tomé los primero 40 valores de n para poder graficarlo, ya que después de 40 el programa comienza a hacerse más lento y más pesado.



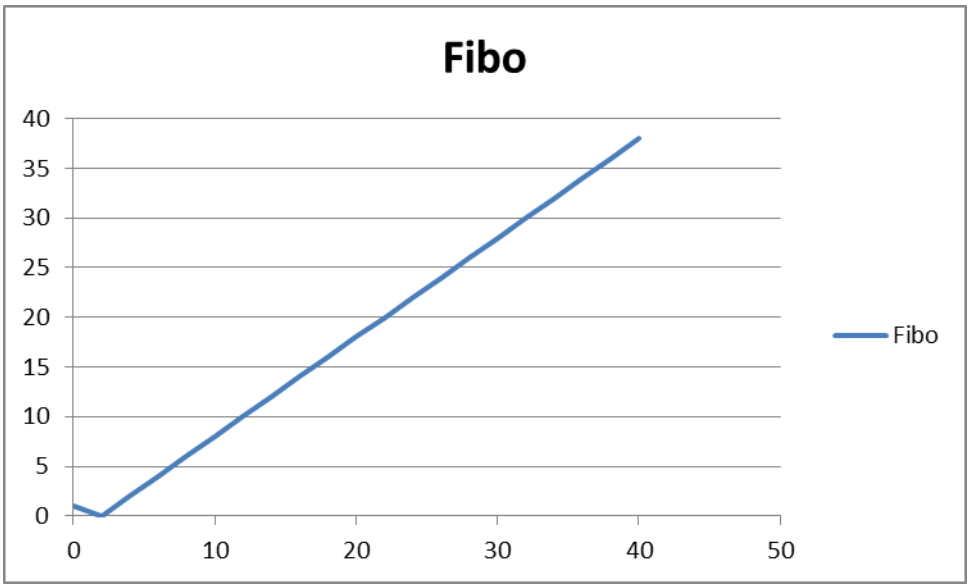
“Fibo”(Fibonacci iterativo)

Este programa realmente es una variante del anterior, ya que cumple el mismo objetivo de encontrar el valor en la serie de Fibonacci para un determinado valor de ‘n’. La diferencia con el anterior es que en este utilizamos un ciclo for para obtener los valores dela serie a partir del 2y al final, el código nos regresaría el valor en la serie (r) y el número de operaciones que se hicieron para n (cnt).

Aquí la muestra del código elaborado:

```
def fibo(n):
        cnt=0
        if n==0 or n==1:
                return(1)
        r,r1,r2=0,1,2
        for i in range (2,n):
                cnt+=1
                r=r1 + r2
                r2=r1
                r1=r
        return r, cnt
```

El número de operaciones de este código es muchísimo menor con respecto al código anterior, como lo podemos apreciar en la siguiente gráfica, que luego se puede comparar con la anterior.



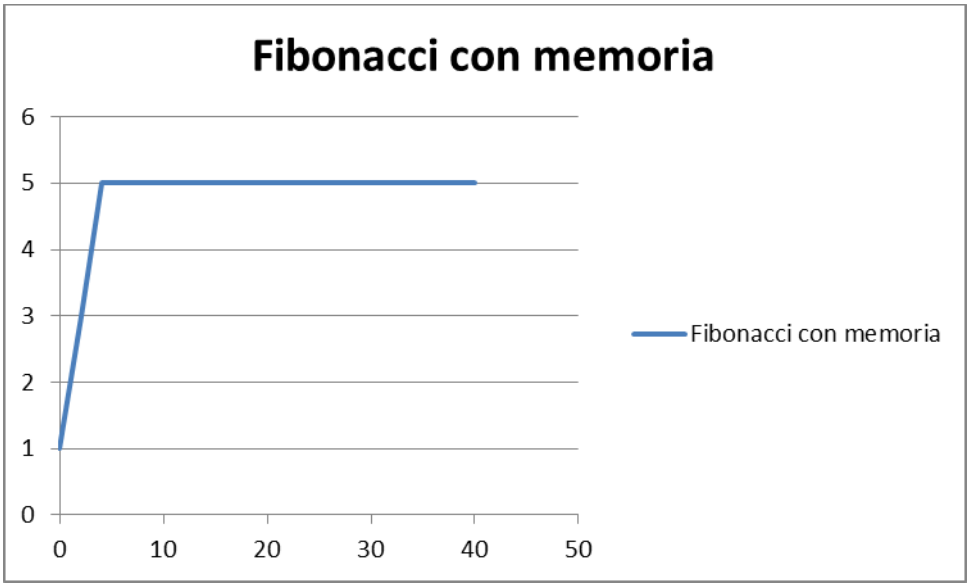
Fibonacci con memoria

Este último código también es una variante de *"Fibonacci()"* visto anteriormente, sin embargo este programa tiene un arreglo llamado memo{} (por memoria) y que además es global, en memoria están los elementos de la serie, así, por ejemplo, si ingresas el valor de Fibonacci(3), te dirá no sólo los de n=3 sino los demás que estén antes de él. Este programa es muchísimo más simple en cuanto al número de operaciones se refiere, y lo podremos apreciar en la última gráfica contenida en este reporte.

Aquí el código de Fibonacci con memoria.

```
>>> memo={}
>>> cnt=0
>>> def fibonacci(n):
        global memo,cnt
        cnt+=1
        if n == 0 or n == 1:
                return (1)
        if n in memo:
                return memo[n]
        else:
                val=fibonacci(n-2)+fibonacci(n-1)
```

Aquí, la gráfica de l número de operaciones hechas por el último programa.



Conclusiones.

Para mí estos códigos no fueron difíciles de entender, y creo que me han ayudado mucho a comprender mejor los ciclos for para python, y otros elementos. Fueron códigos muy sencillos pero creo que me serán de gran utilidad para trabajos futuros.