Reporte de test de Primalidad y sucesión de Fibonacci

Haydee Judith Arriaga Ponce

Matricula: 1659539

l.judithmat@gmail.com

https://github.com/haydee10arriaga/1659539_Mat.Com

14 de octubre de 2017

En este reporte se presenta como implementamos nuestros conocimientos sobre los números y primos y la sucesión de Fibonacci en códigos realizados en Python donde comprendimos su funcionamiento mediante la visualización de las gráficas que realizamos con los datos obtenidos.

1. Test de Primalidad

   Un número es primo cuando es entero positivo, distinto de 0 y 1 y que únicamente se puede dividir por sí mismo y por 1 para dar una solución exacta (por tanto, para todos los otros números por los que intentemos dividir el número primo no dará solución exacta).
   Existen muchos procedimientos y algoritmos para probar si un número natural dado n es primo o no.

   Por cual nos fue posible realizar un código para saber cuántas operaciones le lleva saber si el número ingresado es primo o no a partir de un algoritmo donde comprobamos si el número dado no es dividido por los números entre 2 y uno número antes del otorgado, es decir como el maestro lo explico en la clase que si existiera m donde m es menor igual a la raíz cuadrada de n, por lo que el número es primo si m no divide a n; no es primo si n es menor o igual a 1, se inicializa en 2. Si m llega a ser menor o igual a la raíz de n se tiene que verificar si m divide a n, seria primo n cuando m es mayor a la raíz de n.

Código

```
1  def primo(n):
2      cnt=0
3      for i in 2,(n**0.5):
4          cnt=cnt+1
5          if ((n%i)==0):
6              #return("no es primo")
7              break
8          #return("si es primo")
9          return cnt
10
11
12 def primo(n):
13     cnt=0
14     for i in range(2,round(n**0.5)):
15         cnt=cnt+1
16         if ((n%i)==0):
17             break
18     return cnt
19
20 for w in range(1,10011,50):
21     print(w,primo(w))
22
```

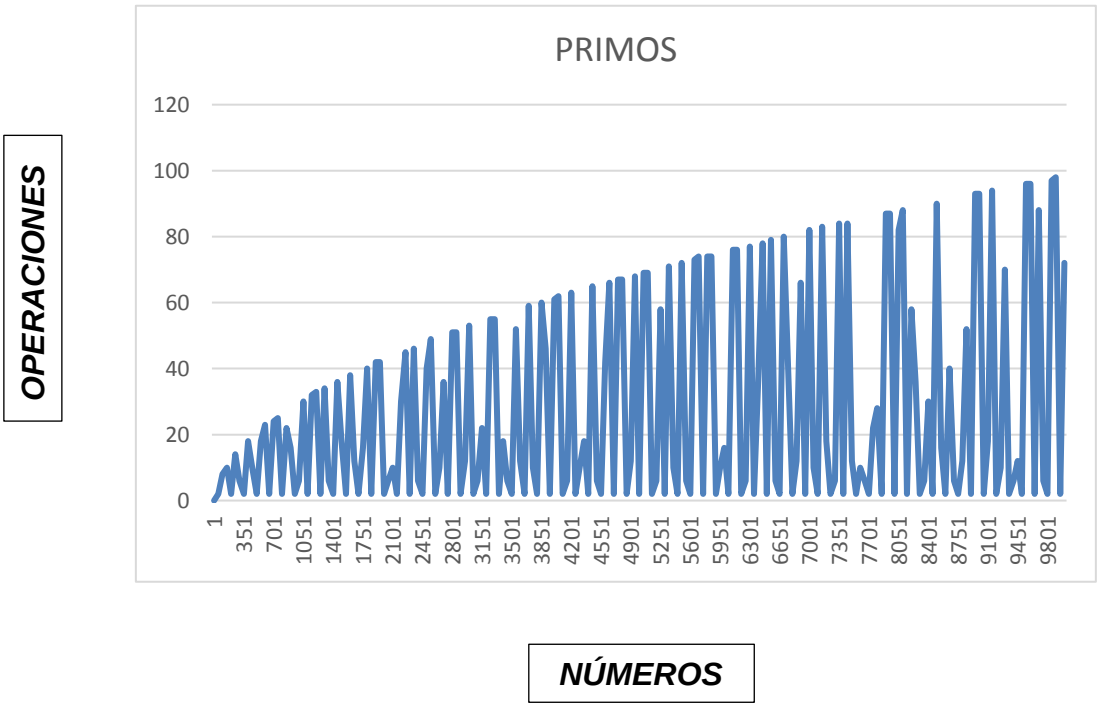
Column1	Column2
1	0
51	2
101	8
151	10
201	2
251	14
301	6
351	2
401	18
451	10
501	2
551	18
601	23
651	2
701	24
751	25
801	2
851	22
901	16
951	2
1001	6
1051	30
1101	2
1151	32
1201	33
1251	2
1301	34
1351	6
1401	2
1451	36
1501	18
1551	2
1601	38
1651	12
1701	2
1751	16
1801	40
1851	2
1901	42
1951	42
2001	2
2051	6
2101	10
2151	2
2201	30
2251	45
2301	2
2351	46
2401	6
2451	2
2501	40
2551	49
2601	2
2651	10
2701	36
2751	2
2801	51
2851	51
2901	2
2951	12
3001	53
3051	2
3101	6
3151	22
3201	2
3251	55
3301	55
3351	2
3401	18
3451	6
3501	2
3551	52
3601	12
3651	2
3701	59
3751	10
3801	2
3851	60
3901	46
3951	2
4001	61
4051	62
4101	2
4151	6
4201	63
4251	2
4301	10
4351	18
4401	2
4451	65
4501	6
4551	2
4601	42
4651	66
4701	2
4751	67
4801	67
4851	2
4901	12
4951	68
5001	2
5051	69
5101	69
5151	2
5201	6
5251	58
5301	2
5351	71
5401	10
5451	2
5501	72
5551	6
5601	2
5651	73
5701	74
5751	2
5801	74
5851	74
5901	2
5951	10
6001	16
6051	2
6101	76
6151	76
6201	2
6251	6
6301	77
6351	2
6401	36
6451	78
6501	2
6551	79
6601	6
6651	2

6701	80
6751	42
6801	2
6851	12
6901	66
6951	2
7001	82
7051	10
7101	2
7151	83
7201	18
7251	2
7301	6
7351	84
7401	2
7451	84
7501	12
7551	2
7601	10
7651	6
7701	2
7751	22
7801	28

7851	2
7901	87
7951	87
8001	2
8051	82
8101	88
8151	2
8201	58
8251	36
8301	2
8351	6
8401	30
8451	2
8501	90
8551	16
8601	2
8651	40
8701	6
8751	2
8801	12
8851	52
8901	2
8951	93

9001	93
9051	2
9101	18
9151	94
9201	2
9251	10
9301	70
9351	2
9401	6
9451	12
9501	2
9551	96
9601	96
9651	2
9701	88
9751	6
9801	2
9851	97
9901	98
9951	2
10001	72

GRÁFICA



2. *LA SUCESIÓN DE FIBONACCI*

De las tantas sucesiones matemáticas que existen, ninguna es tan famosa, tan interesante y tan asombrosa como la que inventó Fibonacci también se lo conocía como Leonardo de Pisa. A lo largo de los años, hombres de ciencia, artistas de todo tipo y arquitectos, la han utilizado para trabajar.

La sucesión de Fibonacci, en ocasiones también conocida como secuencia de Fibonacci o incorrectamente como serie de Fibonacci, es en sí una sucesión matemática infinita. Consta de una serie de números naturales que se suman de a 2, a partir de 0 y 1. Básicamente, la sucesión de Fibonacci se realiza sumando siempre los últimos 2 números (Todos los números presentes en la sucesión se llaman números de Fibonacci) de la siguiente manera:

0,1,1,2,3,5,8,13,21,34...

Fácil, ¿no? (0+1=1 / 1+1=2 / 1+2=3 / 2+3=5 / 3+5=8 / 5+8=13 / 8+13=21 / 13+21=34...) Así sucesivamente, hasta el infinito. Por regla, la sucesión de Fibonacci se escribe así: $x_n = x_{n-1} + x_{n-2}$.

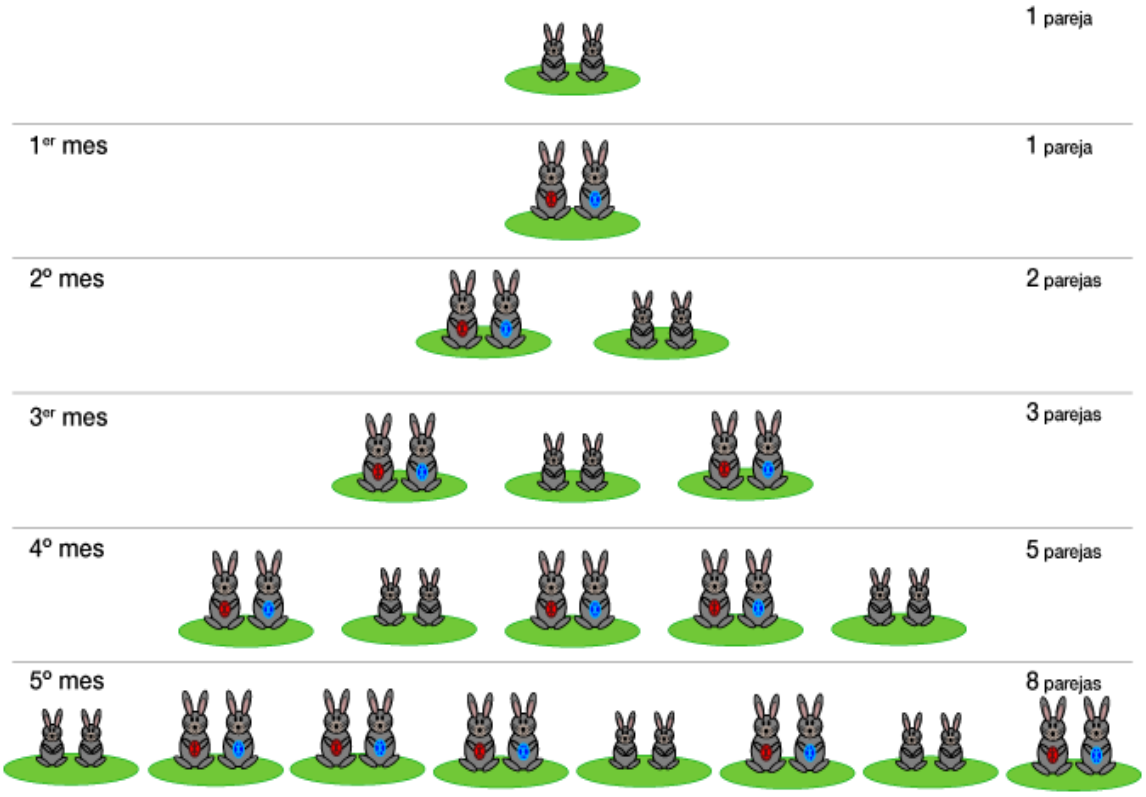
Ahora, ¿qué es lo asombroso de esta secuencia o sucesión matemática tan simple y clara? Que está presente prácticamente en todas las cosas del universo, tiene toda clase de aplicaciones en matemáticas, computación y juegos, y que aparece en los más diversos elementos biológicos.

Ejemplos claros son la disposición de las ramas de los árboles, las semillas de las flores, las hojas de un tallo, otros más complejos y aún mucho más sorprendentes es que también se cumple en los huracanes e incluso hasta en las galaxias enteras, desde donde obtenemos la idea del espiral de Fibonacci.

Todo comenzó con un problema de cría de conejos. Era el siguiente:

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.

Así vemos que:



Un espiral de Fibonacci es una serie de cuartos de círculo conectados que se pueden dibujar dentro de una serie de cuadros regulados por números de Fibonacci para todas las dimensiones.

Y hay una sorpresa. Si tomas dos números de Fibonacci consecutivos (uno detrás del otro), su cociente está muy cerca de la razón aúrea "φ" que tiene el valor aproximado 1.618034...

De hecho, cuanto más grandes los números de Fibonacci, más cerca está la aproximación. Y es más sorprendente todavía esta fórmula para calcular cualquier número de Fibonacci usando la razón de oro:

$$x_n = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}}$$

Increíblemente el valor siempre es un número entero, exactamente igual a la suma de los dos términos anteriores.

La finalidad es calcular números de Fibonacci para lo cual definimos una función que dado n nos calcule el n-ésimo numero de Fibonacci primero utilizamos la forma recursiva y la función fue la siguiente:

```
"""
Created on Sat Oct 14 10:36:49 2017

@author: Angies computer
"""

# Evaluacion 2 "fibonacci"
ley = {0: 0, 1: 1} #Declaracion de los primeros elementos
def fib(x):
    if x not in ley: #Proceso
        ley[x] = fib(x- 1) + fib(x - 2)
    return ley[x]
for w in range(1,51): #Rango a valorar
    print(w,fib(w)) #Imprime valor de ( x) y su posicion
```

Lo único que note que es malo de este código recursivo donde se declaró un arreglo de (1,51) es que se tarda demasiado lo visualice mediante la tabla y la gráfica que realice siguiente:

Column1	Column2
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
30	832040
31	1346269
32	2178309
33	3524578
34	5702887
35	9227465
36	14930352
37	24157817
38	39088169
39	63245986
40	102334155
41	165580141
42	267914296
43	433494437
44	701408733
45	1134903170
46	1836311903
47	2971215073
48	4807526976
49	7778742049
50	12586269025

GRÁFICA Su complejidad es de O(Fn).

Al término de este código realizamos el iterativo es como una optimización hacia el ***ITERATIVO*** lo que se realiza es que solo se calcula solo una vez cada termino en él se generó un arreglo de longitud n+1 y se inicia con los primeros dos términos bases y a partir de ellos se generan los demás y al realizar la tabla y la gráfica se visualiza: su complejidad es de O(n).

Column1	Column2
0	1
1	1
2	2
3	3
4	5
5	8
6	13
7	21
8	34
9	55
10	89
11	144
12	233
13	377
14	610
15	987
16	1597
17	2584
18	4181
19	6765
20	10946
21	17711
22	28657
23	46368
24	75025
25	121393
26	196418
27	317811
28	514229
29	832040
30	1346269
31	2178309
32	3524578
33	5702887
34	9227465
35	14930352
36	24157817
37	39088169
38	63245986
39	102334155
40	165580141
41	267914296
42	433494437
43	701408733
44	1134903170
45	1836311903
46	2971215073
47	4807526976
48	7778742049
49	12586269025
50	20365011074
51	32951280099
52	53316291173
53	86267571272
54	1.39584E+11
55	2.25851E+11
56	3.65435E+11
57	5.91287E+11
58	9.56722E+11
59	1.54801E+12
60	2.50473E+12
61	4.05274E+12
62	6.55747E+12
63	1.06102E+13
64	1.71677E+13
65	2.77779E+13
66	4.49456E+13
67	7.27235E+13
68	1.17669E+14
69	1.90392E+14
70	3.08062E+14
71	4.98454E+14
72	8.06516E+14
73	1.30497E+15
74	2.11149E+15
75	3.41645E+15
76	5.52794E+15
77	8.94439E+15
78	1.44723E+16
79	2.34167E+16
80	3.78891E+16
81	6.13058E+16
82	9.91949E+16

83	1.60501E+17
84	2.59695E+17
85	4.20196E+17
86	6.79892E+17
87	1.10009E+18
88	1.77998E+18
89	2.88007E+18
90	4.66005E+18
91	7.54011E+18

92	1.22002E+19
93	1.97403E+19
94	3.19404E+19
95	5.16807E+19
96	8.36211E+19
97	1.35302E+20
98	2.18923E+20
99	3.54225E+20
100	5.73148E+20

Al final realizamos el recursivo con memoria donde utilizamos una biblioteca y es demasiado evidente que la cantidad de operaciones es menor pero aun así si tiene desventajas con otros algoritmos. A pesar de que también es recursivo como el primero la diferencia es que este no vuelve a calcular los valores que ya se han calculado.

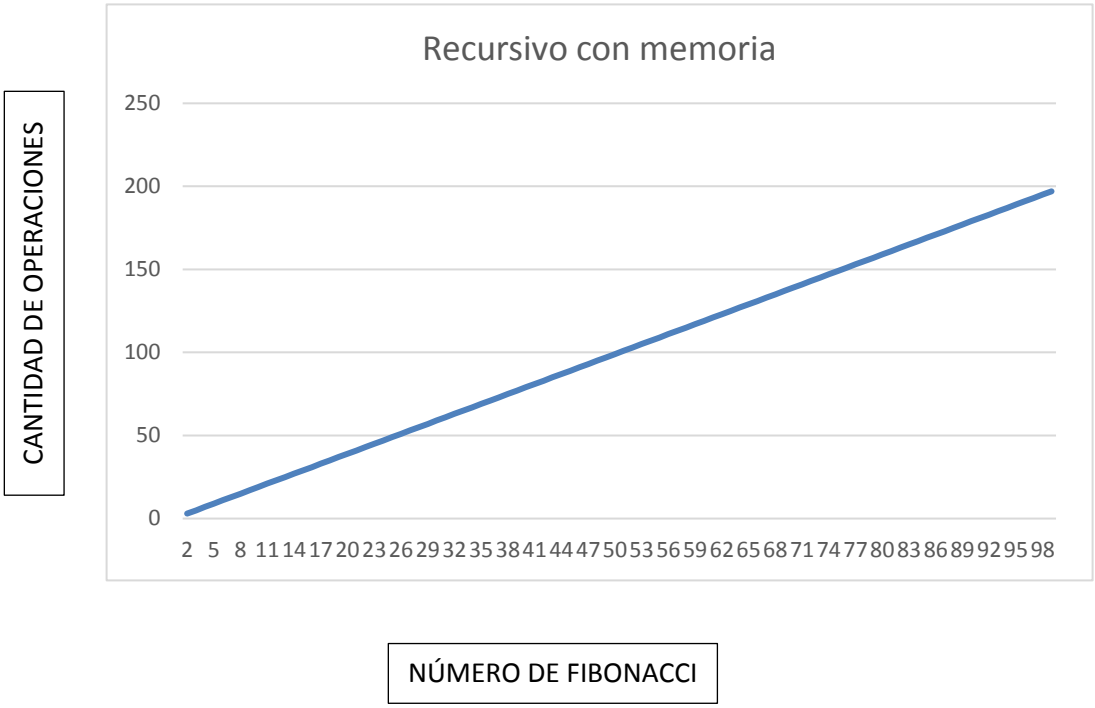
1	**1**
2	3
3	5
4	7
5	9
6	11
7	13
8	15
9	17
10	19
11	21
12	23
13	25
14	27
15	29
16	31
17	33
18	35
19	37
20	39
21	41
22	43
23	45
24	47
25	49

26	51
27	53
28	55
29	57
30	59
31	61
32	63
33	65
34	67
35	69
36	71
37	73
38	75
39	77
40	79
41	81
42	83
43	85
44	87
45	89
46	91
47	93
48	95
49	97
50	99

51	101
52	103
53	105
54	107
55	109
56	111
57	113
58	115
59	117
60	119
61	121
62	123
63	125
64	127
65	129
66	131
67	133
68	135
69	137
70	139
71	141
72	143
73	145
74	147
75	149

76	151
77	153
78	155
79	157
80	159
81	161
82	163
83	165
84	167
85	169
86	171
87	173
88	175
89	177
90	179
91	181
92	183
93	185
94	187
95	189
96	191
97	193
98	195
99	197

GRÁFICA su complejidad es de O(n)



CONCLUSIÓN

Al principio con el algoritmo de los primos yo esperaba obtener como resultado saber si el número ingresado es o no primo pero después de la explicación del maestro comprendí la finalidad de esta actividad de esta evaluación que era saber la cantidad de operaciones que se realizan para determinar si es o no primo y visualizarlo en una gráfica y al final comparar las gráficas los algoritmos después proseguimos con la sucesión de Fibonacci con la cual trabajamos en tres partes para ver los beneficios de cada algoritmo como entre menor operaciones es mejor de cierto modo el algoritmo lo que caracteriza más a cada uno es que el de recursividad es de manera más intuitiva se obtuvo casi de manera más directa a partir de la sucesión de Fibonacci que ya sabemos en cambio en el otro el maestro nos hizo observar que al almacenar los números en un arreglo es de mucha ayuda ya que es para no volver repetir el cálculo en lo personal me pareció de mayor beneficio el recursivo con memoria ya que cualquier persona puede ponerlo en práctica ya sea por rapidez, eficiencia, etc en lo que llevamos del curso este esta evaluación este reporte es el más complicado ya que tuve muchos contratiempos debido a las confusiones con los tres algoritmos de Fibonacci pero gracias al apoyo del maestro el día viernes y sábado lo pude concluir y adquiriendo un gran conocimiento.