Reporte de test de primalidad y sucesión de Fibonacci

Yadira Yaneth Marroquín de la Peña

Matricula: 1701504

yaz_marroquin@outlook.com

https://github.com/yadira02/1701504MatComp

13 de octubre del 2017

En el siguiente documento comentaremos sobre el análisis de los números primos, en el área de las matemáticas se basa sobre la teoría de números para así poder saber si el número es primo o no. En el documento también hablaremos acerca del programa de Fibonacci donde se explica como dentro de un programa se puede saber mucho más fácil para calcular un número en la serie de Fibonacci dentro del programa de Python.

1. Números Primos

Principalmente un número primo debe ser mayor que cero, es decir que tiene exactamente dos divisores positivos. O bien aquel número entero positivo que no puede expresarse como producto de dos números enteros positivos más pequeños que él.

Se presentará un algoritmo para observar si un número es primo o no, a este tipo de algoritmo se le llama test de primalidad, para mí fue un programa muy sencillo donde se puede observar mucho más fácil si el número es primo. Por definición de los números primos decimos que es si se dividen entre 1 y el mismo es primo.

Pseudocodigo de los números primos

```
Proceso Primos

    Escribir "Ingrese la cantidad de numeros primos a mostrar:"
    Leer cant_a_mostrar

    Escribir "1: 2" // el primer primo es 2, los otros son todos impares...
    cant_mostrados <- 1
    n<-3            // ...a partir de 3

    Mientras cant_mostrados<cant_a_mostrar Hacer

        es_primo <- Verdadero // pienso que es primo hasta que encuentre con que dividirlo

        Para i<-3 hasta rc(n) con paso 2 Hacer // ya sabemos que es impar
            Si n MOD i = 0 entonces // si la division da exacta...
                es_primo <- Falso  // ...ya no es primo
            FinSi
        FinPara

        Si es_primo Entonces
            cant_mostrados <- cant_mostrados + 1
            Escribir cant_mostrados, ": ",n
        FinSi

        n <- n + 2
```

Para comprobar si n es primo o no: debemos primero que nada saber si n es menor o igual 1, entonces no es primo. Mientras x sea menor a √ n entonces probar x divide a n.

Comenzamos primero que nada con declarar las variables contardor global para poder iterar dentro un ciclo, dentro de la raiz del numero determinaremos si es primo. Continuación mostraremos el código de números primos en Python.

```
def primo(n):
        cnt=0
        for i in 2,(n**0.5):
                cnt=cnt+1
                if ((n%i)==0):
                        #return("no es primo")
                        break
                #return("si es primo")
                return cnt


def primo(n):
        cnt=0
        for i in range(2,round(n**0.5)):
                cnt=cnt+1
                if ((n%i)==0):
                        break
        return cnt

for w in range(1,10011,50):
    print(w,primo(w))
```

En el código muestra las operaciones que se realiza para poder saber si primo o no, además cada 50 elementos toma hasta llegar al 10000 para poder ver mejor o apreciar la gráfica.

A continuación se muestra los valores que se tomara para graficas:

Elemento	Operaciones
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

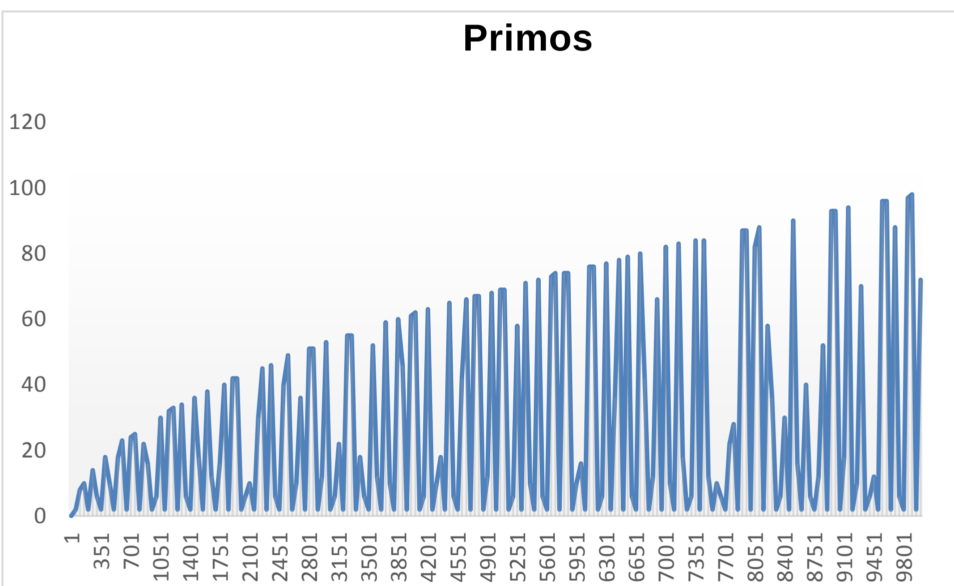
7751	22
7801	28
7851	2
7901	87
7951	87
8001	2
8051	82
8101	88
8151	2
8201	58
8251	36
8301	2

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

Grafica



2. Fibonacci

La secuencia de Fibonacci es 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. En el año 1202, Fibonacci publicó un libro titulado Liber Abaci, en el que incluyó varios problemas y métodos algebraicos. La conocida espiral, denominada "sucesión de Fibonacci" aparece constantemente en la naturaleza.

La sucesión de esta serie, se inicia con 0 y 1 y a partir de ahí cada elemento es la suma de los dos anteriores. A cada elemento que forma esta sucesión se le denomina número de Fibonacci.

La secuencia funciona con una lógica acumulativa, en la cual cada número de la secuencia es la suma de los dos anteriores dentro de la misma secuencia. Cada número representa una unidad de apuesta. Puesto en números, este sería un ejemplo de secuencia Fibonacci:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 ..

La serie empieza con los números 1 y 1, y después los siguientes números que van apareciendo en la sucesión son el resultado de la suma de los dos anteriores. Cómo dato de interés, el sistema Fibonacci surgió en Europa y fue descrito por Leonardo de Pisa, un matemático italiano del siglo XIII, que llamó a este sistema los números de Fibonacci.

Algoritmo recursivo sin memoria

Código

```
# 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
```

En este programa podemos observar declaramos un arreglo del 1 al 50 donde obtuvimos los valores y las operaciones del programa de fibonacci recrusivo sin memoria en Python y ademas observaremos nuestra tabla de los valores que obtuvimos.

Elementos	Posición
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	1.2586E+10

Grafica de recursivo sin memoria

Fibonacci recursivo con memoria

Código

```
global contadora3
contadora3 = 0
cache = {0: 0, 1: 1}

def fib(n):
    global contadora3
    contadora3 = contadora3 +1
    if n not in cache:
        cache[n] = fib(n - 1) + fib(n - 2)
    return cache[n]

x=[]
f=[]
for i in range(1,100):
    x.append(i)
    contadora3=0
    cache = {0: 0, 1: 1}
    fib(i)
    f.append(contadora3)
```

En programa anterior es un poco similar a este no más que este programa lo que hace es que aguarda en una memoria las operaciones que está realizando, de manera que si nombras a ese mismo elemento que se aguardó te regresa el mismo y te lo arroja en la misma posición que te lo había dado antes. Vemos que estos programas nos ayudan para ver cómo es que se comporta la gráfica.

Tabla de datos

Elementos	Operaciones
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

Grafica

Fibonacci iteración

Código

```
def fiboiter(n):
    global cnt
    fib=[1,1]
    for k in range(2,n+1):
        cnt+=1
        fib.append(fib[k-1]+fib[k-2])
    return fib[n]

for n in range(0,101):
    cnt=0
    a=fiboiter(n)
    cntr,cnt=cnt,0
    print(n,fiboiter(n))
```

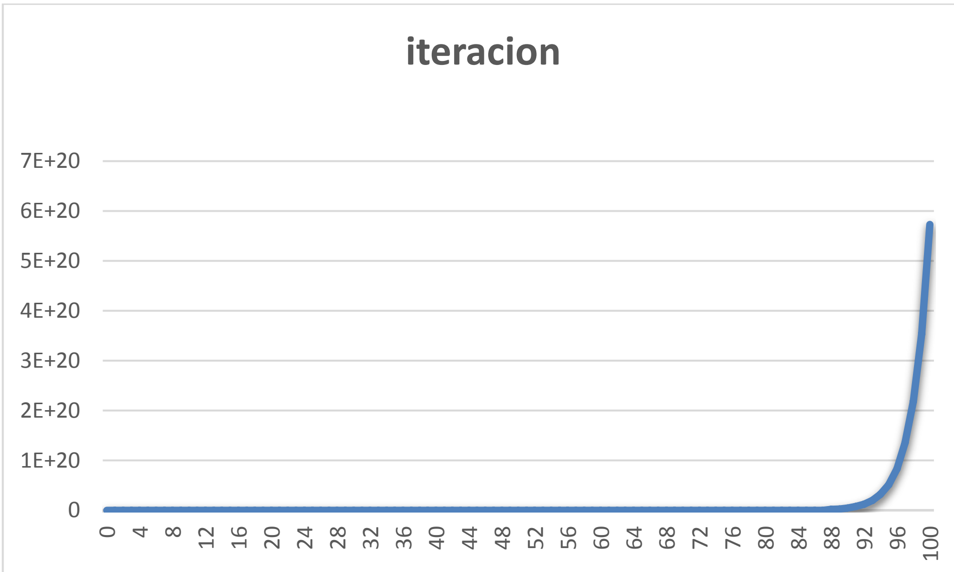
En este código utilizamos para el programa de iteración número del 1 al 100. Podemos calcular el arreglo solamente cada término una vez y además nos permite calcular el n-esimo término usando n suma aproximadamente. Aparte es un algoritmo muy eficaz.

Tabla de datos

Elementos	Operaciones
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	1.2586E+10
50	2.0365E+10
51	3.2951E+10
52	5.3316E+10
53	8.6268E+10
54	1.3958E+11
55	2.2585E+11
56	3.6544E+11
57	5.9129E+11
58	9.5672E+11
59	1.548E+12
60	2.5047E+12
61	4.0527E+12
62	6.5575E+12
63	1.061E+13
64	1.7168E+13
65	2.7778E+13
66	4.4946E+13
67	7.2723E+13
68	1.1767E+14
69	1.9039E+14
70	3.0806E+14
71	4.9845E+14
72	8.0652E+14
73	1.305E+15
74	2.1115E+15
75	3.4165E+15
76	5.5279E+15
77	8.9444E+15
78	1.4472E+16
79	2.3417E+16
80	3.7889E+16
81	6.1306E+16
82	9.9195E+16
83	1.605E+17
84	2.597E+17
85	4.202E+17
86	6.7989E+17
87	1.1001E+18
88	1.78E+18
89	2.8801E+18
90	4.66E+18
91	7.5401E+18
92	1.22E+19
94	3.194E+19
95	5.1681E+19
96	8.3621E+19
97	1.353E+20
98	2.1892E+20
99	3.5422E+20
100	5.7315E+20

93	1.974E+19

Grafica



3. Conclusión

En conclusión con los 4 algoritmos que realizamos en Python ya obteniendo las gráficas y tablas con las operaciones realizadas podemos ver en el algoritmo de los primos tenía una finalidad mostrar que el número era primo o no, para ello primero que nada hacia los elementos y luego no daba operaciones realizadas es decir la variable global. Así mismo determinar que era un número primo para que quedara un poco más claro.

En la parte de Fibonacci se dividió en tres partes en donde se dio una breve explicación acerca de cada uno de los programas en Python. El algoritmo de memoria es un programa muy eficaz porque se realiza de una manera muy rápida las operaciones además que no se borra los datos porque se aguarda al momento de que tu nombre un valor te va dar el mismo porque ya estaba ejecutado. En el algoritmo de sin memoria es un poco similar al de memoria nada más que en este es un poco eficiente y el iterativo se hace solo una operación para así mejorar su eficiencia.

por ultimo este trabajo nos ha dejado muy buenos conocimientos porque nos ayuda saber un poco más acerca de programar cosa que no te tardas mucho en hacerlo en cambio si nosotros como matemáticos hiciéramos a mano para buscar numero primos de una gran cantidad de número tardaremos mucho en cambio al programar es algo mas eficiente.