Algoritmos (primo y Fibonacci)

Nombre: Alexandra Valeria Hernández Quintero **Matricula:** 1663507

Unidad de aprendizaje: Matemáticas Computacionales **Fecha:** 13/10/2017

Resumen: Este reporte trata sobre los algoritmos para saber si un número es primo o no, y calcular el n-ésimo término de la sucesión Fibonacci.

Descripción del problema: crear un algoritmo para determinar si un número es primo o no lo es.

Teorema:

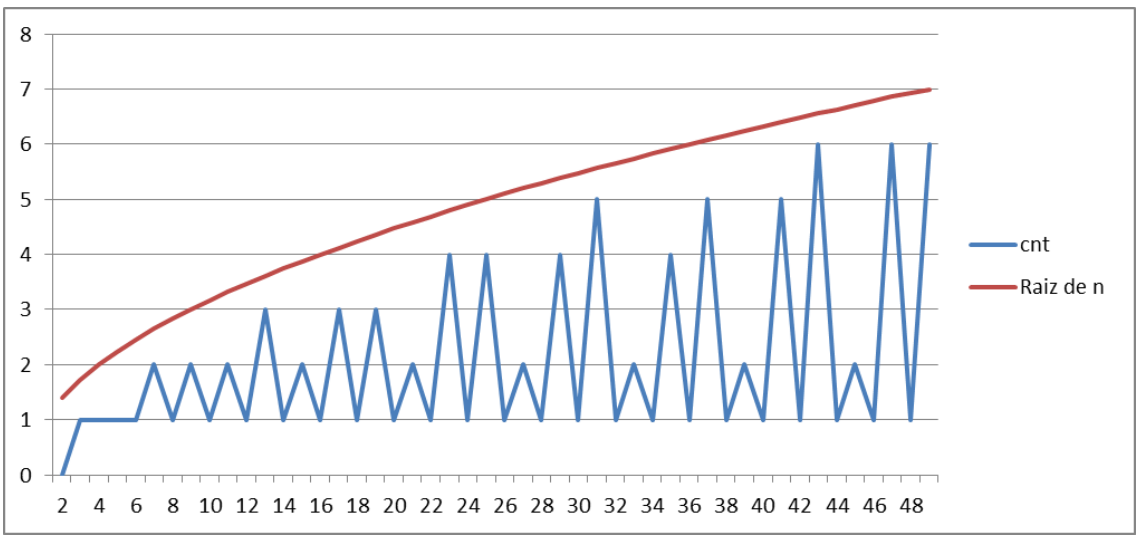
Sabemos que para que un número sea primo solo es divisible entre 1 y sí mismo, también se sabe que si un número "n" no tiene divisores primos menores o iguales que la raíz cuadrada de dicho número "< ó = $n^{1/2}$", entonces dicho número "n "es primo.

Usando el Teorema anterior, es muy sencillo crear el algoritmo para este problema, usaremos una función donde el parámetro de dicha función sea el número "n" que queremos analizar, después usaremos un ciclo For que va desde 2 hasta la raíz del número (2, $n^{1/2}$+1) ya que no nos sirve empezar desde el número uno, ya que todos los números son divisibles entre 1, después se pondrá una condición, para verificar si el número es divisible entre un numero diferente de uno, si cumple con la condición no es primo y si no cumple entonces es primo.

Pseudocódigo

```
definir primo (n):
        para cada  i en el rango(2, round(n**(1/2)+1)):
                Si n%i==0:
                        regresar("no es primo")
        regresar ("es primo")
```

A continuación se muestra una gráfica, con las operaciones que realiza el algoritmo, con los números del 2 al 50, para verificar si es primo o no lo es.



Descripción del problema: Crear un algoritmo para determinar el n-ésimo término de la sucesión de Fibonacci, donde los primeros términos son los siguientes:

1, 1, 2, 3, 5, 8, 13, 21, … , A_n

Donde $A_n = A_{n-1} + A_{n-2,}$ para toda n>2 (1)

Es decir que el n-ésimo término de la sucesión va ser igual a la suma de los dos anteriores términos de la sucesión.

Realizar esto a mano es sencillo si conocemos los dos términos anteriores al término que estamos buscando, pero si queremos el término 100 de la sucesión tardaríamos mucho realizando las operaciones, pero en Python podemos hacerlo rápidamente, sin la necesidad de ingresar los dos términos anteriores del término que estamos buscando. ¿Cómo se hace esto en Python? , pues bien hay distintas formas de hacerlo una de ellas es la que se mencionara a continuación.

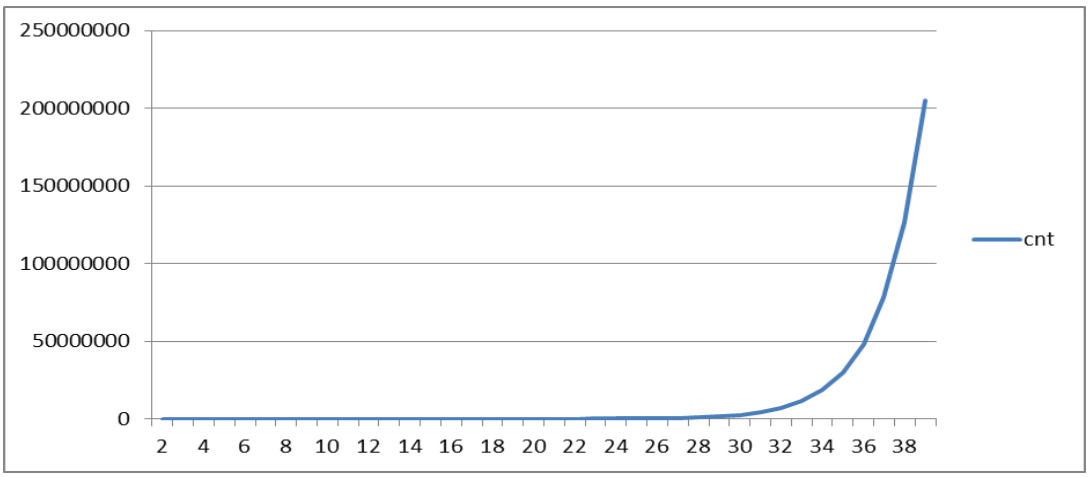
Algoritmo recursivo

Una forma muy sencilla de hacerlo sería crear una función, que tenga como parámetro el n-ésimo (1er, 2do,…etc.) termino que queremos obtener, y bueno como sabemos que el término 0 y 1 son iguales a 1, entonces por default cada vez que se obtenga como dato de entrada el 0 o 1 el término debe ser igual a 1 y la función regresara el valor de 1, para esto se pondrá una condición para la verificación del término. Si n es diferente de 0 y 1 entonces el n-ésimo término será mayor que 2, y para esto usamos la ecuación (1).

Pseudocódigo

```
cnt=0
definir fibonacci(n):
        global cnt
        cnt+=1
        Si n==0 o n==1:
                regresar (1)
        Si no:
                regresar fibonacci(n-2)+fibonacci(n-1)
```

A continuación se muestra una gráfica, con las operaciones que realiza el algoritmo, con los términos del 2 al 40, para encontrar los valores de dichos términos.



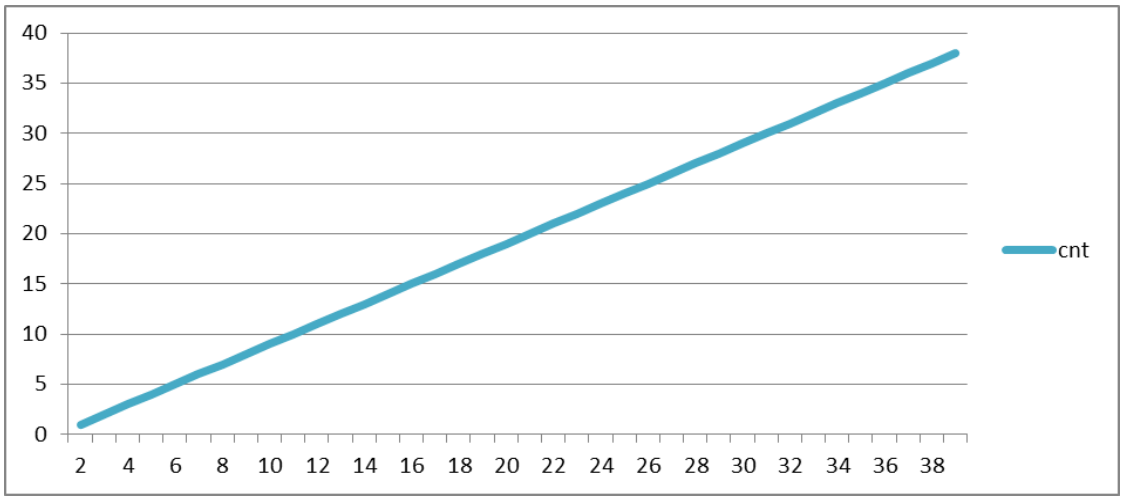
Algoritmo iterativo

Otra forma de hacerlo es mediante otra función, que tenga como parámetro el n-ésimo (1er, 2do,…etc.) termino que queremos obtener, de la misma forma que el algoritmo anterior sabemos que el término 0 y 1 son iguales a 1 la función regresara el valor de 1 cuando el n-ésimo término sea igual a 0 o 1, para esto se pondrá una condición para la verificación del término. Después haremos uso de tres variables, donde la primera será igual a 0, y la segunda y tercera serán iguales a 1, esto se hará con el fin de usar la ecuación (1) más adelante, la segunda y tercera variable están igualadas a 1 por que son los términos 0 y 1, usaremos como herramienta auxiliar un ciclo For que itere desde 2 hasta el término que buscamos, dentro del For se usa la ecuación (1).

Pseudocódigo

```
definir fibonacci(n):
        cnt1=0
        Si n==0 or n==1:
                regresar (1)
        r, r1, r2 = 0, 1 , 1
        Para cada i en el rango (2,n+1):
                cnt1+=1
                r=r1+r2
                r2=r1
                r1=r
        regresar r, cnt1
```

A continuación se muestra una gráfica, con las operaciones que realiza el algoritmo, con los términos del 2 al 40, para encontrar los valores de dichos términos.



Algoritmo con memoria

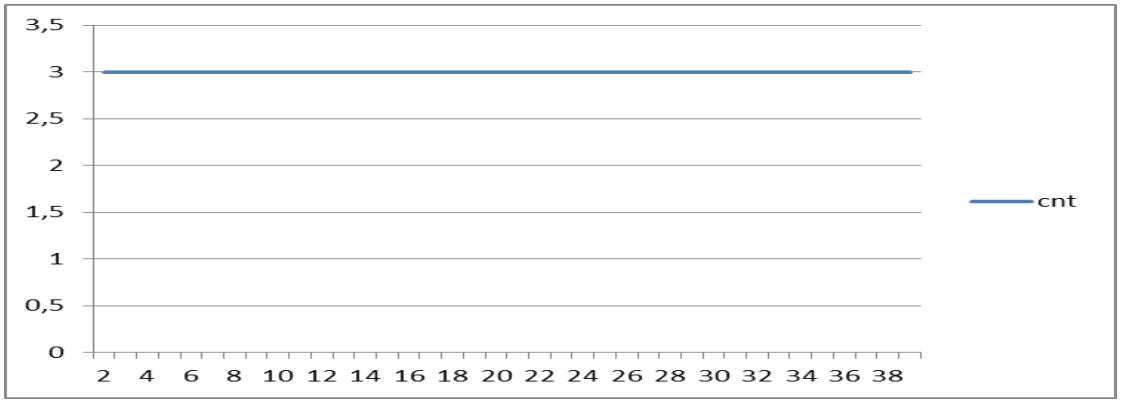
Una tercer forma de obtener el n-ésimo término de la sucesión Fibonacci, es mediante el uso de un diccionario (reserva un espacio de memoria para un conjunto de datos) y una función que tenga como parámetro el n-ésimo (1er, 2do,…etc.) termino que

queremos obtener. De la misma forma que los dos algoritmo anteriores sabemos que el término 0 y 1 son iguales a 1 la función regresara el valor de 1 cuando el n-ésimo término sea igual a 0 o 1, para esto se pondrá una condición para la verificación del término. Después mediante una condición, verificaremos si el n-ésimo termino ya está en el diccionario antes declarado, si ya está en el diccionario regresara el valor del n-ésimo termino, si no está en el diccionario entonces haremos uso de la ecuación (1) de la misma forma como se hizo en el primer algoritmo, pero también le añadiremos la instrucción de que si ya tenemos cual es el n-ésimo termino, lo guardaremos en el diccionario, asi cada vez que mandemos llamar a la función, y le pongamos como parámetro un n-ésimo termino ya antes calculado, la función directamente extraerá el valor el n-ésimo término del diccionario.

Pseudocódigo:

```
memoria={}
cnt2=0
definir fibonacci(n):
        global memoria, cnt2
        cnt2+=1
        si n==0 o n==1:
                regresar(1)
        si n esta en memoria:
                regresar memoria[n]
        si no:
                valor=fibonacci(n-2)+fibonacci(n-1)
                memoria[n]=valor
                regresar valor
```

A continuación se muestra una gráfica, con las operaciones que realiza el algoritmo, con los términos del 2 al 40, para encontrar los valores de dichos términos.



Nota: el número de operaciones es constante ya que, la prueba se hizo dentro de un For, asi que iba guardando los valores de los términos anteriores es por eso que hace solo 3 operaciones, si mandamos llamar a la función Fibonacci con un término donde los dos anteriores términos no están en memoria. Las operaciones dejarían de ser 3.

Conclusiones:

De todo lo anteriormente mencionado podemos notar, que del algoritmo para determinar si un número es primo, pues si es un numero par sólo hará una operación al ser divisibles entre dos todos los números pares y el 2 es el primer valor con el que itera el for a eso se debe que solo haga un operación en los números pares, y cuando se ingresa un numero primo es cuando más operaciones hará el algoritmo. La complejidad del primer algoritmo es $n^{1/2}$.

Claro está que de los algoritmos para calcular el n-ésimo término de la serie Fibonacci es el algoritmo con memoria, siempre y cuando en memoria estén los dos términos anteriores al que buscamos en la memoria, si no están, se podría decir que es igual de ineficiente que el algoritmo recursivo, el algoritmo iterativo podríamos decir que está en un término medio de eficacia en comparación con el algoritmo con memoria y el recursivo.