REPORTE DE ALGORTIMOS DE NÚMEROS PRIMOS Y FIBONACCI

MATEMÁTICAS COMPUTACIONALES

BENNY OZIEL ZAPATA VALLES

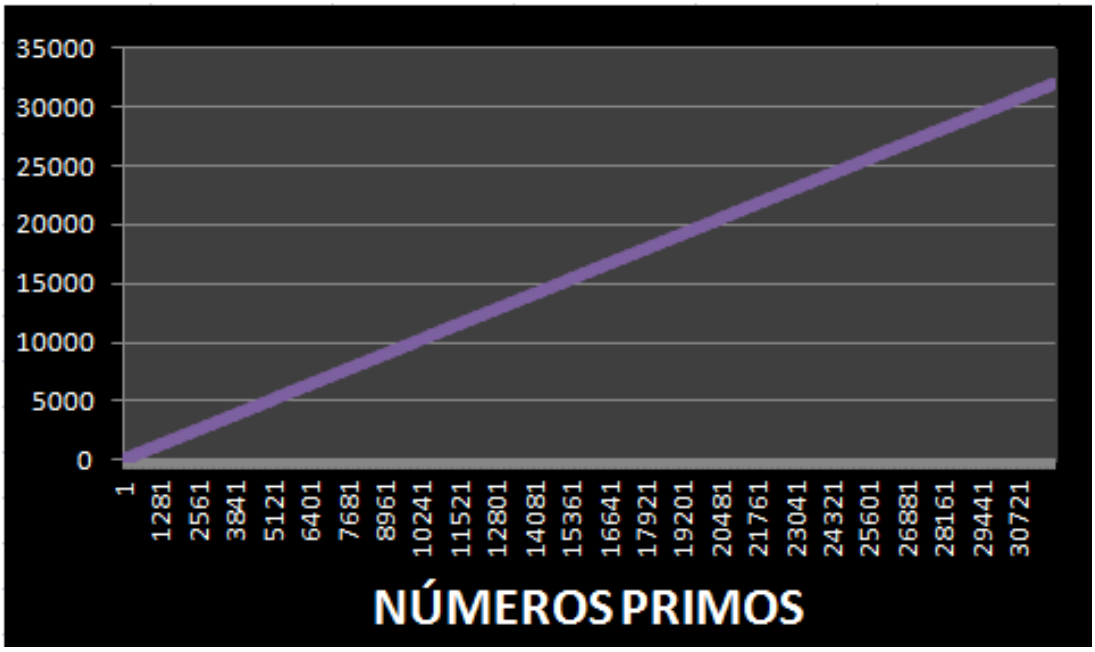
Algoritmo Primo

Durante mucho tiempo se han estudiado las propiedades y ventajas de los números primos, este tipo de número suman complejidad operacional a la de los números naturales, ya que, deben cumplir con un conjunto de características específicas. Estos números no sólo son un problema al momento de realizar sistemas operacionales con ellos, pues, el hecho de encontrar como tal cuales son los números primos es un problema complejo, largo y tedioso. Es por lo anterior que se ha tratado de programar ciertos algoritmos para la obtención de estos números, para así optimizar tiempo y reducir esfuerzo.

El objetivo principal del algoritmo es comparar el módulo, con la operación '***nmodi***', donde '***n***' es el número ingresado para saber si es primo o no, e '***i***' el elemento de una arreglo tal que este último repite la operación hasta que nuestra condición que el módulo sea igual a cero se cumpla, si es así, entonces el ciclo finaliza y se da por termina la parte principal del código.

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

Para el rendimiento de nuestro algoritmo tenemos la siguiente gráfica:



Este algoritmo tiene gran relevancia en cierto campo de las matemáticas, ya que facilita la manera de encontrar número primos, tanto por su no tan alta complejidad, por facilidad de uso, por rendimiento y por optimización de tiempo.

Fibonacci

Este tipo de sucesión fue dada a conocer por un 'problema de conejos', se quería saber la cantidad de conejos que habría en un cierto tiempo, dependiendo de que si, al inicio se tenía una pareja (macho y hembra) tales que estos se reproducían cada semana a excepción de la primera, encontrar ese número de conejos en un tiempo si la pareja tiene a su vez otra pareja con las mismas costumbres de reproducción y la pareja inicial no deja de reproducirse.

Pues bien, esta deducción se realizó gracias a usa sucesión dada donde se relacionan directamente el número de parejas anteriores al día requerido.

La sucesión está dada de la forma

$$f_{n+2} = f_n + f_{n+1}$$

Entonces, necesariamente para encontrar el valor de la sucesión en 'n', hay que saber el valor de la sucesión en n-1 y n-2.

1) Fibonacci Recursivo

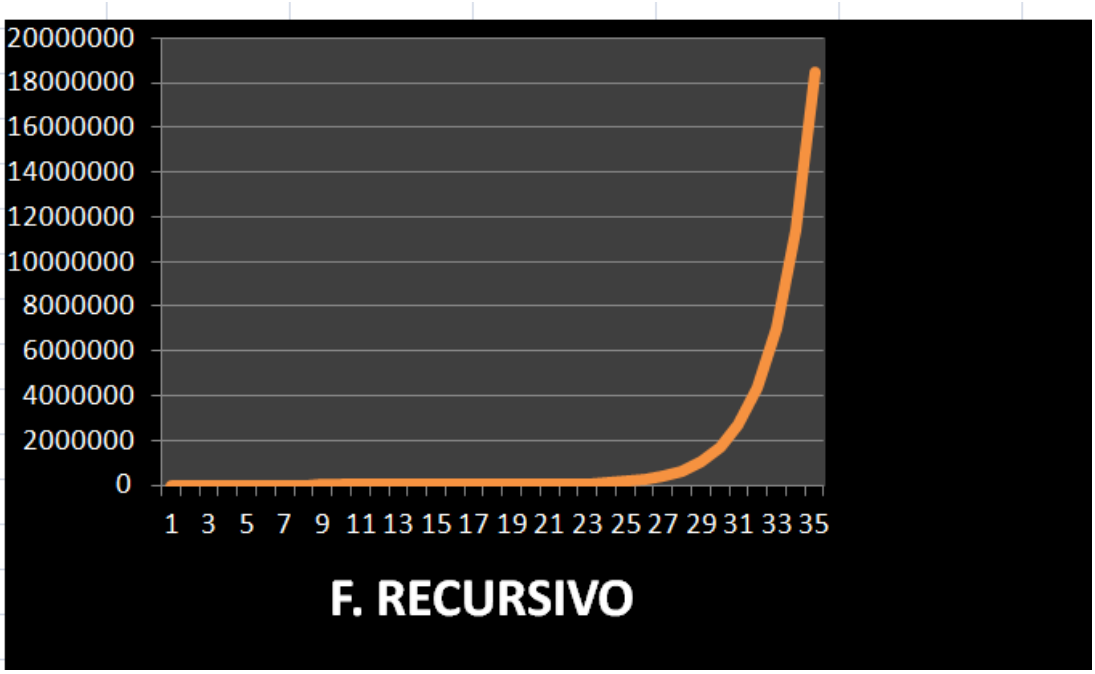
Para este algoritmo se declarará la función que utilizaremos (Fibonacci), para empezar , se tomaran en cuenta dos condiciones necesarias, cuando nuestro número sea igual a 0 o a 1, se toma el valor de 1, ya que así se fuerza que los primeros términos sean 1, que es una de las características de la sucesión Fibonacci, además de esta manera, tenemos ya los primeros dos términos requeridos para la obtención del tercer, y con el segundo y el tercero, la obtención del cuarto, así sucesivamente hasta llegar hasta nuestro número deseado.

Es decir, la suma de dos números ya definidos por el algoritmo encontraremos el siguiente, formando así una especie de ciclo sin necesidad de serlo como tal.

Como se puede ver a continuación:

```
cnt=0

def fibonacci(n): #Declaración de variables

        global cnt #Contador global

        cnt+=1

        if n==0 or n==1: #Condición inicial para determinar los dos primero elementos de la sucesión

                return(1) #Valor de los dos primeros elemntos

        else:

                return fibonacci(n-2)+fibonacci(n-1) #Suma de los valores de los elementos anteriores al elemento buscado
```

Cómo gráfica de rendimiento de este algoritmo tenemos:



Este algoritmo es bueno ya que cumple el objetivo y su grado de entendimiento es muy bajo, por lo cual es sencillo comprender lo que ocurre con cada elemento de la sucesión. Algo malo que podría tomarse de aquí es que llega a ser repetitivo al tomar los valores anteriores al buscado.

2) Fibonacci Iterativo

Al igual que en el algoritmo anterior, tenemos la condición especial para partir de la base que los primeros términos ya estén definidos.

A partir de ahí se toman un conjunto de variables con valores definidos tales que, al iniciarse un ciclo de 2 hasta la posición del número buscado.

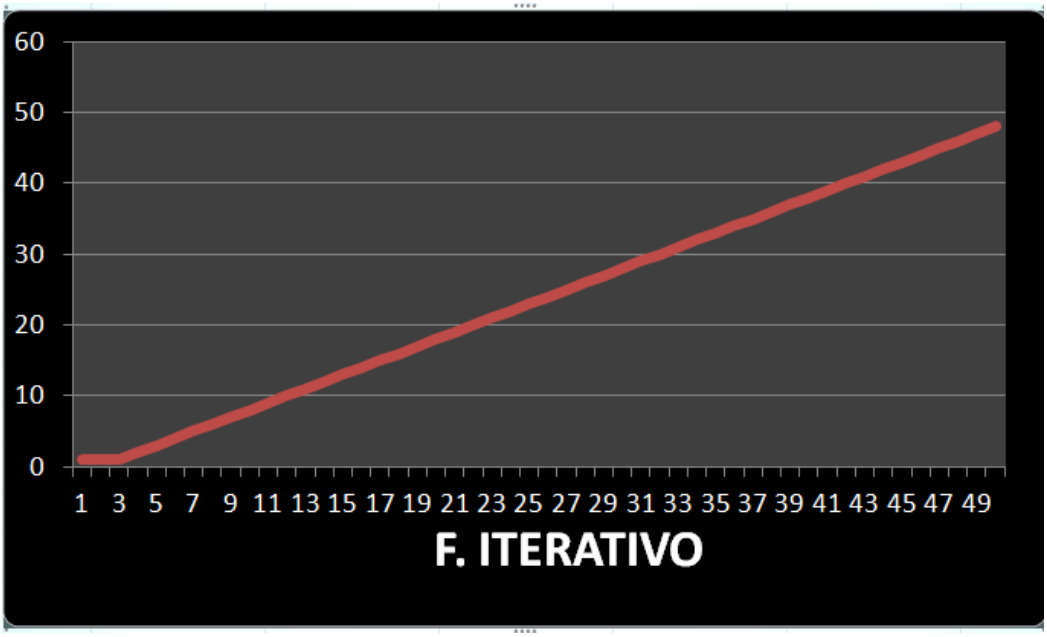
Se 'juega' con las variables y con sus valores, para, poco a poco, ir tomando los valores de las posiciones anteriores a la buscada, no sin antes pasar por todas las anteriores a ellas.

```
def fibo(n): #Definición de variable
    global cnt #Contador
    if n==0 or n==1: #Condicional para los primeros dos términos
```

```
        return(1) #Valor de los primeros térimon
    r, r1, r2 = 0, 1, 2 # Definición de variables
    for i in range (2,n): #Ciclo para repetir el proceso hasta llegar a la posición deseada
        cnt+=1
        r=r1+r2
        r2=r1
        r1=r
    return (r)
```

#Conjunto operacional para la asignación de valores a los elementos de la sucesión

El rendimiento de este algoritmo está dado por:



Este algoritmo puede llegar a ser un poco más complejo a la comprensión, ya que en la parte de las variables es fácil seguir el 'camino' dado dependiendo de la variable que tengamos, pero la deducción es lo que puede complicar el proceso, el saber que valores tomarán cada variable, la forma de acomodarlos, etc. estas dificultades pueden hacer que este proceso sea un poco más difícil de entender al anterior visto.

3) Fibonacci Recursivo Con Memoria

Este algoritmo recursivo con memoria es parecido a los anteriores ya que también se toma como base los dos primero elementos de la sucesión, para después, partir de ahí con los demás. Es casi igual al algoritmo anterior, sólo que con este tenemos una especia de memoria, la cual es un arreglo en la cual guardaremos nuestros valores ya obtenidos para no tener que volver a repetir el proceso como lo hacían los anteriores. El ejemplo siguiente muestra una manera de representar el algoritmo mencionado:

```
arreglo={}
cnt=0
def fibo (n):
 cnt+=1
 if n==0 or n==1:
  return (1)
  if n in arreglo:
   return arreglo[n]
  else:
   val=fibo(n-2)+ fibo(n-1)
   arreglo[n]=val
   return val
```

Para este algoritmo tenemos su representación de rendimiento de esta manera:

La ventaja de este algoritmo con respecto a los anteriores es, como ya mencionamos, que al guardar los valores de los elementos en la sucesión, el proceso se ahorra mucho tiempo al ya no tener que volver a calcularlos.

Cuando analizamos todo lo anterior mostrado, el mejoramiento de nuestro algoritmo puede ser mejor, puede darnos una mejor optimización del tiempo, si tenemos una idea clara del problema que queremos resolver, cuando tenemos bien planteado nuestro problema y encontramos una forma más compleja de ejecutar pero con mejores resultandos. Es decir, no es sencilla llegar al mejor algoritmo para la sucesión Fibonacci pero una vez que se logra es aquel que nos dará un mejor resultado en menor número de operaciones. Nuestro algoritmo menos complejo es el que tiene menor número de operaciones, y el más complejo es el que optimiza mejor el tiempo, entonces, dependiendo del problema que tengamos podemos decidir utilizar uno u otro. En conclusión no siempre la forma más sencilla de hacer algo va a hacer la mejor, puede haber alguna que mejore rendimiento y ahorre tiempo, el problema esencial es encontrar esa forma.