Números primos y Fibonacci

Matemáticas computacionales

Yarethzi Giselle Bazaldúa Parga

13 de octubre de 2017

Resumen

En matemáticas, un número primo es un número natural mayor que 1 que tiene únicamente dos divisores distintos: él mismo y el 1. Algunos ejemplos de números primos son los siguientes: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149. Se llaman números de Fibonacci a aquellos que forman parte de la sucesión infinita de números naturales donde cada número se calcula sumando los dos anteriores a él. La sucesión de Fibonacci es la siguiente sucesión de números enteros positivos: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Números primos:

Éste algoritmo es una herramienta que te facilita saber si un número es primo o no en cuestión de segundos, funciona tratando de dividir al número en cuestión entre números mayores al 1, así si algún numero da como residuo cero, el número es compuesto, de lo contrario, es primo. El algoritmo fue realizado en clase con ayuda del profesor.

```
>>> def primo(n):
        cnt=1
        for i in range(2, math.ceil(math.sqrt(n))):
                cnt = cnt+1
                if ((n%i) == 0):
                        break
        return cnt
```

Fibonacci recursivo (sin memoria):

El algoritmo muestra el número de operaciones que se hicieron para encontrar el número n de la serie de Fibonacci, esto se hace un una variable contadora global que aumenta al mismo tiempo que el algoritmo se repite. Éste algoritmo fue programado también en clase.

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

Fibonacci iterativo:

Éste algoritmo es muy parecido al anterior de recursividad, sólo que hace las operaciones de una forma un poco más “antigua” ya que va dando y almacenando valores a cada r, r1 y r2 hasta llegar a la r que se pidió con ayuda del ciclo for, es un poco más austero pero igual funciona. Se realizó en clase con ayuda del profesor y compañeros.

```
>>> cnt=0
>>> def fibo(n):
        global cnt
        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)
```

Fibonacci recursivo (con memoria):

Trabaja con una memoria y contador global, trabaja casi igual que el primer algoritmo de Fibonacci sólo que éste va guardado los n ya calculados en un arreglo, si el número que le solicitas al algoritmo no ha sido calculado, lo calcula y lo almacena para posibles números que quieras saber después, además guarda el número de veces que se realiza la operación en su variable contadora. Se realizó con ayuda de un compañero de la clase de computacionales.

```
>>> 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)
                memo[n]=val
                return val
```

Conclusiones:

Desde mi punto de vista, el algoritmo más eficiente y eficaz es el Fibonacci recursivo con memoria porque al almacenar lo valores que ya calculó, se ahorra el tiempo de tener que calcularlo de nuevo si se lo piden y así sólo se calculan los nuevos números que se le dan, además te puede imprimir el número Fibonacci correspondiente a la posición que le diste.