# ALGORITMO DE OPTIMIZACIÓN PARA LA DETECCIÓN DE UN NÚMERO PRIMO BASADO EN PROGRAMACIÓN FUNCIONAL UTILIZANDO DRSCHEME

## Algorithm to optimize the detection of a prime number using Functional Programming with DrScheme

**RESUMEN**
Sobre la base de una propuesta para determinar de manera simple si un número es primo o no, el presente artículo presenta dos formas de optimización a partir de la aplicación del paradigma de programación funcional utilizando DrScheme. En esencia lo que se hace es mantener la estructura de funcionamiento recursivo apoyado en tres funciones interrelacionadas sobre las cuales se hacen unas consideraciones notables en relación con el rango sobre el cual se realizan las iteraciones. Al ejecutar las dos versiones optimizadas se nota sustancialmente una diferencia de tiempo de respuesta lo cual, por sí solo, favorece el perfil del objetivo propuesto en este artículo.

**PALABRAS CLAVE:** *Optimización, número primo, programación funcional, Scheme.*

**ABSTRACT:** *Based on a program to determine, on a simple way, whether a number is prime or not, this article presents two forms of optimization from the application of functional programming paradigm using DrScheme. In essence We maintain the structure of recursive operation supported by three interrelated functions on which consider some notable relative to the range over which the iterations are performed. Running the two versions optimized substantially noticeable difference in response time which, by itself, is the target set in this article.*

**KEYWORDS**
Optimization, prime number, functional programming, Scheme

**Msc. OMAR IVAN TREJOS BURITICÁ**
Ingeniero de Sistemas
Espec. Instrumentación Física
Msc. Comunicación Educativa
PhD © Ciencias de la Educación
Profesor Titular
Universidad Tecnológica de Pereira
omartrejos@utp.edu.co

## 1. INTRODUCCIÓN

El análisis de los números primos y la búsqueda de algoritmos que los detecten fácilmente ha sido siempre una preocupación a nivel de programación de computadores. Primeramente porque las mismas características matemáticas de los números primos han sido un factor que ha seducido no solo ha matemáticos a lo largo de la historia sino a programadores en los sesenta años que lleva la programación como área definida de conocimiento. De otra parte la gran cantidad de aplicaciones que se le han encontrado a los números primos, precisamente por sus características matemáticas, ha permitido encontrar en ellos el soporte para poder hacer que algunos procesos sean mucho más seguros y que temas tan modernos como la encriptación tengan componentes matemáticas de gran solidez.

El presente artículo se basa en una propuesta simple para la detección de un número primo formulada por el mismo autor que permite, a partir de unas funciones recursivas y basado plenamente en el paradigma funcional, que ha sido presentado en otras oportunidad. Esta propuesta de solución efectivamente, en un tiempo prudencial, puede determinar si un número dado es primo o no. Lo que se ha pretendido con los dos estadios de optimización que se presentan es reducir significativamente el tiempo de ejecución en la determinación del número primo. Para ello se han hecho pruebas que, articuladas con el reloj interno del computador, muestran los números que se involucran a nivel de segundos y que reflejan el término optimización de la mejor de las formas.

Esto nos lleva a pensar en dos hipótesis en relación con la programación de computadores: en primera instancia que siempre es posible optimizar un proceso dado en función del tiempo de ejecución que éste requiere y, en segundo lugar, que es posible encontrar formas ingeniosas para determinar si un número es primo en un tiempo técnicamente óptimo.

La necesidad de hacer mejor uso de los recursos con que cuenta el computador se ha de convertir en una de las metas para el programador. El tiempo es tal vez el único recurso que tiene el computador que no tiene repuesto y que no se puede encontrar en la tienda de la esquina, por este motivo, cualquier algoritmo que optimice el recurso del tiempo en la ejecución de un programa es interesante no solamente por los resultados que dicho algoritmo arroje sino por el análisis lógico que compete a éste y que

Fecha de Recepción: 15 de Diciembre de 2010
Fecha de Aceptación: 28 de Abril de 2011

se refleja en la reducción significativa y notoria del tiempo de ejecución.

El método utilizado parte del análisis del algoritmo inicial y de los elementos constitutivos de éste que pueden ser optimizables en función del tiempo. Se evalúan las características del algoritmo funcional (escrito bajo la forma de la sintaxis de DrScheme) y sobre ellas se trabajan. Se detectan dos elementos que pueden reducir el tiempo de ejecución y que se plasman en la primera optimización. Finalmente se hace una segunda optimización basado en el análisis de la nueva versión del programa de manera que se vuelva a reducir notoriamente el tiempo de ejecución de dicho programa.

## 2. DEFINICIÓN MATEMÁTICA DE UN NÚMERO PRIMO.

En matemática, se considera como número primo a todo número natural que tiene únicamente dos divisores exactos: él mismo y el número 1. Tal es el caso, por ejemplo, del número 13 cuyos únicos divisores exactos son el número 1 y el mismo número 13. Si un número, además del mismo número y del número 1, tiene otros divisores entonces se llama número compuesto. Un ejemplo de ello podría ser el número 15 cuyos divisores exactos son los números 1, 3, 5 y 15 [1].

Se conoce como ***primalidad*** la propiedad que tiene un número de ser, precisamente, un número primo. Debido a que el único número primo par es el número 2 (dado que cumple con la definición) entonces, con frecuencia, se utiliza el término *número primo impar* para referirse a cualquier número primo mayor que 2 [2].

En relación con el número 1 se tiene en la escena matemática una discusión acerca de si ha de considerarse como tal o no. Puede decirse que ambas posturas matemáticas pueden ser, según el caso, inconvenientes o, según otras situaciones, puede ofrecer ciertas ventajas. Hasta el siglo XIX, la comunidad matemática, en su mayor parte, consideraban el 1 como un número primo. La lista de números primos que publicó Derrick Norman Lehmer en 1956 iba desde el número 1 hasta el número 10.006.721 lo cual evidencia que este matemático consideraba al 1 como un número primo [3].

En la actualidad, la comunidad matemática tiene cierta inclinación a no considerar el número 1 en la lista de los primos. Ello lleva a pensar que principios como el teorema fundamental de la aritmética o algunas propiedades no se cumplen plenamente cuando se trata el número 1. Para efectos de este artículo se considerará el 1 como número primo aunque ha de aclararse que solo se necesita cambiar un valor por otro para que éste quede excluido de la lista, ajustando así la solución que aquí se plantea a la tendencia moderna en cuanto a la no concepción del 1 como número primo.

## 3. DEFINICIÓN FUNCIONAL DE UN NÚMERO PRIMO.

La solución que se presenta en este artículo busca aprovechar al máximo la capacidad computacional moderna para obtener la respuesta acerca de si un número es primo o no esperando que el tiempo de respuesta sea el más breve posible y sabiendo que el guarismo ha de ser introducido por el usuario.

Dado que la definición de número primo establece que éste es todo aquel número natural que tiene solamente dos divisores exactos y que son el número 1 y el mismo número en cuestión, entonces es claro que un número primo podríamos definirlos, para efectos de construir el programa solución, de la siguiente forma: *Un número primo es aquel entero que tiene solamente, y no mas que, dos divisores exactos en el rango [1, n] siendo n el número que se quiere evaluar [4]*.

Esta definición tan simple, y tan próxima a la definición matemática pues deriva de ella, nos permitirá entonces pensar en que si contamos la cantidad de divisores exactos que tiene un número cualquiera y si notamos que esa cantidad es igual a 2, entonces podemos concluir que el número es primo. Es aquí en donde la definición de primalidad en el número 1 podría aparecer en escena y la resolvemos de una forma muy sencilla: tomar el número 1 como número primo para lo cual la definición que se presenta en el párrafo anterior es suficiente y no requiere ninguna modificación ó excluir el número 1 (y de paso excluir, por redundancia, el mismo número que se quiere evaluar).

En este último caso la definición de número primo podría tener la siguiente definición: *Un número primo es aquel entero que no tiene ningún divisor exacto en el rango [2, n-1] siendo n el número que se quiere evaluar.* Y entonces podríamos ser mucho más eficientes y determinar que, sabiendo que de la mitad + 1 de un número n hasta (n-1) no existen divisores exactos. Por ejemplo, el número 1000 no tiene divisores exactos entre el número 501 y el número 999. De forma que una definición más eficiente, en términos computacionales, podría ser: *Un número primo es aquel entero que no tiene ningún divisor exacto en el rango [2,* $\frac{n}{2}$ *] siendo n el número que se quiere evaluar [5]*.

Para efectos de este artículo se utilizará la primera definición que se planteó en esta sección. A manera de optimización, es de aclarar que con el simple hecho de que se encuentre un solo divisor exacto en ese rango es suficiente para que el número no sea primo.

## 4. ALGORITMO DE DETERMINACIÓN SIMPLE DE UN NÚMERO PRIMO.

El algoritmo para la determinación simple de un número primo, cuyo autor es el mismo autor de este documento, está escrito en Lenguaje Scheme y se ha compilado y ejecutado en el IDE (Ambiente Integrado de Desarrollo) DrScheme Versión 4.1.4. Esta versión del programa inicia con una función que determina si un valor es múltiplo de otro. En esta función los dos valores se han llamado a y b y la condición se evalúa haciendo uso de la función *remainder* (que calcula el residuo de una división). En caso de que el residuo de dividir a entre b sea igual a 0 significa que a es múltiplo de b. Si la condición retorna un valor true (Verdadero) entonces se devuelve el valor 1, en caso contrario se devuelve el valor 0 [6].

```
(define (esmulti a b)
  (if (= (remainder a b) 0 )
     1
     0  ))
```

Es una función profundamente sencilla cuya importancia radica en que es el fundamento para desarrollar el proceso de conteo que se hace en la siguiente función. Ésta es una función que cuenta la cantidad de múltiplos que tiene un valor determinado dentro del rango (1, n) siendo n el valor. Si en ese rango, un valor cualquiera, tiene solamente dos múltiplos entonces significará que el número es primo puesto que los dos valores serán el número 1 y el mismo número n. Caso especial constituye el número 1.

Esta función se basa en el principio de la recursividad y toma el valor inicial (n) y lo va dividiendo progresivamente por cada uno de los valores comprendidos entre n y 1 (que en este caso son los valores almacenados en el argumento d). En la medida en que va encontrando múltiplos del valor n, los va contando. Para esto se apoya en la función *esmulti*. Al finalizar su ejecución, la función *cuentamulti* retorna la cantidad de múltiplos que tiene el valor que se recibió inicialmente.

```
(define (cuentamulti n d)
  ( if (= d 0)
     0
     (+ (esmulti n d)
       (cuentamulti n (- d 1)))))
```

La función *primo* recibe su argumento y lo envía dos veces, de esta forma el 1º valor de n carga el valor de n de la función *cuentamulti* y el 2º valor de n carga el argumento d que es el que se va a ir disminuyendo progresivamente de 1 en 1 hasta llegar a 0. Esta función tiene por objetivo permitir que para el usuario sea transparente la utilización del 2º argumento.

```
(define (primo n)
  (cuentamulti n n))
```

Finalmente la función *esprimo* es la que inicia todo el proceso y entrega el resultado final que, basado en lo dicho anteriormente, corresponde a la determinación de si el número solicitado es primo o no y se basa en el conteo de sus múltiplos en el rango (1, n). La respuesta se entrega de una forma muy escueta con dos avisos claros pero cortos.

```
(define (esprimo n)
  (if (= (primo n) 2)
     (display "es primo")
     (display "no es primo")
     ))
```

Esta versión inicial del algoritmo tiene dos grandes ventajas: es un programa bastante simple de entender y es, asimismo, bastante efectivo. Como desventaja podríamos citar el hecho de que ocupa un tiempo considerable cuando se trata de número grandes, sea que éstos sean primos o no. Demora el mismo tiempo en la determinación de cualquiera de las dos respuestas dado que recorre el mismo rango de posibles múltiplos. Este tiempo también implica tener en cuenta los recursos de procesamiento que se consumen en la utilización activa y persistente de la recursión.

## 5. PRIMERA OPTIMIZACIÓN DEL ALGORITMO PARA DETERMINAR SI UN NÚMERO ES PRIMO.

De las cuatro funciones de la versión inicial, la primera o sea la función *esmulti* se deja intacta tal y cual como se programó desde el principio.

```
(define (esmulti a b)
  (if (= (remainder a b) 0 )
     1
     0
     ))
```

La función *cuentamulti* tiene algunos cambios de gran importancia para el efecto de optimización que motiva este artículo. En primera instancia se reconsidera el rango en donde se buscan los posibles divisores exactos del número n. En vez de buscar en el rango (1, n) se busca en el rango $(2, \frac{n}{2})$, de esta forma se reduce a la mitad la cantidad de esos posibles divisores basados en que entre la mitad de un número n y el valor (n – 1) no existen divisores exactos sea cual fuere el número y siempre y cuando éste sea un valor entero. El otro cambio que se incorpora a esta función es que, en el caso de que se encuentre un divisor (el 1º divisor), inmediatamente se retorna el valor 1, en caso contrario se suman 0

seguidamente junto con el resultado del proceso recursivo.

```
(define (cuentamulti n d)
  ( if (= d 2)  ;; tope inferior de evaluación 2 rango (2, n/2)
      0
      (if (= (esmulti n d) 1) ;; se evalua si se encuentra el 1er divisor
          1              ;; entonces se retorna 1
          (+ 0 (cuentamulti n (- d 1)))))) ;; se suman ceros
```

La 3ª función que se usa en la versión original del programa también tiene un elemento de optimización y es que es en ella en donde se establece el tope superior del rango. De esta forma, cuando se llama a la función *cuentamulti* no se envía dos veces el valor de n sino que se envía como 1º argumento el valor de n y como 2º argumento el valor que resulta de tomar la parte entera de dividir el número n entre 2 (o sea la mitad entera del número).

```
(define (primo n)
  (cuentamulti n (floor (/ n 2)))) ;;tope superior de evaluación n/2
```

Por último, la función que origina todo el proceso tiene el último factor de optimización que corresponde a preguntar ya no por la cantidad de divisores exactos que tiene un número n sino que pregunta si se encontró UN divisor exacto, dado que la existencia de al menos un divisor exacto en el rango $(2, \frac{n}{2})$ para cualquier valor de n es suficiente para concluir que el valor n es primo.

```
(define (esprimo n)
  (if (= (primo n) 1) ;; Se evalúa si se encontró el 1er divisor exacto
      (display "no es primo") ;; si fue así el número no es primo en el rango (2, n/2)
      (display "es primo") ;; si no fue asi el número es primo
      ))
```

De acuerdo a esto si se quisiera averiguar si el número 1000000 es un número primo no habría que esperar, como en la primera versión del programa, a que recorriera todo el rango desde 1 hasta 1000000 sino que apenas encuentre el valor 500000, que es el primer divisor que se encuentra al hacer la regresión de 1000000 a 1 (tal como lo hace recursivamente el programa), inmediatamente aparece el mensaje indicando que no es un número primo. No ha de desconocerse que el valor 1 es un caso especial y debe tratarse por fuera de estas funciones. La razón de no incluirlo es que, por ser un caso tan puntual, el gusto del autor de este artículo sugiere tratarlo independientemente, es decir, por fuera de esta solución.

## 6. SEGUNDA OPTIMIZACIÓN DEL ALGORITMO PARA DETERMINAR SI UN NÚMERO ES PRIMO.

En esta segunda optimización se hacen dos cambios que, considerando el tiempo de ejecución, son profundamente significativas y permiten que el tiempo de ejecución se bastante reducido por lo menos cuando el número no es primo pues se retorna en la primera ocurrencia en la cual se encuentra un divisor exacto. La primera función queda tal y como estaba originalmente:

```
(define (esmulti a b)
  (if (= (remainder a b) 0 )
     1
     0
     ))
```

La segunda función es la que tiene dos de los cambios profundos: en primer instancia se cambia el tope inferior que se tenía (pues la progresión iba de n a 1) por el tope superior del rango que esta vez estará dado por la raíz cuadrada del valor n. El otro cambio que tiene esta función es originado por el primero dado que la instrucción recursiva ya no incluye una resta sino una suma pues la progresión de evaluación ya no va de n/2 a 1 sino de 1 a n/2 en términos de aritmética entera.

```
(define (cuentamulti n d)
  ( if (= d ( sqrt n) )  ;; cambio tope inf por sup de evaluación 2 rango (2, n/2)
      0
      (if (= (esmulti n d) 1) ;; se evalúa si se encuentra el 1er divisor
          1              ;; entonces se retorna 1 indicando que el numero no es primo
          (+ 0 (cuentamulti n (+ d 1)))))) ;; Cambió el menos por un mas
```

La 3ª función tiene un cambio de optimización bastante interesante. Los valores que se envían como argumentos a la función cuentamulti son el valor de n y el valor 2 dado que es en el rango optimizado $(2, \sqrt{n})$ en el cual se encuentran los posibles divisores de n.

```
(define (primo n)
  (cuentamulti n 2 ) )
```

La 4ª función, que es la que inicia todo el proceso, se mantiene igual a la versión sin embargo la ejecución no es la misma puesto que en esta versión doblemente optimizada, en caso de que el número a evaluar no sea un número primo se retorna la respuesta con solo encontrar el 1º divisor exacto. Acorde con esto si se quisiera evaluar el valor 1000000, se encontraría la respuesta en la

primera iteración dado que 1000000 es múltiplo de 2 que es el 1º valor que encuentra el programa.

```
(define (esprimo n)
  (if (= (primo n) 1) ;; Se evalúa si se encontró el 1er divisor exacto
      (display "no es primo")
      (display "es primo")
      ))
```

Esta es, finalmente, la versión optimizada especialmente para los casos en los cuales el número no sea un número primo.

## 7. EVALUACIÓN DE LOS RESULTADOS.

Al ejecutar las tres versiones solución al problema planteado se nota una diferencia notoria. En la primera versión (versión original) el tiempo de evaluación de un valor como 1000000 es bastante algo, comparando con los tiempos de proceso que se manejan en un computador moderno. Cuando se realiza esta misma evaluación con la segunda versión del programa solución y, más aún, cuando se hace con la tercera versión de dicho programa se notan diferencias sustanciales en tiempo de proceso que ni siquiera hay que medirlas computacionalmente sino que se perciben de manera inmediata y directa.

## 8. CONCLUSIONES.

La optimización de procesos, y especialmente el enfoque desde el tiempo de procesamiento, es un factor muy importante en el desarrollo y refinamiento de programas que involucran cálculos aritméticos. Algunas limitaciones han de tenerse en cuenta cuando se plantean soluciones que se basan en la recursión como principio de optimización puesto que éste es uno de los factores que más recursos necesitan al momento del procesamiento al punto que si se excede su utilización, como cuando se escriben números demasiado grandes, sale un error que aborta el proceso y no permite su finalización normal. Este es un factor importante cuando se quiera optimizar un programa. Finalmente ha de tenerse en cuenta que el proceso que se ha optimizado es aquel en el cual el valor a evaluar no es un número primo; se abordará el tema de la optimización en los procesos asociados cuando el valor es un número primo en otros documentos.

## 9. REFERENCIAS BIBLIOGRÁFICAS

[1] TREJOS BURITICA, Omar Ivan (2010), Determinación simple de un número primo basado en programación funcional usando Drscheme, Revista Scientia et Technica, No. 45, Año 2010, Universidad Tecnológica de Pereira

[2] A NEW APPROACH TO COMPUTER SCIENCE, (2005), Burns Brendan, University of Massachussets, USA

[3] AN INTRODUCTION TO THE HISTORY OF MATHEMATICS, Howard Eves (1990), Editorial Saunders, ISBN 0-03-029558-0, USA

[4] CONCEPTS, TECHNICS AND MODELS OF COMPUTER PROGRAMMING, (2003), Van Roy Peter, Swedish Institute of Computer Science

[5] HOW TO DESIGN PROGRAM, An Introduction to Programming and Computing, (2003), Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi, The MIT Press,, Cambridge, Massachussets, USA

[6] TREJOS BURITICÁ, Omar Ivan (2004), Fundamentos de Programación, Manizales (Colombia), Editorial Papiro