Algoritmo recursivo para buscar y contar números primos en un rango predefinido usando Programación Funcional

Recursive algorithm for searching and counting prime numbers in a predefined range using Functional Programming

Omar Ivan Trejos Buriticá, PhD[1]

[1] *PhD Ciencias de la Educación, omartrejos@utp.edu.co, Universidad Tecnológica de Pereira*

Fecha de recepción: 18/02/2014 Fecha de aceptación del artículo: 23/05/2014

Resumen

Se acudió a la Programación Funcional para buscar y contar números que cumplan con la característica de primalidad en un rango elemental (1,n) a partir del uso de un algoritmo recursivo. Se planteó la fundamentación teórica de dicho algoritmo y además se aprovechó su formulación lógica para resolver el problema propuesto. El propósito de este artículo fue mostrar una arista útil aplicativa de la eficiencia algorítmica teniendo en cuenta las características tecnológicas modernas y los problemas que la matemática provee partiendo de un principio de simplicidad completamente alcanzable por los estudiantes de ingeniería en sus primeros semestres de formación. Se hizo uso del lenguaje de programación Scheme y se aprovecharon sus potencialidades para implementación de soluciones recursivas. Se demostró que acudiendo a algoritmos simples, la tecnología computacional moderna puede ser de una inmensa utilidad para resolver problemas matemáticos.

Palabras clave

Algoritmo, Matemáticas, Números primos, Programación funcional.

Abstract

In this article we use Functional Programming to find and count numbers with primality characteristic in an primary range (1,n) using a recursive algorithm. You can find the theoric foundations of this algorithm and also its logic solution to solve the problem. The proposal of this article is to show an applicative use of the algorithmic efficiency knowing the modern technologies and the problems coming from math using a simplicity principle absolutely reachable by the first semesters engineering students. We use Scheme programming language y its potentialities in recursive solutions. We found evidence that modern computational technology is very useful to solve math problems.

Keywords

Algorithm, Functional programming, Maths, Prime numbers.

1. Introducción

1.1 Generalidades

Una de las particularidades más especiales de la programación de computadores en el mundo moderno consiste en que cada día se encuentran más aplicaciones tanto para atender las nuevas necesidades de la sociedad que pueden ser resueltas con tecnología computacional como para resolver, por estos medios, problemas que han sido históricamente fatigantes. La búsqueda de soluciones a problemas matemáticos por los caminos que la computación provee es un tema recurrente en el mundo de hoy aprovechando la gran velocidad de procesamiento a la que hoy se puede acceder.

Normalmente, muchas soluciones provienen de propuestas que se construyen a la luz de la complejidad tanto algorítmica como lógica y por ello, a pesar de que dichas soluciones son suficientemente satisfactorias, no deben descartarse caminos simples y sencillos para resolver problemas de las Matemáticas. el propósito de este artículo radica en mostrar esa arista simple que aún está vigente, y que es efectiva, en la construcción de programas con miras a resolver un problema computacional.

Este artículo se deriva del proyecto de investigación "Análisis pedagógico, instrumental y conceptual de algunos paradigmas de programación como contenido de la Asignatura Programación I del Programa Ingeniería de Sistemas y Computación" aprobado por la Vicerrectoría de Investigaciones de la Universidad Tecnológica de Pereira y cuyo objetivo general busca cotejar las metodologías, las estrategias, las herramientas y los elementos de juicio que se ponen a consideración de los estudiantes de ingeniería en la búsqueda de soluciones a determinados programas. Este artículo devela una solución que, siendo demasiado simple, cumple con los requerimientos de proceso, eficiencia y solución que implica un buen programa y que está completamente al alcance de los estudiantes en sus primeras fases de formación.

Si se acudiera a los cánones normales de desarrollo de soluciones en la construcción de programas, poco sentido tendría este artículo. Precisamente lo innovador consiste en que se está planteando una solución a un problema heredado de la matemática a través de caminos construidos a partir de una lógica muy simple. La investigación se ha justificado toda vez que son muchos los docentes de ingeniería que, en cumplimiento de las normas ingenieriles, recorren los caminos formales de la solución de problemas descartando posibles soluciones que pueden ser más simples y, sobre todo, mucho más entendibles para los alumnos.

Durante más de veinte años, el autor de este artículo se ha dedicado a buscar caminos por los cuales los estudiantes de ingeniería de sistemas puedan acceder al conocimiento esencial que involucra la programación de computadores de forma que puedan apropiar, asimilar, modificar y optimizar los elementos de juicio que, desde la lógica, intervienen en la construcción de un programa. Esta preocupación investigativa ha llevado al autor a escribir más de nueve libros de programación siempre bajo la óptica de mostrar el camino simple para llegar a lo más complejo, teoría sobre la cual se fundamenta toda una nueva tendencia en los procesos de enseñanza y aprendizaje de la programación de computadores.

La importancia de esta temática radica en que corresponde a temas propios de la Ingeniería de Sistemas y Computación, el programa de formación profesional de mayor auge, crecimiento y acogida en Colombia y los países latinoamericanos. En países fuera de Latinoamérica tienen igual auge los programas de formación profesional equivalentes.

Es de aclarar que esta solución planteada podrá tener, desde las matemáticas, caminos óptimos y mejorados en relación con la solución que aquí se plantea. Acudiendo al aporte de este artículo valdría la pena preguntarse ¿qué tan clara y entendible es la solución? Dado que entender una solución fácilmente simplifica su mantenimiento y su intervención, lo cual en sí mismo es una gran ventaja de un código ajeno.

Se ha acudido a literatura técnica especializada de carácter científico y pedagógico que comprende temas como los conceptos de algoritmo, recursión y primalidad así como se ha adoptado la programación funcional, el concepto de función, la simplicidad y la eficiencia algorítmica a partir de autores autorizados y especializados en estos temas. Para resolver el problema se tomó como base un algoritmo optimizado recursivo para detectar si un número es primo o no que ha sido propuesto por el autor de este artículo en otras publicaciones y que tiene la característica de ser un ejemplo de alta simplicidad y eficiencia en el logro del objetivo. A partir de este algoritmo se ha implementado el conteo, la búsqueda y el despliegue de los números primos.

La técnica de construcción de este algoritmo solución ha sido compartida con estudiantes del programa Ingeniería de Sistemas y Computación de la Universidad Tecnológica de Pereira y su efecto, como estrategia pedagógica, ha sido monitoreado durante seis semestres. La hipótesis sobre la cual se basa este artículo establece que siempre es posible resolver problemas de la matemática, por complejos que sean, acudiendo a caminos algorítmicos sencillos, fáciles de entender y efectivos en el alcance del objetivo.

El artículo presenta los fundamentos teóricos necesarios para entender la solución, presenta luego el código, su explicación y su forma de uso, se revisan los resultados, se presenta una discusión alrededor de dichos resultados y finalmente se llega a unas conclusiones acompañadas y sustentadas en unas referencias bibliográficas.

1.2 Conceptos básicos

Para el abordaje del tema en cuestión, de su enunciado como problema y de su solución, se hizo necesario tener en cuenta algunos conceptos básicos heredados tanto de la matemática como de la programación pues, finalmente, son los dos caminos que confluyen para resolver el problema planteado. Se conoce como Algoritmo el conjunto de pasos secuenciales y ordenados que permiten lograr un objetivo (Agar, 2003). Que estos pasos sean secuenciales significa que se hacen uno detrás de otro y que solo hasta cuando un paso se ha concluido entonces allí es donde se realiza el siguiente.

Que estos pasos sean ordenados significa que sólo se hacen en determinado orden y que si se altera ese orden, se altera automáticamente el resultado final. Tal es el caso del orden en que se ponen las medias y los zapatos, primero se ponen las medias y luego se ponen los zapatos, como ejemplo de la necesidad del orden algorítmico en la construcción de una solución.

Una función es el núcleo de la programación funcional (Brassard, 2009), es una artista útil y efectiva de la programación estructurada y es la base para la construcción de métodos en la programación orientada a objetos. Una función es un fragmento de programa que tiene varias características: es una unidad lógica independiente, tiene un nombre identificador no único a la luz de la programación moderna, puede recibir argumentos (en programación funcional) o parámetros (en programación estructurada o programación orientada a objetos) y puede retornar valores (Cormen, 2009).

La función es a un programa lo que la célula es a los seres vivos o lo que la familia es a la sociedad. La programación funcional privilegia el concepto de función como base para la construcción de soluciones a problemas que pueden ser resueltos por los caminos que la computación provee, especialmente soluciones a problemas matemáticos. El paradigma de programación funcional es una vertiente de la programación de computadores, derivada de la programación declarativa (Freeman, 2002), en la cual se acude a los recursos que provee la matemática, tanto en notación como en concepción y solución, para resolver problemas apoyados en la tecnología.

Una función puede relacionarse con otras funciones a través de los llamados, frente a los cuales se pueden enviar argumentos o parámetros, y a través de los valores de retorno (Gerequeta, 2009). Cuando la relación se hace a través de llamados entonces se habla de una relación de función llamadora a una función invocada; cuando dicha relación se hace a través de los valores de retorno entonces se llama relación de función invocada a función llamadora. Normalmente las funciones solo pueden retornar un valor sin embargo, usando argucias de la misma programación, es posible retornar un vector y con ello se estarían devolviendo varios valores al tiempo.

La recursión es la técnica que permite que una función se llame a sí misma creando un bucle controlado a través del cual se pueden resolver diversos problemas por un camino muy simple, en términos lógicos, y muy entendible, en términos algorítmicos (Krantz, 2010). La recursión es una de las grandes propiedades que se han heredado de las matemáticas en términos de funciones y en la cual

se ha de establecer un valor que determine el fin del bucle de autollamado para que no se entre en un "loop" infinito.

Una función recursiva, en los términos formales, se compone de tres partes: el nombre de la función y sus argumentos, el condicional de parada y la instrucción recursiva. Estas tres partes, siendo de instrucciones unitarias o no, permiten que la concepción de una función recursiva parta de una estructura que facilita su construcción y, por ende, su incorporación dentro de las soluciones algorítmicas.

La simplicidad algorítmica se define como el arte de construir algoritmos solución que sean muy fáciles de entender y muy fáciles de intervenir (Sedgewick, 2011), esto es, fáciles de cambiarle líneas de código dada su simplicidad. La simplicidad es el gran recurso que exige la programación moderna pues en un mundo en donde la mayoría de soluciones algorítmicas acuden a la complejidad, el camino hacia lo sencillo todavía posibilita soluciones alcanzables para los estudiantes de programación de computadores en sus primeros semestres de formación (Trejos, 2011).

Un algoritmo es eficiente siempre y cuando consuma pocos recursos y alcance fácilmente el objetivo propuesto que, en términos sencillos, ha de ser la solución al problema planteado. La eficiencia se mide en la capacidad que tenga el algoritmo de aprovechar todos los recursos que brinda la computación moderna recorriendo caminos altamente digeribles y que capitalicen las altas velocidades de procesamiento de hoy (Trejos, 2007).

Si bien la recursividad, por definición, es una de las técnicas de programación que más recursos consume (Trejos, 2006), su buena utilización como base para soluciones simples la convierte en el camino excelso para las soluciones simples, sencillas y fácilmente digeribles, al punto que llega a convertirse en una solución óptica que toca la frontera de lo eficiente (Trejos, 2010).

La primalidad de un número se define como la característica que tiene un número natural de ser un número primo, es decir, de ser un número que tenga solamente dos divisores exactos: el número 1 y el mismo número (Rey, 2008). La primalidad ha constituido un problema desde los primeros tiempos de las matemáticas pues se ha reconocido desde la antigüedad la existencia de estos números tal como lo comprueban los huesos de Ishango (20.000 A.C.), las fracciones egipcias (3.500 A.C.) y las reflexiones de Euclides en sus libros Elementos en donde plantea incluso una forma simple de encontrarlos y detectarlos (Scott, 2011).

El algoritmo que se ha utilizado para detectar si un número es primo o no es producto de las investigaciones del autor de este artículo y se ha aprovechado para buscar, encontrar y contar los números primos en un rango determinado tipo rango elemental. Un rango elemental es un rango en el cual la cota inferior siempre es el número 1 y la cota superior es un número entero (Van Santen, 2010). La utilización del entorno DrRacket y del lenguaje de programación Scheme facilita no solo la comprensión de la solución algorítmica sino la implementación a través de procesos recursivos así como la notación que es muy próxima a la notación matemática.

2. Metodología

2.1 Descripción

El algoritmo que se planteó como solución al problema de buscar, encontrar y contar los números primos en un rango elemental se presenta a continuación con las normas de documentación que exige la programación moderna y la sintaxis propia del lenguaje de programación Scheme.

```
;; ============================
;; BUSQUEDA Y CONTEO DE NUMEROS PRIMOS EN UN RANGO ELEMENTAL (1,N)
;; PARA UN VALOR DE N LEIDO
;; En este programa se recibe un valor N como
;; argumento y se cuenta la cantidad de números
;; primos en el rango (1,N)
;; ============================
;; Función determina si un valor es múltiplo de otro
(define (divisor a b) ;; Definición de la función
;; Si al dividir el 1o num entre el 2o
```

```
(if ( = (remainder a b) 0);; el residuo es 0
 1        ;; Entonces retorne 1 (true)
 0        ;; Sino retorne 0 (false)
))        ;; Fin de la función
;; Función que cuenta la cantidad de divisores de un
;; número - rango (2, sqrt n)
(define (cuentadivisores num div) ;; Def. función
;; Si se ha llegó a la raíz cuadrada del num a analizar
 (if (>= div (floor ( / (sqrt num) 1)))
;; devuelva lo que retorne la función divisor
 (divisor num (floor ( / (sqrt num) 1)))
;; Sino sume lo que retorne la función divisor
 ( + (divisor num div)
;; mas lo que retorne esta misma función
 (cuentadivisores num (+ div 1)))
 ))                              ;; Fin de la función
;; Función que retorna si un valor es primo
(define (esprimo n)              ;; Definición de la función
;; Si n no tiene divisores en el rango (2, raiz cuad n)
(if (= (cuentadivisores n 2) 0)
 1        ;; Entonces retorne 1 (Verdadero)
 0        ;; Sino retorne 0 (Falso)
))        ;; Fin de la función
;; Función que muestra los números primos
(define (mostrar n)              ;; Definición de la función
 (if (= n 4) ;; Si el valor analizado es 4
 (begin ;; muestre avisos pertinentes
 (display "3,2")
 (newline)
 (display "Total Nums Primos....")
 0)
 (begin   ;; Sino
 (if (= (esprimo n) 1)       ;; Si el valor es primo
 (begin ;; muéstrelo en pantalla
 (display n)
 (display ",")
 1 ;; retorne un valor verdadero
 ) ;; Fin de la parte Verdadera
 0 ;; Sino retorne un valor Falso
 )))) ;; Fin de la función
;; Función que cuenta los primos de un rango
(define (cuentaprimos n) ;; Definición de la función
 (if (= n 2) ;; Si el valor es 2
 1        ;; cuéntelo como número primo
;; sume lo que retorne la función mostrar
 (+ (mostrar n)
;; con lo que retorne esta misma función
 (cuentaprimos (- n 1)))
 )) ;; Fin de la función
;; Función que inicia el proceso de conteo
(define (cp n) ;; Definición de la función
;; Se recibe tope superior n del rango elemental (1,n)
 (+ 1 ;; Sume el valor 1
 (cuentaprimos n))
;; mas lo que retorne la función cuentaprimos
 )        ;; Fin de la función
```

2.2 Aplicación

El esquema funcional de este programa permite revisar el estado de conexión entre las funciones, los datos que se transfieren de una a otra y las relaciones que permiten ver el programa completo como un todo. Este esquema funcional se presenta en la Figura 1. Este devela que el llamado al programa se hace a través de la función cp a la cual se le envía como argumento el límite superior del rango elemental (1,n). La función cp llama a la función cuentaprimos a la cual le envía como argumento el valor de n (tope superior del rango elemental). La función cuentaprimos llama a la función mostrar a la cual le envía el valor de n y ésta a su vez llama a la función esprimo que también recibe como argumento el valor de n.

La función esprimo llama a la función recursiva cuentadivisores a la cual le envía dos argumentos: num y div. El argumento num contiene el valor al cual se le va a evaluar su primalidad, el argumento div servirá como base para evaluar los posibles divisores del valor num en el rango (2, raíz cuadrada de num). La función cuentadivisores llama a la función divisor a la que le envía los argumentos num y div y con eso se determina si cada uno de los valores de div es un divisor exacto del valor num.

La función divisor retorna un 1 o 0 que equivalen a un Verdadero o un Falso dependiendo de si el argumento div es un divisor exacto del argumento div. La función cuentadivisores retorna el valor de la suma de los divisores exactos de un número. La función mostrar retorna un valor 0 en caso de que el valor recibido no sea primo o muestra el valor

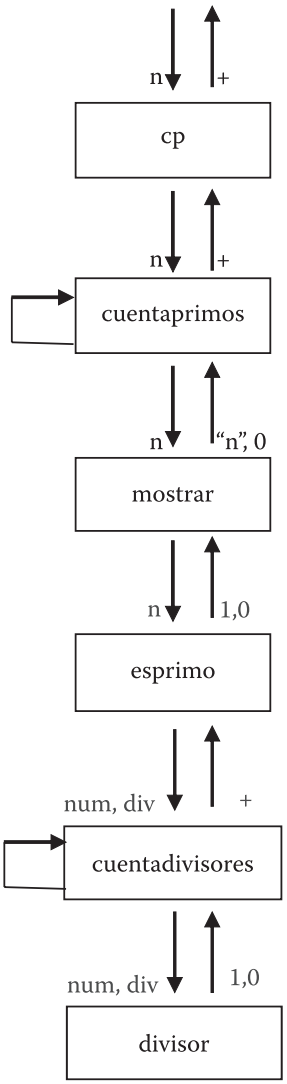

Figura 1. Esquema Funcional

en caso de que si lo sea. La función cuentaprimos retorna la cantidad de números primos que existen en el rango elemental (1, n) y la función cp despliega el resultado final siendo la que inicia y finaliza la secuencia lógica del llamado a las funciones.

3. Resultados y discusión

Al ejecutar el programa, los resultados obtenidos con valores para n iguales a 10, 100 y 1000 son los siguientes:

> (cp 10)
7,5,3,2
Total Nums Primos....4

En esta petición, se solicita que se busquen, se listen y se cuenten la cantidad de números primos comprendidos en el rango elemental (1, 10). El programa efectivamente muestra los números primos comprendidos en dicho rango y los totaliza. Es de recordar que en este rango solamente hay 4 números primos debido a que el número 1 no se considera como tal.

> (cp 100)
97,89,83,79,73,71,67,61,59,53,47,43,41,37,31,29,23,19,17,13,11,7,5,3,2
Total Nums Primos....25

En el rango elemental (1,100) la cantidad de números primos es 25 tal como se muestra en el resultado junto con los números primos como tales. El algoritmo hace una búsqueda eficiente revisando los posibles divisores de cada número buscándolos en el rango (2, raíz cuad de *n*) siendo n el valor evaluado.

> (cp 1000)
997,991,983,977,971,967,953,947,941,937,929,919,911,907,887,883,881,877,863,859,857,853,839,829,827,823,821,811,809,797,787,773,769,761,757,751,743,739,733,727,719,709,701,691,683,677,673,661,659,653,647,643,641,631,619,617,613,607,601,599,593,587,577,571,569,563,557,547,541,523,521,509,503,499,491,487,479,467,463,461,457,449,443,439,433,431,421,419,409,401,397,389,383,379,373,367,359,353,349,347,337,331,317,313,311,307,293,283,281,277,271,269,263,257,251,241,239,233,229,227,223,211,199,197,193,191,181,179,173,167,163,157,151,149,139,137,131,127,113,109,107,103,101,97,89,83,79,73,71,67,61,59,53,47,43,41,37,31,29,23,19,17,13,11,7,5,3,2
Total Nums Primos....168

En el caso del rango elemental (1, 1000), la cantidad de números primos es 168 y son los que aparecen en el resultado del programa. Una de las razones por las cuales se acudió al lenguaje Scheme como herramienta de programación es porque permite la manipulación simple de números muy grandes sin que el programador se tenga que preocupar por rangos de almacenamiento y tipos de datos.

De esta forma se puede invocar la función cp con el número 100000 originando un listado bastante amplio de números primos cuyo resultado total arroja 9592 primos en total. De la misma forma se puede invocar la función cp enviándole como argumento números mucho más grandes. Si el llamado fuera > *(cp 1000000) e*l resultado sería un listado bastante amplio reportando 78498 como el total de los números primos en el rango (1,1000000).

El rango de evaluación de la primalidad de cada uno de los números del rango elemental (1,n) que corresponde al conjunto de números comprendidos entre 2 y la raíz cuadrada del número a evaluar, parte del principio de que si un número entero no tiene divisores exactos en este rango es porque el número es primo. En caso de que existan divisores exactos por fuera de este rango (es decir, mayores que la raíz cuadrada del número a evaluar) se asegura que dicho divisor exacto tiene, a su vez, otro divisor exacto en el rango establecido. Este rango acelera significativamente el proceso de búsqueda de los números primos.

Se ha utilizado el concepto de rango elemental (1,n) debido a que es el rango en el cual normalmente se evalúa la cantidad de números primos que existen, evaluación que se conoce como densidad de los números primos y que constituye el tema central de otro artículo del mismo autor. El algoritmo presentado pone a disposición de los lectores una forma muy sencilla y eficiente de resolver el problema de la búsqueda y conteo de los números primos.

El concepto de optimización busca siempre aprovechar al máximo los recursos que la computación provee para que se encuentre un camino llano a través del cual se puedan implementar soluciones a problemas que, en este caso, son heredados de las matemáticas. La optimización se palpa en funciones como la función cuentadivisores en la cual la búsqueda de los divisores exactos de un número n se hace en el rango (2, raíz cuadrada de n) y no en el rango natural que sería el rango (2,n). De esta manera el factor de eficiencia es alto pues el rango de evaluación se acorta de una manera bastante destacable.

Al contar con un rango de evaluación optimizado, el proceso de búsqueda también se optimiza puesto que al recorrer todo el conjunto de números naturales de un rango elemental los procesos de simplifican y se mejoran en su velocidad de proceso por las razones explicadas en el párrafo anterior.

Una de las fortalezas de este algoritmo es que se cuenta con dos funciones recursivas estratégicamente ubicadas para que el proceso no solo sea eficiente sino veloz tal como se comprueba al momento de ejecutar el programa. No se desconoce que la recursión es una de las técnicas más costosas en términos computacionales pero considerando el estado actual de las tecnologías modernas, este concepto está empezando a ser revaluado dadas las altas velocidades de procesamiento y cálculo que esas mismas tecnologías proveen.

La utilidad de este programa consiste en que se presenta una solución efectiva, simple y eficaz para buscar, encontrar, listar y contar el conjunto de números primos que se encuentran en un rango elemental definido (1,n). Sobre la búsqueda y evaluación de la primalidad de los números son muchos los algoritmos que pueden encontrarse pero ésta es una solución al alcance de los estudiantes de programación en sus primeras fases de formación y, por tanto, su eficacia no solo en el proceso sino en lo formativo supera a muchos otras soluciones.

La densidad de los números primos se define como la cuantificación de la presencia de los números primos en un rango determinado que, normalmente y para efectos matemáticos, hace referencia a un rango elemental, es decir, un rango cuya cota inferior es el número 1 y cuya cota superior es un número entero natural definido por el usuario. Es de anotar que usualmente la densidad de los números primos se calcula con cotas superiores múltiplos de 10 y que sean inicio de un nuevo orden numérico (10, 100, 1000, etc).

En la función cuentadivisores el retorno se hace dividiendo entre el valor 1 debido a que se necesita tomar solo la parte entera de la raíz cuadrada del número a evaluar dado que si se tomara la raíz cuadrada completa jamás se alcanzaría como tope

superior pues ésta podría dar con decimales y, como el rango se basa en números enteros, sería inalcanzable.

En la función es primo se establece como condicional la definición de la cantidad de divisores exactos igual a 0 debido a que en el rango (2, raíz cuadrada de n) un número n tiene 0 divisores exactos. Si el rango de evaluación fuera (1, n) entonces el condicional debería preguntar por una cantidad de divisores exactos igual a 2, como lo establece la definición del número primo, que serían el valor 1 y el mismo número a evaluar.

La función mostrar cuenta con una instrucción "forzada" para el caso de que el número a evaluar sea menor que 4 pues con los valores 2 y 3 la función esprimo presenta una deficiencia dado que la raíz cuadrada de estos números es menor que el tope inferior del rango (o sea el número 2) y entonces el rango, que se supone va de un valor inferior a uno superior, estaría estableciéndose de un número inferior a uno más inferior lo cual es ilógico desde lo conceptual matemático.

La función cuentaprimos, que se encarga de contar cuántos números primos existen en el rango elemental (1,n), evalúa los números en el rango (n,1) que para los efectos matemáticos y conceptuales de este proceso resulta ser exactamente el mismo. En ese mismo sentido al función cp adiciona un valor 1 a la suma que resultado de contar los números primos como mecanismo de compensación de las consideraciones que se hacen sobre los números enteros 2 y 3.

4. Conclusiones

El algoritmo presentado es un ejemplo de que siempre podrá existir un camino fácil y eficiente para encontrar soluciones simples a problemas que históricamente han sido definidos como complejos más aún si el programador se apoya en los fundamentos matemáticos que perfilan, definen y resuelven el problema

Un algoritmo simple y sencillo siempre es mejor, como solución, que un algoritmo complejo de entender toda vez que la eficiencia de ambos sea aproximada

Es posible encontrar un camino expedido entre los problemas que proveen las matemáticas y las soluciones que posibilitan la programación y las tecnologías computacionales modernas

Aprovechando las tecnologías computacionales modernas es posible hacer un poco más flexible el concepto de complejidad y acudir a soluciones muy simples, vistas desde la lógica de los algoritmos, de manera que se capitalice al máximo las características de proceso y rendimiento de los computadores de hoy

Un algoritmo simple siempre será mucho más entendible que un algoritmo complejo y eso conlleva a que cuesta menos entender, asimilar, mantener y mejorar un algoritmo simple así sea más costoso en términos computacionales. Son elementos de juicio a evaluar de acuerdo al caso concreto que se deba resolver

En las primeras fases de un proceso de formación de ingenieros es muy importante que los estudiantes conozcan tanto el concepto de complejidad tal como la teoría de la computabilidad la define así como la lógica que, en esas primeras fases puede ser alcanzable por los mismos estudiantes a partir de resolver problemas de la matemática

Una de las ventajas de la herramienta seleccionadas es que aleja al programador del problema de la capacidad de almacenamiento y los tipos de datos, problema que subyace a otros paradigmas de programación y sus herramientas asociadas, y que permite ocuparse de la lógica de solución más que de las minucias del lenguaje

La programación funcional constituye una excelente alternativa para la resolución de problemas derivados de la matemática pues su notación sintáctica es profundamente cercana a la notación científica

Referencias

1. Agar, J. *The Government Machine. Massachusetts Institute of Technology*. USA. 2003, pp. 102.
2. Brassard, G. Bradley, P. *Fundamentos de Algoritmia*. Prentice Hall. Montreal. 2009, pp. 42.

3. Cormen, T. Leiserson, C. *Introd. to Algorithms*. 3a Ed. MIT Press. USA. 2009, pp. 63.
4. Freeman, W. *El lenguaje de las matemáticas*. Manon Troppo Editorial. Barcelona. 2002, pp. 19.
5. Gerequeta, R. Vallecillo, A. *Técnicas de Diseño de Algoritmos*. Servicio de Publicaciones de la Universidad de Málaga. España. 2007, pp. 100.
6. Krantz, S. *And Episodic History of Mathematics*. MAA TextBooks, USA. 2010, pp. 22.
7. Sedgewick, R. Wayne, K. *Algorithms. 4a Ed. Addison Wesley*. Princeton Univ.. USA. 2011, pp. 33.
8. Trejos, O. *Algoritmo de optimización para la detección de un número primo basado en programación funcional utilizando DrScheme*. Scientia et Technica. Año XVII No. 47. 2011, Pp. 276.
9. Trejos, O. *Algoritmos Problemas Básicos*. Editorial Papiro. Pereira, Colombia. 2007, pp. 51.
10. Trejos, O. *Determinación simple de un número primo aplicando programación funcional a través de DrScheme*. Revista *Scientia et Technica*. Año XVI No. 45. 2010, p. 155.
11. Trejos, O. *Fundamentos de Programación*. Editorial Papiro. Pereira, Colombia. 2006, pp. 72.
12. Rey, P. *Historia de la Matemática*. Gedisa Editorial. Barcelona. España. 2008, pp. 26.
13. Scott, F. *Principles of Programming*. Creative Common Share. San Francisco. USA. 2011, pp. 56.
14. Van Santen. D. *2030 Technology that will change the World*. Oxford Univ. Press. 2010, pp. 197.