Ciencia e Ingeniería Neogranadina
ISSN: 0124-8170
ISSN: 1909-7735
Universidad Militar Nueva Granada

Trejos Buriticá, Omar Iván; Muñoz Guerrero, Luis Eduardo

Dos algoritmos para hallar números primos gemelos en un rango específico usando programación funcional*

Ciencia e Ingeniería Neogranadina, vol. 32, núm. 2, 2022, Julio-Diciembre, pp. 131-144
Universidad Militar Nueva Granada

DOI: https://doi.org/10.18359/rcin.6252

Disponible en: https://www.redalyc.org/articulo.oa?id=91174988009

Ciencia e Ingeniería Neogranadina

2022
Vol. 32(2)
julio-diciembre 2022 ■ ISSN: 0124-8170 · ISSN-e: 1909-7735 ■ pp. 131-144

Editorial Neogranadina

DOI: https://doi.org/10.18359/rcin.6252

Dos algoritmos para hallar números primos gemelos en un rango específico usando programación funcional*

Omar Iván Trejos Buriticá[a] ■ Luis Eduardo Muñoz Guerrero[b]

Resumen: un gran reto que tienen los ingenieros docentes consiste en encontrar aplicaciones en donde la ingeniería específica sea solución a problemas de otras áreas de tal forma que se maximice su potencial conceptual e instrumental. Tal es el caso de la programación de computadores que, como parte de la ingeniería de sistemas, permite resolver problemas gracias a las grandes capacidades tecnológicas para aprovechar lo que el conocimiento humano ha desarrollado. En este artículo se presenta una solución algorítmica al problema de hallar números primos gemelos en un rango especificado por el usuario. Se acudió al paradigma de programación funcional y a conceptos como función, recursividad y ciclos de manera que facilitaron la resolución del problema. Desde esta perspectiva se obtuvieron resultados confiables que cumplen tanto con la definición de los números primos gemelos, como con los principios de eficiencia algorítmica. Se concluye que en la medida en que se puedan compartir con los estudiantes de programación aplicaciones que resuelvan problemas de otras áreas, se harán mucho más entendibles y aplicables conceptos propios de un paradigma y se podrá encontrar el sentido del aprovechamiento de la tecnología en favor del ser humano.

Palabras clave: algoritmo; aprendizaje; función; número primo; paradigma; programación; recursividad

Recibido: 10/05/2022

Aceptado: 05/08/2022

Disponible en línea: 30/12/2022.

Cómo citar: Trejos Buriticá, O. I., & Muñoz Guerrero, L. E. (2022). Dos algoritmos para hallar números primos gemelos en un rango específico usando programación funcional. *Ciencia E Ingeniería Neogranadina, 32*(2), 131-144. https://doi.org/10.18359/rcin.6252

* Artículo de investigación científica y tecnológica.

a Ph. D. en Ciencias de la Educación de la Universidad Tecnológica de Pereira, Pereira, Risaralda, Colombia. Correo electrónico: omartrejos@utp.edu.co
ORCID: https://orcid.org/0000-0002-3751-6014

b Ph. D. en Ciencias de la Educación de la Universidad Tecnológica de Pereira, Pereira, Risaralda, Colombia. Correo electrónico: lemunozg@utp.edu.co
ORCID: http://orcid.org/0000-0002-9414-6187

Two algorithms to find twin prime numbers in a specified range using functional programming

Abstract: a great challenge for teaching engineers is to find applications where specific engineering is a solution to problems in other areas in such a way as to take advantage of its conceptual and instrumental potential. Such is the case of computer programming, which, as part of systems engineering, allows solving problems thanks to today's great technological capabilities that allow taking advantage of what human knowledge has developed. This paper presents an algorithmic solution to the problem of finding twin prime numbers in a range specified by the user. The functional programming paradigm and concepts such as function, recursion, and cycles were used to solve the problem. From this perspective, reliable results were obtained that comply both with the definition of twin prime numbers and with the principles of algorithmic efficiency. It is concluded that to the extent that applications that solve problems in other areas can be shared with programming students, the concepts of a paradigm will become much more understandable and applicable, and the sense of taking advantage of technology in favor of the human being will be found.

Keywords: algorithm; learning; function; prime number; paradigm; programming; recursion

Introducción

La temática que se va a tratar aborda la necesidad permanente de la docencia en ingeniería de encontrar aplicaciones en las que el conocimiento disciplinar se convierta en solución de problemas de otras áreas, de modo tal que estas sean proveedoras de situaciones problémicas y aquellas sean caminos para resolverlas [1].

Todo esto dentro del contexto de la formación de ingenieros en donde con frecuencia se encuentran teorías, modelos y metodologías que, llevadas a una situación específica, son solución, pero no siempre se articulan con enunciados o planteamientos que tal saber resuelve. Para el estudiante será siempre muy enriquecedora la información que puede aplicarse como solución a diferentes problemas, pues allí se convierte en conocimiento y su capacidad para encontrar las aplicaciones, a la postre, permitirá que adquiera las competencias profesionales necesarias para fortalecer su perfil profesional y laboral [2].

El objetivo de este artículo radica en solventar una situación heredada de las matemáticas y plantear dos posibles soluciones desde la perspectiva del paradigma de la programación funcional con una componente de optimización para acelerar los procesos de búsqueda. Se ha adoptado la estructura IMRYD [3] estándar para presentar este escrito y para que después de la introducción se exponga la metodología utilizada, se planteen los resultados y sobre ellos se desarrolle un análisis y una discusión que confluyan en unas conclusiones acompañándolos de las debidas referencias.

El tema tiene que ver con la búsqueda de números primos gemelos, que se explicarán posteriormente, y con el planteamiento de una solución recursiva basada en la construcción de funciones enlazadas en la optimización de los ciclos y en la definición de un rango específico para hacerlo. El problema de investigación apunta a tres elementos: 1) la articulación con el aprendizaje de la programación, cualquiera que fuere el paradigma que se quiera adoptar; 2) la concreción de un problema específico de un área de las ciencias básicas que puede resolverse mediante las capacidades de procesamiento, almacenamiento y respuesta de los sistemas computacionales de hoy y 3) el planteamiento de una solución optimizada basada en el paradigma de programación funcional a la luz de conceptos como función, recursión y condicionamiento, que posibiliten una socialización didáctica dentro de un curso de programación en el contexto de la ingeniería de sistemas.

Para abordar el tema se hizo necesario acudir a algunas temáticas que incluyen no solamente lo puramente disciplinar como el paradigma de programación funcional o lo temático como la definición de números primos y los planteamientos sobre su presencia e infinitud en la recta de los números naturales, sino que también fue necesario abordar las teorías del aprendizaje invisible, del conectivismo y del pensamiento computacional. Con ellas se vislumbraron tres necesidades: 1) definir lo que verdaderamente se aprende en cualquiera de los tres contextos, tal como lo plantea el aprendizaje invisible; 2) fortalecer y crear conocimiento a partir de redes de nodos de saber, tal como sucede en la actualidad y como lo plantea el conectivismo y 3) articularse con el mundo de hoy a partir del pensamiento crítico, la resolución de problemas, la utilización de las tecnologías de la información y las comunicaciones y el desarrollo de soluciones algorítmicas, según lo presenta el pensamiento computacional.

El estudio se justifica debido a que cada vez se hace más necesario convertir la información especializada en conocimiento teniendo en cuenta que la información puede definirse como un conjunto de datos útiles, que son entendibles y que tienen sentido dentro de un contexto, mientras que el conocimiento se define como la aplicación de una información a la solución de determinados problemas en cualquier área del saber, estén conexos con ella o no [4], tal como lo indica la formación por competencias.

Esta necesidad es cada vez mayor debido a que los estudiantes de ingeniería, por su naturaleza como nativos digitales y sabiendo que

cada vez es ***más*** normal para ellos el lenguaje de la tecnología [5], requieren visualizar aplicaciones de su conocimiento temático, enfrentarse a problemas emergentes y resolverlos con el saber disciplinar para que puedan ver en la escena de la academia la utilidad del cúmulo de conocimientos que están adquiriendo [6]. Este artículo es uno de los productos derivados del proyecto código 6-19-11 "Desarrollo de un modelo de enseñanza y aprendizaje que transversalice el conocimiento derivado de las ciencias básicas aprovechando la programación de computadores en ingeniería de sistemas basado en Brain Based Learning y Pensamiento Computacional", tramitado y autorizado por la Vicerrectoría de Investigaciones, Innovación y Extensión de la Universidad Tecnológica de Pereira.

Si bien debe aceptarse que son muchas las aplicaciones de la programación en diferentes áreas, no es usual que tal situación tenga un propósito tan didáctico y académico como el que se presenta en este artículo. En primer lugar, debe destacarse que se pretende posibilitar a los estudiantes una visión práctica de la programación a partir de situaciones problema formuladas desde otras áreas del conocimiento. Esto no solo fortalece el sentido mismo de la programación, como conjunto de herramientas que permiten aprovechar los recursos que proveen las tecnologías modernas desde sus capacidades de almacenamiento, procesamiento y respuesta, sino que, además, brinda una visión más cercana de otras áreas que, siendo de gran importancia en la ingeniería, pocas veces se pueden percibir tan íntimamente conexas con la programación.

En segundo lugar, no se trata de presentar solamente una aplicación más de la programación sino de mostrarla como un recurso didáctico que facilita el aprendizaje y la aplicación de un paradigma específico a la luz de la resolución concreta de un problema heredado, en este caso, de las matemáticas que requieren de todo el potencial tecnológico moderno para resolver problemas, que se encuentran muy por encima de las capacidades posibles del ser humano.

En tercer lugar, la aplicación de la programación puede buscar dos propósitos: visualizarla como un camino para resolver problemas de la sociedad [7] y visualizarla como la senda por la cual se pueden solucionar situaciones y enunciados de otras áreas del conocimiento que si bien los plantean y, por momentos, los solventan teóricamente [8], no siempre facilitan la concreción de la solución, excepto desde un planteamiento general, válido, pero etéreo, en términos prácticos como sucede con muchas de las soluciones que se proponen desde áreas tan importantes como las matemáticas. Por otra parte, compartir con los estudiantes soluciones que requieran utilizar conceptos clave en el paradigma funcional, caso puntual de este artículo, como el término función, la recursividad y los condicionales que dan fin a un proceso repetitivo, hacen que el corpus de conocimiento alrededor de la misma programación se articule entre sí y con otras áreas [9].

El estudio se realizó como una forma de verificar la necesidad de contar con enunciados que realcen el valor de la programación y su relación con otras áreas, que también se recalcan a partir de la resolución de problemas. La hipótesis de esta investigación parte de que se puede posibilitar un aprendizaje más expedito de la programación y un avance en la ingeniería de sistemas, toda vez que se plantean problemas de las áreas de las ciencias básicas para resolverlos con los elementos de juicio que provee un determinado paradigma, sus herramientas y sus instrucciones. El contenido del presente artículo es la respuesta a la hipótesis planteada y la metodología adoptada permitió la verificación propuesta en el marco de los lineamientos de la investigación cualitativa.

Sobre esta base se plantearon los siguientes objetivos específicos: 1) formular la situación problema relacionada con el desarrollo de un algoritmo para encontrar números primos gemelos en un rango específico, usando programación funcional; 2) resolver el enunciado a partir del paradigma de programación funcional; 3) optimizar la solución desde la perspectiva del concepto de función, recursividad de nivel III y evaluación minuciosa de los condicionales de parada; 4) socializar con los estudiantes

de uno de los cursos paralelos de programación que se condujeron durante cada semestre de estudio y 5) recolectar las opiniones cualitativas con respecto al problema planteado, la solución presentada, la lógica utilizada y la percepción de los alumnos sobre su aprendizaje de la programación. Es de anotar que los puntos 4) y 5) de estos objetivos específicos se excluyen de este artículo por razones de extensión, pero forman parte del contenido de otro artículo que presenta los resultados apropiadamente para su análisis.

El fundamento teórico de la presente investigación acude a tres áreas: las matemáticas, la programación de computadores y algunas teorías y modelos de aprendizaje y de pensamiento. Las matemáticas proveen la definición del problema, sus características y los conceptos que se involucran; la programación de computadores provee caminos de solución para resolver el problema planteado y las teorías y modelos de aprendizaje y de pensamiento facilitan la comprensión, en cuanto a la manera como este conocimiento se puede llevar al aula en un entorno de formación en ingeniería, que permita la asimilación clara de los objetivos por alcanzar en relación con el problema y la forma como se ha resuelto por medio del paradigma de programación.

La conjetura de los números primos gemelos determina que siendo a y b dos números naturales se reconocerán como primos gemelos si cumplen con las siguientes condiciones: 1) cada uno es un número primo, es decir, solamente tiene como divisores exactos al número 1 y a sí mismo; 2) entre ellos existe la diferencia de 2 (primer número primo) lo cual implica que, de una pareja de números primos gemelos, el número primo mayor será igual al número primo menor sumándole el valor 2 [10]. Ejemplos de estos números son el 17 y el 19 así como los números 71 y 73. Según la conjetura de los números primos gemelos, existen infinitas parejas de ellos [11] que, por ser una conjetura, aún no se ha demostrado. Debe aclararse que el objetivo de este artículo no radica en su demostración, sino en el planteamiento de un algoritmo basado en programación funcional que permita encontrarlos dentro de un rango definido por el usuario.

Esta conjetura ha sido investigada por diferentes matemáticos expertos en el área de la teoría de números. Desde la matemática se cree que es cierta y las aproximaciones que se han hecho a su demostración, sin lograrlo, se han basado mucho más en evidencias numéricas y en métodos puramente heurísticos que parten de la forma como se distribuyen los números primos, según sus probabilidades. Precisamente, lo que se busca en este artículo es ofrecer una solución algorítmica que no es una demostración en sí misma, pero que sirve como herramienta para sustentar la que posteriormente puede llegar a ser una demostración.

Polignac (1826-1863), matemático francés, planteó a manera de conjetura, que en relación con cualquier número natural siempre existirán infinitos pares ordenados de números que cumplen con ser gemelos, entre los cuales la diferencia siempre será 2xk siendo k un valor igualmente entero [12]. Precisamente la conjetura de los números primos gemelos, tal como se conoce de manera formal, es el equivalente al planteamiento de Polignac cuando la constante k es igual a 1. Debe aclararse que algunos historiadores atribuyen esta conjetura a Legendre (1752-1833), antecesor de Polignac.

El matemático húngaro Paul Erdős (1913-1996) planteó que para una constante c<1 existen infinitos números primos p, de tal forma que $q - p < ln\ p$ en donde q corresponde al número primo que está a continuación de p en la recta de los números naturales y ln es el logaritmo natural de p. Hacia 1986 el matemático alemán Maier (1953) demostró que esta constante podía ser menor que 0,25 o sea que c<0,25 [13]. En relación con este planteamiento, el matemático estadounidense Daniel Alan Goldston (1954) junto con el húngaro János Pintz (1950) y el turco Cem Yalçın Yıldırım (1961) probaron en 2005 que ese resultado es válido si se trata de una constante c que sea mayor que 0, es decir, c>0.

Por su parte, el matemático chino Jingrun Chen (1933-1996) probó que efectivamente

existen infinitos números primos gemelos tal como lo plantea la definición, es decir, pares de números primos que tienen una diferencia de 2, demostración que logró a partir de la teoría de cribas que consiste en un conjunto de metodologías, técnicas y formas que posibilitan el conteo, o mínimamente la estimación, del tamaño de un conjunto específico de números enteros que cumplen con determinada condición [14]. De esta forma, Jingrun Chen pudo abordar la demostración de los números primos gemelos y, de una forma similar por su naturaleza temática, pudo hacer lo propio con la conjetura de Goldbach, según la cual todo número par mayor que 2 puede escribirse como la suma de dos números primos.

Es de anotar que esta conjetura fue originalmente planteada por el filósofo y matemático francés René Descartes (1596-1650) y posteriormente por el matemático prusiano Christian Goldbach (1690-1764), en 1742, conjeturó al matemático suizo Leonhard Paul Euler (1707-1783), cuando afirmó que "cualquier número entero que sea mayor que el valor 5 puede ser escrito como el resultado de sumar tres números primos" [15]. Debe anotarse que existe una forma, matemáticamente impresionante, de generalizar la conjetura de los números primos gemelos, planteada por los matemáticos británicos Godfrey Harold Hardy (1877-1947) y John Edensor Littlewood (1885-1977). Vale la pena recordar que la primera persona en llamarlos "números primos gemelos" fue el matemático alemán Paul Gustav Samuel Stäckel (1862-1919).

Para los efectos de las soluciones algorítmicas que, según la definición, dos números primos gemelos son aquellos números primos cuya diferencia entre el mayor y el menor es 2, el número intermedio es múltiplo de 6. El matemático noruego Viggo Brun (1885-1978) demostró que cuando se suman los inversos (el inverso de un valor x es x^{-1} dado que el producto de los dos números es igual a 1) de los números primos gemelos, este resultado es convergente a un número específico, que se conoce como constante de Brun que se denota como B2. El estadounidense Thomas Ray Nicely (1943-2019), profesor de la Universidad de Lynchburg (Virginia, USA) y quien encontró el error de división en el procesador Pentium de Intel, estimó la constante de Brun en 1,902160578. Hacia 2002 los matemáticos Sebah y Demichel calcularon la constante de Brun hasta 1016 en 1,902160583104 [16].

En la actualidad, aprovechando las potencialidades de la computación, se han calculado número primos gemelos con gran cantidad de dígitos. Los más grandes detectados hasta el momento son el par conformado por (2996863034895 · 21290000 – 1, 2996863034895 · 21290000 + 1), que están compuestos por 388342 dígitos y fueron encontrados en septiembre de 2016. Hasta nuestros días se han hallado 808.675.888.577.436 pares de números primos gemelos que son menores que 10 [18]. En otro artículo se presentará un algoritmo acelerado para calcular la constante de Brun usando programación funcional.

En cuanto al soporte que se requiere de la programación debe decirse que la lógica de programación es el conjunto de reglas que permite convertir un problema humano en un enunciado computacional, es decir, que puede resolverse con la participación del computador a partir de las herramientas, conceptos y facilidades que implica dicha transformación [17]. La lógica de programación posibilita adoptar el concepto de algoritmo como conjunto de pasos ordenados y secuenciales para alcanzar un objetivo que luego será transformado en instrucciones que, de acuerdo con un paradigma de programación, facilitará la implementación de ese algoritmo en los términos que permite un lenguaje de programación.

La programación funcional es un enfoque de programación que se basa en el concepto de función como una unidad básica que tiene un nombre identificativo, puede recibir argumentos, tiene un cuerpo de instrucciones, puede retornar valores y puede ser invocada y cumple con un pequeño objetivo enmarcado dentro de un objetivo mucho más amplio. La conjunción o enlace de varias funciones simples favorece la construcción de soluciones que resuelven problemas de

mayor complejidad desde una perspectiva que posibilita simplificar grandes objetivos, reutilizar el código y realizar pruebas de forma sencilla y totalmente confiable [18]. La programación funcional, por tanto, es un paradigma de programación que puede definirse como una forma de mirar y adoptar el camino de solución para un problema que puede resolverse a partir de las potencialidades de la tecnología actual. En su definición más práctica, desde la programación funcional el mundo se puede interpretar a partir del uso y utilidad de las cosas que lo conforman.

Uno de los conceptos más relevantes en la programación funcional es la recursividad que en términos retóricos significa la definición en la cual se utiliza el término definido. En términos matemáticos es la definición de una función que aprovecha las características de dicha función en su representación en el plano cartesiano y en términos de programación consiste en la posibilidad que ofrecen los lenguajes de programación actuales de que una función se pueda invocar a sí misma [19]. La estructura de una función recursiva, teniendo en cuenta que este concepto es la base de las soluciones que se plantean en el presente artículo, está compuesta por tres partes: 1) la definición de la función; 2) la condición de parada y 3) el llamado recursivo. Dentro de este contexto se define una función recursiva de III nivel como aquella que para alcanzar su objetivo debe apoyarse en otra función igualmente recursiva [20].

Desde la perspectiva del aprendizaje, la presente investigación ha tomado como referentes a:

1) La teoría del aprendizaje invisible [21] según la cual la inmersión de un estudiante en cualquiera de los tres entornos de aprendizaje (del aula, institucional y extrainstitucional) le permiten acceder y asimilar aprendizajes que no están sistematizados ni incluidos en los currículos formales, pero que tienen mucha influencia en su concepción y visión del mundo que les rodea. Esta teoría le aporta a la presente investigación la visión de poder relacionar los conocimientos propios de la programación con problemas de otras áreas en donde las potencialidades de la computación de hoy permiten resolverlos apropiadamente.
2) La teoría del conectivismo [22] que destaca que el conocimiento ya no está dentro del ser humano sino por fuera de él y que este se dinamiza a partir de la interacción entre nodos, que pertenecen a una misma red, tal como sucede con las redes sociales en las que el conocimiento se está cuestionando y revisando permanentemente. Desde esta teoría se plantean dos soluciones posibles a un mismo problema de forma que puedan compararse para sus respectivos cuestionamientos y, de esta forma, se puedan socializar con los estudiantes por medios virtuales, como lo han exigido las condiciones de distanciamiento producto de la COVID-19.
3) La teoría del pensamiento computacional [23] que exige que el ser humano, en los tiempos de hoy y para articularse con las características del mundo tecnológico actual, deba apropiarse de las cuatro componentes este tipo de pensamiento:
   a) El pensamiento crítico o la posibilidad de abordar las situaciones con una mirada que devele las ventajas y desventajas que pudiere encarnar.
   b) La resolución de problemas, en razón a que en la sociedad actual emergen no solo nuevos problemas sino escenarios que no se habían previsto y que esto sucede repentinamente, como ha pasado con la COVID-19.
   c) El aprovechamiento de las TIC, a partir de la gran penetración de todas las herramientas, facilidades, servicios y conceptos que tienden a posibilitar el acceso a la información y al conocimiento, entendiendo que el primero está conformado por un conjunto de datos organizados y entendibles, mientras que el segundo lleva a un estadio superior a la información, cuando se usa bien, sea en contextos relacionados tanto directa como indirectamente y el desarrollo de soluciones a los problemas desde una perspectiva algorítmica que posibilita su implementación posterior en sistemas computacionales.

d) El pensamiento computacional posibilita una visión crítica proactiva de las soluciones planteadas a un mismo problema; también facilita su resolución desde una perspectiva algorítmica y provee herramientas para su implementación a la luz de un paradigma de programación.

Metodología

En esta parte se explicará la forma como se plantearon los dos algoritmos que conforman el corpus del presente artículo tanto desde su perspectiva lógica algorítmica como desde lo funcional. Para ello debe tenerse en cuenta que el problema (enunciado) consiste en encontrar números primos gemelos dentro de un rango especificado por el usuario basándose en el paradigma de programación funcional y, en este caso, acudiendo al lenguaje de programación DrRacket.

Con respecto a este enunciado se plantearon dos soluciones, desde la perspectiva lógica algorítmica:

En la primera solución se recorre el rango especificado por el usuario, se revisa cada número entero contenido en este rango y si se determina que es un número primo entonces se procede a verificar si el resultado de sumarle 2 al número primo hallado, también genera un número primo. En el caso de que esto se cumpla entonces se muestran los dos números primos encontrados pues, por la diferencia que tienen de 2 unidades, significa que son dos números primos gemelos.

En la segunda solución se detecta el múltiplo del número 6 más cercano al límite inferior del rango especificado por el usuario y dentro del cual deben buscarse los números primos gemelos. De allí en adelante se avanzará, dentro de dicho rango, cada seis números y por cada número múltiplo de 6 encontrado se verificará si su antecesor y su sucesor son primos. En caso de ser verdadero entonces se mostrarán en pantalla los dos números (antecesor y sucesor), pues corresponderían a dos números primos gemelos.

Desde la óptica funcional se tuvieron en cuenta las siguientes consideraciones:

Es necesario construir una función que detecte si un número es primo o no. Esta función se podrá concebir de tres formas:

1) Contar los divisores exactos de un número en el rango comprendido entre 1 y el número dado. Si en este rango se encuentran solamente dos divisores exactos, se podrá concluir que el número es primo.

2) Contar los divisores exactos de un número en el rango comprendido entre 2 y la mitad entera del número. En caso de que en este rango no haya divisores exactos entonces se podrá concluir que el número es primo.

3) Contar los divisores exactos de un número en el rango comprendido entre 2 y el resultado entero de calcular la raíz cuadrada del número. En caso de que no se encuentre ningún divisor en este rango entonces se podrá concluir que el número es primo. Este último rango acelera la determinación de primalidad del número por evaluar.

Se construirán las funciones teniendo en cuenta las partes constitutivas de las funciones recursivas (definición de la función, condición de parada y llamado recursivo, según sea el caso).

Para el planteamiento de las dos soluciones que se presentan a continuación, y debido a que se trata de encontrar números primos gemelos, es necesario contar con un fragmento de programa que permita, por la vía de la programación funcional, determinar si un número es primo. Las dos soluciones comparten este fragmento y difieren en la forma como buscan dichos números primos gemelos.

En primer lugar, se define una función que determina si un número es divisible exactamente entre otro. Para ello, se vale de la función de librería *remainder* (propia de DrRacket, pero que tiene equivalente en todos los lenguajes de programación), que calcula el residuo de la división, es decir, su módulo. Según esta función, en caso de que el residuo del primer argumento al ser dividido entre el segundo argumento sea igual a 0 significará que efectivamente el segundo argumento es un divisor exacto del primero.

En tal caso retornará un valor 1 (como confirmación Verdadera #t) y en caso contrario retornará un valor 0 (como confirmación Falsa #f).

```
; Función que determina si un número es
divisor exacto de otro
(define (divexacto a b)
 (if(= (remainder a b) 0)
   1
   0))
```

En segundo lugar, se diseñó una función que cuenta la cantidad de divisores exactos (valiéndose del valor de retorno de la función *divexacto*) que tiene un número n en el rango comprendido entre 2 y la raíz cuadrada de dicho número n. Se parte del principio que indica que si un número no tiene divisores exactos en el rango (2, √n) entonces eso significa que el número es primo. Esta es una función recursiva que, como es de esperarse, su condición de parada se basa en que la variable *contador* (que almacena todos los valores enteros en el subrango especificado), supere el valor entero de √n. Se ha acudido a la función *floor* para tomar el entero más cercano, por debajo, al valor que resulte de √n. En esta función lo que se hace es sumar los valores de retorno de la función *divexacto* por cada uno de los valores enviados.

```
; Función que cuenta los divisores exactos de
un número entero en el rango (2, sqrt(n))
(define (cuentadivex n contador)
 (if(> contador (floor(sqrt n)))
     0
     (+      (divexacto n contador)
             (cuentadivex n (+ contador 1)))
 ))
```

En tercer lugar, se ha diseñado una función que determina si un valor es primo o no por el camino que posibilita que el valor retorne a la función *cuentadivex*. Si esta función retorna el valor 0, teniendo en cuenta que se han evaluado los divisores exactos de n en el rango (2, √n) y que si un número no tiene divisores exactos en este rango significa que es primo y que el valor de n es primo. Para tal caso, la función *esprimo* retornará a 1 (a manera de confirmación de una respuesta Verdadera #t) o retornará 0 confirmando respuesta Falsa #f.

```
; Función que define si un número es primo
o no
(define (esprimo n)
 (if(= (cuentadivex n 2) 0)
   1
   0))
```

El conjunto de estas tres funciones, como solución múltiple, permite determinar por un camino acelerado si un número es primo o no. Esta secuencia funcional algorítmica será utilizada en las soluciones que conforman el corpus del presente artículo por la naturaleza misma de su objetivo.

Ahora bien, la primera solución al hallazgo de los números primos gemelos en un rango definido por el usuario se basa en la función numprimgemelosv1 que recibe dos argumentos: el límite inferior del rango (*liminf*) y el límite superior del rango (*limsup*).

Esta es una función recursiva que tiene como condición de parada la verificación de que el límite inferior haya superado el valor resultante del límite inferior restándole 2, en virtud de que el último número primo gemelo que podría encontrarse será el segundo número primo que, en el caso de coincidir con el límite superior, tendrá una diferencia de 2 con el primer número primo hallado. En caso de que se cumpla la condición de parada, es decir, de que se haya recorrido todo el rango especificado por el usuario entonces aparecerá en pantalla el aviso "…Fin".

Si no se ha cumplido la condición de parada, y es aquí en donde emerge la primera forma de solución, se suma el resultado de la evaluación del límite inferior (*liminf*) por medio de la función *esprimo* con el resultado de la evaluación de sumarle 2 a dicho límite inferior (*liminf + 2*) —escrito en notación prefija por requerimientos del lenguaje DrRacket—, y si esa suma genera el valor 2 significa que tanto con *liminf* como con (+ *liminf 2*) la función *esprimo* retornó a 1 en cada una de ellas. Por tanto, los dos valores son primos y serán dos números primos gemelos,

debido a que la diferencia entre ellos es 2. Finalmente se llama recursivamente a la función numprimgemelosv1 enviándole como argumentos (+ *liminf 1*) que es el número siguiente al valor actual de *liminf* y el mismo *limsup* que no experimenta ningún cambio en toda la función. Es de anotar que para determinar si un número es primo se acude a la función *esprimo* que se explicó anteriormente y que se comparte con la segunda solución.

```
    ; Función que muestra los números primos
gemelos en un rango definido por el usuario
    (define (numprimgemelosv1 liminf limsup)
      (if(> liminf (- limsup 2))
        (display "....Fin")
        (begin
          (if(= (+ (esprimo liminf)
                   (esprimo (+ liminf 2))) 2)
            (begin
              (display liminf)
              (display "\t")
              (display (+ liminf 2))
              (newline)
              )
            )
          (numprimgemelosv1 (+ liminf 1) limsup)
          )
        ))
```

La segunda solución se basa en el enlace de varias funciones. La primera de ellas se ha llamado numprimgemelosv2 que recibe como argumentos *liminf* y *limsup*, solo que esta vez el valor que recibe como *liminf* es el primer múltiplo de 6 a partir del límite inferior definido por el usuario. Para la construcción de esta función se acude a la premisa de que el número intermedio entre dos números primos gemelos siempre será divisible exactamente entre 6, es decir, siempre será un múltiplo de 6.

En la condición de parada de la función se evalúa si el valor de nuevo límite inferior (o sea el que es múltiplo de 6) ha superado al valor del límite superior. En tal caso, deberá aparecer el título "…Fin". Si aún no se ha cumplido la condición de parada de esta función recursiva nivel III, entonces se evalúa si el número anterior al valor *liminf* y el número posterior son números primos, valiéndose del valor de retorno de la función *esprimo*. Si la suma de los dos retornos, al invocar la función *esprimo*, es igual a 2, significa que los dos números son primos gemelos y se procede a desplegarse en la pantalla. Si no se cumple entonces se pasa al número siguiente múltiplo de 6 dentro del rango definido.

```
    ; Función que muestra los números primos
gemelos en un rango definido por el usuario
    (define (numprimgemelosv2 liminf limsup)
      (if(> liminf limsup)
        (display "....Fin")
        (begin
          (if(= (+ (esprimo (- liminf 1))
                   (esprimo (+ liminf 1))) 2)
            (begin
              (display (- liminf 1))
              (display "\t")
              (display (+ liminf 1))
              (newline)
              )
            )
          (numprimgemelosv2 (+ liminf 6) limsup)
          )
        ))
```

La función primermultiplode6, como su nombre lo indica, retorna el primer valor que cumple con la condición de ser múltiplo de 6 a partir del valor del límite inferior que escriba el usuario. Es una función recursiva nivel I.

```
    ; Función que detecta el primer múltiplo de 6
dentro del rango
    (define (primermultiplode6 n)
      (if(= (remainder n 6) 0)
        n
        (primermultiplode6 (+ n 1))
        ))
```

La función interfaz recibe dos argumentos falsos, para que comience a funcionar, y en su interior solicita al usuario los valores de límite inferior (*liminf*) y límite superior (*limsup*) e invoca a la función numprimgemelosv2 enviándole como argumentos el primer múltiplo de 6

por encima de *liminf* y el límite superior (*limsup*) digitado por el usuario.

```
; Función que recibe las fronteras del rango
(define (interfaz liminf limsup)
  (display "Límite inferior del rango: ")
  (set! liminf (read))
  (display "Límite superior del rango: ")
  (set! limsup (read))
  (numprimgemelosv2 (primermultiplode6
liminf) limsup)
  )
```

Resultados

Debe advertirse, en primera instancia, que la ejecución de ambos programas logra los mismos resultados, aunque por caminos diferentes en lo que corresponde a la detección de los números primos, mas no en la determinación de tal caracterización. Cuando se invoca la primera solución, por ejemplo, en el rango (10, 100) se obtienen los siguientes resultados.

```
>(numprimgemelosv1 10 100)
11    13
17    19
29    31
41    43
59    61
71    73
…Fin
```

Cuando se ejecuta la segunda solución, el resultado es exactamente el mismo.

```
>(interfaz 1 1)
Límite inferior del rango: 10
Límite superior del rango: 100
11    13
17    19
29    31
41    43
59    61
71    73
…Fin
```

Como puede observarse los valores obtenidos son los mismos. Realizando una pequeña alteración estética en el despliegue de los datos, en ambas soluciones, se pudieron obtener los siguientes resultados cuando se les escribió como rango (2, 1000).

(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73), (101, 103), (107, 109), (137, 139), (149, 151), (179, 181), (191, 193), (197, 199), (227, 229), (239, 241), (269, 271), (281, 283), (311, 313), (347, 349), (419, 421), (431, 433), (461, 463), (521, 523), (569, 571), (599, 601), (617, 619), (641, 643), (659, 661), (809, 811), (821, 823), (827, 829), (857, 859), (881, 883).

Análisis y discusión de los resultados

Teniendo en cuenta que ambas soluciones parten de una misma base que es una solución compuesta recursiva nivel II, que detecta si un valor es primo o no, debe admitirse que el rango (2, √n), para un valor de n entero, es un rango que posibilita mayor eficiencia en el recorrido de búsqueda de posibles divisores exactos, toda vez que la cantidad de números que conforman dicho rango es significativamente menor que si fuera el rango (1, n) o (2, n/2), rangos a los que también se acude algorítmicamente para buscar los posibles divisores de un número.

Por su parte, la solución 1 permite una búsqueda número a número y cada vez que detecta que un número es primo, moviéndose dentro del rango especificado, busca si también es primo el número que se encuentra 2 enteros más a la derecha de los números naturales. Como lo va haciendo uno por uno, entonces el programa solución recorre todo el rango hasta llegar al límite superior restándole 2 unidades, pues si se presentara el caso de que el límite superior fuera primo, entonces el programa mostraría como gemelos a ese límite y su pareja que ya estaría 2 unidades por fuera del rango. Esta solución aprovecha la velocidad de procesamiento del computador en alta plenitud y encuentra el resultado esperado en tiempo razonable.

La segunda solución hace el mismo recorrido en el rango (2, √n) con dos variantes: 1) detecta el múltiplo de 6 más cercano por encima del límite inferior para comenzar allí su búsqueda

y 2) va avanzando cada seis números teniendo en cuenta que el valor que se encuentra entre los números primos gemelos (dada su diferencia de dos unidades) siempre es un múltiplo de 6. Esta es una solución más eficiente que la anterior puesto que además de aprovechar la velocidad de procesamiento, acelera el proceso avanzando de 6 en 6 y, por tanto, el proceso se reduce a la mitad. Es de anotar que cuando en esta solución se adelanta a seis números más allá del actual, se evalúa tanto el número anterior como el posterior. En el evento de que ambos sean primos, entonces se puede dar fe de que se encontró un par de números primos gemelos.

Ampliando el sentido de la presente solución, que dicho sea de paso conforma el corpus de esta investigación como uno de los productos resultantes del proyecto 6-21-10, se podría hacer toma de tiempos incorporando funciones de biblioteca como *current-time* que pertenece al conjunto de utilidades del lenguaje de programación DrRacket para dimensionar el verdadero valor de la eficiencia de las dos soluciones, pues su diferencia establecerá la ventaja en procesamiento que tiene una con respecto a la otra.

Si bien, con el análisis realizado en los párrafos inmediatamente anteriores, puede deducirse que el recorrido evaluativo dentro de un rango es mucho más eficiente cuando se hace cada seis números, que cuando se hace con todos los números incluidos en el rango, tampoco debe descartarse la medición de tiempos ya sea para confirmar lo dicho o para verificar si efectivamente lo planteado es cierto. De la misma forma, no se descarta que para determinar si un número es primo o no y con el ánimo de mejorar su eficiencia, se pueden adoptar otras fórmulas que proveen las matemáticas y que corresponden a los aportes de grandes investigadores que se han ocupado del tema de la detección temprana y eficiente de los números primos.

Conclusiones

Si se tiene en cuenta que el objetivo de este artículo radica en resolver una situación heredada de las matemáticas y plantear dos posibles soluciones desde la perspectiva del paradigma de la programación funcional con una componente de optimización para acelerar los procesos de búsqueda, puede concluirse que se ha logrado con cinco beneficios, que fortalecen el contenido del presente escrito: 1) se plantean dos soluciones que resuelven un mismo problema por caminos diferentes pero apoyados en la misma base; 2) se le da una aplicación al paradigma funcional en toda su plenitud aprovechando el concepto de función y la recursividad como sus grandes bastiones lógicos para resolver problemas; 3) se fortalece el marco teórico que proveen las matemáticas en cuanto a la definición de los números primos gemelos; 4) se presentan unos resultados que pueden compararse con los publicados en otros medios de difusión, que se han ocupado del mismo problema y 5) al socializar esta solución con los estudiantes, se logra que la programación de computadores y las matemáticas planteen, resuelvan analicen y optimicen un mismo problema, para bien de la comprensión de ambas áreas y para el fortalecimiento del avance académico y el pensamiento científico de los estudiantes.

Referencias

[1] A. Alexander, *Functional Programming*, Boston: CreateSpace Independent Publishing, 2017.

[2] R. Shorey, *Soft Skills for a big impact*, Nueva York: Independently Publisher, 2021.

[3] B. J. Hoogenboom y R. C. Manske, "How to write a scientific article", *International Journal of Sports Physical Therapy*, vol. 7, nº 5, 512-517, 2012.

[4] O. Trejos, *Significado y competencias*, Pereira: Editorial Papiro, 2015.

[5] G. Small, *Digital Brain*, Bogotá: Urano, 2016.

[6] Y. Harari, 21 lecciones para el siglo xxi, Madrid: Editorial Debate, 2018.

[7] D. Deitel, *Programación orientada a objetos*, Nueva York: Independent Press, 2017.

[8] F. M. Tamayo, *Fundamentos de lógica de programación*, Madrid: Editorial Académica Española, 2012.

[9] O. Trejos, *Lógica de programación*, Bogotá: Ediciones de la U, 2017.

[10] E. Diez, *Solución de problemas en teoría de números y geometría*, Nueva York: Independently Publisher, 2022.

[11] D. Wells, *Prime Numbers: the most Mysterious Figures in Math*, Londres: Wiley, 2007.

[12] M. Kline. *El pensamiento matemático. De la antigüedad a nuestros días*, Madrid: Alianza Editorial, 2012.

[13] C. Boyer, *Historia de la matemática*, Madrid: Alianza Editorial, 2010.

[14] I. Stewart, *Incredibles Numbers*, Barcelona: Crítica, 2016.

[15] E. Grothendieck, *Matemáticas: funciones*, Nueva Zelanda: Electronic Press, 2020.

[16] K. Dorling, *The Math Book (Big Ideas)*, Londres: DK, 2019.

[17] C. Boguslaw, *Introduction to programming with C++*, Nueva York: Wiley IEEE Press, 2019.

[18] P. Van Roy, *Functional Programming: Tecniques, Models and Theory*, Boston: MIT Press, 2016.

[19] O. Trejos, *Programación funcional con Racket*, Madrid: RaMa, 2019.

[20] L. Peña, *Introducción a la programación funcional*, Nueva York: Kindle Press, 2021.

[21] C. Cobo y J. Moravec, *Aprendizaje invisible: hacia una nueva ecología de la educación,* Barcelona: Publicacions i Edicions Universitat de Barcelona, 2011.

[22] G. Siemens, *Knowing Knowledges*, Morrisville (NC): Lulu.com, 2006.

[23] J. Wing, *Computational Thinking*, Boston: Oreilly Publishing, 2017.