Algoritmo de búsqueda de ternas pitagóricas basado en programación funcional*[1]

[An algorithm to search Pythagoreans numbers using functional programming]

[Algoritmo de busca baseado em trios pitagóricos programação funcional]

Omar Iván TREJOS BURITICÁ[2]

Recibo: 25.08.2015 – Aprobación: 30.09.2016

Resumen: *Se presenta una posible solución a la búsqueda de números que se constituyan en ternas pitagóricas, mediante una implementación con programación funcional bajo las posibilidades técnicas y sintácticas del lenguaje Scheme entorno DrRacket. La metodología se enmarcó en la investigación educativa de carácter cuantitativo. Los resultados de este algoritmo, y su uso, evidencian la posibilidad de encontrar soluciones muy simples, desde la óptica funcional, para resolver problemas con cierta complejidad así como aprendizajes más significativos y con mayor sentido en el campo de la programación de computadores por parte de los estudiantes. El programa presentado sirve de ejemplo para encontrar una aplicación más a la instancia tecnológica como expresión instrumental de la matemática.*

Palabras clave: *Algoritmo, programación funcional, recursión, ternas pitagóricas.*

Abstract: *The article presents a possible solution to the search for numbers that are formed in Pythagorean triples*

* **Modelo para la citación de este artículo:**
TREJOS BURITICÁ, Omar Iván (2016). Algoritmo de búsqueda de ternas pitagóricas basado en programación funcional. En: Ventana Informática No. 35 (jul-dic). Manizales (Colombia): Facultad de Ciencias e Ingeniería, Universidad de Manizales. p. 63-78. ISSN: 0123-9678

1 Artículo de investigación proveniente del proyecto *Análisis pedagógico, instrumental y conceptual de algunos paradigmas de programación como contenido de la asignatura Programación I del Programa Ingeniería de Sistemas y Computación*, ejecutado en el periodo *20.01-15.12.2013*, e inscrito en el grupo de investigación Informática de la *Vicerrectoría de Investigaciones, Innovación y Extensión de la Universidad Tecnológica de Pereira.*

2 PhD. en Ciencias de la Educación, MSc. en Comunicación Educativa, Especialista en Instrumentación Física, Ingeniero de Sistemas. Docente de planta, Universidad Tecnológica de Pereira (Pereira, Risaralda, Colombia) Correo electrónico: omartrejos@utp.edu.co

by an implementation with functional programming under the technical and syntactical possibilities DrRacket Scheme language environment. The methodology is framed in educational research quantitative. The results of this algorithm and its use, show the possibility of finding simple solutions, from the functional point of view, to solve problems with some complexity and more meaningful learning and more sense in the field of computer programming by the students. The program presented is an example to find a more technological application instance and instrumental expression of mathematics.

Keywords*: Algorithm, functional programming, Pythagorean numbers, recursion.*

Resumo: *O artigo apresenta uma possível solução para a busca de números que são formados em triplos pitagóricos por uma implementação com programação funcional sob o possibilidades técnicas e sintáticos ambiente de idioma Esquema DrRacket. A metodologia está enquadrada em quantitativa pesquisa educacional. Os resultados deste algoritmo e sua utilização mostram a possibilidade de encontrar soluções simples, do ponto de vista funcional, para resolver problemas com alguma complexidade e aprendizagem mais significativa e mais sentido no campo da programação de computadores pela os estudantes. O programa apresentado é um exemplo para encontrar uma instância de aplicação tecnológica mais e expressão instrumental da matemática.*

Palavras-chave*: Algoritmo, programação funcional, recursão, trios pitagóricos.*

Introducción

Trejos (2012, 23), señala la utilización de la programación de computadores como base para la Ingeniería de Sistemas para encontrar soluciones a problemas que, por su naturaleza, tienen cierto nivel de complejidad y, así mismo, para permitir que el computador desarrolle tareas que pudieran ser agotadoras para el ser humano. La búsqueda de aplicaciones de la tecnología para resolver problemas de las matemáticas es uno de los terrenos más fértiles en donde los lenguajes de programación han hecho gala no sólo de utilidad sino también de oportunidad. Desde la perspectiva puramente docente, la programación de computadores es un área que sirve para que el estudiante apropie, asimile, aplique, evalúe y retroalimente las posibilidades de la lógica computacional, base para el desarrollo de los

algoritmos que posteriormente se han de convertir en programas, según señalan Van Roy & Haridi (2004, 52). Normalmente el estudiante, en las asignaturas de programación de computadores, aprende un conjunto de herramientas que le posibilitan la resolución de diferentes problemas. Sin embargo, surge la inquietud, expuesta por Bruner (1969, 18), acerca de ¿cuántas veces en realidad el estudiante se enfrenta a problemas reales que pueda resolver con la programación de computadores? Y posiblemente se encuentre en la respuesta más aplicaciones de orden puramente académico que práctico.

La apropiación, asimilación, aplicación, evaluación y retroalimentación de conceptos derivados de un área de conocimiento se hace mucho más sencilla cuando se aplican en ejemplos y situaciones prácticas según sostiene Ausubel (1998, 39), así, la programación de computadores encuentra en la matemática un espacio amplio y bondadoso para resolver problemas por el camino de la tecnología, que permitan conferirle significado al conocimiento algorítmico y los lenguajes de programación[3]. Esta estrategia se contrapone, desde una óptica mucho más práctica y aplicativa, a aquella en la cual los estudiantes cuando resuelven problemas enfatizan en los procedimientos más que en el razonamiento y en la lógica, de acuerdo con Bruner (1991, 102).

El aprendizaje de los lenguajes de programación está mediado por las relaciones de tecnología con las soluciones implementadas, y entre dichas soluciones y los problemas que resuelven, según Paz Penagos (2011, 216), es donde un estudiante se enfrenta a la retención o aplicación de los conocimientos adquiridos (contenidos de la disciplina), con gran frecuencia dificultada por la apropiación de conceptos y su posterior aplicación.

Que el estudiante aplique la programación de computadores en la solución de problemas prácticos de otras ciencias es un reto para el docente universitario, priorizando el razonamiento lógico y la apropiación del pensamiento algorítmico sobre procedimientos basados más en la memoria. A esto se le puede sumar que existe la posibilidad de encontrar aplicaciones prácticas al pensamiento algorítmico y a la programación de computadores en sí, «*desde las características de los estilos cognitivos de los estudiantes, favorezca un aprendizaje autónomo con responsabilidad que sea más activo,*

3 La programación de computadores es área básica en programas de Ingeniería de sistemas y afines, lo que implica la importancia de su comprensión y aprendizaje.

analítico, colaborativo, interdisciplinario, reflexivo y autorregulado, en el cual los estudiantes construyan conocimientos y significados con sentido» (Paz Penagos, 2014, 54).

La pregunta de investigación se plantea en los siguientes términos: ¿es posible encontrar aplicaciones prácticas en diferentes áreas que permitan aplicar, de manera concreta, el pensamiento algorítmico y la programación de computadores para resolver problemas de dichas áreas de forma que sean tangibles en su aplicación para los estudiantes? La respuesta es el contenido de artículo que responde afirmativamente la pregunta. De aquí se infiere una hipótesis que, en sintonía con el problema, valida la existencia de diferentes espacios de práctica para la programación y el pensamiento algorítmico en diferentes áreas tal que permitan al estudiante aproximarse tanto a la aplicación del conocimiento que la programación provee como a las diferentes áreas en donde dicho conocimiento es la base para la solución de sus problemas.

El marco de referencia de la investigación es la formación científica y tecnológica en ingeniería y se fundamenta en las teorías de aprendizaje significativo de David Paul Ausubel (1998), aprendizaje por descubrimiento de Jerome Seymour Bruner (1991) y la tendencia *Active Learning* a partir de la teoría pedagógica socio-constructivista de Lev Vigotsky (1981) y su aplicación en el campo de la ingeniería de sistemas y computación en el área de programación de computadores. El contenido del presente artículo propone una forma de establecer relaciones entre la programación de computadores y un enunciado específico de las matemáticas a la luz de la construcción de un programa que permita el hallazgo de los factores que satisfacen el teorema de Pitágoras desde la perspectiva que la programación funcional provee aprovechando las facilidades de recursividad derivadas del uso del lenguaje de programación *Scheme* en el entorno *DrRacket*.

1. Fundamento Teórico

El paradigma de programación funcional, de acuerdo con Felleisen et al. (2010, 29), constituye una de las alternativas para resolver un problema lógico tomando como base la fundamentación matemática derivada del cálculo Lambda y el concepto de función como base para la construcción algorítmica de soluciones. La programación funcional basa su teoría, precisamente, en el concepto de función que, desde la perspectiva tecnológica, es un conjunto de instrucciones que puede

recibir argumentos, puede ser identificado con un nombre y puede procesar datos de forma que retorne valores sin acudir a la memoria del computador como intermediario para definir estados de almacenamiento según Van Roy & Haridi (2004, 132). La programación funcional pretende poner a disposición de los programadores un conjunto de herramientas que permitan el procesamiento de datos y respuestas inmediatas a partir de los resultados que arrojen las funciones, como núcleo principal de trabajo.

En su más simple concepción, un programa construido a partir del paradigma funcional es un conjunto de funciones independientes pero, a la luz del objetivo del algoritmo, interdependientes que, en conjunto, resuelven, alcanzan o satisfacen un objetivo. La técnica *divide y vencerás*[4] permite una atomización de las soluciones que simplifica la estructura de dicha solución de manera que encontrar los errores lógicos sea mucho más sencillo, queda resuelto el gran objetivo general de acuerdo con Kline (2012, 61). Cada solución a estos pequeños objetivos se constituye en una función independiente pero cuyos resultados aportan a la solución general a partir de la interacción con otras funciones.

Uno de los recursos más potentes que se usan en programación funcional, precisamente para que las funciones sean altamente *funcionales* –valga esta redundancia- es el concepto de recursividad. Este concepto puede definirse, desde lo puramente tecnológico, como la posibilidad que ofrece un lenguaje de programación para que una función pueda llamarse a sí misma. Desde lo matemático la recursividad se define como la inclusión de la función definida dentro de la misma definición lo cual, en sí mismo, establece un *loop* que, aunque pareciera redundante, puede capitalizarse para que lograr objetivos que se derivan de procesos cíclicos que simplifican y resuelven muchos problemas según lo plantea Rooney (2009, 81). Se ha acudido a la utilización del lenguaje *Scheme* en el entorno *DrRacket* debido a su notable capacidad en procesamiento de datos numéricos, a la posibilidad de manejar datos sin tener que estar atados al concepto de estado (y por ende de tipos de datos) y a la manipulación simple y efectiva que, a partir de la relación e interacción entre funciones, provee el lenguaje de programación.

4 En idioma inglés *divide and conquer*, de acuerdo con Calderín (2014, 7), «*consiste en descomponer el problema en un conjunto de subproblemas* más pequeños. Después se resuelven estos *subproblemas y se combinan las soluciones para obtener la solución para el problema original*» por deducción.

2. Metodología

2.1 Descripción

Para buscar su solución a partir de una propuesta algorítmica funcional se ha escogido el teorema de Pitágoras[5]. El problema planteado desde la matemática consiste en encontrar números enteros que cumplan con dicha ecuación de forma que se muestren en pantalla a manera de ternas pitagóricas, es decir, en conjuntos de números que satisfacen el teorema de Pitágoras, como lo expresa Chabert (2005, 212), donde sus dos valores menores corresponden a la medida de los catetos y el valor mayor corresponde a la medida de la hipotenusa, de acuerdo con Stewart (2012, 55). Desde la programación se pretende construir un programa que encuentre las ternas pitagóricas y que, en pantalla, despliegue la prueba de su validez frente al teorema planteado dentro de un rango determinado para lo cual se asigna un valor tope y se buscan las ternas pitagóricas en el rango 1 hasta el valor tope establecido.

Para la construcción de la solución funcional que resuelva el problema enunciado, se ha planteado la necesidad de pensar en las diferentes posibilidades de validación que se requieren para confirmar el hallazgo de ternas pitagóricas, acorde con la definición. Para ello se pensó en que si el rango de revisión de los números a verificar es de 1 a n para un valor de n leído, entonces las posibles combinaciones estarán dadas por los valores que se muestran en la Tabla 1.

Tabla 1. Posibles valores a validar (Trejos, 2015, 87)

1ª revisión	2ª revisión	3ª revisión	...	N ª revisión
1^2+1^2	2^2+1^2	3^2+1^2	...	n^2+1^2
1^2+2^2	2^2+2^2	3^2+2^2	...	n^2+2^2
1^2+3^2	2^2+3^2	3^2+3^2	...	n^2+3^2
.	.	.	.	.
.	.	.	.	.
1^2+n^2	2^2+n^2	3^2+n^2	...	n^2+n^2

Dicha tabla indica que debe realizarse un proceso que se vaya, progresivamente, verificando en cada par de números el cumplimiento

5 En términos simples, establece que en un triángulo rectángulo, la suma de los cuadrados de los catetos es igual al cuadrado de la hipotenusa, como lo plantea Sparks (2013, 36), que expresado matemáticamente sería $a^2+b^2=c^2$ para valores a, b y c como medida de dichos catetos y dicha hipotenusa, según Boyer (2010, 78).
Debe recordarse, señala Rey Pastor & Babini (2000, 56), que un triángulo es rectángulo si uno de sus ángulos internos es igual a 90º.

del teorema de Pitágoras de forma que se cubra todo el espectro de posibilidades de los números comprendidos entre 1 y el número n asignado como tope superior. Esto pone de presente la necesidad de plantear dos ciclos anidados que recorran los posibles valores. En un proceso cíclico externo deberá hacerse un recorrido desde 1 hasta n (avanzando de 1 en 1) y, dentro de este proceso cíclico externo, debe organizarse un proceso cíclico externo que también recorra todos los valores desde 1 hasta n (avanzando igualmente de 1 en 1). En términos de seudocódigo, la tabla 1 se resume algorítmicamente de la siguiente forma:

Para a = 1 hasta n (1)

Para b = 1 hasta n (1)

Validar $a^2 + b^2$

Una vez detectada la secuencia numérica se procede a detectar el cumplimiento del teorema de Pitágoras: para *Validar a^2+b^2*, se busca si la raíz cuadrada de este resultado es un número entero, para lo cual se acude a la función *floor* (del lenguaje *Scheme*) que obtiene la parte entera de un número real[6]. Se compara si la parte entera de la raíz cuadrada de la suma a^2+b^2 es igual dicha parte entera, en cuyo caso se ha encontrado una terna pitagórica. En ese momento, se procede a mostrar la terna en un formato predefinido, que algorítmicamente podría resumirse de la siguiente forma:

Calcular el resultado de $a^2 + b^2$

Obtener la raíz cuadrada de $a^2 + b^2$

Obtener la parte entera de la raíz cuadrada obtenida

Comparar la parte entera de dicha raíz con la raíz como tal

Si son iguales

Mostrar la terna pitagórica

Si no lo son

Continuar con la siguiente combinación

De esta forma, la solución a encontrar se basa en el algoritmo:

Para a = 1 hasta n (1)

Para b = 1 hasta n (1)

Calcular el resultado de $a^2 + b^2$

Obtener la raíz cuadrada de $a^2 + b^2$

6 Si el número fuera 4.5, al aplicar la función *floor(4.5)* se obtendrá el número 4, como respuesta, que corresponde a la parte entera del valor real 4.5.

Obtener la parte entera de la raíz cuadrada obtenida
Comparar la parte entera de dicha raíz con la raíz como tal
Si son iguales
Mostrar la terna pitagórica
Si no lo son
Continuar con la siguiente combinación
Fin Si
Fin Para
Fin Para

2.2 Aplicación

El programa (Trejos, 2015, 88-89), escrito en lenguaje *Scheme* bajo entorno *DrRacket* versión 6.1 R5RS, cumple con el objetivo planteado.

```
; PROGRAMA QUE ENCUENTRA TERNAS PITAGÓRICAS RANGO 1 A N
PARA UN VALOR N DEFINIDO
; Función simple que calcula la suma a^2+b^2
(define (suma a b)
(+ (expt a 2)(expt b 2))
 )
; Función simple que confirma la existencia de una terna
pitagórica
(define (pitagoras a b)
 (if(= (sqrt(suma a b))(floor(sqrt(suma a b))))
 1
 0
 ))
; Función simple que despliega resultados (solo ternas)
(define (mostrar2 a b)
 (display "Terna Pitagórica --> (")
 (display a)
 (display ",")
 (display b)
 (display ",")
 (display (floor (sqrt (+ (expt a 2)(expt b 2)))))
 (display ")")
 (newline)
 (newline)
 )
; Función simple que despliega resultados (completos)
(define (mostrar1 a b)
 (newline)
```

```
 (display a)
 (display "^2 = ")
 (display (expt a 2))
 (display ", ")
 (display b)
 (display "^2 = ")
 (display (expt b 2))
 (newline)
 (display (expt a 2))
 (display " + ")
 (display (expt b 2))
 (display " = ")
 (display (+ (expt a 2)(expt b 2)))
 (newline)
 (display "Raiz Cuadrada de ")
 (display (+ (expt a 2)(expt b 2)))
 (display " = ")
 (display (floor (sqrt (+ (expt a 2)(expt b 2)))))
 (newline)
 (display "Terna Pitagórica --> (")
 (display a)
 (display ",")
 (display b)
 (display ",")
 (display (floor (sqrt (+ (expt a 2)(expt b 2)))))
 (display ")")
 (newline)
 (newline)
 )

; Función recursiva (proceso cíclico interno) de búsqueda
de números pitagóricos
(define (cicloint i j n)
 (if (> j n)
        0
        (begin
        (if ( = (pitagoras i j) 1 )
                (mostrar1 i j)
                )
        (cicloint i (+ j 1) n)
        )
     )
 )
; Función recursiva (proceso cíclico externo) de búsqueda
de números pitagóricos
```

```
(define (cicloext i j n)
 (if (> i n)
        0
        (begin
          (cicloint i j n)
          (cicloext (+ i 1) j n)
          )
      )
 )
; Función inicial
(define (inicio i j n)
 (cicloext i j n)
 )
; Llamado inicial (100 es el valor definido para n)
(inicio 1 1 100)
```

En la última línea se encuentra el llamado a la función inicial, que articula las funciones para el hallazgo de las ternas pitagóricas. La tabla 2 explica algunas características de cada función.

Tabla 2. Descripción de características de las funciones (Trejos, 2015, 90)

Función	Descripción
suma	Esta función calcula el resultado de a^2+b^2
pitagoras	Esta función retorna Verdadero (1) si los valores a y b corresponden a dos de los tres factores de una terna pitagórica. Retorna Falso (0) si no es cierto
mostrar2	Muestra el resultado de la terna pitagórica en formato (a,b,c)
mostrar1	Muestra el resultado de la terna pitagórica en formato ampliado (ejemplo) *3^2 = 9, 4^2 = 16* *9 + 16 = 25* *Raíz Cuadrada de 25 = 5* *Terna Pitagórica --> (3,4,5)*
cicloint	Genera el proceso cíclico interno que permita validar los valores en el rango 1 a n de acuerdo a la explicación del numeral 3.1 Descripción
cicloext	Genera el proceso cíclico externo que permita validar los valores en el rango 1 a n de acuerdo a la explicación del numeral 3.1 Descripción
inicio	Hace el primer llamado definiendo los valores para i y j como 1 y para n (tope del rango de validación)

3. Resultados y discusión

3.1 Descripción

En cuanto a la ejecución del programa se ha realizado de dos formas: a) mostrando el formato de la función *mostrar2* en donde simplemente se despliegan las ternas, b) mostrando el formato de la función *mostrar1* en donde se despliega el proceso de verificación y cada terna pitagórica.

La tabla 3 muestra algunos resultados con cada llamado. Para efectos de simplicidad se ha ejecutado el programa estableciendo como tope superior del rango de validación el valor 10.

Tabla 3. Resultados obtenidos (Trejos, 2015, 91)

Función	Resultado Obtenido
Usando la función mostrar2	*Terna Pitagórica --> (3,4,5)* *Terna Pitagórica --> (4,3,5)* *Terna Pitagórica --> (6,8,10)* *Terna Pitagórica --> (8,6,10)*
Usando la función mostrar1	*3^2 = 9, 4^2 = 16* *9 + 16 = 25* *Raíz Cuadrada de 25 = 5* *Terna Pitagórica --> (3,4,5)* *4^2 = 16, 3^2 = 9* *16 + 9 = 25* *Raíz Cuadrada de 25 = 5* *Terna Pitagórica --> (4,3,5)* *6^2 = 36, 8^2 = 64* *36 + 64 = 100* *Raíz Cuadrada de 100 = 10* *Terna Pitagórica --> (6,8,10)* *8^2 = 64, 6^2 = 36* *64 + 36 = 100* *Raíz Cuadrada de 100 = 10* *Terna Pitagórica --> (8,6,10)*

Si se quieren obtener las ternas pitagóricas en un rango más amplio, debe cambiarse el valor definido para el argumento *n* en la última línea del programa que corresponde al primer llamado a la función y que para el ejemplo actual tiene un valor de 100. Si por ejemplo, es escribiera el valor 10000 se presentan a continuación los tres primeros resultados y los tres últimos resultados en formato completo:

```
3^2 = 9, 4^2 = 16
9 + 16 = 25
Raíz Cuadrada de 25 = 5
Terna Pitagórica --> (3, 4, 5)
```

```
4^2 = 16, 3^2 = 9
16 + 9 = 25
Raíz Cuadrada de 25 = 5
Terna Pitagórica --> (4, 3, 5)
5^2 = 25, 12^2 = 144
25 + 144 = 169
Raíz Cuadrada de 169 = 13
Terna Pitagórica --> (5, 12, 13)
.
10000^2 = 100000000, 2250^2 = 5062500
100000000 + 5062500 = 105062500
Raíz Cuadrada de 105062500 = 10250
Terna Pitagórica --> (10000, 2250, 10250)
10000^2 = 100000000, 4875^2 = 23765625
100000000 + 23765625 = 123765625
Raíz Cuadrada de 123765625 = 11125
Terna Pitagórica --> (10000, 4875, 11125)
10000^2 = 100000000, 7500^2 = 56250000
100000000 + 56250000 = 156250000
Raíz Cuadrada de 156250000 = 12500
Terna Pitagórica --> (10000, 7500, 12500)
```

3.2 Discusión

Si bien la utilización de la recursividad en programación funcional es un posible recurso para resolver un problema, el uso de ciclos también posibilita la solución de los mismos desde una óptica menos desgastante computacionalmente. El uso de un lenguaje de programación como *Scheme* facilita los cálculos dado que este lenguaje permite un manejo bastante amplio en cuanto a los datos numéricos y eso permite que no se tengan las limitaciones que otros paradigmas de programación involucran debido al problema del estado, capacidad y almacenamiento de memoria.

Si bien la atomización de un programa pudiera parecer un camino más largo para construir soluciones y satisfacer objetivos, es claro que el gran rédito que se obtiene con la técnica *divide y vencerás* es la posibilidad de encontrar fácilmente los errores lógicos cuando éstos se presenten. Debe aclararse que los errores lógicos son diferentes a los de sintaxis, dado que estos no permiten que un programa pueda ejecutarse mientras

que aquellos sí, pero sin que se garanticen resultados acordes con las necesidades.

En el programa que se presenta como solución puede notarse un nivel de atomización suficiente como para diferenciar las funciones operativas de las funciones de I/O (*Input / Output*) o también llamadas funciones de interfaz. En este programa son funciones de interfaz las funciones *mostrar2* (que permite mostrar el resultado de la búsqueda con formato exclusivo de terna pitagórica), *mostrar1* (que permite mostrar la terna pitagórica pero además muestra el proceso que confirma que cumplen con el teorema de Pitágoras) y la función *inicio* (cuyo objetivo es establecer los límites tanto inferior como superior para iniciar el proceso de búsqueda).

Por su parte, en este programa son funciones puramente operativas o de cálculo la función *suma* (que se encarga de calcular el resultado de la expresión a^2+b^2), la función *pitagoras* (que es la que confirma la existencia de un valor entero que equivale a $\sqrt{a^2+b^2}$ y que determina la existencia de una terna pitagórica), la función *cicloext* (que define el proceso para que se recorra desde el valor inicial 1 hasta el valor definido n en la búsqueda de los factores) y la función *cicloint* (que define el mismo proceso pero como complemento al primer ciclo). El hecho de poder clasificar claramente estas funciones permite tener la posibilidad de encontrar fácilmente los errores que se puedan presentar.

En el caso de encontrar errores en los procesos de cálculo entonces es claro éstos podrán estar en las funciones operativas *suma*, *pitagoras*, *cicloext* y *cicloint*. Las funciones reconocidas como de interfaz son, probablemente, las que menos inciden en los errores lógicos salvo en casos excepcionales. Si tenemos en cuenta que el programa cuenta en total con siete funciones y que cuatro de ellas son operativas y que, además, son aquellas en donde pueden estar potencialmente los errores lógicos entonces se puede decir que se ha resuelto dado que una simple revisión somera puede llegar a garantizar las funciones de interfaz.

Las posibilidades que ofrece la programación funcional resulta ser interesante como forma de solución a problemas que la matemática provee aunque no se puede negar que la construcción de ciclos recursivos anidados resulta ser un poco más compleja que la construcción de ciclos estructurados anidados dado que éstos acuden al concepto de estado para almacenar los datos mientras que aquellos acuden al proceso como tal sin la intermediación de la memoria como base para el control del programa.

La exposición de esta solución permite poner a disposición de los estudiantes tanto la posibilidad de encontrar nuevos caminos de construcción en las soluciones algorítmicas y, así mismo, la opción de encontrar nuevos horizontes de problemas en donde la tecnología, la algoritmia y los lenguajes de programación puedan llegar a ser solución. Si bien las ternas pitagóricas pueden calcularse de manera manual, es claro que la intervención de la computación facilita mucho el hallazgo de dichos números y permite que el ser humano pueda dedicarse a analizar y obtener inferencias a partir de los resultados que estos programas entregan, que resultan ser tareas más gratificantes comparando la posibilidades de razonamiento y deliberación de los seres humanos.

Finalmente, vale la pena tener en cuenta que este tipo de ejercicios le confieren vida práctica a tanto a los conceptos que se derivan de la programación de computadores como a los problemas que se pueden derivar de otras ciencias, que en el caso presente son las matemáticas, permitiendo que una sea alimento de la otra y que la otra sea solución de la primera. Todo esto permite que el estudiante encuentre mayor significado y más sentido a los conceptos que intente aprender dentro de un curso de programación de computadores en donde el paradigma funcional sea la base para su contenido temático.

Es importante detenerse un poco para revisar la lógica de la que podría llamarse la función principal de este programa, como es la función *pitagoras*:

```
; Función simple que confirma la existencia de una terna
pitagórica
(define (pitagoras a b)
   (if(= (sqrt(suma a b))(floor(sqrt(suma a b))))
        1
        0
        ))
```

La lógica de esta función es muy sencilla, simplemente se compara el resultado de obtener la raíz cuadrada de la suma de los cuadrados con la parte entera de esa misma raíz. En caso de que estos dos valores sean iguales, significa que existe un valor entero que completa la terna pitagórica pues los dos valores menores serían los catetos y este valor encontrado correspondería a la hipotenusa del triángulo rectángulo. Se acude a esta como una forma sencilla de encontrar la existencia del tercer elemento de la terna pitagórica.

4. Conclusiones

Siempre es posible encontrar campos en diferentes áreas, y especialmente las matemáticas, de donde se puedan extraer problemas que, a su vez, puedan ser resueltos con los conocimientos que la programación provee.

La programación funcional es una alternativa muy interesante para encontrar solución a problemas de las matemáticas, y de manera especial el lenguaje *Scheme*, dado que éste permite la manipulación de datos numéricos con mucha más amplitud que la que permiten los lenguajes imperativos que están sujetos a tamaños y características de almacenamiento definidas por sus respectivos tipos de datos.

Se hace necesario, desde la óptica de la enseñanza de los lenguajes de programación, que se propongan ejercicios que tengan relación con otras ciencias para que se le confiera significado y sentido a ambas áreas de conocimiento.

La técnica *divide y vencerás* posibilita la construcción de soluciones que, si bien pueden estar construidas sobre una base muy variada por la cantidad de funciones que intervienen, también posibilitan la comprensión y el mantenimiento del programa dado que su atomización simplifica la solución en sí, a costa del código como tal.

La posibilidad de poder probar parcialmente un programa, tal como lo permite el entorno *DrRacket* usando el lenguaje de programación *Scheme,* es una fuerte herramienta que nos ayuda a encontrar oportunamente errores tanto lógicos como sintácticos.

El reconocimiento claro de las funciones que son de interfaz y las funciones que son de cálculo u operativas permite que se puedan detectar y superar los problemas lógicos que se pudieren presentar en la ejecución de un programa.

Referencias bibliográficas

AUSUBEL, D. (1998). Sicología Educativa: Un punto de vista cognoscitivo. Ciudad de México (México): Trillas. 1218 p. ISBN: 9682413346

BOYER, C.B. (2010). Historia de la Matemática. Madrid (España): Alianza Editorial. 808 p. ISBN: 8420681865

BRUNER, J.S. (1969). Hacia un teoría de la instrucción. Ciudad de México (México): Hispanoamericana. 314 p. ISBN: 8599860054

BRUNER, J.S. (1991). Actos de significado. Madrid (España): Alianza Editorial. 168 p. ISBN: 8420648124

CALDERÍN ABAD, Y. (2014). Propuesta de acciones interdisciplinarias a desarrollar desde la dinámica de una clase de la asignatura Metodología de la Investigación Científica {en línea]. En: Revista Electrónica EduSol, Vol. 14, No. 48 (jul-sep). Guantánamo (Cuba): Universidad de Guantánamo p. 1- 10. ISSN: 1729-8091 <http://edusol.cug.co.cu/index.php/EduSol/article/view/10/pdf_7> [consulta: 20/10/2016]

CHABERT, J.L. (2005). A history of algorithms: From the Pebble to the Microchip. Heidelberg (Germany): Springer Berlin Heidelberg. p. 199-237. e-ISBN: 978-3-642-18192-4 <doi:10.1007/978-3-642-18192-4_8> [consult: 13/04/2016]

FELLEISEN, M.; FINDLER, R.; FLAT, M. & KRISHNAMURTHI, S. (2010). How to design programs. 2 ed. Boston (MA, USA): MIT Press. 738 p. ISBN: 0262062186

KLINE, M. (2012). El pensamiento matemático de la antiguedad a nuestros días. Madrid (España): Alianza Editorial. 568 p. ISBN: 8420627291

PAZ PENAGOS, H. (2011). ¿Cómo desarrollar la metacognición en la educación superior mediante la resolución de problemas? How to develop metacognition through problem solving in higher education [en línea]. En: Ingeniería e Investigación, Vol. 31 No.1 (apr). Bogotá (Colombia): Universidad Nacional de Colombia. p. 213-233. e-ISSN: 2248-8723 <http://www.redalyc.org/pdf/643/64321170023.pdf> [consulta: 25/01/2016]

PAZ PENAGOS, H. (2014). Aprendizaje autónomo y estilo cognitivo: diseño didáctico, metodología y evaluación. En: Revista Educación en Ingeniería, Vol. 9, No. 17 (ene-jun). Bogotá (Colombia): Asociación Colombiana de Facultades de Ingeniería ACOFI, p.53-65. ISSN:1900-8260 <http://www.educacioneningenieria.org/index.php/edi/article/view/421/194> [consulta: 25/01/2016]

REY PASTOR, J. & BABINI, J. (2000). Historia de la Matemática: De la antigüedad a la baja edad media. Barcelona (España): Editorial Gedisa. 211 p. ISBN: 8497847806

ROONEY, A. (2009). Historia de las Matemáticas: Desde la construcción de la spirámides a la exploración del infinito. Barcelona (España): Oniro. 208 p. ISBN: 9788497544252

SPARKS, J.C. (2013). The Pythagorean Theorem: Crown Jewel of Mathematics. Bloomington (IN, USA): AuthorHouse. 190 p. ISBN: 1482028220

STEWART, I. (2012). Historia de las Matemáticas en los últimos 10000 años. Barcelona (España): Editorial Crítica. 312 p. ISBN: 8498923298

TREJOS BURITICÁ, O.I. (2012). Aprendizaje en Ingeniería: un problema de comunicación. Tesis Doctoral (Doctor en Ciencias de la Educación). Pereira (Colombia): Universidad Tecnológica de Pereira. 442 p.

TREJOS BURITICÁ, O.I. (2015). Comparación de dos algoritmos para hallar ternas pitagóricas usando dos paradigmas de programación diferentes. En: Entre Ciencia e Ingeniería, Vol. 9, No. 18 (jul-dic). Pereira (Colombia): Universidad Católica de Pereira p. 84-94. ISSN: 1909-8367

VAN ROY, P. & HARIDI, S. (2004). Concepts, Techniques and Models of Computer Programming. Cambridge (MA, USA): The MIT Press. 936 p. ISBN: 9780262220699