Entre Ciencia e Ingeniería, ISSN 1909-8367
Año 9 No. 18 - Segundo Semestre de 2015, página 84 - 94

Comparación de dos algoritmos para hallar ternas pitagóricas usando dos paradigmas de programación diferentes[1]

Comparison of two algorithms in order to find Pythagorean triples using two different programming paradigms

O. I. Trejos

Recibido Junio 10 de 2015 – Aceptado Septiembre 23 de 2015

***Resumen*— En este artículo, además de analizar las características algorítmicas de dos versiones de solución al problema de hallar ternas pitagóricas en un rango de 1 hasta n para n como un valor entero definido, se realiza una comparación entre la solución implementada a partir del paradigma de programación funcional, basado en lenguaje Scheme entorno Dr. Racket versión 6.1 y la solución implementada a partir del paradigma de programación imperativa utilizando lenguaje C++ arista estructurada entorno DevC++ Versión 5.8.0. La comparación de estas dos soluciones, vistas desde paradigmas de programación diferentes, se hace a nivel de codificación pura, a nivel de la lógica utilizada para resolverlo y a nivel del rendimiento y la eficiencia, apoyados en la medición de tiempos de proceso. Ambos programas presentan los resultados en dos formatos y la comparación de rendimiento se hace teniendo en cuenta el despliegue en la pantalla. Esta comparación no solo permite establecer relaciones entre los paradigmas de programación (paradigma funcional y paradigma imperativo) sino que posibilita el planteamiento de un híbrido que pudiera ser mucho más eficiente que ambos, pero a partir de lo más eficiente que cada uno puede aportar. Cada una de las implementaciones es totalmente original producto de la investigación y de la iniciativa y la experiencia del autor de este artículo.**

***Palabras clave* — algoritmo, programación funcional, programación imperativa, procesos cíclicos, recursión, ternas pitagóricas**

***Abstract* — in this article we analyze the algorithmic characteristics of two solutions for the problem to find Pythagorean triples in a specific range from 1 to n to a defined value of n as an integer. We also make a comparison between the solution implemented with functional paradigm in Scheme language environment Dr. Racket version 6.1 and the solution implemented with imperative paradigm in C++ language environment Dev C++ version 5.8.0. The comparison of these solutions is made from the perspective of two different programming paradigms. We analyze the pure codification, the logic used in the implementation and the performance and efficiency using the computer timer. Both programs present two formats to show the results in the display. These comparisons not only permit to establish possible relations between the programming computer paradigms (functional and imperative) but also let us to think about a hybrid solution from the best of each solution implemented. Each implementation is absolutely original product of the research and the initiative and experience of the author of this article.**

***Key words*— algorithm, functional programming, imperative programming cyclic process, recursion, Pythagorean triples.**

[1]Producto derivado del proyecto de investigación “Desarrollo de un modelo pedagógico para la asignatura Programación I en Ingeniería de Sistemas basado en aprendizaje significativo, aprendizaje por descubrimiento y el modelo 4Q de preferencias de pensamiento”. Presentado por el Grupo de Investigación en Informática de Ingeniería de Sistemas y Computación, de la Universidad Tecnológica de Pereira.

O.I. Trejos Docente de Planta, Ingeniería de Sistemas y Computación, Universidad Tecnológica de Pereira (Colombia); email: omartrejos@utp.edu.co.

I. Introducción

La programación, según la teoría que se deriva de la ingeniería de sistemas, es uno de los posibles caminos con que se cuenta para encontrar soluciones a determinados problemas que, según sea su contexto, tienen un nivel establecido de complejidad y resolución que le permiten al computador desarrollar las tareas que resuelvan lo que para el ser humano podría ser agotador [1]. Uno de los terrenos más fértiles tanto para el planteamiento de problemas como para su articulación con la programación en la construcción de las posibles soluciones son las matemáticas. En esta área la tecnología provee mecanismos que facilitan la construcción de dichas soluciones. El hallazgo de soluciones a problemas matemáticos desde la óptica de la programación permite que se apropien, se asimilen, se apliquen, se evalúen y, más importante, se retroalimenten planteamientos derivados de la lógica computacional que, no sólo son aplicables en matemáticas, sino en otras áreas del conocimiento a partir del concepto de Algoritmización, Codificación y Puesta a Punto [2]. Se hace necesario que el conocimiento, como el técnico que provee la programación de computadores, se confronte con situaciones prácticas reales de forma que se relacione, en tiempo de solución, con áreas que proveen los problemas, los enunciados y las situaciones problémicas [3]. Esta confrontación es lo que permite imprimirle significado al conocimiento, es decir, el hallazgo de esos espacios donde el conocimiento es útil para el cerebro [4].

El camino del aprendizaje en cualquier proceso de formación y adquisición de conocimientos es mucho más expedito cuando dicho aprendizaje se aplica confiriéndole significado pues esto permite que tienda a ubicarse en la memoria a largo plazo [4]. De allí por qué, la relación entre el conocimiento técnico y los problemas prácticos que provea la matemática es tan importante pues se evidencia aplicación, conocimiento y significado. Las relaciones didácticas entre el conocimiento y su aplicación se pueden apropiar por un camino más sencillo en la medida en que se puedan construir y, además, comparar bajo determinados criterios que apunten a una solución óptima. La estrategia de la comparación de soluciones, desde la óptica de los paradigmas de programación, permite que se puedan enfrentar conceptos para bien de la apropiación y asimilación de conocimientos por parte del estudiante. Cuando se establecen este tipo de comparaciones se le permite al estudiante encontrar aún más significado al conocimiento adquirido y relacionarlo no solo con los conocimientos previos sin con los conocimientos transversales que sirven de base para resolver el mismo objeto de conocimiento. El contenido de este artículo busca que los estudiantes no solo verifiquen, prueben y conozcan soluciones a problemas determinados sino que las comparen en dos paradigmas diferentes desde una óptica práctica basándose más en la lógica y el razonamiento que en procedimientos mecánicos y enteramente memorísticos [5].

La gran utilidad que tiene este artículo es que afianza, desde dos ópticas completamente diferentes, la idea de que el proceso de aprendizaje de la aplicación de los recursos tecnológicos modernos se fundamenta en las posibles relaciones que se pueden establecer entre determinados problemas (caracterizados por su naturaleza) y las soluciones que se puedan implementar para resolverlos. Precisamente una de las dificultades que tiene el docente en estos tiempos modernos, especialmente los docentes de las áreas de aplicación tecnológica, es lograr que el estudiante no solo aplique los conocimientos adquiridos sino que los confronte con conocimientos transversales que están al mismo nivel de utilización [6]. Se ha acudido a las matemáticos como el proveedor de problemas, específicamente del hallazgo de ternas pitagóricas, porque es uno de los terrenos donde el estudiante puede articular teoría con práctica, a lo cual se le puede adicionar el concepto de práctica múltiple, es decir, la práctica que se deriva de aplicar diferentes paradigmas en la resolución de un mismo problema. Los procesos de aplicación comparativa de soluciones tienden a afianzar los conocimientos adquiridos en los procesos de aprendizaje [7].

En cuanto a la pregunta de investigación asociada al contenido del presente artículo, ésta podría describirse en los siguientes términos: ¿es posible encontrar caminos que afiancen el conocimiento adquirido y su significado de aplicación que se basen en la comparación de objetos de estudio similares a partir de los paradigmas de programación que refuercen tanto los conocimientos previos como los nuevos conocimientos? La respuesta es el contenido del artículo donde se presenta una posible alternativa en tratándose de paradigmas de programación y su relación con un problema específico: el hallazgo de ternas pitagóricas. La respectiva hipótesis plantea que es posible lograrlo de manera efectiva en campos como el de la programación de computadores si se formulan soluciones desde, por ejemplo, el paradigma funcional y el paradigma imperativo, que tiendan a resolver un mismo problema (como el hallazgo de ternas pitagóricas).

El contenido de este artículo se fundamenta en la investigación científica y tecnológica apoyado en la teoría del aprendizaje significativo [4], teoría del aprendizaje por descubrimiento [5] así como la tendencia Active Learning y Problem Based Learning que propenden no sólo por hacer al alumno partícipe y responsable de su propio proceso de aprendizaje sino de relacionar permanentemente los conceptos que se revisan desde la teoría con sus aplicaciones prácticas. Adicionalmente, este artículo presenta de una forma aplicada, la estrecha relación que existe entre la matemática, proveedora de problemas, y la programación, como instancia de aquella en lo puramente tecnológico que posibilita algunas soluciones, como una manera de concederle significado a la teoría que se deriva de los paradigmas de programación. Se plantean en el contenido de este artículo dos propuestas de solución algorítmica al problema de la búsqueda y hallazgo de ternas pitagóricas, es decir, aquellos números que cumple con el teorema de Pitágoras. Con estas propuestas se realiza un proceso de comparación en lo sintáctico, en lo lógico y en lo procedimental con el ánimo de afianzar los criterios

que permiten la construcción de cada una de ellas desde la perspectiva de cada paradigma seleccionado.

Para la implementación se ha acudido al lenguaje C++ arista imperativa y a su entorno DevC++ versión 5.8.0 en lo que corresponde a la solución imperativa; por su parte se ha acudido al lenguaje Scheme en su entorno DrRacket versión 6.1 en lo que compete a la solución funcional. De la misma manera el contenido de este artículo es uno de los productos del proyecto de investigación "Desarrollo de un modelo metodológico para la asignatura Programación I en Ingeniería de Sistemas basado en aprendizaje significativo, aprendizaje por descubrimiento y el modelo 4Q de preferencias de pensamiento" Código 6-15-10 de la Vicerrectoría de Investigaciones, Innovación y Extensión de la Universidad Tecnológica de Pereira.

II. Marco teórico

El primer paradigma que se ha utilizado para elaborar una solución al problema del hallazgo de las ternas pitagóricas se conoce como el paradigma funcional, que se basa en la matemática derivada del cálculo Lambda y su concepto más importante es, precisamente, el concepto de función del cual deriva su nombre que constituye la base para la elaboración de soluciones [8]. Este concepto de función puede definirse como un micro programa que alcanza un micro objetivo y que, al formar parte de un programa, junto con otras funciones permite lograr grandes objetivos. La función es un conjunto definido de funciones, que puede tener nombre o puede ser anónima, que puede estar habilitada para recibir argumentos, ser identificada por un nombre y puede, igualmente, retornar valores sin tener que acudir a la memoria del computador como intermediario para definir sus estados de almacenamiento [2]. Lo verdaderamente importante en la programación funcional es la manera como el cuerpo de las funciones interactúa para lograr un resultado que se constituye en el aporte de cada función en un programa.

Desde una óptica muy sencilla, podría decirse que un programa construir sobre los fundamentos del paradigma funcional, consiste en un conjunto de funciones independientes pero, al tiempo, interdependientes en relación con el logro de un determinado objetivo común. La atomización de una solución parte del principio "divide y vencerás" que es un principio de programación según el cual al dividir un gran objetivo a alcanzar en pequeños objetivos, es posible resolver los pequeños objetivos (con funciones) y luego enlazarlos de manera que el gran objetivo original quede resuelto. Es de anotar que resolver cada uno de los pequeños objetivos es mucho más sencillo que resolver el gran objetivo. La solución a esos pequeños objetivos es lo que provee la necesidad de construir funciones y de capitalizar al máximo los resultados de las funciones por encima de las formas posibles de almacenamiento.

Una de las potencialidades mayores que tiene la programación funcional es el concepto de recursión. Como recursión, o recursividad, se entiende la propiedad que provee un lenguaje de programación de permitir que una función se llame a sí misma. En términos matemáticos, la recursión se define como la notación a través de la cual se puede definir una función en sus propios términos. Bajo la perspectiva funcional, la recursión se constituye en uno de los mayores recursos con que cuentan las funciones para implementar procesos cíclicos que no queden en "loops" infinitos, sino que, bajo una determinada lógica y siempre de acuerdo a las necesidades algorítmicas, finalice exitosamente retornando los resultados que se requieran como producto del proceso iterativo. En este artículo se ha utilizado el lenguaje Scheme en el entorno DrRacket por su alta capacidad para procesar datos, por la facilidad de implementar procesos recursivos eficientes y por la simplicidad que provee en el manejo de datos numéricos.

Por su parte, cuando se habla del paradigma de programación imperativa se hace referencia a uno de los modelos computacionales más estables y que han tenido mayor maduración debido a la cantidad de lustros que han permitido su refinamiento y que partieron de los fundamentos que brindaran Alan Turing (1912-1954) y John von Neumann (1903-1957) [1]. El paradigma de programación imperativo se fundamenta en el concepto de máquina de estados, de acuerdo a la cual, un programa se reduce a una cantidad de instrucciones que posibilitan la modificación de los contenidos de la memoria de un computador, que es el espacio físico - lógico donde se almacenan dichos contenidos en formato binario. La programación imperativa basa sus implementaciones en la utilización de tres esquemas conocidos como estructuras básicas, razón por la cual este paradigma también se conoce como paradigma imperativo. Estas tres estructuras, que son el fundamento para la implementación de soluciones, son las secuencias de instrucciones, los condicionales y los ciclos.

Las secuencias de instrucciones se pueden describir como instrucciones escritas y ubicadas bajo un determinado orden que, en las condiciones que describa el problema determinado, es inalterable para la correcta ejecución de la solución (Van Roy, 2008). Una instrucción se ejecuta completamente sólo después de que la anterior instrucción se ha ejecutado completamente y antes de que se comience a ejecutar la siguiente instrucción. Por su parte, la estructura condicional, permite la escogencia de uno de dos caminos lógicos dependiendo de una condición que afecte la decisión y que tiene relación con ambos caminos. La condición normalmente se escribe en términos de operadores aritméticos, relacionales y booleanos. Los ciclos, la tercera estructura básica, posibilitan repetir la ejecución de un conjunto de instrucciones, una cantidad determinada de veces dependiendo, tal como la segunda estructura, de una condición que la regule. Con la programación imperativa se busca que los programadores utilicen al máximo, las potencialidades de las estructuras, de forma que, con su combinación, anidamiento y utilización, se puedan implementar soluciones que satisfagan objetivos computables.

Desde la más simple de las definiciones, un programa basado en el paradigma imperativo se puede concebir como un conjunto de instrucciones independientes que buscan alcanzar un objetivo que, a su vez, satisface un enunciado y que se fundamenta en la interacción y modificación de los contenidos del estado actual de la memoria del computador. Cuando se aprovechan las estructuras básicas de este paradigma es posible llegar a establecer estándares en la construcción de algoritmos de manera que diferentes lógicas aproximen los elementos de juicio que intervinieron en su planteamiento inicial. De la buena utilización de la estructuras y del diseño que se la haga al programa, dependerá el tiempo que se necesite para corregir errores lógicos y que el mantenimiento del código sea eficiente.

La programación imperativa se sirve del concepto de función para lograr programas fáciles de mantener, de modificar y de actualizar, toda vez que las funciones permiten atomizar el objetivo general y construir la solución a partir de objetivos menores que, enlazados, resuelven el problema que se haya planteado [9]. En la programación imperativa la metodología "divide and conquer" es, igualmente, efectiva que en la programación funcional pues permite, en ambos paradigmas, construir soluciones que sean coherentes, simples y entendibles, las tres condiciones que se requieren para que un programa logre el objetivo por el mejor de los caminos. Esta estrategia es el camino para que la solución se atomice a partir del buen uso de las estructuras.

Se ha utilizado el lenguaje C++ en su arista imperativa para la construcción, validación y puesta a punto de la solución planteada, para detectar y hallar ternas pitagóricas, por considerarse el concepto de función, un lenguaje de programación flexible, fácil de codificar y muy fácil de implementar a la luz de las estructuras básicas. Tanto el entorno DevC++ para la codificación de lenguaje C++ como el entorno DrRacket para la codificación de lenguaje Scheme, se consideran suficientemente sólidos en lo sintáctico, veloces en lo compilativo y eficientes en la ejecución de los programas. Cabe anotar que ambos lenguajes permiten la programación orientada a objetos que constituye el tercer paradigma y cuyo alcance va más allá de los objetivos del presente artículo.

El problema que se ha escogido para implementar dos soluciones y compararlas bajo determinados criterios, es el que permite encontrar ternas pitagóricas en un rango determinado. Una terna pitagórica es aquel trío de números que cumple con el teorema de Pitágoras según el cual "la suma de los cuadrados de los catetos es igual al cuadrado de la hipotenusa" [10] que escrito en términos matemáticos correspondería a $a^2+b^2=c^2$ para valores enteros a, b y c como medida de los catetos y de la hipotenusa. Cabe anotar que en la igualdad expuesta, los catetos estarían identificados por las letras a y b y la hipotenusa estaría identificada por la letra c. en el ámbito puramente matemático, el problema consiste en encontrar ternas de números enteros (también conocidos como naturales) que satisfagan la ecuación base de este teorema. Es de anotar, igualmente, que este teorema se hace efectivo en los triángulos rectángulos o sea aquellos en los cuales uno de sus ángulos internos es igual a 90º. La idea de este artículo es establecer elementos de juicio comparativos que, a partir de una solución funcional y de una solución imperativa, permitan develar ventajas y desventajas de cada uno de los paradigmas.

III. Metodología

A. Descripción

Dado que el objetivo de este artículo es establecer criterios comparativos entre dos soluciones algorítmicas a un mismo problema, se presentará la lógica de construcción de ambas soluciones y, posteriormente, los comparativos se harán sobre la codificación de cada una a la luz de un paradigma y utilizando como medio un lenguaje de programación. Tanto para la solución funcional como para la solución imperativa, se han planteado las diferentes variantes de validación que se necesitan para realizar el proceso de búsqueda y hallazgo de las ternas pitagóricas, según lo establecido en la definición.

La lógica de solución parte de las mismas bases y por ello el rango de posibles valores a validar está dado por la tabla I.

Tabla I
Posibles Valores A Validar

01ª 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

La tabla I pareciera sugerir que se requieren dos procesos cíclicos (uno dentro de otro), de manera que se pueda hacer el recorrido apropiadamente. En un proceso cíclico externo deberá recorrerse desde 1 hasta n (de 1 en 1) y, dentro de este ciclo deberá existir un proceso cíclico interno que recorra igualmente los valores desde 1 hasta n (también de 1 en 1).

Algorítmicamente, la secuencia de los números presentados se puede resumir de la siguiente forma:

Para a = 1 hasta n (1)
    Para b = 1 hasta n (1)
        Validar a^2+b^2

La validación de la expresión a^2+b^2, que se constituye en la esencia del problema, se puede realizar de dos formas de manera que cada una esté asociada a cada paradigma tal como lo presenta a continuación.

Solución Funcional
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
Obtener parte entera de raíz cuadrada obtenida
Compararla 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

Solución Imperativa

```
a=1
Mientras a<=n
   b=1
    Mientras b<=n
         Obtener resultado de a^2+b^2
         Almacenar raíz cuadrada de a^2+b^2 en variable entera
         Almacenar raíz cuadrada de a^2+b^2 en variable real
         Comparar los dos valores obtenidos
         Si estos dos valores son iguales
         Desplegar en pantalla la terna pitagórica
         Si no son iguales
         Continuar con la siguiente combinación
         Fin Si
         b=b+1
   Fin Mientras
   a=a+1
Fin Mientras
```

La tabla II muestra la relación entre la lógica funcional y la lógica imperativa en la parte medular de este algoritmo.

TABLA II COMPARACIÓN PARTE MEDULAR

Paradigma Funcional	Paradigma Imperativo
Calcular el resultado de a^2+b^2 Obtener raíz cuadrada de a^2+b^2 Obtener parte entera de la raíz cuadrada obtenida Comparar la parte entera de dicha raíz con la raíz como tal	*Obtener resultado de a^2+b^2 Almacenar raíz cuadrada de a^2+b^2 en una variable entera Almacenar raíz cuadrada de a^2+b^2 en una variable real Comparar los dos valores obtenidos*

En esta parte, desde lo funcional (debido a que se manejan procesos y resultados de funciones) se acude a la obtención del resultado de la raíz cuadrada de la expresión algebraica y luego se compara con la parte entera de la raíz cuadrada obtenida. En la parte imperativa (aprovechando que se maneja el concepto de tipos de datos) se almacena el resultado de la raíz cuadrada de la expresión tanto en una variable entera como en una variable real y luego se comparan los dos valores. En ambos casos, tanto en lo funcional como en lo imperativo, la verificación de igualdad entre los dos valores obtenidos (sea por vía de los resultados de funciones o sea por vía de los valores almacenados en variables de tipo diferente) se constituye en la confirmación de la existencia de una terna pitagórica en la cual los valores a y b, que conforman el valor de los catetos, están definidos por los ciclos y el valor de la hipotenusa está definida por la raíz cuadrada de la raíz cuadrada de la expresión a^2+b^2.

B. Aplicación

Tal como se ha planteado, para la implementación de la solución funcional se acudió al lenguaje Scheme IDE DrRacket versión 6.1 y para la implementación de la solución imperativa se acudió al lenguaje C++ IDE DevC++ Versión 5.8.0. Ambos son entornos integrados de desarrollo para Sistema Operativo Windows. Se presentan a continuación la solución funcional y la solución imperativa.

Solución Funcional

```
; PROGRAMA QUE ENCUENTRA TERNAS
; PITAGÓRICAS EN EL 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)
```

Solución Imperativa

```
// Programa que encuentra ternas pitagóricas en el
// rango 1 a n para un valor n definido

// Inclusión de librerías
#include <iostream>
#include <conio.h>
#include <math.h>

using namespace std;

// Función que calcula la suma a^2+b^2
int suma_factores(int a, int b)
{                          // Inicio función suma_factores
   int r;                  // Variable local
   r=pow(a,2)+pow(b,2);   // Suma de cuadrados
   return(r);
// Retorna suma de cuadrados
}
                           // Fin función suma_factores

// Función que confirma la existencia de una terna
// pitagórica
int respuesta(int a, int b)
{
                                       // Inicio función respuesta
  float rf;
            // Variable local real
   int ri;
// Variable local entera
// raiz cuad (real) de la suma de cuadrados
   rf=sqrt(suma_factores(a,b));
// raiz cuad (entera) de la suma de cuadrados
   ri=sqrt(suma_factores(a,b));
   if(rf==ri)
// si son iguales
      return(1);                             // entonces retorne 1
(Verdadero)
      return(0);                   // Sino, retorne 0 (Falso)
}
// Fin función respuesta

// Función que despliega resultados (solo ternas)
int muestraterna2(int a, int b)
{
                                       // Inicio función muestraterna2
   int c;
            // Variable local
```

```
// Almacena parte entera raiz de suma de cuadrados
   c=int(sqrt(suma_factores(a,b)));
   cout<<"\nTerna Pitagórica --> (";          // Título
// Muestra la terna formato (a,b,c)
   cout<<a<<", "<<b<<", "<<c<<")\n";
}
                              // Fin función muestraterna2

// Función que despliega resultados
// (formato completo)
int muestraterna1(int a, int b)
{
                              // Inicio funcion muestraterna1
   int c;
            // Variable local
   // Muestra cuadrados
   cout<<"\n"<<a<<"^2 = "<<pow(a,2);
   cout<<", "<<b<<"^2 = "<<pow(b,2);
   // Suma de cuadrados
   cout<<"\n"<<pow(a,2)<<" + "<<pow(b,2)<<" = ";
// Obtiene la raiz cuadrada de la suma de cuadrados
   c=pow(a,2)+pow(b,2);
   cout<<c<<"\n";
   cout<<"Raiz Cuad de "<<c<<"="<<sqrt(c) <<"\n";
   cout<<"Terna Pitagórica --> ("; // Título
// Muestra terna
   cout<<a<<", "<<b<<", "<<sqrt(c)<<")\n\n";
}
            // Fin función muestraterna1

// Función que establece los ciclos de validación
int ciclos(int n)
{
                    // Inicio función ciclos
   int i, j;        // Variables locales
   i=1;             // Valor inicial ciclo externo
   while(i<=n)      // Condición ciclo externo
   {
// Inicio ciclo externo
     j=1;           // Valor inicial ciclo interno
     while(j<=n)              // Condición ciclo interno
     {
// Inicio ciclo interno
                    // Si se cumple el teorema de Pitágoras
       if(respuesta(i,j))
      // Muestre la terna pitagórica
            muestraterna1(i,j);
       j++;         // Incremente índice ciclo interno
       }            // Fin ciclo interno
      i++;          // Incremente índice ciclo externo
    }               // Fin ciclo externo
   return(1);       // Al final, retorne 1
         }          // Fin función ciclos

// Función principal
int main( )
{
                    // Inicio función principal
   int n=15;        // Valor de n (rango de 1 a n)
   ciclos(n);       // Llamado a la función ciclos
   getche( );       // Pausa para ver resultados
}                   // Fin función principal
```

Las tablas IIIA y IIIB presentan una descripción breve de las funciones que se han codificado en cada una de las soluciones.

TABLA III A. DESCRIPCION DE FUNCIONES – PARADIGMA FUNCIONAL

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)

TABLA III B. DESCRIPCION DE FUNCIONES - PARADIGMA IMPERATIVO

Función	Descripción
suma_factores	En esta función se calcula el resultado de la expresión a^2+b^2 siendo a y b los valores enteros que se reciben como parámetros
respuesta	Esta es la función clave dentro del programa ya que 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
muestraterna2	En esta función se muestra la terna pitagórica en el formato (a,b,c)
muestraterna1	En esta función se muestra la terna pitagórica acorde con la estructura del siguiente ejemplo
	6^2 = 36, 8^2 = 64 *36 + 64 = 100* *Raíz Cuadrada de 100 = 10* *Terna Pitagórica --> (6, 8, 10)*
ciclos	Esta es la función que recorre los ciclos (externo e interno) que permiten generar los números con los valores analizados
main( )	Esta es la función principal desde donde se define el valor final sobre el cual se quieren hacer las validaciones de las ternas pitagóricas

IV. Resultados

Las tablas IV A y IV B muestran los resultados obtenidos en los programas desarrollados para encontrar las ternas pitagóricas.

TABLA IV A. RESULTADOS OBTENIDOS – PROGRAMA FUNCIONAL

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)*

TABLA IV B-. RESULTADOS OBTENIDOS – PROGRAMA IMPERATIVO

Función	Resultado Obtenido
Usando la función muestraterna2	C:\Users\EQUIPO\Desktop\ArtÝculos Listos\Arti... Terna Pitag¾rica --> (3, 4, 5) Terna Pitag¾rica --> (4, 3, 5) Terna Pitag¾rica --> (6, 8, 10) Terna Pitag¾rica --> (8, 6, 10)
Usando la función muestraterna1	C:\Users\EQUIPO\Desktop\ArtÝculos Listos\Articulo Pro... 3^2 = 9, 4^2 = 16 9 + 16 = 25 Raiz Cuadrada de 25 = 5 Terna Pitag¾rica --> (3, 4, 5) 4^2 = 16, 3^2 = 9 16 + 9 = 25 Raiz Cuadrada de 25 = 5 Terna Pitag¾rica --> (4, 3, 5) 6^2 = 36, 8^2 = 64 36 + 64 = 100 Raiz Cuadrada de 100 = 10 Terna Pitag¾rica --> (6, 8, 10) 8^2 = 64, 6^2 = 36 64 + 36 = 100 Raiz Cuadrada de 100 = 10 Terna Pitag¾rica --> (8, 6, 10)

Como puede notarse en la tabla IV A y IV B, los resultados son similares vistos desde la óptica del usuario, es decir, desde la persona que ejecuta el programa para obtenerlos. La diferencia fundamental está en la manera como se concibió la lógica para aprovechar cada uno de los recursos que cada paradigma provee y, de esa forma, encontrar una solución efectiva al problema planteado.

V. DISCUSIÓN

El espíritu del presente artículo radica en establecer algunos criterios que permitan la comparación de los programas presentados como soluciones al objetivo planteado, de manera que se puedan correlacionar las soluciones en cada paradigma, a la luz de la lógica, de los recursos sintácticos y de las posibilidades que tanto el paradigma como el lenguaje proveen. La tabla VII muestra la forma como se ha implementado la función para calcular el resultado de la expresión a^2+b^2 que es la base para la demostración del teorema de Pitágoras, ya que ésta es el resultado de sumar el valor de los catetos elevados cada uno al cuadrado. Posteriormente es a este valor, al cual se le va a obtener la raíz cuadrada y de esa forma podremos verificar la existencia de una terna pitagórica. Luego la expresión a^2+b^2, su validación correcta y su apropiada generación de resultado es el centro de toda la demostración matemática y de los programas que se presentan en este artículo.

En su versión funcional, esta función acude a la función de librería *expt* que retorna el resultado de elevar un valor a una potencia determinada (que en este caso es la potencia 2). El resultado de sumar el cuadrado de cada uno de los valores recibidos como argumentos es el valor que retorna esta función que aprovecha dicho cálculo para retornarlo sin variables intermedias que lo almacenen. Por su parte en la versión imperativa se hace muy útil, aunque pudiera no ser estrictamente necesario, que dicho resultado se almacene en una variable local para retornarse por parte de la función. Se ha utilizado en esta versión la función *pow* que se encuentra en la librería *math.h* que es la que contiene, en lenguaje C++, las funciones matemáticas. Es de anotar que en ambas funciones se reciben los mismos valores, en la versión funcional como argumentos y en la versión imperativa como parámetros. Bien puede decirse que, a pesar de basarse en paradigmas diferentes, son dos funciones simples que contienen las similaridades del retorno, pues en ambas funciones son equivalentes, sin embargo acuden a funciones diferentes y la función como tal, mantiene la filosofía que caracteriza cada paradigma: priorizar el proceso y priorizar el almacenamiento para los paradigmas funcional e imperativo, respectivamente.

TABLA V. FUNCION QUE CALCULA LA SUMA a^2+b^2

Paradigma	Código
Funcional	*; Función simple que calcula la suma a^2+b^2* *(define (suma a b)* *(+ (expt a 2)(expt b 2))* *)*

Imperativo	*// Función que calcula la suma a^2+b^2*
	int suma_factores(int a, int b)
	{ // Inicio función suma_factores
	int r; // Variable local
	r=pow(a,2)+pow(b,2); // Suma de cuadrados
	return(r) // Retorna suma de cuadrados
	}
	// Fin función suma_factores

En la tabla VI se presenta la función que confirma la existencia de una terna pitagórica. En la versión funcional lo que se hace es comparar el resultado de la raíz cuadrada

de la suma de los cuadrados con el valor entero de dicha raíz cuadrada. Si estos valores son iguales, significa que se ha encontrado una terna pitagórica. Para confirmarlo se retorna un valor 1 como expresión de un valor lógico Verdadero y para negarlo se retorna un valor 0, como expresión de un valor lógico Falso. Es de anotar que se pudo haber acudido a la utilización de las constantes booleanas #t (para un valor lógico Verdadero) y #f (para un valor lógico Falso) sin embargo, para mantener alguna relación con la codificación en el paradigma imperativo, se optó por utilizar el mismo mecanismo de confirmación (Verdadero y Falso) en ambas funciones equivalentes.

Por su parte, en la versión imperativa, si bien es cierto que la función recibe los mismos parámetros (llamados argumentos en la versión funcional) también lo es que la lógica es un poco diferente. En esta versión se aprovecha el concepto de tipo de dato que es completamente ausente en la versión funcional. A la luz de este concepto se declaran dos variables locales: una de tipo real llamada *rf* (como para indicar que es la respuesta tipo *float*) y otra de tipo entero llamado *ri* (para indicar que es la respuesta tipo *int*). En cada una de estas variables se almacena el mismo valor que corresponde al resultado de calcular la raíz cuadrada de la suma de los cuadrados de los catetos (que para el caso corresponden a los valores a y b recibidos como parámetros). El hecho de que el mismo valor se almacene en una variable entera y en una variable real, permite que posteriormente se comparen estos resultados. Si son iguales, significa que se ha encontrado una terna pitagórica puesto que solo serán iguales cuando el resultado de la raíz cuadrada de la suma de los cuadrados sea un valor entero, dado que solo así se podrán ignorar los decimales que pudieren acompañar el valor obtenido. Finalmente, y en consonancia con la versión funcional, se retorna un valor 1 (Verdadero) o 0 (Falso) dependiendo de que se cumpla o no el condicional que pregunta si el valor entero es igual al valor real.

TABLA VI. FUNCIÓN QUE CONFIRMA LA EXISTENCIA DE UNA TERNA PITAGÓRICA

Paradigma	Código
Funcional	; 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))

Imperativo	// Función que confirma la existencia de una terna
	// pitagórica
	int respuesta(int a, int b)
	{ // Inicio función respuesta
	float rf; // Variable local real
	int ri; // Variable local entera
	// raiz cuad (real) de la suma de cuadrados
	rf=sqrt(suma_factores(a,b));

	// raiz cuad (entera) de la suma de cuadrados
	ri=sqrt(suma_factores(a,b));
	if(rf==ri) // si son iguales
	return(1); // entonces retorne 1 (Verdadero)
	return(0); // Sino, retorne 0 (Falso)
	} // Fin función respuesta

En la tabla VII se presenta el conjunto de instrucciones que generan y controlan los procesos cíclicos que se explicaron en la tabla I. En estas dos versiones sí son notorias las características de cada uno de los paradigmas. En la versión funcional se acude a una alta utilización del concepto de recursión. Para generar los ciclos que se necesitan, se construye primero una función recursiva llamada *cicloext* que controla el 1º valor que se va a utilizar en el proceso validatorio. Desde esta función se invoca a la función *cicloint*, que también es otra función recursiva, que genera los valores correspondientes al otro de los catetos desde una posición lógica subordinada al llamado y control de la función *cicloext*. En caso de que se encuentre una terna pitagórica se procede a mostrar el resultado de la terna hallada. Para este fin se han construido dos funciones de salida, una que muestra los resultados concretos y la otra que muestra adicionalmente el proceso que confirma la existencia de dicha terna. La solución imperativa es comparativamente sencilla dado que los dos procesos cíclicos se han anidado utilizando las estructuras cíclicas que provee el ciclo *while* y que permite que en una sola función podamos tener control de ambas variables, la que representa un cateto y la que representa el otro cateto, y de todo el proceso de manera condensada. En esta función se han declarado dos variables locales, una de ellas servirá como índice controlador del ciclo externo y la otra servirá como índice controlador del ciclo interno, ambas gobernadas por el mismo valor tope. Al igual que en la solución funcional, en caso de que se encuentre una terna pitagórica entonces se muestran los resultados con cualquiera de las dos funciones que para tal fin se han previsto: una que presenta los resultados de forma simplificada y la otra que muestra, además de los resultados, el proceso que permite confirmar la demostración de la terna pitagórica.

TABLA VII FUNCIONES CICLICAS

Paradigma	Código
Funcional	*; Función recursiva (ciclo 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 (ciclo 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))))*
Imperativo	*// Función que establece los ciclos de validación*

	*int ciclos(int n)
	{ // Inicio función ciclos
	int i, j; // Variables locales
	i=1; // Valor inicial ciclo externo
	while(i<=n) // Condición ciclo externo
	{ // Inicio ciclo externo
	j=1; // Valor inicial ciclo interno
	while(j<=n) // Condición ciclo interno
	{ // Inicio ciclo interno
	// Si se cumple el teorema de Pitágoras
	if(respuesta(i,j))
	// Muestre la terna pitagórica
	muestraterna1(i,j);
	j++; // Incremente índice ciclo interno
	} // Fin ciclo interno
	i++; // Incremente índice ciclo externo
	} // Fin ciclo externo
	return(1); // Al final, retorne 1
	} // Fin función ciclos*

Finalmente la tabla VIII presenta lo que podría llamarse, en cada solución, el llamado principal (para la versión funcional) o la función principal (para la versión imperativa). En la versión funcional se hace un llamado enviando un valor determinado como tope para las búsquedas de las ternas pitagóricas. Ese llamado permite invocar una función que, a su vez, invoca a la función que controla el ciclo externo recursivo que se ha implementado. En la parte imperativa, la función principal, también conocida como *main( )*, tiene declarada una variable local que es la variable en al cual se almacena el tope de evaluación para los posibles valores de los catetos, se invoca a la función ciclos (que implementa en una sola función los dos ciclos anidados que, en la versión funcional, requiere dos funciones recursivas separadas) y finalmente se acude a una instrucción para esperar la verificación del resultado.

TABLA VIII. FUNCION PRINCIPAL

Paradigma	Código
Funcional	*; Función inicial (define (inicio i j n) (cicloext i j n)) ; Llamado inicial (10 = valor definido para n) (inicio 1 1 10)*

Imperativo	*// Función principal
	int main( )
	{ // Inicio función principal
	int n=15; // Valor de n (rango de 1 a n)
	ciclos(n); // Llamado a la función ciclos
	getche( ); // Pausa para ver resultados
	}*

Es de anotar que este programa se ha realizado sin utilizar recursos gráficos de usuario que permitan una interfaz gráfica un poco más amable para el usuario pues se ha desarrollado para aprovechar los recursos sintácticos, lógicos y tecnológicos de cada paradigma.

VI. Conclusiones

- Para concederle significado a los conceptos que se derivan del estudio de la programación de computadores, es muy importante buscar ejemplos que tengan relación práctica con el conocimiento y el área de las matemáticas son un excelente espacio de donde se puede acceder a diversos problemas que cumplen con este objetivo.

- Se constituye en un reto para los docentes de programación y de matemáticas, a nivel universitario, permitir que los estudiantes establezcan ese nexo de significado entre lo teórico y lo práctico y proveerles de problemas que puedan ser resueltos con las herramientas básicas que la programación de computadores provee.

- Tanto la programación funcional como la programación imperativa, cada una con sus características, ventajas y limitaciones, son excelentes fuentes para construir soluciones a problemas de las matemáticas.

- Aunque en este artículo se ha acudido a dos herramientas específicas (DrRacket y DevC++), no se descarta que otras soluciones puedan ser implementadas con mayor eficiencia desde diferentes lenguajes y diferentes entornos.

- Es de gran importancia que los estudiantes de las áreas de programación reciban tanto el conocimiento fundamental como su respectiva instancia tecnológica y su aplicación práctica para que el conocimiento no quede ad portas del olvido sino que pueda ubicarse en la memoria a largo plazo producto de su aplicación en situaciones prácticas.

- El concepto de función simplifica la construcción de soluciones tanto cuando se utiliza el paradigma funcional a través de algún lenguaje de programación como cuando se usa el paradigma imperativo.

- Vale la pena resaltar que las funciones de interfaz son altamente similares en ambas soluciones y que las funciones operativas o de cálculo aprovechan al máximo las características de cada paradigma para posibilitar la implementación de una solución que, a la luz de cada paradigma, se aproxime a la ideal

Referencias

[1] Trejos Buriticá, O. I. (2012). Aprendizaje en Ingeniería: un problema de comunicación. Pereira (Colombia): Tesis Doctoral - Universidad Tecnológica de Pereira.

[2 Van Roy, P. (2008). Concepts, Techniques and Models of Computer Programming. Estocolmo: Université catholique de Louvain.

[3 Bruner, J. S. (1969). Hacia un teoría de la instrucción. Ciudad de México: Hispanoamericana.

[4] Ausubel, D. (1986). Sicología Educativa: Un punto de vista cognoscitivo. Ciudad de México. Trillas.

[5] Bruner, J. S. (1991). Actos de significado. Madrid: Alianza Editorial.

[6] Paz Penagos, H. (2014). Aprendizaje autónomo y estilo cognitivo: diseño didáctico, metodología y evaluación. Revista Educación en Ingeniería, 9(17), 53-65.

[7] Paz Penagos, H. (2009). How to develop metacognition through problem solving in higher education? Revista de Ingeniería e Investigación, 31(1), 75-80.

[8] Felleisen, M. et al. (2013). How to design programs (2ª Ed). Boston. MIT Press

[9] Trejos Buriticá, O. I. (2006). Fundamentos de Programación. Pereira: Papiro.

[10] Sparks, J. (2008). The Pythatorean Theorem. Bloomington (Indiana): AuthorHouse.

Omar Ivan Trejos Buriticá. Ingeniero de Sistemas. Especialista en Instrumentación Física. Magister en Comunicación Educativa. PhD en Ciencias de la Educación, actualmente es docente de tiempo completo de la Universidad Tecnológica de Pereira.