

**UNIVERSIDAD DE LAS FUERZAS ARMADAS**

**ESPE**

**TEMA:**

# **RECURSIVIDAD**

**ASIGNATURA:**

**ESTRUCTURA DE DATOS**

**ING. FERNANDO SOLIS**

**DOCENTE TC DE DCCO**

**Sangolquí- Ecuador, noviembre de 2020**

**ÍNDICE**

**TABLA DE CONTENIDOS**

[**RECURSIVIDAD** 1](#_Toc64837030)

[**A.** **ANTECEDENTES** 3](#_Toc64837031)

[**B.** **OBJETIVO** 3](#_Toc64837032)

[**C.** **DESARROLLO** 3](#_Toc64837033)

[**1.** **PARA LA ELABORACIÓN DEL RESUMEN** 3](#_Toc64837034)

[**a)** **Realice la lectura exploratoria del recurso teórico recomendado.** 3](#_Toc64837035)

[**b)** **Identifique las ideas más importantes del contenido estudiado.** 3](#_Toc64837036)

[c) **Elabore una frase sinóptica, concisa, que abarque el contenido total y que responda a la pregunta ¿de qué trata la lectura?** 6](#_Toc64837037)

[**d)** **Reúnalas ideas más importantes y, de existir, los ejemplos más sobresalientes.** 6](#_Toc64837038)

[**e)** **Elimine los párrafos redundantes o accesorios y los ejemplos abundantes.** 7](#_Toc64837039)

[**f)** **Integre las ideas importantes en un escrito coherente que exprese, con sus propias palabras, la síntesis de o que se explica en el tema, sin cambiarlas ideas centrales.** 8](#_Toc64837040)

[**g)** **Tenga en cuenta que el resumen puede repetir literalmente las ideas principales tal como las expone el autor, siempre y cuando se las vincule adecuadamente.** 9](#_Toc64837041)

[**2.** **RESOLUCIÓN DE INFORMES DE LABORATORIOS / PRÁCTICAS O TALLERES.** 9](#_Toc64837042)

[**a)** **Realice la lectura del ejercicio del recurso teórico que corresponda.** 9](#_Toc64837043)

[**b)** **Acceda a otra información teórica en relación con el tema a desarrollar. Utilice diferentes estrategias para la búsqueda, selección y organización de información.** 10](#_Toc64837044)

[**c)** **Identifique los conceptos principales y secundarios de la temática.** 11](#_Toc64837045)

[**d)** **Utilice la metodología indicada en la resolución de los ejercicios y del Documento o Informe del Laboratorio.** 12](#_Toc64837046)

[**e)** **Elabore el documento o Informe del Laboratorio con los ejercicios indicados y en el formato dado.** 12](#_Toc64837047)

[**D.** **CONCLUSIONES** 13](#_Toc64837048)

[**E.** **RECOMENDACIONES** 13](#_Toc64837049)

[**BIBLIOGRAFÍA** 13](#_Toc64837050)

# **ANTECEDENTES**

La recursividad es un concepto fundamental en el área de matemáticas y en ciencias de la computación. Es una alternativa diferente para implementar estructuras de repetición que no utilicen estructuras iterativas. Esta técnica de programación se puede usar en toda situación en la cual una solución pueda ser expresada como una secuencia de pasos o transformaciones guiadas por un conjunto de reglas no ambiguas.

Este tipo de estructuras repetitivas tienen la capacidad de abarcar problemas muy complejos y dar una solución corta y limpia en código. Según (Aguilar, 2008), Una solución recursiva es normalmente menos eficiente en términos de tiempo de computadora que una solución iterativa debido a las operaciones auxiliares que llevan consigo las llamadas suplementarias a las funciones; sin embargo, en muchas circunstancias el uso de la recursión permite a los programadores especificar soluciones naturales, sencillas, que serían, en caso contrario, difíciles de resolver.

Los algoritmos o funciones recursivas ofrecen soluciones cortas, fáciles y elegantes a costa de un mayor tiempo de ejecución y reserva de memoria debido a las sucesivas llamadas que hará la función recursiva así misma, por lo tanto, se deben tomar en cuenta las restricciones que se deben cumplir para no implementar funciones recursivas incorrectas.

# **OBJETIVO**

Elaborar un resumen sobre la recursividad para la solución de problemas de programación, sus características, variantes, ventajas y desventajas de su uso frente a la iteración y la estructura de una función recursiva.

# **DESARROLLO**

## **PARA LA ELABORACIÓN DEL RESUMEN**

### **Realice la lectura exploratoria del recurso teórico recomendado.**

Información obtenida del libro Fundamentos de programación 4ta Edición de Luis Joyanes Aguilar

### **Identifique las ideas más importantes del contenido estudiado.**

* La recursividad es la propiedad que posee una función de llamarse a sí misma.
* Permite a los programadores especificar soluciones naturales y sencillas.
* Un subprograma recursivo es un subprograma que se llama a su mismo ya sea directa o indirectamente.
* En matemáticas, circunstancias y situaciones de la vida ordinaria tienen carácter recursivo.
* El método de definición de una función en términos de sí misma se llama en matemáticas una definición inductiva y conduce naturalmente a una implementación recursiva.
* Por ejemplo, el clásico proceso de calcular la factorial de un número o el producto entre dos números.

Text

Description automatically generatedGraphical user interface, text

Description automatically generated

* Si una función, procedimiento o método se invoca a sí misma, el proceso se denomina recursión directa.
* Si una función, procedimiento o método puede invocar a una segunda función, procedimiento o método que a su vez invoca a la primera, este proceso se conoce como recursión indirecta o mutua.
* Un ejemplo de recursión indirecta es la impresión del abecedario, con dos funciones A y B, en este cao A llama a B y B llama a la función A.

Text, letter

Description automatically generated Text, letter

Description automatically generated Text

Description automatically generated

* Para que un algoritmo recursivo no genere una secuencia infinita de llamadas sobre sí mismo debe incluir un componente base (condición de salida).
* En recursión directa el código del subprograma recursivo F contiene una sentencia que invoca a F.
* En recursión indirecta el subprograma F invoca al subprograma G que invoca a su vez al subprograma P, y así sucesivamente hasta que se invoca de nuevo al subprograma F.
* Si no se considera una condición de terminación el subprograma continuaría indefinidamente llamándose a sí mismo y llegaría un momento en que la memoria se podría agotar.
* Tanto la iteración como la recursión se basan en una estructura de control.
* La iteración utiliza una estructura repetitiva y la recursión utiliza una estructura de selección.
* La iteración y la recursión implican ambas repeticiones: la iteración utiliza explícitamente una estructura repetitiva mientras que la recursión consigue la repetición mediante llamadas repetidas.
* La iteración y recursión implican una prueba de terminación (condición de salida).
* La iteración termina cuando la condición del bucle no se cumple mientras que la recursión termina cuando se reconoce un caso base o la condición de salida se alcanza.
* La recursión invoca repetidamente al mecanismo de recursividad y en consecuencia se necesita tiempo suplementario para realizar las mencionadas llamadas.
* La recursividad puede resultar cara en tiempo de procesador y espacio de memoria.
* Cada llamada de una función recursiva produce que otra copia de la función (realmente sólo las variables de función) sea creada consumiendo memoria considerablemente.
* La iteración se produce dentro de una función, de modo que las operaciones suplementarias de las llamadas a la función y asignación de memoria adicional son omitidas.
* La razón fundamental del uso de recursividad es que existen numerosos problemas complejos que poseen naturaleza recursiva y son más fáciles de implementar con algoritmos de este tipo.
* Recursividad vs iteración

Text, letter

Description automatically generatedText

Description automatically generated

**Factorial Iterativo**  **Factorial Recursivo**

* + Cuando el consumo de tiempo y memoria sean decisivos o concluyentes para la resolución del problema, la solución a elegir debe ser, normalmente, la iterativa.
  + Cualquier problema que se puede resolver recursivamente se puede resolver también iterativamente.
  + Un enfoque recursivo se elige normalmente con preferencia a un enfoque iterativo cuando el enfoque recursivo es más natural para la resolución del problema y produce un programa más fácil de comprender y depurar.
  + Otra razón para elegir una solución recursiva es que una solución iterativa puede no ser clara ni evidente.
  + Por ejemplo, los recorridos de árboles binarios.
  + Se ha de evitar utilizar recursividad en situaciones de rendimiento crítico o exigencia de altas prestaciones en tiempo y memoria, ya que las llamadas recursivas emplean tiempo y consumen memoria adicional.
  + Si una solución de un problema se puede expresar iterativa o recursivamente con igual facilidad, es preferible la solución iterativa.
  + Una solución iterativa se ejecuta más rápidamente (no existen llamadas adicionales a funciones que consumen tiempo de proceso) y utiliza menos memoria (la pila necesaria para almacenar las sucesivas llamadas necesarias en la recursión).
* Directrices en la toma de decisión iteración/recursión
  + Considérese una solución recursiva sólo cuando una solución iterativa sencilla no sea posible.
  + Utilícese una solución recursiva sólo cuando la ejecución y eficiencia de la memoria de la solución esté dentro de límites aceptables considerando las limitaciones del sistema.
  + Si son posibles las dos soluciones, iterativa y recursiva, la solución recursiva siempre requerirá más tiempo y espacio debido a las llamadas adicionales que se realizan.
  + En ciertos problemas, la recursión conduce naturalmente a soluciones que son mucho más fáciles de leer y comprender que su correspondiente iterativa. En estos casos los beneficios obtenidos con la claridad de la solución suelen compensar el coste extra (en tiempo y memoria) de la ejecución de un programa recursivo.
* La iteración y la recursión pueden producirse infinitamente. Un bucle infinito ocurre si la prueba o test de continuación de bucle nunca se vuelve falsa; una recursión infinita ocurre si la etapa de recursión no reduce el problema en cada ocasión de modo que converja sobre el caso base o condición de salida.
* La recursión infinita significa que cada llamada recursiva produce otra llamada recursiva y ésta a su vez otra llamada recursiva y así para siempre.
* La recursión infinita se ejecutará hasta que la computadora agota la memoria disponible y se produzca una terminación anormal del programa.
* El flujo de control de un algoritmo recursivo requiere tres condiciones para una terminación normal:
  + Una prueba para detener (o continuar) la recursión (condición de salida o caso base).
  + Una llamada recursiva (para continuar la recursión)
  + Un caso final para terminar la recursión.
* Cuando se realizan llamadas recursivas se han de pasar argumentos diferentes de los parámetros de entrada, por ejemplo, la función suma, el argumento en la función recursiva es n-1 y el parámetro es n.
* Muchos problemas de computadora tienen una formulación simple y elegante que se traduce directamente a código recursivo.

### **Elabore una frase sinóptica, concisa, que abarque el contenido total y que responda a la pregunta ¿de qué trata la lectura?**

La recursividad es una herramienta de programación que permite la implementación de soluciones más naturales y fáciles de entender, en donde la solución a una problemática usa su misma definición dando lugar a un ciclo repetitivo de llamadas de una función así misma, ya sea de manera directa o indirecta, almacenadas en un apila para su ejecución, dichas funciones tienen en su estructura una condición de salida, el caso base o final, y el caso recursivo. Frente a la iteración las soluciones recursivas son menos eficientes en cuanto a tiempo de ejecución y consumo de memoria.

### **Reúnalas ideas más importantes y, de existir, los ejemplos más sobresalientes.**

* La recursividad es la propiedad que posee una función de llamarse a sí misma.
* Permite a los programadores especificar soluciones naturales y sencillas.
* El método de definición de una función en términos de sí misma se llama en matemáticas una definición inductiva y conduce naturalmente a una implementación recursiva.
* Por ejemplo, el clásico proceso de calcular la factorial de un número o el producto entre dos números.
* Si una función, procedimiento o método se invoca a sí misma, el proceso se denomina recursión directa.
* Si una función, procedimiento o método puede invocar a una segunda función, procedimiento o método que a su vez invoca a la primera, este proceso se conoce como recursión indirecta o mutua.
* Un ejemplo de recursión indirecta es la impresión del abecedario, con dos funciones A y B, en este cao A llama a B y B llama a la función A.
* Tanto la iteración como la recursión se basan en una estructura de control.
* La iteración utiliza una estructura repetitiva y la recursión utiliza una estructura de selección.
* La iteración y la recursión implican ambas repeticiones: la iteración utiliza explícitamente una estructura repetitiva mientras que la recursión consigue la repetición mediante llamadas repetidas.
* La iteración y recursión implican una prueba de terminación (condición de salida).
* La iteración termina cuando la condición del bucle no se cumple mientras que la recursión termina cuando se reconoce un caso base o la condición de salida se alcanza.
* La recursividad puede resultar cara en tiempo de procesador y espacio de memoria.
* Cada llamada de una función recursiva produce que otra copia de la función (realmente sólo las variables de función) sea creada consumiendo memoria considerablemente.
* La razón fundamental del uso de recursividad es que existen numerosos problemas complejos que poseen naturaleza recursiva y son más fáciles de implementar con algoritmos de este tipo.
* Recursividad vs iteración
  + Cuando el consumo de tiempo y memoria sean decisivos o concluyentes para la resolución del problema, la solución a elegir debe ser, normalmente, la iterativa.
  + Cualquier problema que se puede resolver recursivamente se puede resolver también iterativamente.
  + Otra razón para elegir una solución recursiva es que una solución iterativa puede no ser clara ni evidente.
  + Por ejemplo, los recorridos de árboles binarios.
  + Una solución iterativa se ejecuta más rápidamente (no existen llamadas adicionales a funciones que consumen tiempo de proceso) y utiliza menos memoria (la pila necesaria para almacenar las sucesivas llamadas necesarias en la recursión).
  + Considérese una solución recursiva sólo cuando una solución iterativa sencilla no sea posible.
* La iteración y la recursión pueden producirse infinitamente. Un bucle infinito ocurre si la prueba o test de continuación de bucle nunca se vuelve falsa; una recursión infinita ocurre si la etapa de recursión no reduce el problema en cada ocasión de modo que converja sobre el caso base o condición de salida.
* La recursión infinita se ejecutará hasta que la computadora agota la memoria disponible y se produzca una terminación anormal del programa.
* El flujo de control de un algoritmo recursivo requiere tres condiciones para una terminación normal:
  + Una condición para detener (o continuar) la recursión.
  + El caso recursivo.
  + Un caso final para terminar la recursión.
* Cuando se realizan llamadas recursivas se han de pasar argumentos diferentes de los parámetros de entrada, por ejemplo, la función suma, el argumento en la función recursiva es n-1 y el parámetro es n.

### **Elimine los párrafos redundantes o accesorios y los ejemplos abundantes.**

* La recursividad es la propiedad que posee una función de llamarse a sí misma.
* Permite a los programadores especificar soluciones naturales y sencillas.
* El método de definición de una función en términos de sí misma se llama en matemáticas una definición inductiva y conduce naturalmente a una implementación recursiva.
* Si una función, procedimiento o método se invoca a sí misma, se denomina recursión directa, por ejemplo, la factorial de un número, pero si una función invoca a una segunda función, y que a su vez invoca a la primera, este se conoce como recursión indirecta o mutua por ejemplo la impresión del abecedario.
* Tanto la iteración como la recursión se basan en una estructura de control, la iteración utiliza una estructura repetitiva y la recursión utiliza una estructura de selección.
* Ambas implican repeticiones: la iteración utiliza explícitamente una estructura repetitiva mientras que la recursión consigue la repetición mediante llamadas repetidas.
* La iteración termina cuando la condición del bucle no se cumple mientras que la recursión termina cuando se reconoce un caso base o la condición de salida se alcanza.
* La recursividad puede resultar cara en tiempo de procesador y espacio de memoria.
* Cada llamada de una función recursiva produce que otra copia de la función (sólo las variables de función) sea creada consumiendo memoria considerablemente.
* La razón fundamental del uso de recursividad es que existen numerosos problemas complejos que poseen naturaleza recursiva y son más fáciles de implementar con algoritmos de este tipo.
* Recursividad vs iteración
  + Cuando el consumo de tiempo y memoria sean decisivos o concluyentes para la resolución del problema, la solución a elegir debe ser, normalmente, la iterativa.
  + Cualquier problema que se puede resolver recursivamente se puede resolver también iterativamente.
  + Una solución iterativa se ejecuta más rápidamente (no existen llamadas adicionales a funciones que consumen tiempo de proceso) y utiliza menos memoria (la pila necesaria para almacenar las sucesivas llamadas necesarias en la recursión).
  + Considérese una solución recursiva sólo cuando una solución iterativa sencilla no sea posible, por ejemplo, los recorridos de árboles binarios.
* Se puede producir un bucle infinito, en la iteración si la prueba o test de continuación de bucle nunca se vuelve falsa; en la recursión ocurre si no se reduce el problema en cada ocasión de modo que converja sobre el caso base o condición de salida, y se ejecutará hasta que la computadora agota la memoria disponible y se produzca una terminación anormal del programa.
* El flujo de control de un algoritmo recursivo requiere tres condiciones para una terminación normal:
  + Una condición para detener o continuar la recursión.
  + El caso recursivo.
  + Un caso final para terminar la recursión.
* Cuando se realizan llamadas recursivas se han de pasar argumentos diferentes de los parámetros de entrada, por ejemplo, la función suma, el argumento en la función recursiva es n-1 y el parámetro es n.

### **Integre las ideas importantes en un escrito coherente que exprese, con sus propias palabras, la síntesis de o que se explica en el tema, sin cambiarlas ideas centrales.**

#### **Recursividad**

La recursividad es la propiedad que posee una función de llamarse a sí misma, permitiendo a los programadores especificar soluciones más naturales y sencillas, en matemáticas una definición inductiva conduce naturalmente a una implementación recursiva.

Una función recursiva se compone de 3 partes:

* Una condición para detener o continuar la recursión.
* El caso recursivo donde la función se llama así misma.
* El caso final o caso base para terminar la recursión.

Si una función o proceso se invoca a sí misma, se denomina **recursión directa**, por ejemplo, la factorial de un número, pero si una función invoca a una segunda función, y que a su vez invoca a la primera, este se conoce como **recursión indirecta o mutua** por ejemplo la impresión del abecedario.

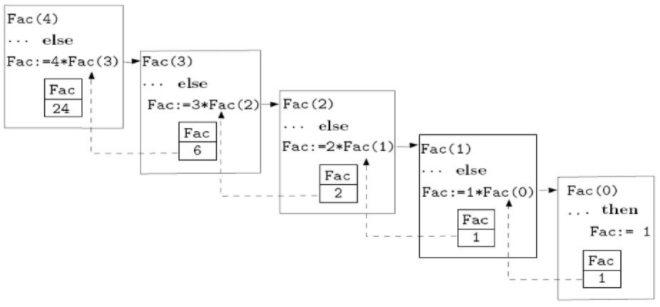
Tanto la iteración como la recursión se basan en una estructura de control, la iteración utiliza una estructura repetitiva y la recursión utiliza una estructura de selección. En donde ambas implican repeticiones, la iteración utiliza explícitamente una estructura repetitiva mientras que la recursión consigue la repetición mediante llamadas repetidas así misma. La iteración termina cuando la condición del bucle no se cumple mientras que la recursión termina cuando se reconoce el caso base o la condición de salida.

La recursividad puede resultar cara en tiempo de procesador y espacio de memoria ya que cada llamada de una función recursiva produce que otra copia de la función sea creada consumiendo memoria considerablemente en si copias de las variables de la función.

La razón fundamental del uso de recursividad es la existencia de numerosos problemas complejos que poseen naturaleza recursiva y son más fáciles de implementar con algoritmos de este tipo.

#### **Recursividad vs iteración**

Cuando el consumo de espacio de memoria y tiempo de ejecución sean condiciones esenciales para la resolución de un problema, normalmente se eligen procesos iterativos, aunque cualquier problema que se puede resolver recursivamente también tiene una solución iterativa, una solución iterativa se ejecutara más rápido al no existir llamadas adicionales a funciones que consumen tiempo de proceso y utilizan menos memoria, ya que la recursividad forma una pila en donde se almacenan las llamadas sucesivas necesarias en la recursión.



Se considera una solución recursiva cuando la solución iterativa no es fácil de implementar, por ejemplo, los recorridos de árboles binarios.

Un grave error de programación es la ejecución de un bucle infinito, en la iteración se produce si la prueba o test de continuación del bucle nunca se vuelve falsa, mientras que en la recursión ocurre si no se reduce el problema en cada ocasión de modo que no se ejecutara la condición de salida o caso base, por lo tanto, la función recursiva se ejecutará hasta que la computadora agote la memoria disponible y se produzca una terminación anormal del programa.

### **Tenga en cuenta que el resumen puede repetir literalmente las ideas principales tal como las expone el autor, siempre y cuando se las vincule adecuadamente.**

## **RESOLUCIÓN DE INFORMES DE LABORATORIOS / PRÁCTICAS O TALLERES.**

### **Realice la lectura del ejercicio del recurso teórico que corresponda.**

#### **Torres de Hanói**

Muchos problemas de computadora tienen una formulación simple y elegante que se traduce directamente a código recursivo.

El problema en cuestión supone la existencia de 3 varillas o postes en los que se alojaban discos, cada disco es ligeramente inferior en diámetro al que está justo debajo de él, y pretende determinar los movimientos necesarios para trasladar los discos de una varilla a otra cumpliendo las siguientes reglas:

* En cada movimiento sólo puede intervenir un disco.
* Nunca puede quedar un disco sobre otro de menor tamaño.

A picture containing diagram

Description automatically generated

Al final todos los discos deben ser trasladados de la varilla I a la varilla F cumpliendo las 2 reglas.

A picture containing chart

Description automatically generated

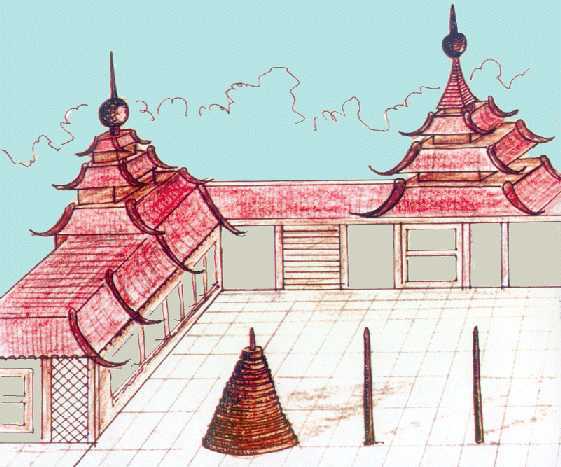
### **Acceda a otra información teórica en relación con el tema a desarrollar. Utilice diferentes estrategias para la búsqueda, selección y organización de información.**

#### **Historia de las torres de Hanói**

El juego de las torres de Hanói, o también conocido como, las torres de Brahma o el problema del fin del mundo, es un juego que consiste en mover una pila de discos de una torre a otra cumpliendo con 2 sencillas normas, en cada movimiento sólo puede intervenir un disco y nunca puede quedar un disco sobre otro de menor tamaño.

#### **Las torres de Brahma o el problema del fin del mundo**

Cuenta la leyenda que, en Benarés, durante el reinado del Emperador Fo Hi, existía un templo con una cúpula que marcaba el centro del mundo. Debajo de la cúpula yacía una base de bronce, en donde se encontraban acomodadas tres agujas de diamante, cada una del grueso del cuerpo de una abeja y de una altura de unos cincuenta centímetros aproximadamente. En una de esas agujas, en el comienzo de los tiempos, Brahma, dios hindú de la creación, colocó sesenta y cuatro discos de oro puro, el mayor descansando sobre la base de bronce, y el resto decreciendo en tamaño conforme se va ascendiendo. Día y noche, incesantemente, los sacerdotes del templo se turnaban el trabajo de mover los discos de una aguja a otra de acuerdo con las leyes inmutables impuestas por Brahma, que requerían que siempre hubiese algún sacerdote trabajando, que no se moviese más de un disco a la vez y que debían colocar cada disco en alguna de las agujas de modo que nunca cubriese a un disco de radio menor. Cuando los sesenta y cuatro discos hayan sido transferidos de la aguja en la que Brahma los colocó a otra aguja, el templo y los sacerdotes se convertirán en polvo, y junto con ellos el mundo desaparecerá.



El número de movimientos para mover los sesenta y cuatro por lo que los monjes necesitarán una cantidad de movimientos, suponiendo que los monjes mueven un disco por segundo les costaría pasar la torre de la varilla inicial a la final aproximadamente 585 mil millones de años.

El algoritmo óptimo, para mover una torre de n discos de la primera varilla a la tercera es el siguiente algoritmo recursivo:

* Si n = 0, entonces no hacer nada.
* Si n = 1, entonces mover el único disco de la primera varilla a la tercera.
* Si n > 1, entonces:

1. Mover la torre de n−1 discos de la primera varilla a la segunda, aplicando el mismo algoritmo considerando la segunda varilla como la tercera y viceversa.
2. Mover el n-ésimo disco de la primera varilla a la tercera.
3. Mover la torre de n−1 discos de la segunda varilla a la tercera, aplicando el mismo algoritmo considerando la primera varilla como la segunda y viceversa.

El número de movimientos está considerado por la expresión donde n es el número de discos.

### **Identifique los conceptos principales y secundarios de la temática.**

**Las torres de Hanoi están compuestas por 3 torres, una inicial, una intermedia y una final.**

La torre inicial cuenta con n discos.

El juego consiste en mover los discos de la torre inicial a la torre final

Para mover los discos se deben considerar las 2 reglas del juego.

* En cada movimiento sólo puede intervenir un disco.
* Nunca puede quedar un disco sobre otro de menor tamaño.

### **Utilice la metodología indicada en la resolución de los ejercicios y del Documento o Informe del Laboratorio.**

### **Elabore el documento o Informe del Laboratorio con los ejercicios indicados y en el formato dado.**

#### **ARGUMENTACIÓN**

Las torres de Hanoi son un clásico ejemplo para explicar el significado y uso de la recursividad para el desarrollo de algoritmos con una implementación más elegante y natural, además, son más fáciles de entender y depurar.

Una solución iterativa tendría una implementación y desarrollo más complejo, ya que al usar una estructura repetitiva, esta no tendrá las mismas ventajas de una estructura recursiva, aunque no todo es bueno en cuanto al uso de la recursividad debido al aumento de tiempo de ejecución y consumo de memoria que produce una implementación recursiva, esto debido a las copias de las variables que hace la función al llamarse una y otra vez, y la pila de funciones a procesarse que se forma que son necesarias para la recursividad.

La recursividad es eficiente cuando la solución no implica una gran cantidad de llamadas de una función así misma, y su implementación es más fácil y sencilla de escribir que una iterativa en casos muy complejos.

#### **Introducción**

La recursividad es una propiedad que tienen varios lenguajes de programación, ayudando a los programadores a implementar algoritmos más sencillos y elegantes frente a problemas complejos.

Los algoritmos recursivos e iterativos tienen ventajas y desventajas frente a problemas específicos que se presentan en el desarrollo de software, los más evidentes son el tiempo de ejecución, consumo de memoria y código limpio, los algoritmos recursivos se implementan tras una inducción matemática que demuestre que un proceso necesite de su misma definición para resolverse en varios casos en las que se concluya con una solución recursiva.

La implementación de funciones recursivas depende tanto del problema en cuestión y de las condiciones que se propongan para su implementación, por lo tanto, dependerá del programador escoger una solución recursiva o una iterativa.

Las torres de Hanoi son un claro ejemplo del uso de la recursividad, en donde una función simple puede resolver un problema complejo.

**Objetivo**

**p.e.**

Desarrollar un programa que imprima los pasos a seguir para mover los discos de las torres de Hanoi.

**Instrucciones**

**p.e.**

1. En cada movimiento sólo puede intervenir un disco.
2. Nunca puede quedar un disco sobre otro de menor tamaño.
3. Calcule el número de movimientos que harán los discos.
4. Utilice algoritmos recursivos para el movimiento de los discos y calcular el número de movimientos.
5. Utilice información consultada en Internet y conocimiento adquirido en clase.

**Actividades**

1. **Ubicación de recursos**

**p.e.**

1. Instalar un IDE que permite el uso de lenguaje C++.
2. **Planteamiento del problema**

**p.e. Ejercicio No 1**

Escribir un programa (es decir una función *main*) que haga la llamada a 2 funciones recursivas, una que calcule el número de movimientos para mover los discos de la torre inicial a la final, y otra que realice el movimiento de los discos e imprima los pasos que se deben seguir para mover los discos de una torre a otra. Para ello la función *main* debe tener el prototipo estándar **int main(int argc, char \*argv[])**.

1. **Entregable (s)**

**p.e.**

**#include <stdio.h>**

**#include <iostream>**

**#include <stdlib.h>**

**class Hanoi**

**{**

public:

void moverdisco(int ,int ,int ,int );

int pasos(int );

void hanoi();

**};**

**#include "Hanoi.h"**

**int Hanoi::pasos(int discos)**

**{**

if (discos == 1)

{

return 1;

}

else {

return 2 \* pasos(discos - 1) + 1;

}

**}**

**void Hanoi::moverdisco(int disco, int torre1, int torre2, int torre3)**

**{**

if (disco == 1)

{

std::cout << "Mover el disco: " << disco << " de la torre: " << torre1 << " hacia la torre: " << torre3 << std::endl;

}

else

{

moverdisco(disco - 1, torre1, torre3, torre2);

std::cout << "Mover el disco: " << disco << " de la torre: " << torre1 << " hacia la torre: " << torre3 << std::endl;

moverdisco(disco - 1, torre2, torre1, torre3);

}

**}**

**void Hanoi::hanoi() {**

int discos = 0, torre1 = 1, torre2 = 2, torre3 = 3;

do {

system("cls");

std::cout << "Ingrese el numero de discos: ";

std::cin >> discos;

} while (!(discos > 0 && discos < 10));

std::cout << "El numero de pasos para resolver es: " << pasos(discos) << " , los pasos son: " << std::endl;

moverdisco(discos, torre1, torre2, torre3);

**}**

**int main(int argc, char \*argv[])**

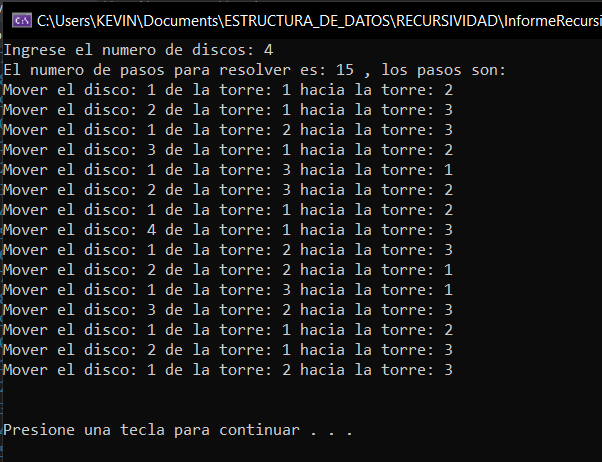
**{**

Hanoi hni;

hni.hanoi();

return 0;

**}**



#### **Conclusión del ejercicio planteado**

Las funciones para el cálculo de numero de movimientos que se hacen en base al número de discos, y la función para mover los discos fueron implementadas de una forma muy simple, limpia y elegante.

# **CONCLUSIONES**

* La recursividad es una forma de implementar soluciones simples a problemas complejos.
* La recursividad se divide en 2 tipos, la recursividad directa, y la recursividad indirecta en donde una función llama a una segunda función, la segunda a una tercera y así sucesivamente, hasta que se haga la llamada a la primera.
* Una función recursiva se llama así misma, y su estructura se compone de una condición para detener o continuar con la recursión, el caso base y el caso recursivo.
* Las funciones recursivas consumen más memoria y tiempo de ejecución al crearse copias de sus variables y la formación de la pila de llamadas que se hace así misma.
* Se demostró que la recursividad es un concepto de programación con la capacidad de resolver problemas complejos como lo es Hanoi de manera simple, pero depende mucho de la experiencia y lógica del programador.

# **RECOMENDACIONES**

* Para la implementación de algoritmos recursivos se debe tomar en cuenta el tipo de problema que se esta tratando, las condiciones para resolver dicho problema y analizar si una solución recursiva es mejor que una iterativa.
* Las funciones recursivas consumen un mayor tiempo de ejecución y espacio de memoria, por lo tanto, se debe tomar en cuenta que el numero de repeticiones que hará la función recursiva no sea tan grande.
* La declaración de variables no se debe hacer en el cuerpo de una función recursiva, de preferencia deben estar como parámetros de la función, esto debido a las copias que hará cada llamada así misma que hará a la función.
* Se debe tomar muy en cuenta la condición de salida para que se ejecute el caso base de la función recursiva, caso contrario esta generara una llamada infinita así misma hasta que la memoria del computador se agote provocando una terminación anormal del programa.

# **BIBLIOGRAFÍA**

Aguilar, L. J. (2008). *Fundamentos de Programacion.* Madrid: McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S. A. U.

Amaia, G. L., Ester, L. E., Oihane, R. D., & Josué, 0. C. (n.d.). *Torres de Hanoi.* curso “Combinatoria” de la Licenciatura en Matemáticas de la Universidad del País Vasco, Vasco.

Sangolquí, 21 de Febrero de 2021

|  |  |  |  |
| --- | --- | --- | --- |
| **Elaborado Por:** | **Revisado Por:** | **Supervisado por:** | **Aprobado por:** |
| UNIDAD DE  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Nombre  **Cargo** | UNIDAD DE  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Nombre.  **Cargo** | UNIDAD DE  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Nombre.  **Cargo** | UNIDAD DE  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Nombre.  **Cargo** |