La recursividad

The recursion.

Autor: Yuliana Melissa Vera Jaramillo

*Universidad Tecnológica de Pereira, Pereira, Colombia*

Correo: yuliana.vera@utp.edu.co

***Resumen*— “La recursividad es la forma en la cual se especifica un proceso basado en su propia definición”. En programación se define como la esencia de que una función se llame a sí misma, resolviendo así problemas de grado de dificultad alto, con el fin de hacer su solución más corta y fácil.**

***Palabras clave—* Recursividad, llamado, programación, bucle.**

***Abstract*— Recursion is the way in which a process is specified based on its own definition. Programming is defined as the essence of a function calling itself, thus solving problems of high difficulty degree, in order to make your solution shorter and easier**

***Key Word* —** **Recursion, calling, programming, looping.**

1. INTRODUCCIÓN

“La recursividad, también llamada recursión o recurrencia, es la forma en la cual se especifica un proceso basado en su propia definición. O sea, si se tiene un problema de tamaño N, este puede ser dividido en instancias más pequeñas que N del mismo problema y conociendo la solución de las instancias más simples, se puede aplicar inducción a partir de estas asumiendo que quedan resueltas.”

1. CONTENIDO

“Recursividad:

La recursividad es una técnica de programación que se utiliza para realizar una llamada a una función desde ella misma, de allí su nombre. El ejemplo más utilizado por su fácil comprensión es el cálculo de números factoriales. El factorial de 0 es, por definición, 1. Los factoriales de números mayores se calculan mediante la multiplicación de 1 \* 2 \* ..., incrementando el número de 1 en 1 hasta llegar al número para el que se está calculando el factorial.

Un algoritmo recursivo es un algoritmo que expresa la solución de un problema en términos de una llamada a sí mismo. La llamada a sí mismo se conoce como llamada recursiva o recurrente.

#include <iostream>

#include <cstdlib>

using namespace std;

int Factorial(int n);

int main(){

int valor;

system("clear");

cout << "Introduzca numero a calcular: ";

cin >> valor;

cout << "\nEl Factorial de " << valor << " es: " << Factorial(valor) << endl;

return 0;

}

int Factorial(int n){

if (n < 0){

cout << “No existe el factorial de un numero negativo.\n”;

}else if(n < 2){

return 1;

}else

return n \* Factorial(n-1);

}

Generalmente, si la primera llamada al subprograma se plantea sobre un problema de tamaño u orden N, cada nueva ejecución recurrente del mismo se planteará sobre problemas, de igual naturaleza que el original, pero de un tamaño menor que N. De esta forma, al ir reduciendo progresivamente la complejidad del problema a resolver, llegará un momento en que su resolución sea más o menos trivial (o, al menos, suficientemente manejable como para resolverlo de forma no recursiva). En esa situación diremos que estamos ante un caso base de la recursividad.

Es frecuente que los algoritmos recurrentes sean más ineficientes en tiempo que los iterativos aunque suelen ser mucho más breves en espacio.”

1. Recursividad directa vs indirecta.

“Cuando en una subrutina hay llamadas a ella misma se habla de recursividad directa, en contraposición, cuando se tienen varias subrutinas y éstas se llaman unas a otras formando ciclos se dice que la recursión es indirecta.

Subrutina\_A → Subrutina\_A → Subrutina\_A

Subrutina\_A → Subrutina\_B → Subrutina\_C → Subrutina\_D → Subrutina\_A”

1. Propiedades de las definiciones o algoritmos recursivos.

“● No debe generar una secuencia infinita de llamadas así mismo, dicho de otro modo ha de existir al menos un caso base.

● Una función recursiva f debe definirse en términos que no impliquen a f al menos en un argumento o grupo de argumentos.

● Debe existir una "salida" de la secuencia de llamadas recursivas.

● Cada llamada recurrente se debería definir sobre un problema de menor complejidad (algo más fácil de resolver).”

1. Programación Recursiva.

“Es mucho más difícil desarrollar una solución recursiva en un lenguaje determinado para resolver un problema específico cuando no se tiene un algoritmo. No es solo el programa sino las definiciones originales y los algoritmos los que deben desarrollarse. En general, cuando encaramos la tarea de escribir un programa para resolver un problema no hay razón para buscar una solución recursiva. La mayoría de los problemas pueden resolverse de una manera directa usando métodos no recursivos. Sin embargo, otros pueden resolverse de una manera más lógica y elegante mediante la recursión.

Volviendo a examinar la función factorial. El factor es, probablemente, un ejemplo fundamental de un problema que no debe resolverse de manera recursiva, dado que su solución iterativa es directa y simple. Sin embargo, examinaremos los elementos que permiten dar una solución recursiva. Antes que nada, puede reconocerse un gran número de casos distintos que se deben resolver. Es decir, quiere escribirse un programa para calcular 0!, 1!, 2! Y así sucesivamente. Puede identificarse un caso "trivial" para el cual la solución no recursiva pueda obtenerse en forma directa. Es el caso de

0!, que se define como 1. El siguiente paso es encontrar un método para resolver un caso "complejo" en términos de uno más "simple", lo cual permite la reducción de un problema complejo a uno más simple. La transformación del caso complejo al simple resultaría al final en el caso trivial. Esto significaría que el caso complejo se define, en lo fundamental, en términos del más simple.

Ejemplo de la sucesión de Fibonacci

En matemáticas, la sucesión de Fibonacci es la siguiente sucesión infinita de números naturales:

0,1,1,2,3,5,8,13,21,34,55,89,144...

El primer elemento es 0, el segundo es 1 y cada elemento restante es la suma de los dos anteriores:

0 si i = 0

fi = 1 si i = 1

f(i-2)+f(i-1) si i > 1

A cada elemento de esta sucesión se le llama número de Fibonacci. Esta sucesión fue descrita en Europa por Leonardo de Pisa, matemático italiano del siglo XIII también conocido como Fibonacci.

El código en C++ que representa la función Fibonacci es el siguiente:

#include <iostream>

#include <cstdlib>

using namespace std;

int Fibonacci(int n);

int main(){

int valor;

system("clear");

cout << "Introduzca numero a calcular: ";

cin >> valor;

cout << "\nEl Fibonacci de " << valor << " es: " << Fibonacci(valor) << endl;

return 0;

}

int Fibonacci(int n){

if (n < 0){

cout << “No existe Fibonacci para numeros negativos.”;

} else if (n == 0) {

return 0;

} else if (n == 1) {

return 1;

}else

return Fibonacci(n-2) + Fibonacci(n -1);

}

”

1. Imágenes recursivas: efectos Droste y Escher.

“El paradigma de la publicidad bien podría ser una imagen que se publicita a sí misma. Eso mismo debieron de pensar los directores publicitarios de la marca de productos alimenticios Droste en los años 60.

Posteriormente conocido como efecto Droste es uno de los ejemplos más sencillos de recursividad aplicado a una imagen. Digo más sencillos porque las imágenes recursivas ofrecen un gran número de posibilidades creativas, tan variadas como originales. Aunque no se trataba de una idea original pues las imágenes recursivas ya se conocían seiscientos años antes lo que si es cierto es que la publicidad en aquel siglo no tenía el mismo impacto en la población que la del siglo XX.

¿Qué es la recusividad? Según la definición de wikipedia recursividad es la forma en la cual se especifica un proceso basado en su propia definición. Esto es un poco difícil de adaptar al mundo de la imagen o el diseño gráfico, por ese motivo es mejor recurrir a un ejemplo más visual. Imaginad una cuadrícula bidimensional. Una cuadrícula es un ejemplo perfecto de imagen recursiva sin tener que recurrir a las matemáticas. Cada cuadrado está formado por cuadrados de menor tamaño que a su vez están formados por otros cuadrados más pequeños y así una y otra vez. No importa realmente que sean cuadrados, triángulos equiláteros, rectángulos áureos, curvas de Koch,… No importa tanto la forma como que el contenido de la imagen se utiliza a sí misma hasta el infinito.

Al principio de esta publicación he mencionado el efecto Droste pues es uno de los más claros. La imagen original del efecto Droste reproducía una joven sosteniendo una bandeja con un tazón y un envoltorio de cacao con la imagen impresa de una joven sosteniendo una bandeja con un tazón y un envoltorio de cacao con la imagen impresa… etc. El efecto Droste presenta un tipo específico de recursividad en el que cada imagen contiene una reproducción más pequeña de sí misma. No requiere ningún tipo de proceso matemático pues el único requisito es encontrar en la imagen el punto adecuado en el que reproducir la réplica. El límite lo estable la resolución del documento. Es posible observar multitud de ejemplos en la web que utilizan este efecto aunque, cuidado, no todas las imágenes que aparecen en las búsquedas usan el efecto Droste pero, sí son imágenes recursivas.

Uno de los tipos de imágenes recursivas que en ocasiones se confunden con el efecto Droste son las originadas a partir de la técnica efecto Escher en honor al pintor holandés Maurits Cornellis Escher (1898-1972) cuyas litografías exploraron diferentes técnicas especialmente enfocas a jugar con el espacio. De esta forma en sus obras se destacan las teselas, retículas, imágenes recursivas, estudios de las superficies y la partición regular del plano.

Actualmente es posible reproducir con facilidad tanto el efecto Escher como el Droste con ayuda de algoritmos matemáticos, cálculos fractrales y la aplicación Mathematica.”

1. Gödel.

“Gödel puso en claro por primera vez que el descubrimiento de un procedimiento universal de decisión para la lógica general sería suficiente para permitir una solución a la serie de problemas irresueltos de la teoría de los números. Pero en el contexto en que formuló esta reflexión no había pretensión por apoyar alguna filosofía de la matemática, más bien trataba de sugerir la inutilidad de todo intento por descubrir un procedimiento de decisión universal para la lógica general. La imposibilidad de tal descubrimiento pudo probarse más tarde a partir del desarrollo de la propia técnica de Gödel.

Esta técnica está contenida en la llamada “función recursiva”.

La aritmetización, que era una idea presente en Leibniz, con Gödel se pone en práctica de manera sistemática por primera vez. Intuitivamente, la noción de recursividad corresponde a la de un cálculo que se puede llevar a cabo por aproximaciones sucesivas hacia un resultado. Una función recursiva es una función cuyos valores pueden ser calculados progresivamente, partiendo de valores previamente conocidos. Esta función se define como “esquema de recursión” y es la función “sucesor” la que da una idea de ella; esta idea, de manera sencilla, surge cuando la función “sucesor” se aplica a un entero, le hace aumentar una unidad (existen fórmulas para la suma, la multiplicación, la exponenciación).

La operación que se aplica a cualquier categoría de objetos es representable recursivamente si se establece una correspondencia biunívoca entre estos objetos y una determinada clase de números enteros, y se encuentra entonces una función “F” tal que cuando se aplica esta función y esta operación a los elementos estos corresponden.

La función “F” es una representación recursiva de esta operación. Aquí lo importante es que en la operación de relacionar objetos con otros objetos, elementos de conjuntos con otros conjuntos, se introduce la idea de “función”. Esta idea es el movimiento u operación misma en el siguiente ejemplo: 2 pasa a 4, 4 pasa a 16, o 50 a 2500, cuando al multiplicar se dan valores o números y se obtienen otros, es decir lo dado es otro número y lo que se ha terminado es una multiplicación. Este triple movimiento se representa con símbolos y su acción sobre ellos mismos: F(2)=4. Si en lugar de un numeral se introduce una variable se tiene F(x)=x (al cuadrado), cuya posibilidad la constituyen innumerables operaciones mentales. A partir de la idea de función Gödel introduce una clase especial, la de funciones especiales recursivas primitivas.”

1. Johann Sebastián Bach.

El 21 de marzo de 1685 nació el clavecinista, organista y compositor alemán: Johann Sebastián Bach. Sus obras son consideradas piezas artísticas debido a su profundidad intelectual, su perfección técnica y su belleza melódica. Además, es reconocido como el último gran maestro del arte de contrapunto, técnica de composición musical que relaciona dos voces independientes para crear un equilibrio armónico.

El alemán compuso sus piezas musicales pensando en Fuga y Canon, que son elementos musicales que usan la repetición de melodías en diferente escala o velocidad con o sin contrapunto, pero que no precisamente son la misma melodía. El uso del canon y la fuga es similar a la naturaleza fractal pero este término no se conocía en 1685 sino hasta 1975 cuando fue acuñado por el matemático Benoît Mandelbrot.

En piezas autosemejantes de Bach los mismos motivos son repetidos una y otra vez con distintas variaciones dentro de una región mayor de la pieza. Así, por ejemplo, distintas voces se repiten al doble de velocidad la melodía de la voz principal.

A partir de la música de Bach se descubrió que repetir un proceso con el objetivo de alcanzar una meta deseada, en conjunto con las matemáticas, resultan elementos que se aplican como una extensión de la composición musical. Así, los fractales proveen inesperadas conexiones entre las artes musicales y muchos procesos naturales, ya que mezclan cualidades determinantes y probables para producir naturalmente un agradable balance entre lo que se espera y la novedad, pues al escuchar la repetición de cada una de las notas por separado, la pieza resultaría un caos, al igual que una imagen que no entra en armonía con todos los elementos.”

1. CONCLUSIONES

La recursividad es una herramienta a la cual podemos recurrir al momento de enfrentarnos a problemas de gran tamaño o dificultad.

RECOMENDACIONES

Se recomienda tener claro el concepto de recursividad para hacer un buen uso de ella en la solución de problemas.

REFERENCIAS

1. <https://www.ecured.cu/Recursividad>
2. <http://webdelprofesor.ula.ve/nucleotrujillo/jalejandro/Prog2/Unidad3.pdf>
3. <https://mimoriarty.wordpress.com/2011/04/11/imagenes-recursivas-efectos-droste-y-escher/>
4. <https://books.google.com.co/books?id=EaspYiU7p9EC&pg=PA33&lpg=PA33&dq=recursividad+Godell&source=bl&ots=4OslTn2Hrq&sig=ACfU3U1lsmcFipfT66n11iywluUxdZBfUg&hl=es&sa=X&ved=2ahUKEwiS9taYk9_iAhVMs1kKHbhmCJgQ6AEwBXoECAgQAQ#v=onepage&q=recursividad%20Godell&f=false>
5. <https://culturacolectiva.com/musica/los-fractales-en-la-musica-de-sebastian-bach>