Tecnológico Nacional de México

Instituto Tecnológico de Orizaba

Asignatura: Estructura de Datos

Tema 2: Recursividad

Alumnos:

Bandala Hernández Sebastián (21010921)

Brito García Sebastián (21010929)

Jiménez Jiménez Carlos Yael (21010975)

Maestra: Martínez Castillo María Jacinta

Grupo: 3g2B

Hora: 17:00 – 18:00 hrs

Fecha de Entrega: 27 de marzo de 2023

Introducción

La recursividad es un concepto fundamental en la programación y se refiere a la capacidad de una función para llamarse a sí misma durante su ejecución. Esta técnica permite la resolución de problemas complejos dividiéndolos en subproblemas más simples y ordenándolos de manera recursiva. La recursividad se utiliza comúnmente en algoritmos de búsqueda, ordenación, estructuras de datos y en muchos otros problemas de programación. La capacidad de una función para llamarse a sí misma puede ser muy poderosa, pero también puede llevar a problemas de eficiencia y a errores de desbordamiento de pila si no se utiliza adecuadamente. Comprender los conceptos básicos de la recursividad y saber cómo aplicarlos en la programación puede ayudar a los desarrolladores a crear soluciones más elegantes y eficientes a problemas complejos.

Competencia Específica

Con base al tema que vimos en clase, aprendemos y elaboramos nuestros programas con los diferentes tipos de estructuras de datos que pudimos ver, esto con el fin de almacenar diferentes datos que el usuario puede dar y nosotros como programadores debemos almacenarlos en una base de datos.

Marco Teórico

2.1 Definición

Es una característica de los lenguajes de programación que permite que un subprograma se invoque a sí mismo. La recursividad es útil para resolver problemas definibles en sus propios términos. La recursividad es, en cierta medida, análoga al principio de inducción

2.2 Procedimiento iterativo

El proceso iterativo es la práctica de elaborar, refinar y mejorar un proyecto, producto o iniciativa. Los equipos que usan procesos de desarrollo iterativos crean, prueban y hacen revisiones hasta que se sienten satisfechos con el resultado final.

2.3 Procedimiento recursivo

Un procedimiento recursivo es aquel que se llama a sí mismo. En general, esta no es la manera más eficaz de escribir código de Visual Basic. El procedimiento siguiente usa recursividad para calcular el factorial de su argumento original.

2.4 Cuadro comparativo de procedimientos iterativos y recursivos

ITERATIVO	RECURSIVO
Repetición de un proceso dentro de un algoritmo.	El procedimiento se llama así mismo lo que hace que se repita un cierto número de veces.
Ventajas	**Ventajas**
- Permite dar soluciones a problemas complejos por partes. - Aumenta la productividad y permite optimizar el proceso en el corto plazo.	- Soluciona problemas recurrentes. - Programas cortos. - Soluciones simples y claras. - Soluciones elegantes. - Soluciones a problemas complejos.
Desventajas	**Desventajas**
- No se puede predecir el número de operaciones necesarias para obtener una solución en particular. - El tiempo de ejecución y la exactitud del resultado puede depender de la elección juiciosa de los parámetros. - Generalmente, no se gana tiempo por iteración si la matriz de coeficientes es simétrica. Un método genérico puede reducir en este caso el tiempo de ejecución a la mitad.	- Creación de muchas variables. - Puede necesitar muchas variables. - Ineficiencia inherente de algunos algoritmos recursivos. - Sobrecargas asociadas a las llamadas a sub-logaritmos, ya que una simple llamada puede generar un gran número de llamadas recursivas.

- Análisis de algoritmos: Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema

- Complejidad en el tiempo: es el número de operaciones que realiza un algoritmo para completar su tarea (considerando que cada operación dura el mismo tiempo).

- Complejidad en el espacio: es la cantidad de espacio en memoria que un algoritmo emplea al ejecutarse.

- Eficiencia de los algoritmos: es medible de acuerdo con la utilización de espacio y tiempo que requieren; a través del análisis es posible generar una función matemática que representa su eficiencia; a esto lo debemos llamar análisis de complejidad algorítmica.

Materiales: y equipo:

• Computadora.

• Software y versión usados.

• Materiales de apoyo para el desarrollo de la práctica

Desarrollo de prácticas: Recursividad

1. funciónRecursiva: el método recibe dos valores que después evalúa el primer valor de j es menor que el valor de n y si es así imprime el valor de j, después se llama al método para poder incrementar el valor de j y así ir mostrando los demás números hasta llegar a n.

```
public static void funcionRecursiva(int j,int n) {
    if(j<=n) {
        System.out.print(j);
        funcionRecursiva(j+1,n);
    }
}
```

Resultado: `123456`

2. funcionRecursivaString: el método hace al comienzo lo mimos que el anterior solo que este lo hace en un cuadro de texto y al cumplirse la condición concatena los valores uno debajo de otro y cuando ya no se cumple la condición se termina y muestra un espacio en blanco.

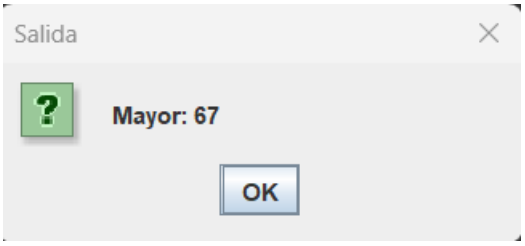
```
public static String funcionRecursivaString(int j,int n) {
            if(j<=n) {
                return j+"\n"+funcionRecursivaString(j+1,n);
            }
            else {
                return "";
            }
}
```

Resultado:

3. numMayorRecursivo: este método lo que hace es recibir dos parámetros y después de esto evalúa la j con la condición y si se cumple esta se pude ingresar un numero y si el dato es mayor a la variable de mayor se le asigna ese valor para después llamar al método y así hacer que se incremente el valor de j y al terminar de cumplirse la primera condición se regresa el valor mayor

```
public static int numMayorRecursivo(byte j, int mayor) {
    if(j<=5) {
        int dato=Tools.leerInt("Dato: ");
        if(dato>mayor) {
            mayor=dato;
            return numMayorRecursivo((byte) (j+1),mayor);
        }
    }
    return mayor;
}
```

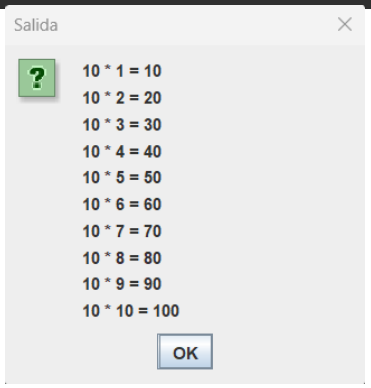
Resultado:



4. tablaMultiRecursivo: este método recibe dos parámetros y se crea una variable String donde se concatenará el valor de j y después esa misma variable se evaluará en una condición y después en esa condición se concatenará en esa variable y después se llamará al método para aumentar el valor de j, al terminar este se retornará un espacio vacío.

```
public static String tablaMultiRecursivo(byte j, int num) {
    String cad ="";
    if(j<=10) {
        return cad+=num+" * "+j+" = "+(num*j)+"\n"+tablaMultiRecursivo((byte) (j+1),num);
    }
    return "";
}
```

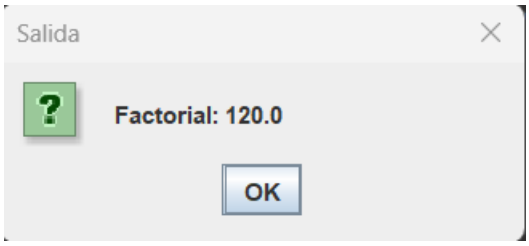
Resultado:



5. numFactorialRecursivo: este método tiene 3 valores como parámetro y después la variable dato será evaluada y si se cumple esta condición se retornará el valor de 1, sino se evaluara la j con el valor ingresado y se comenzara a llamar el método donde se incrementara y se multiplicara el valor de j y al terminar este se regresara el valor del factorial que se obtiene al multiplicarlo por la j.

```
public static double numFactRecursivo(byte j, byte fac,int dato) {

    if(dato==0 || dato==1) {
        return 1;
    }

    if(j<=dato) {
        return numFactRecursivo((byte) (j+1),(byte) (fac*=j),dato);
    }
    return fac;
}
```

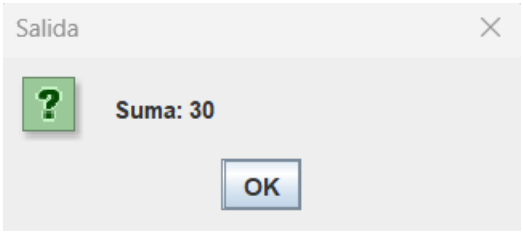
Resultado:



6. sumDivisionesRecursivo: el método tiene 2 parámetros, y la variable suma inicializada con el valor de 0, después se evalúa k que debe ser menor al dato y después se evalúa el resultado del dato%k si es igual a 0 y si eso se cumple la variable de dato se le suma el valor de k y después se retorna el valor de la suma y se llama al método y se incrementa el valor de k, al terminar esto se retorna el valor de suma.

```
public static int sumaDivisoresRecursivo(int dato,int k) {
    int suma=0;

    if(k<dato) {
        if (dato % k==0) {
        suma+=k;
        return suma+sumaDivisoresRecursivo(dato, k+1);
        }
    }
    return suma;
}
```

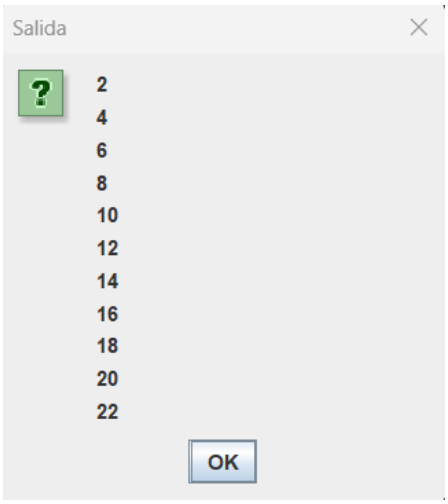
Resultado:



7. numPares: el método recibe un parámetro y se crea un variable de tipo String llamada cad, donde se concatenará el valor de k, después de esto se evalúa el valor de k%2 si es diferente de 0 y si lo es a k se le suma 1, se vuelve a evaluar y si k es menor o igual a 144 se empieza a guardar el valor en cad y se llama al método que ira incrementando el valor de k de 2 en 2, al final al terminar regresara un espacio vacío.

```
public static String numPares(int k) {
    String cad="";

    if(k%2!=0) {
    k+=1;}
    if(k<=144) {
        return cad+=k+"\n"+numPares(k+=2);
    }
    return "";
}
```

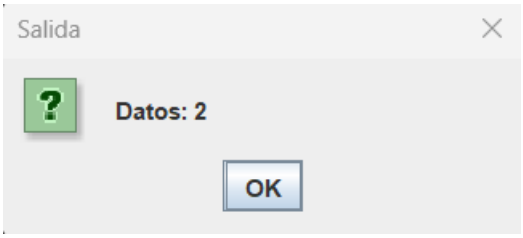
Resultado:



8. ctaDigitos: este método recibe dos parámetros donde el primero será avaluado si es diferente de 0 y si es así se llama al método y se divide el dato entre 10 y c se incrementa, al dar cero retornara el valor de c.

```
public static String ctaDigitos(int dato,byte c) {

    if(dato!=0)
    {
        return ctaDigitos(dato/=10, (byte) (c+1));
    }

    return "Datos"+c;
}
```

Resultado:



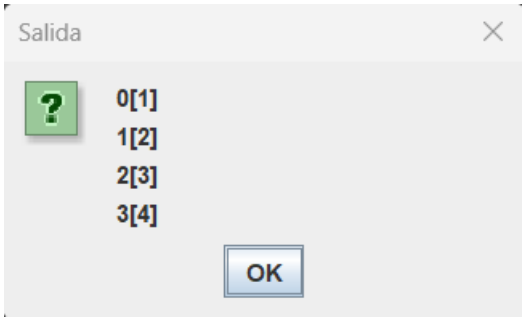
9. ordenarBurbuja: el método se compone de 2 métodos el primero tiene 2 parámetros, en el primero se evalúa si la i en menor a la longitud menos 1 del arreglo y si es así se llaman a los métodos incrementando el valor de i en los dos de uno en uno, después en el segundo método que tiene 3 parámetros y se evalúa si j es menor el tamaño del arreglo, si se cumple se hace otra evaluación si a[i]>a[j] si se cumple se asigna al auxiliar a[i], a[i]=a[j] y a[j]=aux, al terminar esto se llama al método y se incrementa la j

```
public static void ordenaBurbujai(int a[],int i) {
    if(i<a.length-1) {
        ordenaBurbujaj(a,i,i+1);
        ordenaBurbujai(a,i+1);
    }
}

public static void ordenaBurbujaj(int a[] , int i, int j) {
    int aux=0;
    if(j<a.length) {
        if(a[i]>a[j]) {
            aux=a[i];
            a[i]=a[j];
            a[j]=aux;
        }
        ordenaBurbujaj(a, i, j+1);
    }
}
```

10. imprimeArrayRec: el primer método tiene 2 parámetros y se evalúa si i es menor a el tamaño del arreglo y después si se cumple concatena los valores en una variable String, después de esto se llama al método y hace que se incremente el valor de i, en el segundo método se hace la misma evaluación y si se cumple se retorna la posición mas el valor de la posición del arreglo y se llama al método para poder incrementar le valor de i, al terminar solo retorna un espacio en blanco cuando no se cumple la condición.

```
public static void imprimeArrayRec(int a[],int i) {
    String cad="";

    if(i<a.length) {
        cad+=i+"["+a[i]+"]\n";
        imprimeArrayRec(a,i+1);
    }

    Tools.imprimirMSJE("Datos de arreglo\n"+cad);
}

public static String imprimeArrayRec2(int a[],int i) {

    if(i<a.length) {
        return i+"["+a[i]+"]\n"+imprimeArrayRec2(a,i+1);

    }

    return"";
}
```

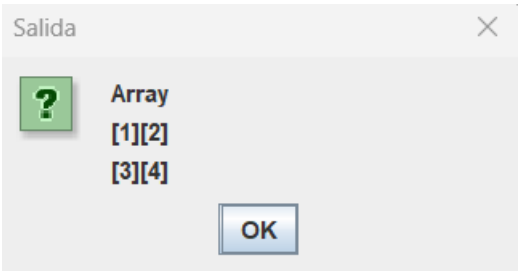
Resultado:



11. verMatrizRecursivo: se le pone un parámetro al primer método que será un arreglo un los métodos, se inicializa un variable entera que se inicializa con el valor de 0 y se compara con el tamaño del arreglo, se concatenara un salto de línea en un dato de valor String y después se llamará al método y por último se imprimirá lo que se concateno en ca, en el segundo método se evalúa si j en menor al tamaño del arreglo si se cumple se concatena los calores de la tabla en ca, sino se cumple regresa un valor vacío.

```
public static void verMatrizRecursivo(int tabla[][]) {
    int i=0;
    String ca="";
    if(i<tabla.length) {
        ca+="\n";
        verMatrizRecursivo(tabla);
    }
    Tools.imprimirMSJE("Array\n"+ca);
}

public static String verMatrizRecursivo2(int tabla[][]) {
    int i = 0,j=0;
    String ca="";
    if(j<tabla.length) {
        ca+="["+tabla[i][j]+"]"+"";
    }
    return "";
}
```

Resultado:



Conclusión:

En conclusión, cada uno de los métodos tienen la similitud que cada uno maneja recursividad sin importar el tipo, tratando de mejorar los diferentes métodos ya hechos antes, haciendo mejores y diferentes de manejar los métodos de esta forma haciendo los más rápidos en cierto modo y así mejorarlos, en comparación con sus versiones anteriores de forma iterativa que no hacen tan rápidamente el mismo proceso, de este modo se trato de mejorar de la mejor forma los programas hechos cona anterioridad, creemos que se logra hacer en la mayoría de casos y así darles una mayor optimización.

Referencias:

School, T. (2022, May 11). ¿Qué es la recursividad en la programación con Java | Tokio. Tokio School. https://www.tokioschool.com/noticias/recursividad-programacion-java/

Recursividad: Conceptos básicos. (2023). Tutorialesprogramacionya.com. https://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?codigo=123&punto=&inicio=

Cecilio Álvarez Caules, & Cecilio Álvarez Caules. (2020, October 28). Java Recursividad y manejo de Ficheros. Arquitectura Java. https://www.arquitecturajava.com/java-recursividad-y-manejo-de-ficheros/

Nivardo. (2020, September 9). ▷ Recursividad en JAVA → 【Tutorial de Java – 2023】. Java. https://oregoom.com/java/recursividad/

Programación en Java/Funciones recursivas - Wikilibros. (2019). Wikibooks.org. https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java/Funciones_recursivas

Ejercicios resueltos de recursividad en JAVA. (2022, November 19). Aula En La Nube. https://aulaenlanube.com/zona-programacion/java/ejercicios-recursividad-java/