Instituto tecnológico de Orizaba

Estructura De Datos

Unidad 2: Recursividad

Carrera: Ingeniería en Sistemas Computacionales

Grupo: 3g2B

Hora: 17:00 – 18:00 hrs

Profesora: Martínez Castillo María Jacinta

Alumno: Moran De la Cruz Aziel

Introducción

La recursividad es un concepto importante en la programación que permite a una función o método llamarse a sí mismo para resolver un problema de manera repetitiva. En Java, la recursividad se logra mediante la técnica de llamada recursiva, donde una función se llama a sí misma dentro de su propia definición.

La recursividad ofrece una forma elegante y poderosa de abordar problemas que se pueden descomponer en subproblemas más pequeños y similares al problema original. Al dividir el problema en instancias más pequeñas y resolverlos de forma recursiva, se puede lograr una solución más clara y concisa.

Competencia Específica

La capacidad de resolver problemas de forma eficiente utilizando estructuras de datos recursivas. Un ejemplo común es la manipulación de estructuras de datos como árboles y listas enlazadas mediante la recursividad.

Por ejemplo, supongamos que tenemos un árbol binario y queremos realizar un recorrido en profundidad (preorden, inorden o postorden) para procesar cada nodo. Podemos implementar esta operación utilizando la recursividad. La función recursiva se llamaría a sí misma para procesar los hijos izquierdo y derecho del nodo actual, y así sucesivamente hasta llegar a los nodos hoja.

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

Los materiales utilizados fueron

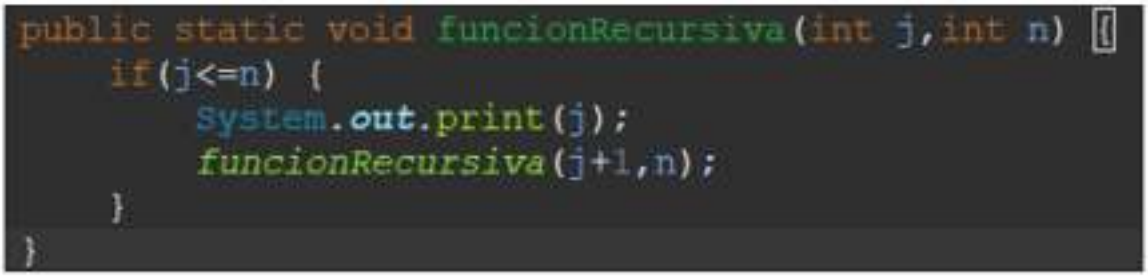
Computadora

Netbeans

Material de apoyo para la práctica

Desarrollo

1. funciónRecursiva: el método recibe dos valores que después evalúa el privalor de j es menor que el valor de n y si es así imprime el valor de j, despuse llama al método para poder incrementar el valor de j y así ir mostrandodemás 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 antersolo 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

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.

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.

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.

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.

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.

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

La recursividad es una técnica poderosa en la programación que permite resolver problemas de manera elegante y concisa. Al utilizar la recursividad, una función o método puede llamarse a sí mismo para abordar subproblemas más pequeños y similares al problema original.

La recursividad ofrece ventajas como la claridad y la simplicidad en la solución de problemas complejos. Permite dividir un problema en instancias más pequeñas y resolverlos de forma recursiva hasta alcanzar un caso base. Esto facilita el diseño de algoritmos y mejora la legibilidad del código.