2024-I

UNAM
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

MATERIA

ESTRUCTURA DE DATOS

TEMA

Recursión

PROFESOR

Mtro. ISC. MIGUEL ANGEL SÁNCHEZ HERNÁNDEZ

Ing. Miguel Angel Sánchez Hernández

Introducción

La recursión es un concepto importante en informática, muchos algoritmos se pueden describir mejor en término de recursividad, podemos decir que recursión es un método que hace, directa o indirectamente, una llamada así mismo. Sus desventajas es que podemos llegar hacer un ciclo infinito si no tenemos cuidado y a veces es difícil de crear si no tenemos experiencia. La forma de hacer llamadas recursivas es en forma directa, es decir cuando una función se llama a sí misma, o de forma indirecta una función A, llama a una función B y esta a su vez llama a la función A.

Las propiedades que debe tener una función recursiva es la siguiente:

1.- Debe de existir un criterio (criterio base) en donde le procedimiento no se llama a si mismo, es decir debe existir una solución no recursiva.

2.- Cuando se hace cada llamada, esta se debe acercar más al criterio base, es decir describir el problema en términos del mismo, pero más pequeño.

El ejemplo más general de recursión es el cálculo de factorial:

n! = 1 * 2 * 3 ... * (n-2) * (n-1) * n
0! = 1
1! = 1
2! = 1 * 2
3! = 1 * 2 * 3
...
n! = 1 * 2 * 3 ... * (n-2) * (n-1) * n
=> n! = n* (n-1)!
si n = 0 entonces n! = 1
si n > 0 entonces n! = n * (n-1)!

```
Function factorial(n:integer):longint:
begin
        if ( n = 0 ) then
                factorial:=1
        else
                factorial:= n * factorial(n-1);
end;
```

Rastreo de ejecución:
Si n = 6
Nivel
1) Factorial(6)= 6 * factorial(5) = 720
2) Factorial(5)=5 * factorial(4) = 120
3) Factorial(4)=4 * factorial(3) = 24
4) Factorial(3)=3 * factorial(2) = 6
5) Factorial(2)=2 * factorial(1) = 2
6) Factorial(1)=1 * factorial(0) = 1
7) Factorial(0)=1

Tenemos ahora un seudocódigo para calcular la serie de fibonacci.

0, 1, 1, 2, 3, 5, 8, ...
F0 = 0
F1 = 1
F2 = F1 + F0 = 1
F3 = F2 + F1 = 2
F4 = F3 + F2 = 3
...
Fn = Fn-1 + Fn-2

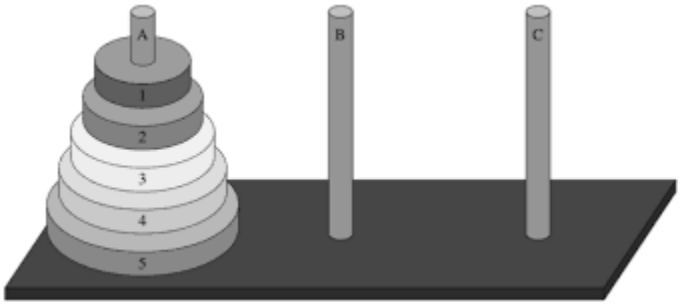
si n= 0, 1 Fn = n
si n >1 Fn = Fn-1 + Fn-2

```
function fibo ( n:integer) : integer;
begin
        if ( n = 0) or ( n = 1) then
                fibo:=n
        else
                fibo:=fibo(n-1) + fibo(n-2);
        end;
end;
```

Ejercicio 1: Ocupa el seudocódigo anterior para calcular la serie fibonacci e impleméntalo.

Torres de Hanoi

Supongamos que dan tres postes, A, B y C y suponga que el poste A se encuentra un número finito de n discos de tamaño decreciente como se muestra en la figura de abajo.



El objetivo del juego es mover los discos del poste A al poste C, usando el B como poste auxiliar. Las reglas del juego son las siguientes:

a) Sólo se pueden mover un disco cada vez, tomando en cuenta que solo el disco superior de un poste es el que se puede mover a otro poste.
b) Nunca pude haber un disco mayor sobre un disco menor.

Si nosotros ocupamos la siguiente notación podemos simplificar el proceso por ejemplo X→Y mover el disco superior del poste X al poste Y.

Si n=1
A→C
Si n=2
A→B, A→C, B→
Si n=3
A→B, A→C, B→C, A→B, C→A, C→B, A→B,A→C,B→C,B→A,C→A,B→C,A→B,A→C,B→C.

Ocupando las técnicas de recursión para n>1 podemos decir.

a) Mover los n-1 discos superiores del poste A al poste B
b) Mover el disco superior del A al poste C: A→C
c) Mover los n-1 discos superiores del poste B al poste C.

Seudocódigo de Torres-Hanoi

```
COM=POSTE INICIAL
FIN=POSTE FINAL
AUX=POSTE AUXILIAR

Torre(n,COM,AUX,FIN){
        Si n=1
                Escribir: COM→FIN
                Volver
        //mover los n-1 discos del poste COM al poste AUX
        Torre(n-1,COM,FIN,AUX)
        Escribir: COM→FIN
        //mover los n-1 discos del poste AUX al poste FIN
        Torre(n-1,AUX,COM,FIN)
}
```

Ejercicio 2: Implementa el seudocódigo anterior para resolver las torres de hanoi.

Implementación de procedimientos recursivos mediante Pila

Recordemos que cada llamada recursiva a una función puede tener parámetros así como variables locales, por lo que cada vez que se llama, se tiene que almacenar estos parámetros y variables locales y aparte la dirección de vuelta, todo esto se almacena en la pila de ejecución, si el proceso hace varias llamadas podemos tener un error "stack overflow", es decir se lleno la pila de ejecución, en java la maquina virtual su pila es de 256 MB por predeterminación. Para resolver esto podemos ocupar la estructura de pila, es decir cambiar las llamadas recursivas por una pila.

Ejercicio 3: Implementa factorial con pila.
Ejercicio 4: Implementa fibonacci con pila.
Ejercicio 5: Implementa las torres de Hanoi ocupando una pila.

Ejercicio 6: Considera el siguiente código recursivo para generar permutaciones ocupando una lista, impleméntalo y después cambia su recursividad ocupando una pila.

```
public class Pruebas {
public static void main(String[] args) {
      Lista elementos=new Lista();
      elementos.agregarCola(new String("a"));
      elementos.agregarCola(new String("b"));
      elementos.agregarCola(new String("c"));
      elementos.agregarCola(new String("d"));
      elementos.agregarCola(new String("e"));
      int n = 3;                //Tipos para escoger
      int r = elementos.getLongitud();   //Elementos elegidos
      Pruebas app = new Pruebas();
      app.permuta(elementos, "", n, r);
   }

   public void permuta(Lista elem, String act, int n, int r) {
      if (n == 0) {
         System.out.println(act);
      } else {
         for (int i = 0; i < r; i++) {
           // Controla que no haya repeticiones
            if (!act.contains((CharSequence) elem.obtenerNodo(i))) {
                     permuta(elem, act + elem.obtenerNodo(i) + ", ", n - 1, r);
            }
         }
      }
   }
}
```