ESTRUCTURA DE DATOS

TEMA 3. RECURSIVIDAD

Presenta: Mtro. David Martínez Torres
Universidad Tecnológica de la Mixteca
Instituto de Computación
Oficina No. 37
dtorres@gs.utm.mx

Contenido

1. Directa e indirecta
2. Comparación entre funciones iterativas y recursivas
3. Funciones recursivas con arreglos
4. Ejemplo de transformación de un algoritmo recursivo a iterativo
5. Ejemplo de transformación de un algoritmo iterativo a recursivo
6. Referencias

Introducción

Una **función recursiva** es una función que se llama a sí misma, ya sea directa o indirecta a través de otra función.

```
int factorial(int n) {
int fact;
if(n==0||n==1)
      fact=1;
else
      fact=n*factorial(n-1);
return fact;
}
```

Tiene dos componentes:

1. **El caso base:** es el resultado más simple, lo que conoce la función.
2. **El segundo, el paso de recursión:** Problema poco menos complejo que el original. También puede incluir la palabra reservada **return**.

1. Recursión directa e indirecta

Según el modo en que se realiza la llamada:

Directa: Cuando una función se invoca así mismo. Ejemplo factorial, potencia, etc.

```
int factorial(int n) {
int fact;
if(n==0||n==1)
      fact=1;
else
      fact=n*factorial(n-1);
return fact;
}
```

Indirecta: Cuando una función puede invocar a una segunda función que a su vez invoca a la primera

1. Recursión directa

Realice una prueba de escritorio para n=4. Después, codifique el programa

```
int factorial(int n) {
int fact;
if(n==0||n==1)
      fact=1;
else
      fact=n*factorial(n-1);
return fact;
}
```

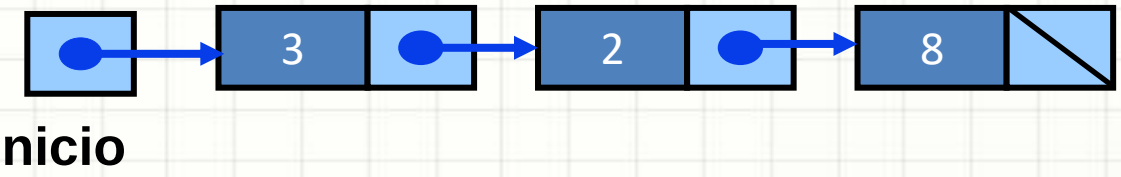
1. Recursión directa

Utilizando la siguiente definición matemática recursiva de la potencia, escriba la función correspondiente en un programa.

$$\text{POTENCIA} \qquad x^n = \begin{cases} x * x^{n-1} & \text{si } n \geq 1 \\ 1 & \text{si } n = 0 \end{cases}$$

1. Recursión directa

Realice una prueba de escritorio para la siguiente función, después implemente en un programa




```
void imprimir(tipoListaPtr listaT){
if(listaT==NULL)
        printf("NULL\n");
else {
        printf("%d   ",listaT->dato);
        imprimir(listaT->sig);
}
}
```

1. Recursión directa

Utilizando la siguiente definición recursiva de la serie de Fibonacci, escriba la función y realice una prueba de escritorio. Después codifique en un programa.

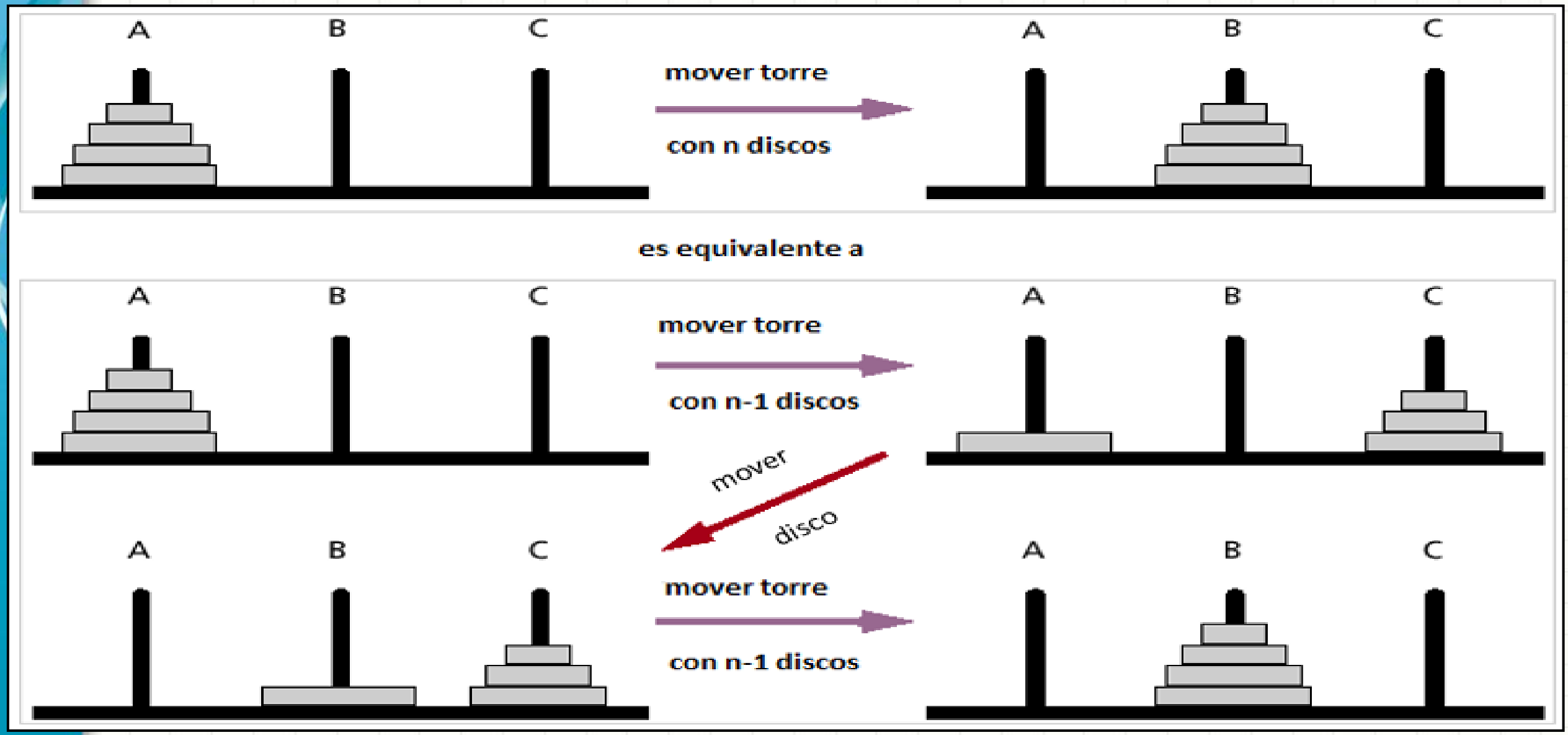
La serie Fibonacci (Leonardo de Pisa)
0,1,1,2,3,5,8,13,21,34, …

fibonacci(0)=0

fibonacci(1)=1

fibonacci(n)=fibonacci(n-1)+fibonacci(n-2)

1. Recursión directa: Torres de Hanoi



1. Recursión directa: Torres de Hanoi

Realice una prueba de escritorio para n=4, después, implemente el programa indicando también el número de disco en cada paso.

```
void moverTorres( int n, char desde, char
hacia,char temp){
if( n== 1 )
      printf("Mueve de %c  a %c .\n",desde,hacia);
else {
      moverTorres( n- 1, desde, temp, hacia);
      printf("Mueve de %c  a %c .\n",desde,hacia);
      moverTorres( n- 1, temp, hacia, desde);
}
}
```

```
int main(){
system("cls");
A('D');
printf("\n");
system("pause");
return 0;
}
void A(char c){
if(c>'A')
      B(c);
putchar(c);
}

void B(char c){
      A(--c);
}
```

1. Recursión indirecta

Realice una prueba de escritorio. Después, implemente el programa y verifique.

2. Comparación entre funciones recursivas e iterativas

Se basan en una **estructura de control**:

Las **iterativas** utilizan una estructura de repetición; las **recursivas** una estructura de selección.

Incluyen un **ciclo de repetición**:

La **iteración** utiliza una estructura de repetición explícita; la **recursión** lo hace mediante llamadas de función repetidas.

Incluyen una **prueba de terminación**:

La **iteración** termina cuando falla la condición de continuación del ciclo; la **recursión** termina cuando se reconoce el caso base.

2. Comparación entre funciones recursivas e iterativas

Rendimiento

En el caso de la recursión el invocar repetidamente la misma función, puede resultar costosa en tiempo de procesador y espacio de memoria.

Por el contrario la iteración se produce dentro de una función.

2. Comparación entre funciones recursivas e iterativas

¿Cuándo elegir recursión?

La razón fundamental es que existen numerosos problemas complejos que poseen naturaleza recursiva, por lo cual son más fáciles de implementar con este tipo de algoritmos

Sin embargo, en condiciones críticas de tiempo y de memoria, la solución a elegir debe ser normalmente de forma iterativa.

3. Funciones recursivas con arreglos

Realice una prueba de escritorio para v[5]={2,1,4,3,5]. Después codifique y verifique en un programa.

```
int suma(int vector[], int fin){
int result;
if(fin==0)
      result=vector[0];
else
      result=vector[fin]+suma(vector,fin-1);
return result;
}
```

3. Funciones recursivas con arreglos de estructuras

Considere que tiene un arreglo de estructuras con datos de alumnos (nombre, edad, promedio). Escriba las siguientes funciones recursivas:

- Encuentre y devuelva el alumno que tiene el mayor promedio
- Calcule y devuelva la suma de las edades de los alumnos, posteriormente en la función principal calcule el promedio.

4. Transformación de un algoritmo recursivo a iterativo

Todo **algoritmo recursivo** puede ser transformado en otro de tipo **iterativo**, pero para ello a veces se necesita utilizar pilas donde almacenar los cálculos parciales y el estado actual del subprograma recursivo.

Es posible **estandarizar** los pasos necesarios para convertir un algoritmo recursivo en iterativo, aunque el algoritmo así obtenido requerirá una posterior labor de optimización.

4. Transformación de un algoritmo recursivo a iterativo [6]

Tipos de transformaciones:

- Recursivas finales
- Recursivas no finales

4.1 Recursivas finales

Esquema recursivo

```
tipo f(tipo x){
tipo res;
if(Condicion(x))
        res = (CasoBase)
else
        res = f(Prec(x))
return res;
}
```

Esquema iterativo

```
tipo f(tipo x){
tipo res;
while (!Condicion(x))
        res = Prec(x);
res = (CasoBase)
return res;
}
```

4.1 Recursivas finales: ejemplo 1

Transformar el algoritmo recursivo, que calcula el residuo de la división de dos enteros, a su correspondiente algoritmo iterativo

Esquema recursivo

```
tipo f(tipo x){
tipo res;
if(Condicion(x))
    res = (CasoBase)
else
    res = f(Prec(x))
return res;
}
```

Esquema recursivo

```
int residuo(int a, int b){
int res;
if(a<b)
      res=a;
else
      res=residuo(a-b,b);
return res;
}
```

Equivalencia

x	→ (a,b)
<Condicion (a,b)>	→ a<b
<CasoBase>	→ a, si a<b
Prec(a,b)	→ (a-b,b)

4.1 Recursivas finales: ejemplo 1

x	→ (a,b)
<Condicion (a,b)>	→ a<b
<CasoBase>	→ a, si a<b
Prec(a,b)	→ (a-b,b)

Ejemplo, pruebe con a=6 , b=2; y, a=5 , b=2.

Esquema recursivo

```
tipo f(tipo x){
tipo res;
if(Condicion(x))
    res = (CasoBase)
else
    res = f(Pred(x))
return res;
}
```

Esquema iterativo

```
tipo f(tipo x){
tipo res;
while (!Condicion(x))
        res = Pred(x);
res = (CasoBase)
return res;
}
```

Esquema recursivo

```
int residuo(int a, int b){
int res;
if(a<b)
    res=a;
else
    res=residuo(a-b,b);
return res;
}
```

Esquema iterativo

```
int residuo(int a, int b){
int res;
while(!(a<b))
    a=a-b;
return res=a;
}
```

4.1 Recursivas finales: ejemplo 2

Transformar el algoritmo recursivo, que imprime el contenido de una lista enlazada, a su correspondiente algoritmo iterativo, revise el patrón.

Esquema recursivo

```
void imprimirL(tipoListaPtr lista){
if(lista == NULL)
    printf("NULL");
else {
    printf("%d ->", lista->dato);
    imprimir(lista->sig);
}
}
```

Esquema iterativo

```
void imprimirL(tipoListaPtr lista) {
while(!(lista == NULL)) {
    printf("%d ->", lista->dato);
    lista=lista->sig;
}
printf("NULL");
}
```

4.2 Recursivas no finales

Esquema recursivo

```
tipo f(tipo x){
tipo result;
if (Condicion(x))
      result = (CasoBase);
else
      result = C(x,f(Prec(x)));
return result;
}
```

Esquema iterativo

```
tipo f(tipo x){
tipo result;
result= (CasoBase);
while (!Condicion(x)) {
      result= C(result,x);
      x= Prec(x);
}
return result;
}
```

4.2 Recursivas no finales: ejemplo

Transformar el algoritmo recursivo, que calcula la suma de los elementos de un vector a su correspondiente iterativo

Recursivas no finales: Ejemplo

Esquema recursivo

```
int suma (int A[], int i){
int result;
if (i < 0)
      result= 0;
else
      result = A[i] + suma (A, i-1);
return result;
}
```

- Equivalencias
  - Recursiva

x →(A,i)

<Condicion(A,i)> → i<0

<CasoBase>para (A,-1) = 0

C(x,f(Pred(x)))=A[i]+suma(A, i-1)

  - Iterativa

C(result,x) = result+A[i]

Prec(x) = i-1

- **result** es la pila donde se almacenan las llamadas

Ejemplo de recursivas no finales

- Equivalencias recursiva

x $\rightarrow$(A,i)

<Condicion(A,i)> $\rightarrow$ i<0

<CasoBase>para (A,-1) = 0

C(x,f(Pred(x)))=A[i]+suma(A, i-1)

- Equivalencias recursiva

C(result,x) = result+A[i]

Prec(x) = i-1

result es la pila donde se almacenan las llamadas

Esquema recursivo

```
tipo f(tipo x){
tipo result;
if (Condicion(x))
      result = (CasoBase);
else
      result =
   C(x,f(Prec(x)));
return result;
}
```

Esquema iterativo

```
tipo f(tipo x){
tipo result;
result= (CasoBase);
while !(Condicion(x)){
   result= C(result,x);
   x= Prec(x);
}
return result;
}
```

Esquema recursivo

```
int suma (int A[], int i){
int result;
if (i < 0)
   result= 0;
else
   result = A[i] + suma (A, i-1);
return result;
}
```

Esquema iterativo

```
int suma (int A[], int i){
int result;
result=0;
while (!(i <0)){
     result=result+A[i];
     i=i-1;
}
return  result;
}
```

Recursivas no finales: Ejemplo

- Ahora convierta la función factorial recursiva a una función iterativa siguiendo el algoritmo anterior.

```
int factorial(int n) {
int fact;
if(n==0||n==1)
        fact=1;
else
        fact=n*factorial(n-1);
return fact;
}
```

6. Referencias

1. Langsam Y., Augenstein M. J., Tenenbaum A. M. Estructuras de Datos en C. Prentice-Hall, 1997.
2. Wirth, Niklaus. Algoritmos y estructura de Datos. Prentice-Hall.
3. Joyanes A. L, et. al. Estructuras de datos en C. McGraw-Hill, 2005.
4. Cairó O., Guardati S. Estructuras de datos . 3ª edición. McGraw-Hill, 2010.
5. Kerrighan y Ritchie. El lenguaje de programación. Prentice Hall
6. Gottfried, Byron. Programación en C. McGraw-Hill.
7. H. Schildt. C++ from the Ground Up. McGraw-Hill, 1998
8. Kelley A., Pohl I. A Book on C: Programming in C. 4th edition. Addison-Wesley, 1998
9. Serrano Montero, Monserrat. Estructura de datos. Universidad de Valladolid.