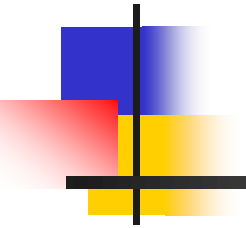


ESTRUCTURA DE DATOS:
Tema 3. Recursividad

Presenta: David Martínez Torres

Universidad Tecnológica de la Mixteca

Instituto de Computación

Oficina No. 37

dtorres@mixteco.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.

Tiene dos componentes, el caso base: es el resultado más simple, lo que conoce la función.

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.

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

1. Recursión directa

Ejemplificar

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

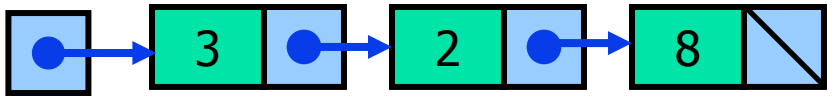
1. Recursión directa

Escriba una función potencia con recursión directa.

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

1. Recursión directa

Ejemplificar

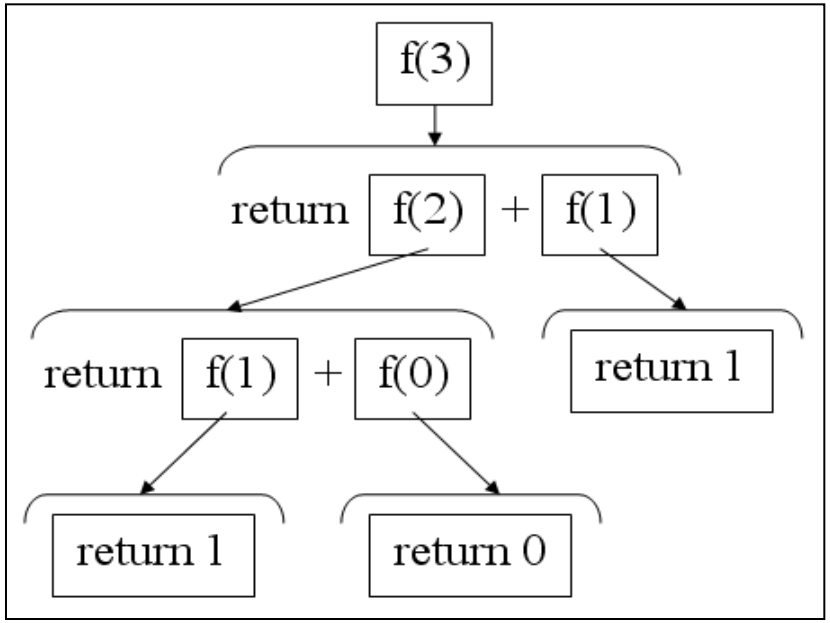


inicio

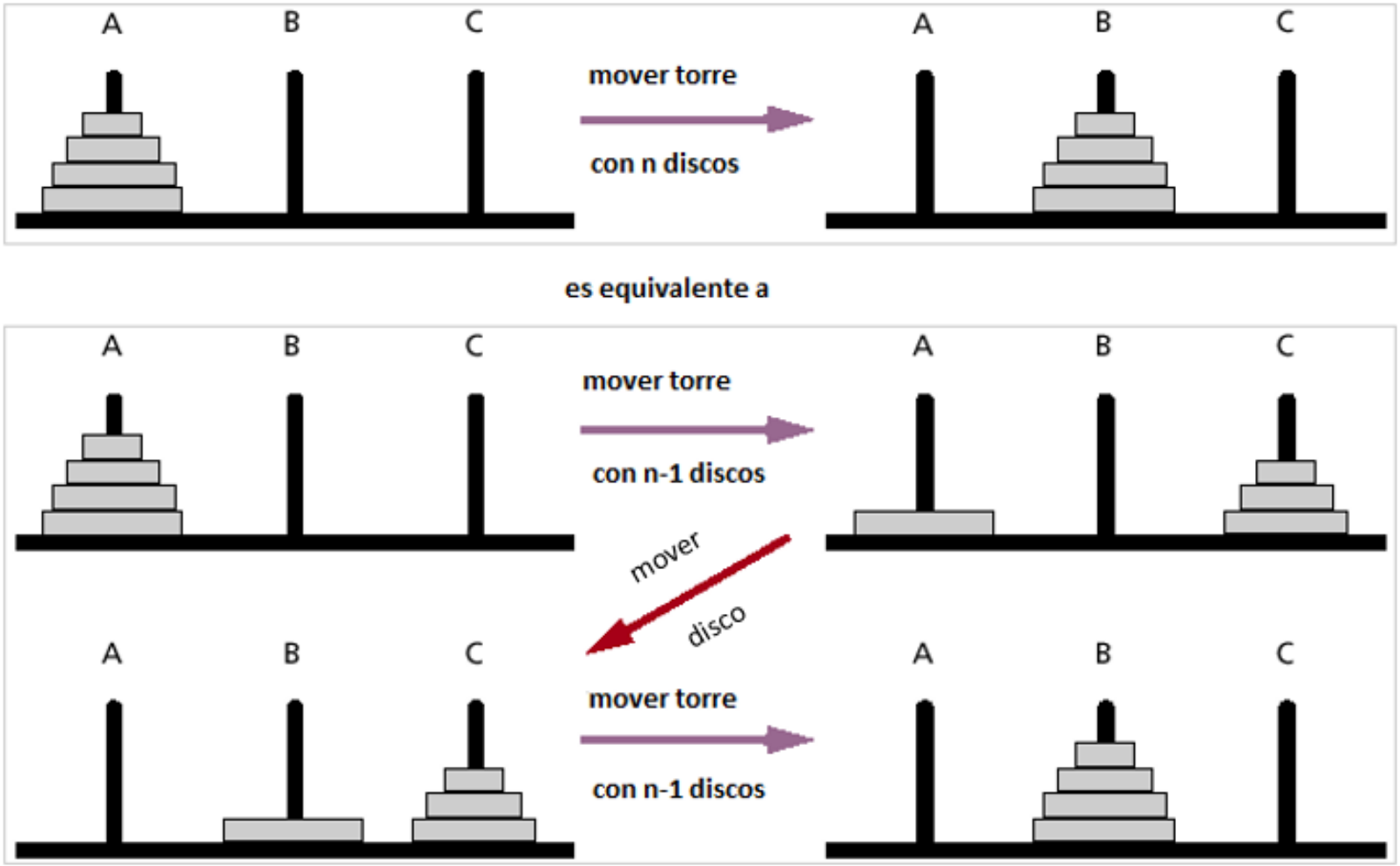
```
void imprimir(tipoListaPtr temp){
if(temp==NULL)
        printf("NULL\n");
else {
        printf("%d ",temp->dato);
        imprimir(temp->sig);
}
}
```

1. Recursión directa

La serie Fibonacci
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

```
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);
}
}
```

1. Recursión indirecta

```
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);
}
```

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

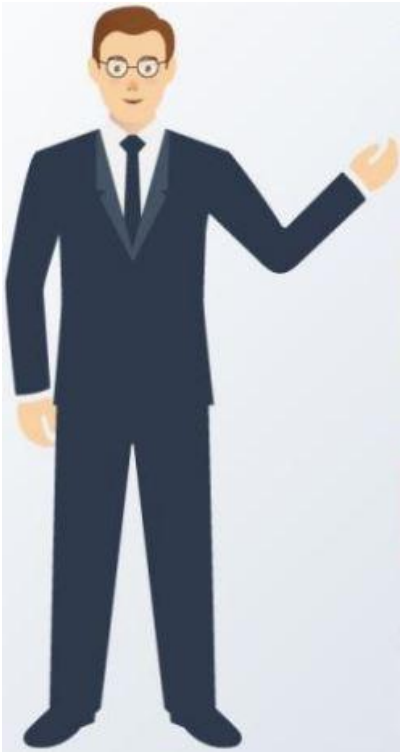
Sin embargo, en condiciones críticas de tiempo y de memoria, la solución a elegir debe ser normalmente de forma iterativa.

3. Funciones recursivas con arreglos

Ejemplo. La suma de los elementos del arreglo

```
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. Escriba las siguientes funciones recursivas:

- Encuentre el alumno que tiene el mayor promedio
- La suma de las edades de los alumnos
- Busque un alumno en el grupo y devuelva su posición en el arreglo

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

- 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(Pred(x))
return res;
}
```

Esquema iterativo

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

```
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
Pred(a,b)	→ (a-b,b)

4.1 Recursivas finales: ejemplo 1

Equivalencia

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

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

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(Pred(x)));
return result;
}
```

Esquema iterativo

```
tipo f(tipo x){
tipo result;
result= (CasoBase);
while !(Condicion(x)) {
        result= C(result,x);
        x= Pred(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]

Pred(x) = i-1

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

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;
}
```

Esquema iterativo

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

7. Referencias

1. Joyanes Aguilar, Luis (1996) *Fundamentos de programación, Algoritmos y Estructura de datos.* McGraw-Hill, México.
2. Deitel & Deitel (2001) *C++ Como programar en C/C++.* Prentice Hall
3. Kerrighan y Ritchie "*El lenguaje de programación".* Prentice Hall
4. Gottfried, Byron (1999) *"Programación en C"* McGrawHill, México.
5. Levine Gutierrez, Guillermo (1990) *Introducción a la computación y a la programación estructurada.* McGraw-Hill, México.
6. Levine Gutierrez, Guillermo (1990) *Introducción a la computación y a la programación estructurada.* McGraw-Hill, México.
7. H. Schildt, C++ from the Ground Up, McGraw-Hill, Berkeley, CA, 1998
8. Keller,,AL;Pohl,Ira. A Book on C. 3 ª edición. Edit.Benjamin umnings.1995