3 De recursivo a iterativo

ALGORÍTMICA Y COMPLEJIDAD

Camilo Palazuelos Calderón

Curso 2023-2024

Transformación de recursivo a iterativo

- Estudiaremos estrategias automáticas de transformación de diseños recursivos a iterativos

- Nos centraremos en la recursión lineal (una sola llamada recursiva) y sus dos posibles casos
  - *Recursión lineal final:* cuando la propia llamada recursiva es el último paso de nuestro algoritmo
  - *Recursión lineal no final:* cuando modificamos el resultado de la llamada recursiva después de esta
  - También existen estrategias automáticas de transformación de algoritmos recursivos múltiples a iterativos, pero requieren técnicas algorítmicas algo más sofisticadas para su implementación

- Estas estrategias son de gran utilidad en computación
  - En su caso más general, imitan el comportamiento de un compilador
  - Recorren de manera natural el árbol de llamadas

Recursión lineal final, I

```
RECURSIVO(X):
    if ESSENCILLO(X)
        Y ← DIRECTO(X)
    else
        Y ← RECURSIVO(F(X))
    return Y
```

```
ITERATIVO(X):
    X' ← X   ▷ Solo se copian los parámetros que cambian
    while not ESSENCILLO(X')
        X' ← F(X')
    Y ← DIRECTO(X')
    return Y
```

- Ejemplo: Suma de los elementos de un array
  - ESSENCILLO es TRUE si $i < n$
  - DIRECTO es $s + A[i]$
  - F cambia s por $s + A[i]$
  - F cambia i por $i + 1$

```
SUMAR1(A[1..n]):
    return SUMAR1(A[1..n], 0, 1)

SUMAR1(A[1..n], s, i):
    if i ≥ n
        return s + A[i]
    else
        return SUMAR1(A[1..n], s + A[i], i + 1)
```

Recursión lineal final, II

- Ejemplo: Suma de los elementos de un array

```
SUMAR1(A[1..n]):
    return SUMAR1(A[1..n], 0, 1)

SUMAR1(A[1..n], s, i):
    if i ≥ n
        return s + A[i]
    else
        return SUMAR1(A[1..n], s + A[i], i + 1)
```

```
SUMAR2(A[1..n]):
    return SUMAR2(A[1..n], 0, 1)

SUMAR2(A[1..n], s, i):
    s' ← s
    i' ← i
    while i' < n
        s' ← s' + A[i']
        i' ← i' + 1
    return s' + A[i']
```

```
SUMAR3(A[1..n]):
    s ← 0
    i ← 1
    while i < n
        s ← s + A[i]
        i ← i + 1
    return s + A[i]
```

Recursión lineal no final sin pila, I

```
RECURSIVO(X):
    if ESSENCILLO(X)
        Y ← DIRECTO(X)
    else
        Y ← C(RECURSIVO(F(X)), X)
    return Y
```

```
ITERATIVO(X):
    X' ← X   ▷ Solo se copian los parámetros que cambian
    while not ESSENCILLO(X')
        X' ← F(X')
    Y ← DIRECTO(X')
    while X' ≠ X
        X' ← F⁻¹(X')
        Y ← C(Y, X')
    return Y
```

- Para poder aplicar este esquema, debe existir F^{-1}
  - Si no existe, se utiliza una *pila* para almacenar los valores que recuperar posteriormente
  - Si, por ejemplo, $F(X') = 2X' \Longrightarrow F^{-1}(X')$ *existe y es* $X'/2$, pero si $F(X') = \lfloor X'/2 \rfloor \Longrightarrow F^{-1}(X')$ *no existe*

Recursión lineal no final sin pila, II

- Ejemplo: Suma de los elementos de un array

```
SUMAR1(A[1..n]):
    return SUMAR1(A[1..n], 1)

SUMAR1(A[1..n], i):
    if i ≥ n
        return A[i]
    else
        return SUMAR1(A[1..n], i + 1) + A[i]
```

```
SUMAR2(A[1..n], i):
    i' ⟵ i
    while i' < n
        i' ⟵ i' + 1
    Y ⟵ A[i']
    while i' ≠ i
        i' ⟵ i' – 1
        Y ⟵ Y + A[i']
    return Y
```

```
SUMAR3(A[1..n]):
    i ⟵ n
    Y ⟵ A[i]
    while i ≠ 1
        i ⟵ i – 1
        Y ⟵ Y + A[i]
    return Y
```

Recursión lineal no final con pila, I

```
RECURSIVO(X):
    if ESSENCILLO(X)
        Y ← DIRECTO(X)
    else
        Y ← C(RECURSIVO(F(X)), X)
    return Y
```

```
ITERATIVO(X):
    X' ← X   ▷ Solo se copian los parámetros que cambian
    P ← ()
    while not ESSENCILLO(X')
    APILAR(X', P)
        X' ← F(X')
    Y ← DIRECTO(X')
    while not VACÍA(P)
        X' ← DESAPILAR(P)
        Y ← C(Y, X')
    return Y
```

- Ejemplo: Suma de los dígitos de un número

```
SUMAR1(n):
    if n < 10
        return n
    else
        return SUMAR1(⌊n/10⌋) + n mod 10
```

Recursión lineal no final con pila, II

- Ejemplo: Suma de los dígitos de un número
  - (1) EsSencillo es True si $n < 10$, (2) Directo es n, (3) F cambia n por $\lfloor n/10 \rfloor$
  - (4) Si Y es el resultado de la llamada recursiva, C cambia Y por $Y + n \bmod 10$

```
SUMAR1(n):
    if n < 10
        return n
    else
        return SUMAR1(⌊n/10⌋) + n mod 10
```

```
SUMAR2(n):
    n' ← n
    P ← ()
    while n' ≥ 10
        APILAR(n', P)
        n' ← ⌊n'/10⌋
    Y ← n'
    while not VACÍA(P)
        n' ← DESAPILAR(P)
        Y ← Y + n' mod 10
    return Y
```