Desarrollo Sistemático de Programas

Trabajo Práctico N.º 3

Algoritmos Iterativos y Recursivos – Resolución explicada

Consigna general. Para cada ejercicio se propone un algoritmo iterativo y otro recursivo, se calcula su orden y se indica cuál de los dos resulta más conveniente y por qué.

1) Sumar los elementos pares de un vector de dimensión N

Qué se pide. Dado un vector de números, sumar solamente los elementos pares. No se utiliza una función Par(); la paridad se verifica directamente con el resto de la división por 2.

A) Algoritmo iterativo

```
Función SumaParesIterativa(A, N) retorna entero
    suma ← 0
    Para i desde 1 hasta N hacer
        Si A[i] mod 2 = 0 entonces
            suma ← suma + A[i]
        FinSi
    FinPara
    Retorna suma
FinFunción
```

Explicación. Se recorre el vector una sola vez. Cada vez que un elemento resulta par, se acumula en la variable suma.

Orden. $\Theta(N)$

B) Algoritmo recursivo

```
Función SumaParesRecursiva(A, N) retorna entero
    Si N = 0 entonces
        Retorna 0
    Sino
        Si A[N] mod 2 = 0 entonces
            Retorna A[N] + SumaParesRecursiva(A, N - 1)
        Sino
            Retorna SumaParesRecursiva(A, N - 1)
        FinSi
    FinSi
FinFunción
```

Explicación. El caso base ocurre cuando ya no quedan elementos por analizar. En cada llamada se observa el último elemento del vector considerado y se reduce el tamaño del problema en una unidad.

Orden. $\Theta(N)$

¿Cuál es mejor?

Los dos algoritmos son lineales. En la práctica conviene el iterativo, porque usa menos memoria y evita el costo adicional de las llamadas recursivas.

2) Calcular la potencia n de un número x

Qué se pide. Se desea calcular x^n.

A) Algoritmo iterativo

```
Función PotenciaIterativa(x, n) retorna entero
    pot ← 1
    Para i desde 1 hasta n hacer
        pot ← pot * x
    FinPara
    Retorna pot
FinFunción
```

Explicación. Se parte de 1 y se multiplica por x exactamente n veces.

Orden. Θ(n)

B) Algoritmo recursivo

```
Función PotenciaRecursiva(x, n) retorna entero
    Si n = 0 entonces
        Retorna 1
    Sino
        Retorna x * PotenciaRecursiva(x, n - 1)
    FinSi
FinFunción
```

Explicación. Se usa la identidad $x^n = x \cdot x^{n-1}$. El caso base es n = 0, donde el resultado vale 1.

Orden. Θ(n)

¿Cuál es mejor?

Ambos algoritmos tienen orden lineal. El iterativo vuelve a ser más conveniente por simplicidad y por menor consumo de memoria.

3) Indicar si una palabra es palíndromo

Qué se pide. Dado un vector de caracteres, indicar si la palabra se lee igual de izquierda a derecha que de derecha a izquierda.

A) Algoritmo iterativo

```
Función EsPalindromoIterativo(A, N) retorna lógico
    i ← 1
    j ← N
    Mientras i < j hacer
        Si A[i] <> A[j] entonces
            Retorna Falso
        FinSi
        i ← i + 1
        j ← j - 1
    FinMientras
    Retorna Verdadero
FinFunción
```

Explicación. Se comparan los extremos del vector: primero con último, segundo con anteúltimo, y así sucesivamente. Si alguna pareja no coincide, la palabra no es palíndromo.

Orden. Θ(N)

B) Algoritmo recursivo

```
Función EsPalindromoRecursivo(A, izq, der) retorna lógico
    Si izq >= der entonces
        Retorna Verdadero
    Sino
        Si A[izq] <> A[der] entonces
            Retorna Falso
        Sino
            Retorna EsPalindromoRecursivo(A, izq + 1, der - 1)
        FinSi
    FinSi
FinFunción
```

Explicación. El caso base aparece cuando los índices se cruzan o se encuentran. Si los extremos son iguales, el problema se reduce al subvector interior.

Orden. Θ(N)

¿Cuál es mejor?

En eficiencia práctica conviene el iterativo. Sin embargo, la versión recursiva expresa muy naturalmente la idea del problema.

4) Generar un vector de pares y otro de impares

Qué se pide. A partir de un vector original, construir dos nuevos vectores: uno con los elementos pares y otro con los impares.

A) Algoritmo iterativo

```
Procedimiento SepararParImparIterativo(A, N, P, I, cp, ci)
    cp ← 0
    ci ← 0
    Para k desde 1 hasta N hacer
        Si A[k] mod 2 = 0 entonces
            cp ← cp + 1
            P[cp] ← A[k]
        Sino
            ci ← ci + 1
            I[ci] ← A[k]
        FinSi
    FinPara
FinProcedimiento
```

Explicación. Se recorre el vector una sola vez. Según la paridad del elemento, se lo agrega al vector de pares o al vector de impares.

Orden. Θ(N)

B) Algoritmo recursivo

```
Procedimiento SepararParImparRecursivo(A, N, k, P, I, cp, ci)
    Si k > N entonces
        FinProcedimiento
    Sino
        Si A[k] mod 2 = 0 entonces
            cp ← cp + 1
            P[cp] ← A[k]
        Sino
            ci ← ci + 1
            I[ci] ← A[k]
        FinSi
        SepararParImparRecursivo(A, N, k + 1, P, I, cp, ci)
    FinSi
FinProcedimiento
```

Explicación. Cada llamada procesa un elemento del vector y luego invoca recursivamente el procedimiento para la siguiente posición.

Orden. Θ(N)

¿Cuál es mejor?

Los dos algoritmos recorren todos los elementos una sola vez. El iterativo resulta más práctico y directo para este tipo de tarea secuencial.

5) Contar los caracteres que sean letras del alfabeto

Qué se pide. Dado un vector de caracteres, contar cuántos corresponden a letras del alfabeto.

A) Algoritmo iterativo

```
Función ContarLetrasIterativo(A, N) retorna entero
    cont ← 0
    Para i desde 1 hasta N hacer
        Si (A[i] >= 'A' y A[i] <= 'Z') o (A[i] >= 'a' y A[i] <= 'z')
        entonces
            cont ← cont + 1
        FinSi
    FinPara
    Retorna cont
FinFunción
```

Explicación. Se analiza cada carácter y se comprueba si pertenece al rango de letras mayúsculas o minúsculas.

Orden. Θ(N)

B) Algoritmo recursivo

```
Función ContarLetrasRecursivo(A, N) retorna entero
    Si N = 0 entonces
        Retorna 0
    Sino
        Si (A[N] >= 'A' y A[N] <= 'Z') o (A[N] >= 'a' y A[N] <= 'z')
        entonces
```

```
            Retorna 1 + ContarLetrasRecursivo(A, N - 1)
        Sino
            Retorna ContarLetrasRecursivo(A, N - 1)
        FinSi
    FinSi
FinFunción
```

Explicación. El caso base se alcanza cuando no quedan caracteres por revisar. Si el carácter actual es letra, se suma 1; si no, se continúa sin sumar.

Orden. Θ(N)

¿Cuál es mejor?

El iterativo es preferible por simplicidad y menor uso de memoria, aunque ambos tienen el mismo orden.

6) Determinar la cantidad de divisores de un entero positivo

Qué se pide. Dado un número entero positivo n, determinar cuántos divisores posee.

A) Algoritmo iterativo

```
Función CantDivisoresIterativo(n) retorna entero
    cont ← 0
    Para i desde 1 hasta n hacer
        Si n mod i = 0 entonces
            cont ← cont + 1
        FinSi
    FinPara
    Retorna cont
FinFunción
```

Explicación. Se prueban todos los valores entre 1 y n. Cada vez que n es divisible por i, se incrementa el contador.

Orden. Θ(n)

B) Algoritmo recursivo

```
Función CantDivisoresRecursivo(n, i) retorna entero
    Si i > n entonces
        Retorna 0
    Sino
        Si n mod i = 0 entonces
            Retorna 1 + CantDivisoresRecursivo(n, i + 1)
        Sino
            Retorna CantDivisoresRecursivo(n, i + 1)
        FinSi
    FinSi
FinFunción
```

Explicación. La idea es la misma que en el caso iterativo, pero en lugar de un ciclo se utiliza un parámetro i que avanza recursivamente desde 1 hasta n.

Orden. Θ(n)

¿Cuál es mejor?

Ambas propuestas son lineales. La iterativa se considera mejor porque evita el costo de las llamadas recursivas y es más sencilla de implementar.

Conclusión general

En todos los ejercicios propuestos, los algoritmos iterativos y recursivos tienen el mismo orden asintótico. Sin embargo, en términos prácticos, el algoritmo iterativo suele resultar más conveniente porque utiliza menos memoria y evita el costo adicional de las llamadas recursivas. La recursión, por su parte, puede ser útil para expresar de forma más natural algunos problemas, como el de palíndromos.