Análisis de algoritmos iterativos

Pedro O. Pérez M., MTI

Análisis y diseño de algoritmos
Tecnológico de Monterrey

pperezm@tec.mx

01-2019

Contenido

Herramientas a utilizar

- A todas las divisiones se le aplicará la función *floor*. La función *floor*(x) devuelve el entero más pequeño o igual a x. Por ejemplo, $floor(3.3) = floor(3.99999) = floor(3.5) = 3$.
- Propiedades de los logaritmos.
  - $\log_b a$ es una función estrictamente creciente y uno a uno.
  - $\log_b 1 = 0$.
  - $\log_b b^a = a$.
  - $\log_b(XY) = \log_b X + \log_b Y$
  - $\log_b X^a = a \log_b X$
  - $X^{\log_b Y} = Y^{\log_b X}$

Herramientas a utilizar

$$(1)\ \sum_{i=1}^{n} c = c * n$$
$$(2)\ \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$
$$(3)\ \sum_{i=1}^{n} i^2 = \frac{2n^3+3n^2+n}{6}$$

Sentencias simples

La sentencias simples son aquellas que ejecutan operaciones básicas, siempre y cuando no trabajen sobre variables estructuradas cuyo tamaño está relacionado con el tamaño del problema. La inmensa mayoría de las sentencias simples requieren un tiempo constante de ejecución y su complejidad es $O(1)$.
Ejemplos:

$x \leftarrow 1$
$y \leftarrow z + x + w$
print x
read x

Condicionales

Los condicionales suelen ser $O(1)$, a menos que involucren un llamado a un procedimiento, y siempre se debe tomar la peor complejidad posible de las alternativas del condicional, bien en la rama afirmativa o bien en la rama positiva. En decisiones múltiples (*switch*) se tomará la peor de todas las ramas. Ejemplo:

if $a > b$ **then**
  for $i \leftarrow 1$ **to** n **do**
    $sum \leftarrow sum + 1$
  end for
else
  $sum \leftarrow 0$
end if

Ciclos (while, for, repeat-until)

En los ciclos con un contador explícito se distinguen dos casos: que el tamaño n forme parte de los límites del ciclo, con una complejidad basada en n, o que dependa de la forma como avanza el ciclo hacia su terminación.

Si el ciclo se realiza un número constante de veces, independientemente de n, entonces la repetición solo introduce una constante multiplicativa que puede absorberse, lo cual da como resultado $O(1)$.
Ejemplo:

for $i \leftarrow 1$ **to** k **do**
  sentencias simples $O(1)$
end for

Si el tamaño n aparece como límite de las iteraciones, entonces la complejidad será: n * $O(1) \rightarrow O(\text{n})$.
Ejemplo:

for $i \leftarrow 1$ **to** n **do**
  sentencias simples $O(1)$
end for

Si los ciclos son aninadados...
Ejemplo:

for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** n **do**
    sentencias simples $O(1)$
  end for
end for

En este caso, la complejidad sería: n * n * $O(1) \rightarrow O(n^2)$.

Para ciclos anidadados pero con variables independientes:
Ejemplo:

for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** i **do**
    sentencias simples $O(1)$
  end for
end for

$\sum_{i=1}^{n} \sum_{j=1}^{i} O(1) = \sum_{i=1}^{n} i = \frac{n(n-1)}{2} = O(n^2)$

A veces aparecen ciclos multiplicativos, donde la evolución de la variable de control no es lineal (como en los casos anteriores):
Ejemplo:

$c \leftarrow 1$
while $c < n$ **do**
  $c \leftarrow c * 2$
end while

El valor inicial de la variable c es 1, y llega a 2^n al cabo de n iteraciones $\rightarrow \log_2 n$.

Y la combinación de los anteriores:
Ejemplo:

```
for i ← 1 to n do
  c ← i
  while c > 0 do
    c ← c/2
  end while
end for
```

Se tiene un ciclo interno de orden $O(\log_2 n)$ que se ejecuta n veces en el ciclo externo; por lo que, el ejemplo es de orden $O(n \log_2 n)$.

Llamada a procedimientos

La complejidad de llamar a un procedimiento viene dada por la complejidad del contenido del procedimiento en sí.
Ejemplo:

$a \leftarrow 10$
$b \leftarrow 20$
$c \leftarrow FACTORIAL(a)$
$z \leftarrow a + b + c$

Si se tiene un ciclo con un llamado a una función:
Ejemplo:

for $i \leftarrow 1$ **to** n **do**
  $x \leftarrow FACTORIAL(i)$
end for

Si hay un ciclo que se realiza n veces, lo que generaría una complejidad O(n); pero como en su interior hay un llamado a la función *FACTORIAL*, la complejidad del ciclo es multiplicado por la complejidad de la función; en este caso sería O(n) * O(n) $\rightarrow O(n^2)$

Si hay dos o más llamadas a funciones:
Ejemplo:

$QUICKSORT(array, n)$
$DISPLAY(array, n)$

La complejidad del $QUICKSORT$ es de complejidad $O(\log_2 n)$ y que $DISPLAY$ simplemente muestra el contenido del arreglo en la pantalla con una complejidad de O(n), la complejidad total será mayor de los dos llamadas a las funciones, $O(n \log_2 n)$.

Procedure 1 MAX - Return the greatest element of an array

Input: $A : Array$

```
val ← A[1]
for i ← 2 to A.length do
    if A[i] > val then
        val = A[i]
    end if
end for
return val
```

Procedure 2 AVERAGE - Calculate the average of the elements of an array

Input: $A : Array$

$acum \leftarrow 0$
for $i \leftarrow 1$ **to** $A.length$ **do**
$acum \leftarrow acum + A[i]$
end for
return $acum/A.length$

Ejercicio de revisión

Procedure 3 POW - $y = x^n$

Input: $x : Real, n : Integer$

```
acum ← 1
for i ← 1 to n do
    acum ← acum * x
end for
```

Procedure 4 BUBLE_SORT -Sort the elements of an array

Input: A : *Array*

```
for i ← A.length to 2 do
    for j ← 1 to i − 1 do
        if A[j] > A[j + 1] then
            SWAP(A, j, j + 1)
        end if
    end for
end for
```

Procedure 5 INSERTION_SORT - Sort the elements of an array

Input: A : *Array*

```
for i ← 2 to A.length do
    j ← i
    while j >= 2 and A[j] < A[j − 1] do
        SWAP(A, j, j − 1)
        j ← j − 1
    end while
end for
```

Procedure 6 BINARY_SEARCH - Return the position in which a given element is in an array; otherwise, return the position where the element should be.

Input: $A : Array, key : Object$
  $low \leftarrow 1$
  $high \leftarrow A.length$
  while $low < high$ **do**
    $mid \leftarrow (high + low)/2$
    if $key = A[mid]$ **then**
      return mid
    else if $key < A[mid]$ **then**
      $high \leftarrow mid - 1$
    else
      $low \leftarrow mid + 1$
    end if
  end while
  return low

Ejercicio de revisión

Procedure 7 SELECTION_SORT - Sort the elements of an array

Input: $A : Array$
  for $i \leftarrow A.length$ **to** 2 **do**
    $higher \leftarrow 1$
    for $j \leftarrow 2$ **to** i **do**
      if $A[higher] < A[j]$ **then**
        $higher \leftarrow j$
      end if
    end for
    if $higher <> j$ **then**
      $SWAP(A, higher, j)$
    end if
  end for

Ejercicio de revisión

Procedure 8 POW2 - Calculate exponentiation by squaring

result $\leftarrow 1$
while $n > 0$ **do**
  if $n \mod 2 = 1$ **then**
    result $\leftarrow$ *result* $* x$
  end if
  $n \leftarrow n/2$
  $x \leftarrow x * x$
end while
return *result*