Ejemplos de algoritmos con su análisis de complejidad.

Díaz Hernández Marcos Bryan Grupo: 1 Tarea virtual - 1

5 - Cinco ejemplos de códigos o algoritmos con su respectivo análisis de complejidad.

1: El primer ejemplo es un arreglo con elementos distintos, con la salida si todos los elementos son iguales o cambian.

- A[0, n-1]: input
- Output: Retur "true" si todos los elementos son distintos

Código:
```
for i ← n-2 do
    for j ← i+1 to n-1 do
        if A[i] = A[j] return false
return true
```

Peor caso: contiene la mayor complejidad

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1)-(i+1)+1] = \sum_{i=0}^{n-2} (n-1-i)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i = (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2} = (n-1)^2 - \frac{(n-2)(n-1)}{2}$$

$$= \frac{(n-1)n}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2). \therefore \text{la complejidad es } \underline{O(n^2)}$$

2: El segundo ejemplo es un algoritmo recursivo, que consiste en el planteamiento de la torre de Hanoi, donde se deben colocar los discos desde el mas grande al mas pequeño.

n = número de movimientos

input: $M(n) = M(n-1) + 1 + M(n-1)$ para $n>1$

$M(n) = 2M(n-1) + 1 \quad n>1$

Para $n=1 \quad C_M(1) = 1$

Solución.

$M(n) = 2M(n-1) + 1$ sustitución $M(n-1) = 2M(n-2)+1$

$M(n-2) = 2M(n-3)+1$

$= 2[2M(n-2)+1]+1 = 2^2 M(n-2) + 2 + 1$

$= 4[2M(n-3)+1] + 2 + 1 = 2^3 M(n-3) + 2^2 + 2 + 1$

Ejemplos de algoritmos con su análisis de complejidad.

con $i=$ $M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \dots + 2 + 1 = 2^i M(n-i) + 2^i - 1$

donde $i = n-1$ se obtiene: $M(n) = 2^{n-1}[M(n-(n-1))] + 2^{n-1} - 1$

$= 2^{n-1} M(1) + 2^{n-1} - 1 = 2^{n-1} + 2^{n-1} = 2^n - 1$

Siendo el caso más simple, aunque la forma es la más sencilla en base a la cantidad de movimientos.

∴ la complejidad es $(GS) = \sum_{i=0}^{n-1} 2^i = 2^n - 1 = O(2^n)$

3: Algoritmo recursivo (BinRec(n))

- Input: A positive decimal integer n
- Output: Representación binaria. (# elementos)

```
if n=1 return 1
else return BinRec(n/2)+1
```

Cambio de forma a: $A(n) = A(n/2) + 1 \quad n > 1$

- Condición inicial: $A(1) = 0$

Se hace un cambio de $n = 2^k$ que sirve para hacer una aproximación del crecimiento de los valores de n.

$A(2^k) = A(2^{k-1}) + 1 \quad k > 0$

$A(2^0) = 0$

$A(2^k) = A(2^{n-1}) + 1$

$= [A(2^{k-2}) + 1] + 1 = A(2^{k-2}) + 2$

$= [A(2^{n-3}) + 1] + 2 = A(2^{k-3}) + 3$

$= A(2^{k-i}) + i$

$= A(2^{k-k}) + k$

Subsituto $A(2^{n-1}) = A(n-2) + 1$

$A(2^{k-2}) = A(2^{n-3}) + 1$

∴ $A(2^k) = A(1) + k = k$

Si $n = 2^k \quad k = \log_2 n$

$A(n) = \log_2 n \in O(\log n)$

∴ la complejidad es $O(\log n)$

4: Merge sort: algoritmo de ordenamiento. Código:

1) Merge-Sort(A, p, q, r)

```
if (p<r)
    then q ← (r+p)/2
    Merge-Sort(A, p, q)
    Merge-sort(A, q+1, r)
    Merge(A, p, q, r)
    n1 ← q-p+1
    n2 ← r-q
```

2) create arrays $L[1 \dots N1+1]$ and $R[1 \dots N2+1]$

```
for i ← 1 to N1
    do L[i] ← A[p+i-1]
for j ← 1 to N2
    do R[j] ← A[q+j]
L[N1+1] ← ∞
R[N2+1] ← ∞
i ← 1
j ← 1
```

Ejemplos de algoritmos con su análisis de complejidad.

3) for k ← p to r
do if Z[i] ≤ R[j]
then A[k] ← Z[i]
i ← i+1
else A[k] ← R[j]
j ← j+1

2a Combinacion de dos arreglos con elementos donde se busca el orden del menor al mayor puede ser el peor caso por estar dividida la cantidad de elementos mayores.

Demostración: se tiene que Zet T(N) = # número de comparaciones entre los elementos de los arreglos

1) $T(N) = N-1+2T(N/2)$
$T(1) = 0$
$T(N/2) = N/2 - 1 + 2T(N/4)$
$T(N) = N-1+N-2+4T(N/4)$
$= N-1+N-2+N-4+8T(N/8)$ → Sustitución para generalizar
$= N-1+N-2+N-2^{k-1}+2^k T(N/2^k)$ → $k = \log_2 N$

$T(N) = N\log N - N + 1$ ∴ $T(N) = O(N\log_2 N)$ Para el peor caso

2) $T(N) = N/2 - 5T(N/2)$
$T(1) = 0$
$T(N/2) = N/4 + 2T(N/4)$ → Se reemplaza en la 1)
$T(N) = N/2 + N/2 + 4T(N/4)$
$T(N) = kN/2 + 2^k T(N/2^k)$ → $k = \log_2 N$
$T(N) = N\log_2 N/2$ ∴ $T(N) = O(N\log 2N)$ Para el mejor caso

5: Quich sort: algoritmo de ordenamiento. Código:

1) void quich sort (int [] a, int lo, int hi)
{
int i = lo, j = h, h;
int x = a[h%*N];
do
{
while (a[i] < x) i++;
while (a[j] > x) j--;
if (i <= j)
}

2) h = a[i]; a[i] = a[j];
a[j] = h;
i++; j--;
}
} while (i <= j);
if (lo < j) quicksort (a, lo, j);
if (i < hi) quicksort (a, i, hi);

Ejemplos de algoritmos con su análisis de complejidad.

El algoritmo divide un arreglo y obtiene un término de pivote el cual va intercambiando de posición hasta encontrar su posición del menor a mayor, de forma consecutiva utiliza un pivote y lo va colocando hasta obtener orden.

Demostración: donde el T(N) es el numero de comparaciones para el tamaño de un arreglo

$T(N) = N-1 + T(N-1)$

$T(1) = 0$

$T(N-1) = (N-2) + T(N-2)$

$T(N-2) = N-3 + T(N-3)$

$T[N-(N-1)] = [N-(N-1)] + T[N-(N-1)]$

$= 1 + T(0) = 1$

$T(N) = (N-1) + T(N-1)$

$= (N-1) + (N-2) + T(N-2)$

$= (N-1) + (N-2) + (N-3) + \dots + T(N(N-1))$

$= (N^2 + N)/2$

$= O(N)^2 \therefore \underline{T(N) = O(N^2)}$

- Peor caso

$T(N) = N-1 + 2T(N-1)/2)$

$T(1) = 0$

$T(N-1)/2) = 2T(N-3)/4) + (N-1)/2 - 1$

$T(N) = 2[(2T(N-3)/4) + N(-1)/2 - 1)] + N-1$

$= 2^2 T((N-3)/2^2) + N-3 + N-1$

$T((N-3)2^2) = 2T((N-1)2^3) + (N-3) + N-1$

$T(N) = 2^n T\left(\frac{N-1+N-3+\dots N-(2n-1)2^n}{2^n}\right)$

$= 2^n T\left(\frac{N-2^n+1}{2^n}\right) + nN - (-n+2^{n+1})$

si $\frac{N-2^n+1}{2^n} = 1$

$\therefore T(N) = (N+1)\log(N+1) - 2N$

$= \underline{T(N) = O(N \log N)}$

- Mejor caso

Referencias:

- Anany I. (2007). The design & analysis of algorithms. México: Pearson.
- Song Q. Merge sort algorithm. 20104120 20. Florida Institute of Technology: Recuperado de: https://pdfs.semanticscholar.org/6804/987ab63d1879aa55ba68224dced142ce8774.pdf
- Song Q. Quick sort algorithm. 20104120 20. Florida Institute of Technology. Recuperado de: https://pdfs.semanticscholar.org/1268/4/3dbfc27055a99c1692f14d28e7777e0bdc7b.pdf.