Scopul analizei complexităţii

Analiza complexităţii unui algoritm are ca scop stabilirea *resurselor* necesare pentru execuţia algoritmului pe o maşină de calcul. Prin resurse înţelegem:

- *Spaţiul de memorie* necesar pentru stocarea datelor pe care le prelucrează algoritmul.
- *Timpul* necesar pentru execuţia tuturor prelucrărilor specificate în algoritm.

Timp de execuţie

Vom nota cu *T(n)* timpul de execuţie al unui algoritm destinat rezolvării unei probleme de dimensiune *n.*
Pentru a estima timpul de execuţie trebuie stabilit un *model de calcul* şi o *unitate de măsură*.

Model de calcul

Vom consideră un model de calcul (numit RAM) caracterizat prin:

- Prelucrările se efectuează în mod secvenţial.
- Operaţiile *elementare* sunt efectuate în timp constant *indiferent* de valoarea operanzilor.
- Timpul de acces la informaţie nu depinde de poziţia acesteia.

Unitatea de măsură

A stabili o unitate de măsură înseamnă a stabili care sunt *operaţiile elementare* şi a considera ca unitate de măsură timpul de execuţie a acestora. In acest fel timpul de execuţie va fi exprimat prin numărul de operaţii elementare executate.

Exemplu

Exemplul 1. Considerăm problema calculului $\sum_{i=1}^{n} i$. Dimensiunea acestei probleme este n. Algoritmul de rezolvare poate fi descris prin:

```
suma(n)
1:    S ← 0
2:    i ← 1
3:    WHILE i <= n DO
4:        ||S ← S + i
5:        ||i ← i + 1
RETURN S
```

unde operaţiile ce sunt contorizate sunt numerotate. Timpul de execuţie a algoritmului poate fi determinat folosind tabelul de costuri:

Operaţie	Cost	Nr. repetări
1	c_1	1
2	c_2	1
3	c_3	$n+1$
4	c_4	n
5	c_5	n

T(n)

Însumând timpii de execuţie ai prelucrărilor elementare se obţine: $T(n) = n(c_3 + c_4 + c_5) + c_1 + c_2 + c_3 = k_1 n + k_2$, adică timpul de execuţie depinde liniar de dimensiunea problemei. Costurile operaţiilor elementare influenţează doar constantele ce intervin în funcţia $T(n)$.

Exemplu

Inmulţire matrice ($a[1..m, 1..n]$, $b[1..n, 1..p]$)

1: **for** $i \leftarrow 1,m$ **do**

2: **for** $j \leftarrow 1,n$ **do**

3: $c[i,j] \leftarrow 0$

4: **for** $k \leftarrow 1,p$ **do**

5: $c[i,j] \leftarrow c[i,j]+a[i,k]*b[k,j]$

return $c[1..m, 1..p]$

Exemplu

Căutare ($x[1..n]$,v)
1: *găsit* $\leftarrow$ FALS
2: $i \leftarrow 1$
3: **while** (*găsit* = FALS) **and** $i \leq n$ **do**
4: **if** $v = x[i]$
5: **then** *găsit* $\leftarrow$ TRUE
6: **else** $i \leftarrow i+1$
return *găsit*

Exemplu

$m \leftarrow 1$
FOR $i \leftarrow 1, n$ DO
    $m \leftarrow 3 * m$
    FOR $j \leftarrow 1, m$ DO
        prelucrare $\Theta(1)$ {prelucrare de cost constant }

Exemplu

$h \leftarrow 1$
WHILE $h \leq n$ DO
$\|\ldots(\Theta(1))$
$\|h \leftarrow 2 * h$

Analiza timpului de execuţie

Având în vedere faptul că pentru seturi de date de intrare diferite acelaşi algoritm foloseşte timpi de execuţie diferiţi este necesar a lua în consideraţie:

- • ***timpul în cazul cel mai favorabil*** (durata minimă pentru execuţia algoritmului);
- • ***timpul mediu*** (raportul dintre suma timpilor necesari pentru toate seturile de date posibile şi numărul acestor seturi);
- • ***timpul în cazul cel mai defavorabil*** (durata maximă pentru execuţia algoritmului).

Exprimarea timpului de execuţie

Există câteva funcţii tipice de exprimare a timpului de execuţie a unui algoritm. Putem obţine, în mod obişnuit, o astfel de funcţie printr-o relaţie de recurenţă.
Cel mai adesea se va întâlni funcţia de forma

$$T(n) = cg(n) + tm$$

unde $c > 0$ este o constantă, n reprezintă dimensiunea intrării, iar *tm* este un "termen mic" care este semnificativ doar pentru valori mici ale lui n sau pentru algoritmi sofisticaţi.

Exprimarea timpului de execuţie

Funcţia $g(n)$ poate fi:

- constantă (algoritmul se execută în timp constant);
- log n (algoritmul se execută în timp log);
- n (d = 1, 2…)(algoritmul se execută în timp polinomial);

Cazuri particulare:

a) n (algoritmul se execută în timp liniar);

b) nlog n (algoritmul se execută în timp liniar log);

- $n!$ (algoritmul se execută în timp factorial);
- k^n ($k > 1$, constantă) (algoritmul se execută în timp exponenţial).

Viteza de creştere a funcţiilor

n	$\lg n$	$n\lg n$	n^2
1	0	0	1
16	4	64	256
256	8	2 048	65 536
4 096	12	49 152	16 777 216
65 536	16	1 048 565	4 294 967 296
1 048 476	20	20 969 520	1 099 301 922 576
16 775 616	24	402 614 784	281 421 292 179 456

Complexitatea asimptotică

O simplă caracterizare a eficienţei unui algoritm constă în viteza de creştere a timpului său de execuţie care oferă posibilitatea de a compara performanţele relative ale unor algoritmi alternativi.

Când datele de intrare au dimensiuni suficient de mari, astfel încât numai viteza de creştere a timpului de execuţie să fie relevantă, vom studia eficienţa ***asimptotică*** a algoritmilor.

Timpul asimptotic de execuţie a unui algoritm.

Pentru exprimarea unui astfel de timp se va folosi un limbaj riguros care cuprinde următoarele simboluri asimptotice: Θ, O , o , Ω, ω. Se vor folosi funcţii al căror domeniu de definiţie îl reprezintă mulţimea numerelor naturale N.

Fie mulţimea funcţiilor nenegative care au ca domeniul de definiţie mulţimea numerelor naturale N = {0, 1, 2, ...}. Fie funcţiile $f(n)$, $g(n)$.

Θ - notaţia

Definiţie:

$f(n) = \Theta(g(n)), (n \to \infty)$ dacă există constantele $c1 > 0, c2 > 0$ şi $n_0 \in N$, astfel încât

$$\forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

Vom spune că funcţia $g(n)$ constituie o ***margine asimptotică tare sau strânsă*** (inferioară şi superioară) pentru funcţia $f(n)$.

O - notaţia

Definiţie:

$f(n) = O(g(n)), (n \to \infty)$ dacă există o constantă $c > 0$, şi $n_0 \in N$, astfel încât

$$\forall n \geq n_0, 0 \leq f(n) \leq c\, g(n)$$

Notaţia O oferă o margine asimptotică superioară care nu este neapărat o margine asimptotică tare.

Ω - notaţia

Definiţia 1.3.3. $f(n) = \Omega(g(n))\ (n \to \infty)$ dacă $g(n) = O(f(n))\ (n \to \infty)$.

$$\Omega(g(n)) = \{ f(n) \in \Re \mid \exists c > 0, \exists n_0 \in N$$

astfel încât $f(n) \geq cg(n) \geq 0, \forall n \geq n_0 \}$