Analiza complexității

Capitolul 2

- **Timpul de execuție**
- **Cazul cel mai defavorabil şi cazul mediu**
- **Ordinul de complexitate**

În cadrul acestui capitol vom prezenta câteva detalii referitoare la complexitatea algoritmilor. Vom arăta că ordinul de complexitate al unui algoritm depinde de dimensiunea datelor de intrare, vom prezenta noțiunile de caz mediu şi caz cel mai defavorabil şi vom introduce noțiunea de ordin de complexitate.

2.1. Timpul de execuție

Este evident faptul că durata execuției unui program depinde de dimensiunea şi de particularitățile datelor de intrare. De exemplu, putem spune cu certitudine că sortarea unui şir care conține o mie de numere va dura mai mult decât sortarea unui şir care conține o sută de numere.

În general, timpul de execuție al unui algoritm creşte pe măsură ce creşte dimensiunea datelor de intrare. Din acest motiv, este natural să descriem timpul de execuție al unui program ca o funcție care depinde de această dimensiune.

Totuşi, conceptele de timp de execuție şi dimensiune a datelor de intrare sunt relativ vagi. În continuare vom defini aceste noțiuni din punctul de vedere al analizei complexității algoritmilor.

2.1.1. Datele de intrare

Definiția dimensiunii datelor de intrare depinde de problema care trebuie rezolvată. De exemplu, pentru problema sortării unui şir de N numere, dimensiunea intrării este N.

Cea mai des utilizată "măsură" a dimensiunii datelor este numărul "obiectelor" care constituie intrarea. Totuşi, nu în toate cazurile este utilizată o astfel de măsură. De

exemplu, pentru adunarea a două numere, o măsură ar putea fi numărul biților folosiți pentru a reprezenta cele două numere.

Aşadar dimensiunea datelor de intrare este o funcție care depinde de anumite variabile. În foarte multe cazuri avem o singură astfel de variabilă, dar există şi situații în care sunt necesare mai multe. De exemplu, dacă intrarea constă în descrierea unui graf neorientat, putem descrie dimensiunea datelor de intrare în funcție de două variabile: numărul nodurilor şi numărul muchiilor.

2.1.2. "Măsurarea" timpului

În principiu, timpul de execuție al unui program este durata (exprimată în secunde, milisecunde, nanosecunde etc.) necesară pentru ca programul să preia datele de intrare, să efectueze anumite operații asupra lor şi să furnizeze datele de ieşire. Evident, acest timp depinde întotdeauna de calculatorul pe care este executat programul. Nu ne putem aştepta ca un program care sortează un milion de numere să necesite acelaşi timp de execuție pe un calculator dotat cu un procesor Pentium II şi pe un calculator dotat cu un procesor Pentium 4.

Chiar şi pentru calculatoarele de ultimă generație vom avea timpi de execuție diferiți care vor depinde, în primul rând, de arhitectura şi frecvența procesorului. Considerând din nou exemplul programului care sortează un milion de numere, vom obține timpi de execuție diferiți pe calculatoare din familii diferite. Programul nu va rula în acelaşi timp pe calculatoare cu procesoare Celeron, Pentium 4, Itanium, Athlon XP, Athlon 64 sau Opteron.

Chiar dacă avem procesoare din aceeaşi familie, vom observa diferențe vizibile între un procesor cu frecvența de 2 GHz şi un procesor cu frecvența de 3 GHz.

Chiar dacă avem aceeaşi frecvență, vom observa, din nou, diferențe între un Pentium 4 şi un Athlon XP.

De fapt, chiar dacă avem acelaşi tip de procesor şi aceeaşi frecvență, pot apărea unele diferențe de timp datorate altor factori cum ar fi: sistemul de operare, cantitatea de memorie, rata de transfer a discului etc.

În concluzie, măsurarea exactă a timpului de execuție nu este un criteriu valid pentru a descrie timpul de execuție al unui algoritm.

S-ar putea crede că un algoritm care are nevoie de t secunde pentru a rula pe un calculator al cărui procesor are frecvența de 1 GHz va avea nevoie de t / 2 secunde pentru a rula pe un calculator al cărui procesor are frecvența de 2 GHz. Din nefericire nici o astfel de presupunere nu este validă, deoarece dublarea frecvenței nu înseamnă dublarea performanțelor.

O măsură mult mai utilă a timpului de execuție este dată de numărul paşilor elementari necesari execuției unui algoritm. Noțiunea de pas trebuie definită astfel încât să nu depindă de tipul calculatorului pe care este executat programul.

Cu o oarecare aproximare, putem considera că o linie descrisă în pseudocod se execută întotdeauna în acelaşi interval de timp. Evident, pentru linii diferite vom avea

intervale diferite, dar o aceeaşi linie se va executa întotdeauna în acelaşi timp. Există câteva argumente împotriva unei astfel de alegeri. De exemplu, este posibil ca anumite linii să nu se execute în timp constant. Dacă avem o linie în care se precizează că se sortează un şir, atunci timpul de execuție al acestei linii va depinde de dimensiunea şirului. Totuşi, o astfel de linie "ascunde" de fapt o mulțime de paşi care nu au mai fost prezentați pentru a simplifica descrierea. De aceea, va trebui să luăm în considerare astfel de cazuri în momentul în care analizăm timpii de execuție ai algoritmilor.

Aşadar, dacă dimensiunea intrării este n, putem exprima numărul de paşi sub forma unei funcții $T(n)$. Dacă algoritmul constă într-un număr de paşi egal cu pătratul valorii lui n vom avea $T(n) = n^2$. Dacă, din diverse motive, mai avem nevoie de încă n paşi, vom avea $T(n) = n^2 + n$.

De obicei, timpul de execuție al unui algoritm este întotdeauna acelaşi pentru date de intrare identice. Există şi situații în care intervine nedeterminismul (de exemplu dacă utilizăm numere generate aleator), caz în care timpul de execuție poate varia chiar şi pentru date de intrare identice.

2.2. Cazul cel mai defavorabil şi cazul mediu

Timpul de execuție al unui algoritm nu depinde doar de dimensiunea datelor de intrare, ci şi de configurația acestora. De exemplu, pentru a sorta un şir de numere "aproape sortat" vom avea nevoie, în principiu, de mai puțin timp decât pentru un şir "sortat în ordine inversă".

Din acest motiv au fost introduse noțiunile de caz mediu şi caz cel mai defavorabil. Timpul de execuție în cazul mediu reprezintă o medie a timpilor de execuție pentru toate configurațiile posibile ale unei intrări de dimensiune dată. Timpul de execuție în cazul cel mai defavorabil reprezintă cel mai mare dintre timpii de execuție corespunzători configurațiilor posibile ale unei intrări de dimensiune dată.

De obicei, în analiza algoritmilor se utilizează cazul cel mai defavorabil. Există trei motive pentru această alegere.

În primul rând, timpul de execuție în cel mai defavorabil caz reprezintă o limită superioară pentru timpul de execuție corespunzător oricărei configurații de dimensiune dată. Este garantat faptul, că pentru nici o configurație, nu vom avea un timp de execuție mai mare.

În al doilea rând, în cazul multor algoritmi cazul cel mai defavorabil apare relativ frecvent. De exemplu, pentru căutarea unei valori într-un vector, cazul cel mai defavorabil apare atunci când valoarea nu există, iar o astfel de situație este destul de frecventă.

În al treilea rând, în marea majoritate a cazurilor, cazul mediu este "aproape la fel de nefavorabil" ca şi cel mai nefavorabil caz. De exemplu, pentru a determina maximul unui şir, indiferent ce metodă aplicăm, va trebui întotdeauna să parcurgem toate elementele şirului.

Totuşi, în anumite situaţii, timpul de execuţie pentru cazul mediu se poate dovedi util. Cea mai mare dificultate întâmpinată în momentul în care se încearcă determinarea acestui timp este dată de faptul că, în marea majoritate a cazurilor, nu vom putea determina timpii de execuţie pentru toate cazurile pentru ca apoi să calculăm o medie. De aceea, va trebui să estimăm o astfel de medie fără a fi siguri că obţinem media exactă. În principiu, în acest scop putem considera că orice configuraţie a datelor de intrare are aceleaşi şanse să apară ca şi oricare alta. Din nefericire, în practică, această presupunere se dovedeşte deseori falsă.

2.3. Ordinul de complexitate

Funcţiile obţinute pentru timpii de execuţie pot fi uneori foarte complicate şi, din acest motiv, devin inutile. Tocmai de aceea a fost introdus ordinul de complexitate al unui algoritm. Acesta reprezintă o caracterizare a funcţiei respective, o descriere a ordinului de mărime.

Ordinul de complexitate va indica doar ordinul de mărime al funcţiei, eliminându-se toţi termenii care nu sunt importanţi.

De exemplu, dacă avem $T(n) = 2 \cdot n^2 + 5 \cdot n + 3$, ordinul de complexitate va fi $O(n^2)$. Din acest exemplu "deducem", în primul rând, că ordinul de complexitate se notează prin O şi se obţine prin eliminarea tuturor coeficienţilor şi a termenilor "neimportanţi".

Noţiunea de termen "neimportant" este foarte vagă, motiv pentru care vom arăta modul în care putem decide care termen este mai important.

În primul rând, putem compara doi termeni doar dacă în cadrul lor apare aceeaşi variabilă. Termenul mai "important" este numit *termen dominant*. În tabelul următor vom prezenta modul în care se stabileşte termenul dominant (coeficienţii au fost eliminaţi).

Termen dominant	Termen "dominat"	Condiţii
$n!$	a^n	întotdeauna
a^n	b^n	$a > b > 1$ sau $0 < b < a < 1$
a^n	n^b	întotdeauna
n^a	n^b	$a > b > 0$ sau $b < a < 0$
n^a	n	$a > 1$
n	$\log_a n$	$a > 1$
$\log_a n$	1	$a > 1$

Ar putea părea că ar trebui să avem şi o linie care să indice că $\log_a n$ "domină" pe $\log_b n$ dacă $a > b$. Oarecum surprinzător, nu este cazul. Câteva calcule matematice simple ne arată că toţi termenii logaritmici sunt la fel de importanţi. Avem:

$$\log_a x = \frac{\log_b x}{\log_b a} = \frac{1}{\log_b a} \cdot \log_b x = c \cdot \log_b x .$$

Am notat prin c raportul $1 / \log_b a$, pentru a sugera faptul că este o constantă. Datorită faptului că valorile constante (coeficienții) nu sunt luați în considerare, o funcție logaritmică nu poate domina o altă funcție logaritmică. Din acest motiv, pentru descrierea ordinului de complexitate nici nu mai este prezentă baza logaritmului, notația fiind $O(\log n)$.

În foarte multe situații putem estima mult mai uşor timpii de execuție (şi ordinele de complexitate) pentru anumite secțiuni ale algoritmilor. În această situație, este mult mai uşor să determinăm ordinul de complexitate al întregului algoritm, deoarece se reduc semnificativ calculele necesare.

În continuare vom presupune că funcția $f(n)$ domină funcția $g(n)$. În această situație vom avea întotdeauna:

- $O(f(n)) + O(g(n)) = O(f(n))$;
- $O(f(n)) \cdot O(g(n)) = O(f(g(n)))$;
- $\min(O(f(n)), O(g(n))) = O(g(n))$;
- $\max(O(f(n)), O(g(n))) = O(f(n))$.

Pe baza acestor relații ne va fi mult mai uşor să stabilim ordinul de complexitate al unui algoritm.

De exemplu, presupunem că am reuşit să stabilim că ordinul de complexitate al operației de citire a datelor de intrare este $O(n)$, ordinul de complexitate al operației de prelucrare al acestora este $O(n^2)$ şi ordinul de complexitate al operației de scriere a datelor de ieşire este $O(n)$. În acest caz, ordinul de complexitate al algoritmului va fi $O(n) + O(n^2) + O(n) = O(n^2)$.