Analiza eficienței unui algoritm

Sumar

Competenţe generale

- *aplicarea algoritmilor fundamentali în prelucrarea datelor*
- *elaborarea algoritmilor de rezolvare a problemelor*

Competenţe specifice

- *alegerea unui algoritm eficient de rezolvare a unei probleme*
- *analiza comparativă a eficienţei diferitelor metode de rezolvare a aceleiaşi probleme şi alegerea unui algoritm eficient de rezolvare a unei probleme*

Analiza unui algoritm:

- identificarea resurselor necesare executării unui algoritm:
    - timpul de executare
    - memoria utilizată

Complexitatea algoritmilor:

- complexitate temporală:
  - numărul de operații elementare executate de algoritm

- complexitate spațială:
  - memoria necesară variabilelor utilizate de algoritm

Optimizarea algoritmilor:

optimizare = proces prin care se produce cod care este cât mai eficient posibil

optimizare timp = accentul se pune pe viteză în detrimentul memorie

optimizare memorie = accentul se pune pe spațiu în detrimentul vitezei

Analiza complexității unui algoritm:

I. **Analiza complexității unui algoritm din punct de vederea al timpului de executare**

II. **Analiza complexității unui algoritm din punct de vederea al memoriei utilizate**

I. Analiza complexității unui algoritm din punct de vederea al timpului de executare

Eficiența unui algoritm:
- evaluarea timpului necesar pentru executarea algoritmului

Timpul de executare al algoritmului:
- reprezintă numărul de operații elementare pe care algoritmul le execută

Se poate aprecia timpul de execuție al unui algoritm și nu al unui program, deoarece programul depinde de mai mulți factori:

- dimensiunea și natura datelor de intrare;
- caracteristicile sistemului de calcul;
- eficiența codului produs de compilator.

În general, pentru analiza complexității unui algoritm, din punct de vederea al timpului de executare, nu se calculează numărul tuturor operațiilor pe care algoritmul le efectuează, ci se calculează numărul operațiilor importante, numite *operații de bază*, de care depinde timpul de rulare al algoritmului.

Operația de bază este o operație elementară sau o succesiune de operații elementare.

Operații de bază:

- atribuire
- operații aritmetice
- operații logice
- comparații

Există 3 cazuri în procesul de analiză:

1. Cazul cel mai bun – cazul favorabil
   - se efectuează cele mai puține operații
   - timpul minim
   - notația Ω
2. Cazul cel mai rău – cazul defavorabil
   - se efectuează cele mai multe operații
   - timpul maxim
   - notația O
3. Cazul mediu
   - cel mai greu de determinat
   - media aritmetică a timpilor de execuție corespunzătoare tuturor cazurilor posibile
   - notația Θ

Concluzie: în analiza complexității se determină timpul maxim de execuție al algoritmului.

Notații:

- notația pentru reprezentarea complexității algoritmilor este **O** și reprezintă ordinul de complexitate al algoritmului

- notația **O** este pentru a mărginii cazul cel mai defavorabil

- timpul estimat de calcul se trece sub formă aproximativă:
  O(număr estimat de execuții ale operației de bază)

Exemplu: pentru un algoritm care execută aproximativ **n** operații de bază, complexitatea sa se notează **O(n)**.

Clase de complexitate:

Ordin de complexitate	Tip complexitate	Exemplu
O(1)	constantă	testare număr par
O(log n)	logaritmică	căutare binară
O(n)	liniară	minimul dintr-un șir
O(n·log n)	liniar logaritmică	sortare prin interclasare – Merge Sort
$O(n^2)$	pătratică	sortare prin metoda bulelor
$O(n^3)$	cubică	înmulțirea a două matrici
$O(n^k)$	polinomială	determinarea valorii unui polinom
$O(2^n)$	exponențială	turnurile din Hanoi
O(n!)	factorială	generarea permutărilor unei mulțimi

Etapele analizei eficienței de timp:

Pasul 1: se identifică dimensiunea problemei

Pasul 2: se identifică operația de bază/dominantă

Pasul 3: se determină numărul de execuții ale operației de bază

Pasul 4: dacă numărul de execuții ale operației de bază depinde de proprietățile datelor de intrare atunci se analizează:
- cazul cel mai favorabil
- cazul cel mai defavorabil
- cazul mediu

Pasul 5: se stabilește ordinul/clasa de complexitate

Exemplu:

Suma primelor **`n`** numere naturale nenule.

Algoritm constant	Algoritm liniar	Algoritm pătratic
O(1)	O(n)	$O(n^2)$

`s=n*(n+1)/2;`	`s=0;`
	`for(i=1;i<=n;i++)`
	`  s=s+i;`
		`s=0;`
		`for(i=1;i<=n;i++)`
		`  for(j=1;j<=i;j++)`
		`  s=s+1;`

`n` – mărimea problemei/intrării

Analiza complexității algoritmilor care determină suma:

	Algoritm constant	Algoritm liniar	Algoritm pătratic
Ordin de complexitate	**O(1)**	**O(n)**	**O(n^2)**

Algoritmul de rezolvare	`s=n*(n+1)/2;`	`s=0;`
		`for(i=1;i<=n;i++)`
		`  s=s+i;`
			`s=0;`
			`for(i=1;i<=n;i++)`
			`  for(j=1;j<=i;j++)`
			`  s=s+1;`

Operația de bază	`s=n*(n+1)/2;`	`s=s+i;`	`s=s+1;`
Atribuiri	1	2n+2	n^2+2
Comparații	0	n+1	(n^2+3n+2)/2
Adunări/scăderi	1	n	n(n-1)/2
Înmulțiri/împărțiri	2	0	0
Total operații	4	4n+3	2n^2+n+3

II. Analiza complexității unui algoritm din punct de vederea al memoriei utilizate

Complexitate spațială:

Estimează cantitatea de memorie necesară unui algoritm pentru a memora:

- datele de intrare;
- datele de ieșire;
- datele de manevră.

Spațiul de stocare:

Spațiul unui algoritm este alcătuit din:

1. spațiul pe disc
   - dependent de hard și de limbaj

2. cantitatea de memorie alocată în timpul procesării
   - memoria sau depozitul nevolatil utilizat de variabile

Notații: se folosește notația de la complexitatea temporală.

Complexitate de memorie:

- relația dintre necesarul de memorie exprimat în numărul de locații elementare și dimensiunea problemei

Exemplu:

1. variabilă simplă
   - complexitate de memorie **O(1)**

2. vector cu **n** elemente
   - complexitate de memorie **O(n)**

3. matrice cu **2** linii și **n** coloane
   - complexitate de memorie **O(n)**

4. matrice cu **n** linii și **n** coloane
   - complexitate de memorie **$O(n^2)$**

Evaluarea necesarului de memorie:

- dimensiunea problemei = volumul de memorie necesar problemei pentru a stoca toate datele de intrare ale problemei

- dimensiunea problemei se exprimă în funcție de:
  1. *numărul de componente* ale datelor de intrare (valori întregi, reale, caracter, etc.)
  2. *numărul de biți* necesar stocării datelor de intrare

Evaluarea necesarului de memorie:

- se calculează numărul de variabile nestructurate (întregi, reale, booleene, caracter)
- se determină numărul de octeți necesari reprezentării fiecărei componente/variabile

Necesar memorie:

`număr octeți * număr componente`

Exemple:

1. minimul din tablou
   - dimensiunea problemei: **`n`**

2. calculul valorii unui polinom de ordin **`n`**
   - dimensiunea problemei: **`n`**

3. suma a două matrici cu **`m`** linii și **`n`** coloane
   - dimensiunea problemei: **`mxn`**

Concluzii:

- nu există formulă generală pentru analiza eficienței unui algoritm din punct de vedere al timpului de execuție
- în general, nu este posibil să obținem simultan un timp de execuție mai scurt, precum și un necesar de memorie mai mic
- procesele tehnologice din ultima vreme impun drept criteriu primordial criteriul timp

Fişe de lucru

- Analiza eficienței unui algoritm

7. Bibliografie & webografie

1. Miloşescu M., *Informatică. Manual pentru clasa a XI*, Editura Didactică şi Pedagogică, Bucureşti, 2006
2. https://ro.wikipedia.org/wiki/Teoria_complexit%C4%83%C8%9Bii
3. https://ro.wikipedia.org/wiki/Complexitate_%C3%AEn_timp