Proiectarea algoritmilor

Lucrare de laborator nr. 2

Complexitatea algoritmilor

Cuprins

1 Evaluarea algorimilor

1.1 Consideraţii generale

Evaluarea algorimilor din punctul de vedere al performanţelor obţinute de aceştia în rezolvarea problemelor este o etapă esenţială în procesul de decizie a utilizării acestora în aplicaţii.

La evaluarea (estimarea) algoritmilor se pune în evidenţă consumul celor două resurse fundamentale: timpul de execuţie şi spaţiul de memorare a datelor.

În funcţie de priorităţile alese, se aleg limite pentru resursele timp şi spaţiu. Algoritmul este considerat eligibil dacă consumul celor două resurse se încadrează în limitele stabilite.

1.2 Timp şi spatiu

Fie P o problemă şi A un algoritm pentru P. Fie $c_0 \vdash_A c_1 \cdots \vdash_A c_n$ un calcul finit al algoritmului A. Notăm cu $t_A(c_i)$ timpul necesar obţinerii configuraţiei c_i din $c_{i-1},\ 1 \leq i \leq n$, şi cu $s_A(c_i)$ spaţiul de memorie ocupat în configuraţia $c_i,\ 0 \leq i \leq n$. Fie A un algoritm pentru problema P, $p \in P$ o instanţă a problemei P şi $c_0 \vdash c_1 \vdash \cdots \vdash c_n$ calculul lui A corespunzător instanţei p.

Timpul necesar algoritmului A pentru rezolvarea instanţei p este:

$$T_A(p) = \sum_{i=1}^{n} t_A(c_i)$$

Spaţiul (de memorie) necesar algoritmului A pentru rezolvarea instanţei p este:

$$S_A(p) = \max_{0 \leq i \leq n} s_A(c_i)$$

1.3 Mărimea unei instanţe

Asociem unei instanţe $p \in P$ o mărime $g(p)$, care este un număr natural, pe care o numim *mărimea* instanţei p. De exemplu, $g(p)$ poate fi suma lungimilor reprezentărilor corespunzând datelor din instanţa p.

Dacă reprezentările datelor din p au aceeaşi lungime, atunci se poate considera $g(p)$ egală cu numărul datelor. Dacă p constă dintr-un tablou atunci se poate lua $g(p)$ ca fiind numărul de elemente ale tabloului. Dacă p constă dintr-un polinom se poate considera $g(p)$ ca fiind gradul polinomului (= numărul coeficienţilor minus 1). Dacă p este un graf se poate lua $g(p)$ ca fiind numărul de vârfuri, numărul de muchii sau numărul de vârfuri + numărul de muchii etc.

2 Cazul favorabil şi nefavorabil

2.1 Consideraţii generale

Fie A un algoritm pentru problema P. Spunem că A rezolvă P în *timpul* $T_A^{fav}(n)$ dacă:

$$T_A^{fav}(n) = \inf\{T_A(p) \mid p \in P, g(p) = n\}$$

Spunem că A rezolvă P în *timpul* $T_A(n)$ dacă:

$$T_A(n) = \sup\{T_A(p) \mid p \in P, g(p) = n\}$$

Spunem că A rezolvă P în spaţiul $S_A^{fav}(n)$ dacă:

$$S_A^{fav}(n) = \inf\{S_A(p) \mid p \in P, g(p) = n\}$$

Spunem că A rezolvă P în spaţiul $S_A(n)$ dacă:

$$S_A(n) = \sup\{S_A(p) \mid p \in P, g(p) = n\}$$

Funcţia T_A^{fav} (S_A^{fav}) se numeşte *timpul de execuţie al algoritmului (spaţiul utilizat de algoritmul) A pentru cazul cel mai favorabil*
Funcţia T_A (S_A) se numeşte *timpul de execuţie al algoritmului (spaţiu utilizat de algoritmul) A pentru cazul cel mai nefavorabil*.

2.2 Exemplu de analiză - Problema căutării unui element într-o secvenţă de numere întregi

Intrare: $n, (a_0, \ldots, a_{n-1}), z$ numere întregi.

Ieşire: $poz = \begin{cases} \min\{i \mid a_i = z\} & \text{dacă } \{i \mid a_i = z\} \neq \emptyset, \\ -1 & \text{altfel.} \end{cases}$

Presupunem că secvenţa $(a_0, \ldots, a_{n-1})$ este memorată în tabloul $(\texttt{a[i]} \mid 0 \leq i \leq n-1)$.

```
/* algoritmul A1 */
i ← 0
while (a[i] ≠ z) and (i < n-1) do
   i ← i+1
if (a[i] = z)
   then poz ← i
   else poz ← -1
```

Considerăm ca dimensiune a problemei numărul n al elementelor din secvenţa în care se caută. Deoarece suntem în cazul când toate datele sunt memorate pe câte un cuvânt de memorie, vom presupune că toate operaţiile necesită o unitate de timp. Cazul cel mai favorabil este obţinut când $a_0 = z$ şi se efectuează trei comparaţii şi două atribuiri. Rezultă $T_{A_1}^{fav}(n) = 3+2 = 5$. Cazul cel mai nefavorabil se obţine când $z \notin \{a_0, \ldots, a_{n-1}\}$ sau $z = a[n-1]$, în acest caz fiind executate $2n+1$ comparaţii şi $1+(n-1)+1 = n+1$ atribuiri. Rezultă $T_{A_1}(n) = 3n+2$. Pentru simplitatea prezentării, nu au mai fost luate în considerare operaţiile `and` şi operaţiile de adunare şi scădere. Spaţiul utilizat de algoritm, pentru ambele cazuri, este $n+7$ (tabloul `a`, constantele `0`, `1` şi `-1`, variabilele `i`, `poz`, `n` şi `z`).

3 Cazul mediu

3.1 Consideraţii generale

Timpii de execuţie pentru cazul cel mai favorabil nu oferă informaţii relevante despre eficienţa algoritmului. Mult mai semnificative sunt informaţiile oferite de timpii de execuţie în cazul cel mai nefavorabil: în toate celelalte cazuri algoritmul va avea performanţe mai bune sau cel puţin la fel de bune. Pentru evaluarea timpului de execuţie nu este necesar întotdeauna să numărăm toate operaţiile. În exemplul anterior, observăm că operaţiile de atribuire (fără cea iniţială) sunt precedate de comparaţii. Putem număra numai comparaţiile, pentru că numărul acestora determină numărul atribuirilor. Putem să mergem chiar mai departe şi să numărăm numai comparaţiile între z şi componentele tabloului. Uneori, numărul instanţelor p cu $g(p) = n$ pentru care $T_A(p) = T_A(n)$ sau $T_A(p)$ are o valoare foarte apropiată de $T_A(n)$ este foarte mic. Pentru aceste cazuri, este preferabil să calculăm comportarea în medie a algoritmului.

Pentru a putea calcula comportarea în medie este necesar să privim mărimea $T_A(p)$ ca fiind o variabilă aleatoare (o experienţă = execuţia algoritmului pentru o instanţă p, valoarea experienţei = durata execuţiei algoritmului pentru instanţa p) şi să precizăm legea de repartiţie a acestei variabile aleatoare.

Comportarea în medie se calculează ca fiind media acestei variabile aleatoare (considerăm numai cazul timpului de execuţie):

$$T_A^{med}(n) = M(\{T_A(p) \mid p \in P \wedge g(p) = n\})$$

Dacă mulţimea valorilor variabilei aleatoare $T_A(p) = \{x_1, \dots\}$ este finită sau numărabilă ($T_A(p) = \{x_1, \dots, x_i \dots\}$ şi probabilitatea ca $T_A(p) = x_i$ este p_i, atunci media variabilei aleatoare T_A (timpul mediu de execuţie) este:

$$T_A^{med}(n) = \sum_i x_i \cdot p_i$$

3.2 Exemplu de evaluare pentru cazul mediu

Considerăm problema căutării unui element într-o secvenţă de numere întregi, definită anterior. Mulţimea valorilor variabilei aleatoare $T_{A_1}(p)$ este $\{3i+2 \mid 1 \le i \le n\}$. Facem următoarele presupuneri: probabibilitatea ca $z \in \{a_0, \dots, a_{n-1}\}$ este q şi probabilitatea ca z să apară prima dată pe poziţia $i-1$ este $\frac{q}{n}$ (indicii i candidează cu aceeaşi probabilitate pentru prima apariţie a lui z). Rezultă că probabilitatea ca $z \notin \{a_0, \dots, a_{n-1}\}$ este $1-q$. Probabilitatea ca $T_{A_1}(p) = 3i+2$ ($poz = i-1$) este $\frac{q}{n}$, pentru $1 \le i < n$, iar probabilitatea ca $T_{A_1}(p) = 3n+2$ este $p_n = \frac{q}{n} + (1-q)$ (probabilitatea ca $poz = n-1$ sau ca $z \notin \{a_0, \dots, a_{n-1}\}$).

Timpul mediu de execuţie este:

$$\begin{aligned}
T_{A_1}^{med}(n) &= \sum_{i=1}^{n} p_i x_i = \sum_{i=1}^{n-1} \frac{q}{n} \cdot (3i+2) + (\frac{q}{n} + (1-q)) \cdot (3n+2) \\
&= \frac{3q}{n} \cdot \sum_{i=1}^{n} i + \frac{q}{n} \sum_{i=1}^{n} 2 + (1-q) \cdot (3n+2) \\
&= \frac{3q}{n} \cdot \frac{n(n+1)}{2} + 2q + (1-q) \cdot (3n+2) \\
&= \frac{3q \cdot (n+1)}{2} + 2q + (1-q) \cdot (3n+2) \\
&= 3n - \frac{3nq}{2} + \frac{3q}{2} + 2
\end{aligned}$$

Pentru $q = 1$ (z apare totdeauna în secvenţă) avem $T_{A_1}^{med}(n) = \frac{3n}{2} + \frac{7}{2}$ şi pentru $q = \frac{1}{2}$ avem $T_{A_1}^{med}(n) = \frac{9n}{4} + \frac{11}{4}$.

4 Calcul asimptotic

4.1 Consideraţii generale

În practică, atât $T_A(n)$, cât şi $T_A^{med}(n)$ sunt dificil de evaluat. Din acest motiv se caută, de multe ori, margini superioare şi inferioare pentru aceste mărimi. Următoarele clase de funcţii sunt utilizate cu succes în stabilirea acestor margini:

$$\begin{aligned}
O(f(n)) &= \{g(n) \mid (\exists c > 0, n_0 \ge 0)(\forall n \ge n_0)|g(n)| \le c \cdot |f(n)|\} \\
\Omega(f(n)) &= \{g(n) \mid (\exists c > 0, n_0 \ge 0)(\forall n \ge n_0)|g(n)| \ge c \cdot |f(n)|\} \\
\Theta(f(n)) &= \{g(n) \mid (\exists c_1, c_2 > 0, n_0 \ge 0)(\forall n \ge n_0) c_1 \cdot |f(n)| \le |g(n)| \le c_2 \cdot |f(n)|\}
\end{aligned}$$

Cu notaţiile O, Ω şi Θ se pot forma expresii şi ecuaţii. Considerăm numai cazul O, celelalte tratându-se similar. Expresiile construite cu O pot fi de forma:

$$O(f_1(n)) \text{ op } O(f_2(n))$$

unde „op”poate fi $+, -, *$ etc. şi notează mulţimile: $\{g(n) \mid (\exists g_1(n), g_2(n), c > 0, n_0 \ge 0)$

$$[(\forall n) g(n) = g_1(n) \text{ op } g_2(n)] \wedge [(\forall n \ge n_0) g_1(n) \le c f_1(n) \wedge g_2(n) \le c f_2(n)]\}$$

De exemplu:

$$O(n) + O(n^2) = \{g(n) = g_1(n) + g_2(n) \mid (\forall n \ge n_0) g_1(n) \le cn \wedge g_2(n) \le cn^2\}$$

Utilizând regulile de asociere şi prioritate, se obţin expresii de orice lungime:

$$O(f_1(n)) \text{ op}_1 O(f_2(n)) \text{ op}_2 \cdots$$

Orice funcţie $f(n)$ poate fi gândită ca o notaţie pentru mulţimea cu un singur element $f(n)$ şi deci putem alcătui expresii de forma:

$$f_1(n) + O(f_2(n))$$

ca desemnând mulţimea:

$$\{f_1(n) + g(n) \mid g(n) \in O(f_2(n))\} =$$

$$\{f_1(n) + g(n) \mid (\exists c > 0, n_0 > 1)(\forall n \geq n_0) g(n) \leq c \cdot f_2(n)\}$$

Peste expresii considerăm formule de forma:

$$expr1 = expr2$$

cu semnificaţia că mulţimea desemnată de $expr1$ este inclusă în mulţimea desemnată de $expr2$.

4.2 Exemplu de formule

$$n \log n + O(n^2) = O(n^2)$$

Justificare:
$(\exists c_1 > 0, n_1 > 1)(\forall n \geq n_1) n \log n \leq c_1 n^2$, $g_1(n) \in O(n^2)$ implică
$(\exists c_2 > 0, n_2 > 1)(\forall n \geq n_2) g_1(n) \leq c_2 n^2$ c si de aici
$(\forall n \geq n_0) g(n) = n \log n + g_1(n) \leq n \log n + c_2 n^2 \leq (c_1 + c_2) n^2$, unde
$n_0 = \max\{n_1, n_2\}$.

De remarcat nesimetria ecuaţiilor: părţile stânga şi cea din dreapta joacă roluri distincte. Ca un caz particular, notaţia $g(n) = O(f(n))$ semnifică, de fapt, $g(n) \in O(f(n))$.

4.3 Calculul timpului asimptotic de execuţie pentru cazul cel mai nefavorabil

Un algoritm poate avea o descriere complexă şi deci evaluarea sa poate pune unele probleme. Deoarece orice algoritm este descris de un program, în continuare considerăm A o secvenţă de program. Regulile prin care se calculează timpul de execuţie sunt date în funcţie de structura lui A:

A este o instrucţiune de atribuire. Timpul de execuţie a lui A este egal cu timpul evaluării expresiei din partea dreaptă. A este forma A_1 A_2 Timpul de execuţie al lui A este egal cu suma timpilor de execuţie ai algoritmilor A_1 şi A_2. A este de forma `if` e `then` A_1 `else` A_2. Timpul de execuţie al lui A este egal cu maximul dintre timpii de execuţie ai algoritmilor A_1 şi A_2 la care se adună timpul necesar evaluării expresiei e. A este de forma `while` e `do` A_1. Se determină cazul în care se execută numărul maxim de iteraţii ale buclei `while` şi se face suma timpilor calculaţi pentru fiecare iteraţie. Dacă nu este posibilă determinarea timpilor pentru fiecare iteraţie, atunci timpul de execuţie al lui A este egal cu produsul dintre timpul maxim de execuţie al algoritmului A_1 şi numărul maxim de execuţii ale buclei A_1.

4.4 Timp polinomial

Teorema 4.1 *Dacă g este o funcţie polinomială de grad k, atunci* $g = O(n^k)$.

Demonstraţie:
Presupunem $g(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \cdots + a_1 \cdot n + a_0$.
Efectuând majorări în membrul drept, obţinem: $g(n) \leq |a_k| \cdot n^k + |a_{k-1}| \cdot n^{k-1} + \cdots + |a_1| \cdot n + |a_0| < n^k \cdot \underbrace{(|a_k| + |a_{k-1}| + |a_0|)}_{c} < n^k \cdot c$ pentru $\forall n > 1 \Rightarrow g(n) < c \cdot n^k$, cu $n_0 = 1$.
Deci $g = O(n^k)$.

5 Clasificarea algoritmilor

Următoarele incluziuni sunt valabile în cazul notaţiei O:

$$O(1) \subset O(\log n) \subset O(\log^k n) \subset O(n) \subset O(n^2) \subset \cdots \subset O(n^{k+1}) \subset O(2^n)$$

Pentru clasificarea algoritmilor cea mai utilizată totaţie este O. Cele mai cunoscute clase sunt:

$\{A \mid T_A(n) = O(1)\}$ = clasa algoritmilor constanţi;
$\{A \mid T_A(n) = O(\log n)\}$ = clasa algoritmilor logaritmici;
$\{A \mid T_A(n) = O(log^k n)\}$ = clasa algoritmilor polilogaritmici;
$\{A \mid T_A(n) = O(n)\}$ = clasa algoritmilor liniari;
$\{A \mid T_A(n) = O(n^2)\}$ = clasa algoritmilor pătratici;
$\{A \mid T_A(n) = O(n^k)\}$ = clasa algoritmilor polinomiali;
$\{A \mid T_A(n) = O(2^n)\}$ = clasa algoritmilor exponenţiali.

Cu notaţiile de mai sus, doi algoritmi, care rezolvă aceeaşi problemă, pot fi comparaţi numai dacă au timpii de execuţie în clase de funcţii (corespunzătoare notaţiilor O, Ω şi Θ) diferite. De exemplu, un algoritm A cu $T_A(n) = O(n)$ este mai eficient decât un algoritm A' cu $T_{A'}(n) = O(n^2)$. Dacă cei doi algoritmi au timpii de execuţie în aceeaşi clasă, atunci compararea lor devine mai dificilă pentru că trebuie determinate şi constantele cu care se înmulţesc reprezentanţii clasei.

6 Sarcini de lucru şi barem de notare

Sarcini de lucru:

Pentru crearea unei liste circulare simplu înlănţuite studentul Ionescu reţine pointerul către ultimul element al listei, inserând elementul nou introdus după ultimul, actualizând apoi pointerul către ultimul element. Student Popescu reţine pointerul către primul element, pentru inserţia unui element în listă parcurgând lista până la găsirea ultimului, realizând inserţia după acesta.

a) Pornind de la lista vidă, inserând 10000 de elemente şi considerând ca pas elementar comparaţia a 2 pointeri, câte operaţii efectuează în plus studentul Popescu faţă de studentul Ionescu?

b) Ajutaţi-l pe studentul Popescu scriind pentru poblema dată o funcţie C/C++ de complexitate $O(1)$, care primeşte ca parametru pointerul către primul element al listei circulare simplu înlănţuite şi valoarea de inserat la sfârştul listei circulare!

Observaţii:

- Se vor număra operaţiile elementare și se va măsura timpul experimentului (durata de execuţie a programului fără a considera operațiile de citire a datelor de intrare).
- În cazul în care optați pentru limbajul C++, puteți găsi informații despre afișarea valorilor reale cu o anumită precizie urmărind link-ul următor:
  http://www.cplusplus.com/reference/iostream/ios_base/precision/

Barem de notare:

1. Problema a): 4p
2. Problema b): 5p
3. Baza: 1p

7 Determinarea timpului de rulare

Pentru determinarea duratei necesare efectuării experimentului se poate adapta codul de mai jos.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main( void )
{
      clock_t start, finish;
      long loop;
      double result,r2, elapsed_time;

      printf( "Inmultirea a doua numere reale de un miliard de ori...\n" );
      result = 3.1415926535897;

      start = clock(); //inceputul experimentului
      for( loop = 0; loop < 1000000000; loop++ ) {
            r2=result*16.0;
      }
      finish = clock(); //sfarsitul operatiilor cronometrate

      elapsed_time = (double)(finish - start) / CLOCKS_PER_SEC;
      printf("\nProgramul necesita %6.2f secunde.\n",elapsed_time );

      return 0;
}
```

Bibliografie

[1] Lucanu, D. şi Craus, M., *Proiectarea algoritmilor*, Editura Polirom, 2008.