Proiectarea algoritmilor

Complexitatea algoritmilor

Lucrare de laborator nr. 2

Cuprins

Evaluarea algorimilor

- Evaluarea algorimilor din punctul de vedere al performanțelor obținute de aceștia în rezolvarea problemelor este o etapă esențială în procesul de decizie a utilizării acestora în aplicații.
- La evaluarea (estimarea) algoritmilor se pune în evidență consumul celor două resurse fundamentale: timpul de execuție și spațiul de memorare a datelor.
- În funcție de prioritățile alese, se aleg limite pentru resursele timp și spațiu.
- Algoritmul este considerat eligibil dacă consumul celor două resurse se încadrează în limitele stabilite.

Timp și spatiu

- Fie P o problemă și 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 obținerii configurației c_i din c_{i-1}, $1 \leq i \leq n$, și cu $s_A(c_i)$ spațiul de memorie ocupat în configurația c_i, $0 \leq i \leq n$.
- Fie A un algoritm pentru problema P, $p \in P$ o instanță a problemei P și $c_0 \vdash c_1 \vdash \cdots \vdash c_n$ calculul lui A corespunzător instanței p.
  - *Timpul* necesar algoritmului A pentru rezolvarea instanței p este:

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

  - *Spațiul* (de memorie) necesar algoritmului A pentru rezolvarea instanței p este:

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

Mărimea unei instanțe

- Asociem unei instanțe $p \in P$ o mărime $g(p)$, care este un număr natural, pe care o numim *mărimea* instanței p.
  - De exemplu, $g(p)$ poate fi suma lungimilor reprezentărilor corespunzând datelor din instanța p.
- Dacă reprezentările datelor din p au aceeași 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 coeficienților 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.

Cazul favorabil și nefavorabil

- 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 spațiul $S_A^{fav}(n)$ dacă:
  $$S_A^{fav}(n) = \inf\{ S_A(p) \mid p \in P, g(p) = n\}$$
  - Spunem că A rezolvă P în spațiul $S_A(n)$ dacă:
  $$S_A(n) = \sup\{ S_A(p) \mid p \in P, g(p) = n\}$$
- Funcția T_A^{fav} (S_A^{fav}) se numește *timpul de execuție al algoritmului (spațiul utilizat de algoritmul) A pentru cazul cel mai favorabil*
- Funcția T_A (S_A) se numește *timpul de execuție al algoritmului (spațiu utilizat de algoritmul) A pentru cazul cel mai nefavorabil*.

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

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

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

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

```
/* 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
```

Problema căutării unui element într-o secvență de numere întregi - continuare

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

```
/* 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
```

Cazul mediu

- Timpii de execuție pentru cazul cel mai favorabil nu oferă informații relevante despre eficiența algoritmului.
- Mult mai semnificative sunt informațiile oferite de timpii de execuție în cazul cel mai nefavorabil: în toate celelalte cazuri algoritmul va avea performanțe mai bune sau cel puțin la fel de bune.
- Pentru evaluarea timpului de execuție nu este necesar întotdeauna să numărăm toate operațiile.
- În exemplul anterior, observăm că operațiile de atribuire (fără cea inițială) sunt precedate de comparații.
- Putem număra numai comparațiile, pentru că numărul acestora determină numărul atribuirilor.
- Putem să mergem chiar mai departe și să numărăm numai comparațiile între z și componentele tabloului.
- Uneori, numărul instanțelor 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.

Cazul mediu - continuare

- Pentru a putea calcula comportarea în medie este necesar să privim mărimea $T_A(p)$ ca fiind o variabilă aleatoare (o experiență = execuția algoritmului pentru o instanță p, valoarea experienței = durata execuției algoritmului pentru instanța p) și să precizăm legea de repartiție a acestei variabile aleatoare.
- *Comportarea în medie* se calculează ca fiind media acestei variabile aleatoare (considerăm numai cazul timpului de execuție):

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

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

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

Exemplu de calcul al timpului pentru cazul mediu

- Considerăm problema căutării unui element într-o secvență de numere întregi, definită anterior.
- Mulțimea valorilor variabilei aleatoare $T_{A_1}(p)$ este $\{3i+2 \mid 1 \leq i \leq n\}$.
- Legea de repartiție:
  - Facem următoarele presupuneri: probabibilitatea ca $z \in \{a_0, \ldots, a_{n-1}\}$ este q și probabilitatea ca z să apară prima dată pe poziția $i-1$ este $\frac{q}{n}$ (indicii i candidează cu aceeași probabilitate pentru prima apariție a lui z). Rezultă că probabilitatea ca $z \notin \{a_0, \ldots, a_{n-1}\}$ este $1-q$.
  - Probabilitatea ca $T_{A_1}(p) = 3i+2$ ($poz = i-1$) este $\frac{q}{n}$, pentru $1 \leq i < n$, iar probabilitatea ca $T_{A_1}(p) = 3n+2$ este $p_n = \frac{q}{n} + (1-q)$ (probabilitatea ca $poz = n-1$ sau ca $z \notin \{a_0, \ldots, a_{n-1}\}$).

Exemplu de calcul al timpului pentru cazul mediu - continuare

- Timpul mediu de execuție 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 secvență) avem $T_{A_1}^{med}(n) = \frac{3n}{2} + \frac{7}{2}$ și pentru $q = \frac{1}{2}$ avem $T_{A_1}^{med}(n) = \frac{9n}{4} + \frac{11}{4}$.

Calcul asimptotic

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

$$O(f(n)) = \{g(n) \mid (\exists c > 0, n_0 \geq 0)(\forall n \geq n_0)|g(n)| \leq c \cdot |f(n)|\}$$
$$\Omega(f(n)) = \{g(n) \mid (\exists c > 0, n_0 \geq 0)(\forall n \geq n_0)|g(n)| \geq c \cdot |f(n)|\}$$
$$\Theta(f(n)) = \{g(n) \mid (\exists c_1, c_2 > 0, n_0 \geq 0)(\forall n \geq n_0)c_1 \cdot |f(n)| \leq |g(n)| \leq c_2 \cdot |f(n)|\}$$

- Cu notațiile O, Ω și Θ se pot forma expresii și ecuații. 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. și notează mulțimile:
$\{g(n) \mid (\exists g_1(n), g_2(n), c > 0, n_0 \geq 0)$

$$[(\forall n)g(n) = g_1(n) \text{ op } g_2(n)] \wedge [(\forall n \geq n_0)g_1(n) \leq cf_1(n) \wedge g_2(n) \leq cf_2(n)]\}$$

  - De exemplu:

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

Calcul asimptotic - continuare

- Utilizând regulile de asociere și prioritate, se obțin expresii de orice lungime:

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

- Orice funcție $f(n)$ poate fi gândită ca o notație pentru mulțimea cu un singur element $f(n)$ și deci putem alcătui expresii de forma:

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

ca desemnând mulțimea:

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

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

- Peste expresii considerăm formule de forma:

$$expr1 = expr2$$

cu semnificația că mulțimea desemnată de *expr*1 este inclusă în mulțimea desemnată de *expr*2.

Calcul asimptotic - 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 ecuațiilor: părțile stânga și cea din dreapta joacă roluri distincte.
- Ca un caz particular, notația $g(n) = O(f(n))$ semnifică, de fapt, $g(n) \in O(f(n))$.

Calculul timpului asimptotic de execuție pentru cazul cel mai nefavorabil

- Un algoritm poate avea o descriere complexă și deci evaluarea sa poate pune unele probleme.
- Deoarece orice algoritm este descris de un program, în continuare considerăm A o secvență de program.
- Regulile prin care se calculează timpul de execuție sunt date în funcție de structura lui A:
  - A este o instrucțiune de atribuire. Timpul de execuție a lui A este egal cu timpul evaluării expresiei din partea dreaptă.
  - A este forma $A_1\ A_2$ Timpul de execuție al lui A este egal cu suma timpilor de execuție ai algoritmilor A_1 și A_2.
  - A este de forma `if` e `then` A_1 `else` A_2. Timpul de execuție al lui A este egal cu maximul dintre timpii de execuție ai algoritmilor A_1 și 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 iterații ale buclei `while` și se face suma timpilor calculați pentru fiecare iterație. Dacă nu este posibilă determinarea timpilor pentru fiecare iterație, atunci timpul de execuție al lui A este egal cu produsul dintre timpul maxim de execuție al algoritmului A_1 și numărul maxim de execuții ale buclei A_1.

Timp polinomial

Theorem (1)

Dacă g este o funcție polinomială de grad k, atunci $g = O(n^k)$.

Demonstrație.

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, obținem:

$$g(n) \leq |a_k| \cdot n^k + |a_{k-1}| \cdot n^{k-1} + \cdots + |a_1| \cdot n + |a_0| < n^k \cdot \underbrace{(|a_k| + |a_{k-1}| + |a_0|)}_{c} < n^k \cdot c$$

pentru $\forall n > 1 \Rightarrow g(n) < c \cdot n^k$, cu $n_0 = 1$.
Deci $g = O(n^k)$. $\square$

Clasificarea algoritmilor

- Următoarele incluziuni sunt valabile în cazul notației 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 totație este O.
- Cele mai cunoscute clase sunt:

$$\begin{aligned}
\{A \mid T_A(n) = O(1)\} &= \text{clasa algoritmilor constanți;}\\
\{A \mid T_A(n) = O(\log n)\} &= \text{clasa algoritmilor logaritmici;}\\
\{A \mid T_A(n) = O(log^k n)\} &= \text{clasa algoritmilor polilogaritmici;}\\
\{A \mid T_A(n) = O(n)\} &= \text{clasa algoritmilor liniari;}\\
\{A \mid T_A(n) = O(n^2)\} &= \text{clasa algoritmilor pătratici;}\\
\{A \mid T_A(n) = O(n^k)\} &= \text{clasa algoritmilor polinomiali;}\\
\{A \mid T_A(n) = O(2^n)\} &= \text{clasa algoritmilor exponențiali.}
\end{aligned}$$

- Cu notațiile de mai sus, doi algoritmi, care rezolvă aceeași problemă, pot fi comparați numai dacă au timpii de execuție în clase de funcții (corespunzătoare notațiilor O, Ω și 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 execuție în aceeași clasă, atunci compararea lor devine mai dificilă pentru că trebuie determinate și constantele cu care se înmulțesc reprezentanții clasei.

Sarcini de lucru și barem de notare

Sarcini de lucru:

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

a) Pornind de la lista vidă, inserând 10000 de elemente și considerând ca pas elementar comparația a 2 pointeri, câte operații efectuează în plus studentul Popescu față de studentul Ionescu?

b) Ajutați-l pe studentul Popescu scriind pentru poblema dată o funcție C/C++ de complexitate $O(1)$, care primește ca parametru pointerul către primul element al listei circulare simplu înlănțuite și valoarea de inserat la sfârștul listei circulare!

Observații:

- Se vor număra operațiile elementare și se va măsura timpul experimentului (durata de execuție 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

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;
}
```