Notații de complexitate

23 octombrie 2022

1 Ce înseamnă și de ce depinde complexitatea?

Un algoritm performant consumă cât mai puține resurse temporale și spațiale. Cantitatea de resurse folosite depinde de:

- formatul datelor de intrare (ex: în general se consumă mai puțin pentru ordonarea unui vector gata sortat decât a unuia cu elemente aranjate aleator)
- dimensiunea datelor de intrare - notată tipic **n** (ex: se consumă mai mult timp sortând un vector de 1000 de elemente față de unul de 3 elemente)

Măsurarea experimentală a timpului consumat de algoritm introduce dependențe de hardware (mașina pe care se rulează) sau software (limbajul de programare) – lucruri care nu țin de calitatea intrinsecă a algoritmului. Vom prefera o **estimare matematică a numărului de pasi** parcurși de algoritm. În calculul de complexitate ținem cont numai de **operațiile critice** (operațiile care prin natura lor sunt foarte consumatoare sau sunt efectuate de un număr semnificativ de ori; ex: într-un algoritm de sortare o operație critică este comparația de elemente, întrucât se produce de multe ori).

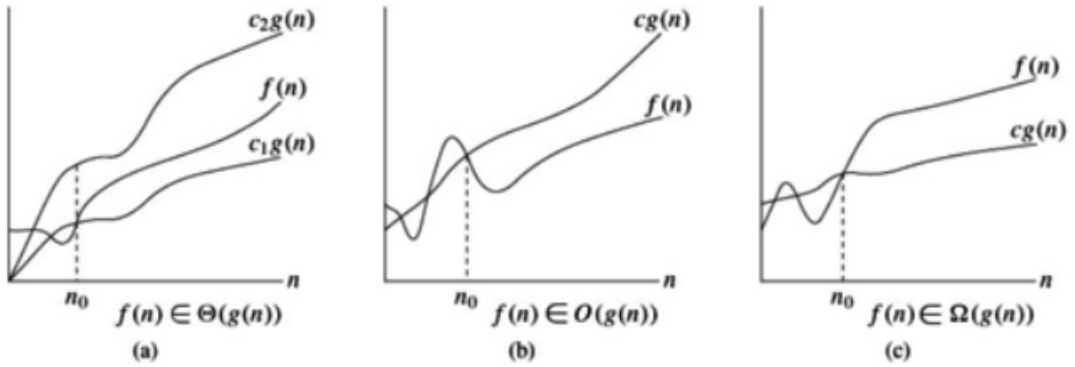
Este dificil de calculat exact un număr de pași, iar ceea ce va interesa este "cam de ce ordin" este funcția de complexitate (în ce **clasă de complexitate**) sau, altfel spus, **cum crește funcția de complexitate** (linear/logaritmic/exponențial). Astfel, se deduce cum se va comporta algoritmul pe dimensiuni mari ale datelor de intrare, acolo unde diferențele se simt cel mai acut. Această idee stă la baza **analizei asimptotice** de complexitate, bazată pe notațiile asimptotice de complexitate.

2 Tipuri de analiză de complexitate

- **cazul cel mai defavorabil** (ce complexitate rezultă pentru cel mai neprietenos input): analiza cea mai frecventă, e ușor de realizat și oferă utilizatorului o garanție: algoritmul nu se va purta niciodată mai rău decât atât;
- **cazul mediu** (la ce complexitate ne putem aștepta în cazul inputurilor aleatoare): o analiză utilă, însă dificil de realizat; necesită cunoașterea unei distribuții statistice a posibilelor inputuri, pentru a pondera pe fiecare cu probabilitatea sa de apariție și a calcula apoi o astfel de medie ponderată a complexităților;
- **cazul cel mai favorabil** (ce complexitate rezultă pentru cel mai prietenos input): analiza cea mai puțin frecventă, utilă cel mult pentru probleme care tind să aibă inputuri favorabile; în plus, este ușor de trișat, prin plasarea unui test pentru un input anume la începutul algoritmului, caz în care se dă direct rezultatul, cu minim de efort;

3 Notațiile asimptotice de complexitate (O, Ω, Θ, o, ω)

Funcțiile de complexitate sunt funcții asimptotic crescătoare de tip $N \to R_+$ (în figură, f(n) reprezintă numărul de pași efectuați de algoritm pentru o intrare de dimensiune n).



1. $O(g(n)) = \{f : N \to R_+ \,|\, \exists\, ct.\, c \in R_+, c > 0, \text{ și } n_0 \in Na.i.\, pt\, \forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$

   - $f(n) \in O(g(n))$ înseamnă că, pentru valori mari ale dimensiunii intrării, $c * g(n)$ este o **limită superioară** pentru $f(n)$; algoritmul se va purta mereu mai bine decât această limită;

2. $\Omega(g(n)) = \{f : N \to R_+ \,|\, \exists\, ct.\, c \in R_+, c > 0, \text{ și } n_0 \in Na.i.\, pt\, \forall n \geq n_0, 0 \leq cg(n) \leq f(n)\}$

   - $f(n) \in \Omega(g(n))$ înseamnă că, pentru valori mari ale dimensiunii intrării, $c * g(n)$ este o **limită inferioară** pentru $f(n)$; algoritmul se va purta mereu mai rău decât această limită;

3. $\Theta(g(n)) = \{f : N \to R_+ \,|\, \exists\, ct.\, c_1, c_2 \in R_+, c_1 > 0, c_2 > 0 \text{ și } n_0 \in Na.i.\, pt\, \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

   - $f(n) \in \Theta(g(n))$ înseamnă că, pentru valori mari ale dimensiunii intrării, $c_1 g(n)$ este o **limită inferioară** pentru $f(n)$, iar $c_2 g(n)$ o **limită superioară**; algoritmul tinde să se comporte "cam ca" $g(n)$.
   - **Pentru analize de complexitate cât mai precise preferăm notația Θ.**

4. $o(g(n)) = \{f : N \to R_+ \,|\, \forall c \in R_+, c > 0, \exists\, n(c) \in Na.i.\, pt\, \forall n \geq n(c), 0 \leq f(n) < cg(n)\}$

   - $f(n) \in o(g(n))$ înseamnă că $g(n)$ crește **strict** mai repede decât $f(n)$

5. $\omega(g(n)) = \{f : N \to R_+ \,|\, \forall c \in R_+, c > 0, \exists\, n(c) \in Na.i.\, pt\, \forall n \geq n(c), 0 \leq cg(n) < f(n)\}$

   - $f(n) \in \omega(g(n))$ înseamnă că $f(n)$ crește **strict** mai repede decât $g(n)$

Recomandare: Pentru continuarea capitolului de complexitate, revizuiți proprietățile logaritmilor, exponențialelor, seriilor aritmetice și geometrice.

Redefinirea notațiilor asimptotice folosind limite de funcții:

- $\lim_{n\to\infty}(\frac{f(n)}{g(n)}) = L \; cu \; 0 \leq L < \infty \Rightarrow f(n) \in O(g(n))$
- $\lim_{n\to\infty}(\frac{f(n)}{g(n)}) = L \; cu \; 0 < L < \infty \Rightarrow f(n) \in \Theta(g(n))$
- $\lim_{n\to\infty}(\frac{f(n)}{g(n)}) = L \; cu \; 0 < L \leq \infty \Rightarrow f(n) \in \Omega(g(n))$
- $\lim_{n\to\infty}(\frac{f(n)}{g(n)}) = 0 \Rightarrow f(n) \in o(g(n))$
- $\lim_{n\to\infty}(\frac{f(n)}{g(n)}) = \infty \Rightarrow f(n) \in \omega(g(n))$

4 Proprietăți ale notațiilor asimptotice de complexitate:

Tranzitivitatea:

- $f(n) \in O(h(n)) \wedge h(n) \in O(g(n)) \Rightarrow f(n) \in O(g(n))$
- $f(n) \in \Omega(h(n)) \wedge h(n) \in \Omega(g(n)) \Rightarrow f(n) \in \Omega(g(n))$
- $f(n) \in \Theta(h(n)) \wedge h(n) \in \Theta(g(n)) \Rightarrow f(n) \in \Theta(g(n))$
- $f(n) \in o(h(n)) \wedge h(n) \in o(g(n)) \Rightarrow f(n) \in o(g(n))$
- $f(n) \in \omega(h(n)) \wedge h(n) \in \omega(g(n)) \Rightarrow f(n) \in \omega(g(n))$

Reflexivitatea:

- $f(n) \in O(f(n))$
- $f(n) \in \Omega(f(n))$
- $f(n) \in \Theta(f(n))$

Simetria:

- $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$

Antisimetria:

- $f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$

Altele:

- $f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in O(g(n)) \wedge f(n) \in \Omega(g(n))$
- $f(n) \in o(g(n)) \Leftrightarrow f(n) \in O(g(n)) \wedge not(f(n) \in \Theta(g(n)))$
- $f(n) \in \omega(g(n)) \Leftrightarrow f(n) \in \Omega(g(n)) \wedge not(f(n) \in \Theta(g(n)))$
- $\Theta(f(n) + g(n)) = \Theta(f(n)) + \Theta(g(n))$
- $\Theta(f(n) * g(n)) = \Theta(f(n)) * \Theta(g(n))$

Exemplu de analiză de complexitate: insertion-sort

```
1: function INSERTION-SORT(v)                          ▷ Nr. execuții/instrucțiune
2:     for j ← 2 to n do                                ▷ n
3:         elem ← v[j]                                  ▷ n-1
4:         i ← j − 1                                    ▷ n-1
5:         while i > 0 and elem < v[i] do               ▷ S1
6:             v[i + 1] ← v[i]                          ▷ S2
7:             i ← i − 1                                ▷ S2
8:         end while
9:         v[i + 1] ← elem                              ▷ n-1
10:    end for
11: end function
```

1. **Cazul cel mai favorabil:** v deja sortat, nu se intră în while
   - S1 = n-1 (prin fiecare instrucțiune while se trece fix o dată, de n-1 ori)
   - S2 = 0 (nu se ajunge la instrucțiunile din interiorul while-ului) $\Rightarrow T(n) = 5n - 4 \Rightarrow T(n) \in \Theta(n)$
2. **Cazul cel mai defavorabil:** v sortat invers, fiecare while merge până la i=0
   - $S1 = \sum_{j=2}^{n} j = n(n+1)/2–1$
   - $S2 = \sum_{j=2}^{n}(j-1) = n(n-1)/2 \Rightarrow T(n) \in \Theta(n^2)$

Observație: Ca regulă generală când stabilim clasa de complexitate, în funcția de complexitate **ignorăm termenii cu creștere mai înceată și ignorăm constanta din fața termenului dominant.**

Exerciții:

1. Demonstrați proprietățile notațiilor de complexitate:
   (a) tranzitivitatea
   (b) simetria
   (c) reflexivitatea
   (d) ...
2. Demonstrați folosind notațiile asimptotice de complexitate sau cele cu limite:
   (a) $5 * \log n + 6 \in O(n)$
   (b) $n^2 \in \Theta(n^2 + n)$
   (c) $\log_a n \in \Theta(\log_2 n)$
   (d) $\sqrt{n} \in \Omega(\log n)$
   (e) $O(n^2 + n) = O(n^2)$
3. Identificați răspunsul pentru fiecare subpunct:
   (a) $\Omega(g(n)) \cap \Theta(g(n)) = ?$

(b) $O(g(n)) \cup \Theta(g(n)) =?$

(c) $\omega(g(n)) + \Theta(g(n)) =?$

(d) $2^{2n} \in \Theta(2^n) =?$

4. Rezolvați:

   (a) Știind că f(n) $\in o(n^2)$ și h(n) $\in \Theta(n^3)$, determinați complexitatea f(n) * h(n).

   (b) Știind că f(n) $\in \Omega(n)$ și h(n) $\in O(n^2)$, determinați complexitatea h(n) / f(n).

   (c) Știind că f(n) $\in O(n^5)$ și g(n) $\in \Theta(\sqrt{n})$, este adevărată afirmația: f(g(n)) $\in \omega(n^2)$?

   (d) Știind că f(n) $\in O(g(n))$, care este limita asimptotică pentru: o(f(n)) + o(g(n))?

   (e) Știind că f(n) $\in O(\log(n))$, este adevărată afirmația: $2^{f(n)} \in$ O(n)?

5. Calculați complexitatea următoarelor bucăți de cod:

   (a)
   ```
   for (int i=1; i<c; i++) do
       //operații cu complexitate O(1)
   end for
   ```

   (b)
   ```
   for (int i=1; i<=n; i+ = c) do
       //operații cu complexitate O(1)
   end for
   ```

   (c)
   ```
   for (int i=n; i>0; i− = c) do
       for (int j=i+1; j<=n; j+ = c) do
           //operații cu complexitate O(1)
       end for
   end for
   ```

   (d)
   ```
   for (int i=n; i>0; i−−) do
       for (int j=1; j<n; j∗ =10) do
           for (int k=0; k<j; k++) do
               ... //număr constant c1 de operații
           end for
       end for
   end for
   ```

   (e)
   ```
   for (int i=1; i<n; i∗ =2) do
       for (int j=n; j>0; j/ =2) do
           for (int k=j; k<n; k++) do
               ... //număr constant c2 de operații
           end for
       end for
   end for
   ```

   (f)
   ```
   for (int i=1; i<=n; i∗ =2) do
       for (int j=0; j<i; j++) do
           for (int k=0; k<n; k+ =2) do
               ... //număr constant c3 de operații
           end for
           for (int k=1; k<n; k∗ =2) do
               ... //număr constant c4 de operații
           end for
   ```

end for
end for