Analiza eficienței algoritmilor

SD 2014/2015

Conținut

Analiza eficienței algoritmilor

Analiza funcțiilor recursive

- Funcții recursive
- Metoda substituției
- Metoda iterației
- Arborele de recursie
- Teorema Master

Clase de eficiență

Clasa	Notație	Exemplu
logaritmic	$O(\log n)$	căutare binară
liniar	$O(n)$	căutare secvențială
pătratic	$O(n^2)$	sortare prin inserție
cubic	$O(n^3)$	înmulțirea a două matrici $n \times n$
exponențial	$O(2^n)$	prelucrarea submulțimilor unei mulțimi cu n elemente
factorial	$O(n!)$	prelucrarea permutărilor de ordin n

Analiza empirică a eficienței algoritmilor

- Utilizată atunci când analiza teoretică a eficienței este dificilă.

- Scop:
  - formularea unei ipoteze inițiale privind eficiența algoritmului;
  - verificarea unei afirmații (ipoteze) privind eficiența;
  - compararea algoritmilor;
  - analiza eficienței unei implementări.

Analiza empirică a eficienței algoritmilor

- Se stabilește scopul analizei.
- Se alege o masură a eficienței
  Exemplu: numărul de execuții ale unor operații, timpul, etc.
- Se stabilesc caracteristicile setului de date de intrare.
- Se implementează algoritmul.
- Se generează datele de intrare.
- Se execută programul pentru toate datele de intrare; se înregistrează rezultatele.
- Se analizează rezultatele.

Conținut

Conținut

Funcții recursive

- Funcția f() **apelează direct** funcția g() dacă în definiția lui f() există un apel la g().
- Funcția f() **apelează indirect** funcția g() dacă f() apelează direct o funcție h(), iar h() apelează direct sau indirect funcția g().
- Funcția f() este definită **recursiv** dacă ea se auto-apelează direct sau indirect.

Funcții recursive

Definiția unei funcții recursive cuprinde:

- **Testarea cazului de bază** – condiția de oprire a apelului recursiv.
- **Apelul recursiv (cazul general)**: o variabilă (întreagă) este transmisă ca parametru funcției însăși, în așa fel ca după un număr de pași să se atingă cazul de bază.

Observație: Există și funcții recursive fără parametri.

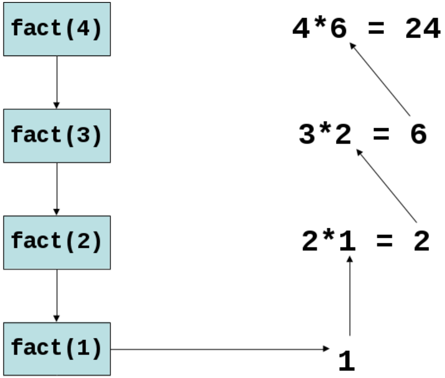
Funcții recursive

Exemplul 1. Definiția funcției factorial:

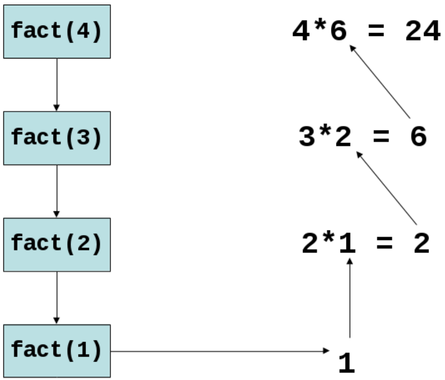
- Cazul de bază: $0! = 1$;
- Cazul general: $n! = n \times ((n-1)!), n > 0$.

Function *factorial(n)*
begin
  if $n <= 1$ **then**
    return 1
  else
    return $(n * factorial(n-1))$
end

factorial(4) - apel recursiv



factorial(4) - apel recursiv



- algoritmii recursivi: ușor de implementat;
- costuri suplimentare: la fiecare apel recursiv se plasează o serie de informații într-o zonă de memorie specifică (stiva programului).

factorial(n) - varianta iterativă

```
Function factorial(n)
begin
    produs ← 1
    while n > 1 do
        produs ← produs * n
        n ← n − 1
    return produs
end
```

Observație: Valoarea returnată de *factorial*(n) este corectă doar pentru valorile lui n pentru care n! este mai mic sau egal decât cea mai mare constantă întreagă pe care o putem reprezenta !

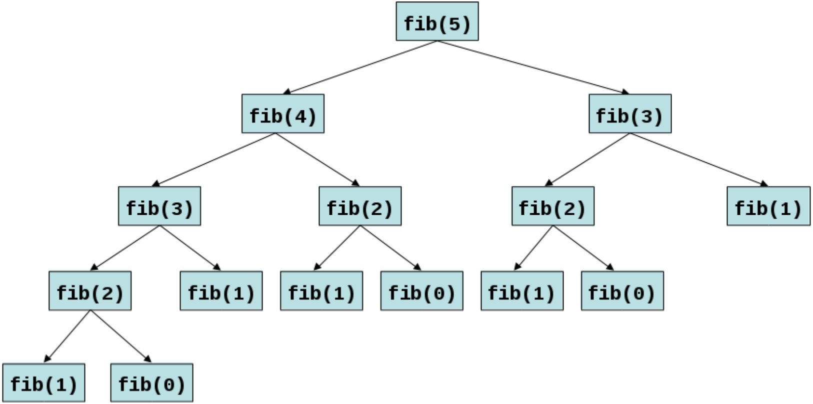
Recursie vs. iterație. Fibonacci recursiv

Exemplul 2. Șirul lui Fibonacci:

- $f(0) = 0, f(1) = 1,$
- $f(n) = f(n-1) + f(n-2), n > 1.$

Function *fib(n)*
begin
    if $n <= 1$ **then**
        return n
    else
        return $fib(n-1) + fib(n-2)$
end

Fibonacci recursiv: arbore apeluri



$O(\phi^n)$

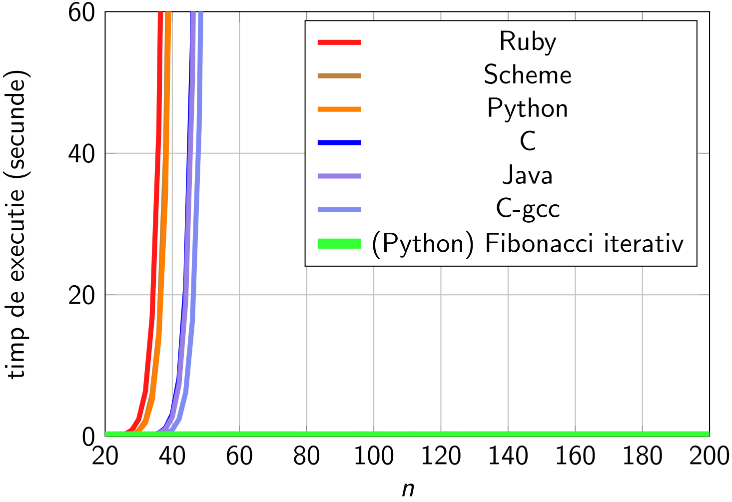
Număr de apeluri

n	fib(n)	apeluri
2	1	3
24	46’368	150’049
42	267’914’296	866’988’873
43	433’494’437	1’402’817’465

Recursie vs. iterație: Fibonacci iterativ

Function *ifib(n)*
begin
    $f0 \leftarrow 0$
    $f1 \leftarrow 1$
    if $n <= 1$ **then**
        return n
    else
        for $k \leftarrow 2$ **to** n **do**
            $temp \leftarrow f1$
            $f1 \leftarrow f1 + f0$
            $f0 \leftarrow temp$
        return $f1$
end

Comparaţie Fibonacci recursiv/iterativ



Eficiența algoritmilor recursivi

- Pentru estimarea timpului de execuție:
  - se stabilește relația de recurență care exprimă legatura dintre timpul de execuție corespunzător problemei inițiale și timpul de execuție corespunzător problemei reduse;
  - se rezolvă relația de recurență.
- Exemplu: pentru calculul factorialului, relația de recurență pentru timpul de execuție este:

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 1, & n > 1 \end{cases}$$

Rezolvarea recurențelor

1. **Metoda substituției.** Se ghicește o limită și apoi se utilizează inducția matematică pentru a demonstra corectitudinea.

2. **Metoda iterației.** Se iterează recurența și se exprimă ca o sumă de termeni care depind doar de dimensiunea problemei și de condițiile inițiale.

3. **Arborele de recursie.** Convertește recurența într-un arbore (nodurile reprezintă costuri).

4. **Metoda master.** Furnizează limite pentru recurențe de forma

$$T(n) = aT(n/b) + f(n)$$

.

Conținut

- Se ghicește soluția.
- Se utilizează inducția matematică pentru a determina constantele și pentru a demonstra că soluția este corectă.

Metoda substituției - exemplu

Determinarea unei limite superioare pentru relația $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Metoda substituției - exemplu

Determinarea unei limite superioare pentru relația $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- Ghicim soluția: $T(n) = O(n \log n)$.
- Demonstrăm prin inducție că $T(n) \leq cn \log n$, pentru $c > 0$.

Metoda substituției - exemplu

Determinarea unei limite superioare pentru relația $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- Ghicim soluția: $T(n) = O(n \log n)$.
- Demonstrăm prin inducție că $T(n) \leq cn \log n$, pentru $c > 0$. Presupunem că limita are loc pentru toate valorile pozitive $m < n$, în particular pentru $m = \lfloor n/2 \rfloor$: $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)$.

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)) + n \\ &\leq cn \log(\lfloor n/2 \rfloor) + n \\ &= cn \log n - cn \log 2 + n \\ &= cn \log n - cn + n \\ &\leq cn \log n, \text{ pentru } c \geq 1 \end{aligned}$$

Metoda substituției - exemplu

Determinarea unei limite superioare pentru relația $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- Ghicim soluția: $T(n) = O(n \log n)$.
- Demonstrăm prin inducție că $T(n) \leq cn \log n$, pentru $c > 0$.
  Presupunem că limita are loc pentru toate valorile pozitive $m < n$, în particular pentru $m = \lfloor n/2 \rfloor$: $T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)$.

$$\begin{aligned}
T(n) &\leq 2(c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)) + n \\
&\leq cn \log(\lfloor n/2 \rfloor) + n \\
&= cn \log n - cn \log 2 + n \\
&= cn \log n - cn + n \\
&\leq cn \log n, \text{ pentru } c \geq 1
\end{aligned}$$

Trebuie să arătăm că soluția este validă și pentru condițiile limită.
$T(1) = 1 \leq c1 \log 1 = 0$
Cazuri de bază: T(2) și T(3) ($n_0 = 2$)
$T(2) = 4$ și $T(3) = 5$, $T(2) \leq c2 \log 2$ și $T(3) \leq c3 \log 3 \Rightarrow c \geq 2$.

Metoda substituției - subtilități

- Scăderea unui termen de ordin inferior (pentru a consolida ipoteza inductivă).

Exemplu: $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

- Ghicim soluția: $T(n) = O(n)$.
- Demonstrăm prin inducție că $T(n) \leq cn$, pentru $c > 0$.

$$\begin{aligned} T(n) &\leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\ &= cn + 1 \end{aligned}$$

- Demonstrăm prin inducție că $T(n) \leq cn - d$, $d >= 0$ const.

$$\begin{aligned} T(n) &\leq (c\lfloor n/2 \rfloor - d) + (c\lceil n/2 \rceil - d) + 1 \\ &= cn - 2d + 1 \\ &\leq cn - d, \text{ pentru } d \geq 1 \end{aligned}$$

Trebuie să alegem constanta c suficient de mare pentru a satisface condițiile limită.

Metoda substituției - subtilități

- Evitarea capcanelor

  Exemplu: $T(n) = 2T(\lfloor n/2 \rfloor) + n$

  Demonstrăm "fals" ca $T(n) = O(n)$ ghicind $T(n) \leq cn$ și argumentând:

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor) + n \\ &\leq cn + n \\ &= O(n), \quad \Leftarrow \text{fals!!} \end{aligned}$$

  Eroarea: nu am demonstrat *forma exactă* a ipotezei inductive.

Metoda substituției - subtilități

- Schimbare de variabilă.

  Exemplu: $T(n) = 2T(\lfloor sqrt(n) \rfloor) + \log n$

  Simplificăm recurența printr-o schimbare de variabilă $m = \log n$.
  $T(2^m) = 2T(2^{m/2}) + m$

  Redenumim $S(m) = T(2^m)$, și avem $S(m) = 2S(m/2) + m$.
  $S(m) = O(m \log m)$,
  $T(n) = T(2^m) = S(m) = O(m \log m) = O(\log n \log \log n)$.

Conținut

Iterarea unei recurențe

Metoda substituției: implică ghicirea soluției (!)

Iterarea relației de recurență:

- **directă**
  - se pornește de la cazul particular și se construiesc termeni succesivi folosind relația de recurență;
  - se identifică forma termenului general $T(n)$;
  - se verifică prin calcul direct sau inducție matematică.
- **inversă**
  - se pornește de la cazul $T(n)$ și se inlocuiește $T(h(n))$ cu valoarea corespunzătoare, apoi se inlocuiește $T(h(h(n)))$ și așa mai departe, până se ajunge la cazul particular;
  - se efectuează calculele și se obține $T(n)$.

Iterarea unei recurențe - exemplu $n!$

$$T(n) = \begin{cases} 0, & n = 1 \\ T(n-1) + 1, & n > 1 \end{cases}$$

Iterarea unei recurențe - exemplu $n!$

$$T(n) = \begin{cases} 0, & n = 1 \\ T(n-1) + 1, & n > 1 \end{cases}$$

Iterare directă

$T(1) = 0$
$T(2) = 1$
$T(3) = 2$
$\ldots$
$T(n) = n - 1$

Iterarea unei recurențe - exemplu $n!$

$$T(n) = \begin{cases} 0, & n = 1 \\ T(n-1) + 1, & n > 1 \end{cases}$$

Iterare directă

$T(1) = 0$
$T(2) = 1$
$T(3) = 2$
$\ldots$
$T(n) = n - 1$

Iterare inversă

$T(n) = T(n-1) + 1$
$T(n-1) = T(n-2) + 1$
$\ldots$
$T(2) = T(1) + 1$
$T(1) = 0$

$T(n) = n - 1$

Iterarea unei recurențe - exemplu

$T(n) = 3T(\lfloor \frac{n}{4} \rfloor) + n$

$$
\begin{aligned}
T(n) &= n + 3(\lfloor \frac{n}{4} \rfloor) + 3T(\lfloor \frac{n}{16} \rfloor) \\
&= n + 3\lfloor \frac{n}{4} \rfloor + 9(\lfloor \frac{n}{16} \rfloor + 3T(\lfloor \frac{n}{64} \rfloor)) \\
&= n + 3\lfloor \frac{n}{4} \rfloor + 9\lfloor \frac{n}{16} \rfloor + 27T(\lfloor \frac{n}{64} \rfloor) \\
\dots \\
&\leq n \sum_{i=0}^{\infty} (\frac{3}{4})^i + \Theta(n^{log_4 3} x T(1)) \\
&= 4n + \Theta(n^{log_4 3} x T(1)) \\
&= O(n)
\end{aligned}
$$

Observație: utilizarea seriilor geometrice:

$1 + x + x^2 + ... + x^n = \frac{1-x^{n+1}}{1-x}$, pentru $x \neq 1$

Conținut

2. Arborele de recursie

- **Arborele de recursie**:
  - permite vizualizarea ieterării unei recurențe;
  - fiecare nod reprezintă costul unei subprobleme;
  - se calculează suma costurilor pe nivele și apoi se însumează aceste costuri pentru a determina costul total al recursiei.
- Arborele de recursie poate fi utilizat pentru a genera o valoare pentru metoda substituției.

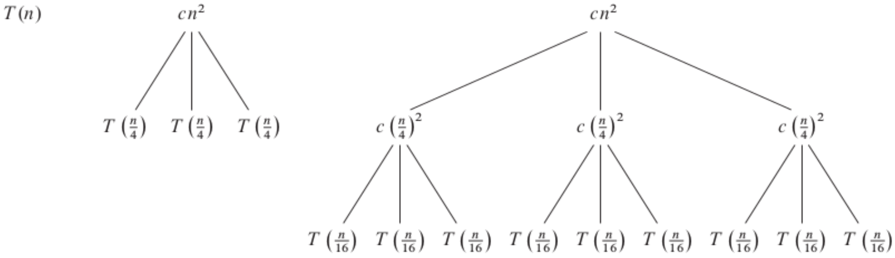
Arborele de recursie - exemplu

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

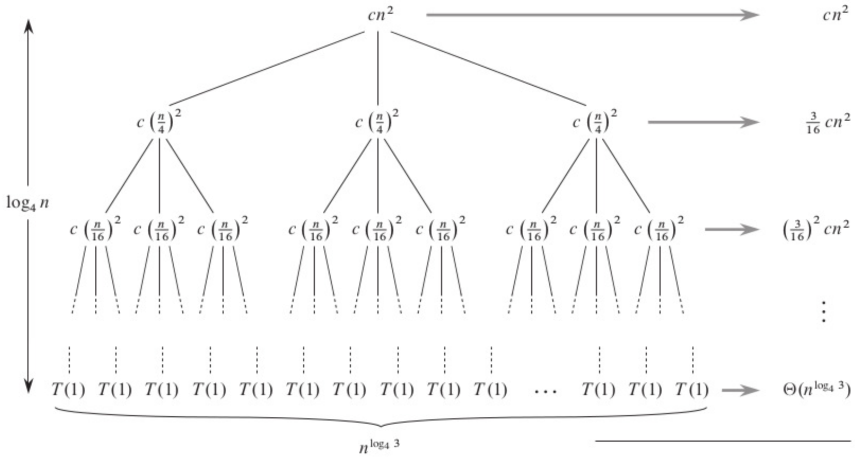
Arborele de recursie - exemplu

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

- Creăm arborele de recursie pentru $T(n) = 3T(n/4) + cn^2$, $c > 0$



Arborele de recursie - exemplu



Arborele de recursie - exemplu

- Dimensiunea unei subprobleme corespunzătoare unui nod de adâncime i: $n/4^i \Rightarrow$ dimensiunea subproblemei ajunge la $n = 1$ cand $n/4^i = 1 \Leftrightarrow i = log_4 n \Rightarrow$ arborele are $log_4 n + 1$ nivele.
- Numărul de noduri de la nivelul i: 3^i.
- Fiecare nod de pe nivelul i are costul: $c(n/4^i)^2$.
- Costul total al nodurilor de la nivelul i: $3^i c(n/4^i)^2 = (3/16)^i cn^2$ (Ultimul nivel $log_4 n$: $n^{log_4 3} T(1)$.)

Arborele de recursie - exemplu

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \ldots + (\frac{3}{16})^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n - 1} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1 - (3/16)} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2)
\end{aligned}
$$

Arborele de recursie - exemplu

- Utilizăm metoda substituției pentru a verifica că $T(n) = O(n^2)$ este o limită superioară pentru relația $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$.

- Arătăm ca $T(n) \leq dn^2$, pentru $d > 0$

$$\begin{aligned} T(n) &\leq 3T(\lfloor n/4 \rfloor) + cn^2 \\ &\leq 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\leq 3d(n/4)^2 + c(n^2) \\ &= \frac{3}{16}dn^2 + cn^2 \\ &\leq dn^2, \text{ pentru } d \geq (16/13)c \end{aligned}$$

Conținut

3. Teorema Master

- Furnizează o metodă de rezolvare a recurențelor de forma $T(n) = aT(n/b) + f(n)$ unde $a \geq 1$ și $b > 1$ sunt constante, iar $f(n)$ este o funcție asimptotic pozitivă.

- **Teorema Master**:
  Fie $a \geq 1$ și $b > 1$ constante, $f(n)$ o funcție și $T(n)$ definită pe numere întregi nenegative prin relația de recurență: $T(n) = aT(n/b) + f(n)$. Avem:
  1. Dacă $f(n) = O(n^{log_b a - \epsilon})$ pentru $\epsilon > 0$ constant, atunci $T(n) = \Theta(n^{log_b a})$.
  2. Dacă $f(n) = \Theta(n^{log_b a})$, atunci $T(n) = \Theta(n^{log_b a} \log n)$.
  3. Dacă $f(n) = \Omega(n^{log_b a + \epsilon})$ pentru $\epsilon > 0$ constant, și dacă $af(n/b) \leq cf(n)$ pentru $c < 1$ și n suficient de mare, atunci $T(n) = \Theta(f(n))$.

Teorema Master - exemple

- $T(n) = 9T(n/3) + n$

  $a = 9, b = 3, f(n) = n$ și $n^{log_b a} = n^{log_3 9} = \Theta(n^2)$.
  Cum $f(n) = O(n^{log_3 9 - \epsilon})$, cu $\epsilon = 1$, putem aplica cazul 1 al teoremei master $\Rightarrow T(n) = \Theta(n^2)$.

- $T(n) = T(2n/3) + 1$

  $a = 1, b = 3/2, f(n) = 1$ și $n^{log_b a} = n^{log_{3/2} 1} = n^0 = 1$.
  Cum $f(n) = \Theta(n^{log_b a}) = \Theta(1)$, putem aplica cazul 2 al teoremei master $\Rightarrow T(n) = \Theta(\log n)$.

Teorema Master - exemple

- $T(n) = 3T(n/4) + n \log n$

  $a = 3, b = 4, f(n) = n \log n$ și $n^{log_b a} = n^{log_4 3} = O(n^{0.793})$
  Cum $f(n) = \Omega(n^{log_4 3+\epsilon})$, cu $\epsilon \approx 0.2$, putem aplica cazul 3 al teoremei master dacă are loc condiția:
  $af(n/b) = 3(n/4)\log(n/4) \leq (3/4)n \log n = cf(n)$ pentru $c = 3/4$ și n suficient de mare. Rezultă $T(n) = \Theta(n \log n)$.

- Metoda master nu se poate aplica pentru $T(n) = 2T(n/2) + n \log n$

  $a = 2, b = 2, f(n) = n \log n$ și $n^{log_b a} = n$
  Cum $f(n) = n \log n$ este asimptotic mai mare decât $n^{log_b a} = n$, putem aplica cazul 3 (fals!!).
  $f(n)$ nu este *polinomial* mai mare.
  $f(n)/n^{log_b a} = (n \log n)/n = \log n$ este asimptotic mai mic decât n^{ϵ}, pentru orice constantă pozitivă ϵ.