Calcularea complexității algoritmilor recursivi

October 25, 2017

Spre deosebire de un algoritm iterativ, unde putem număra operațiile efectuate în fiecare buclă, un algoritm recursiv necesită o analiză mai atentă. Un algoritm recursiv poate fi definit prin relația:

$$T(n) = cost\_recursiv(n) + cost\_nerecursiv(n)$$

Costul recursiv al unui apel se referă la câte noi apeluri recursive lansează un anumit apel și la dimensiunea parametrilor acestor noi apeluri, în timp ce costul nerecursiv reprezintă orice altceva mai face un apel (alte operații).

Relații de recurență uzuale:

1. Divide et impera: $T(n) = bT(n/c) + f(n)$
2. Chip and conquer: $T(n) = T(n-c) + f(n)$
3. Chip and be conquered: $T(n) = bT(n-c) + f(n)$

În primul rând, pentru un algoritm recursiv, trebuie definită o funcție care să descrie complexitatea sa. Având funcția, putem aplica diverse metode pentru a determina complexitatea acesteia.

Vom exemplifica aceste metode prin studiul algoritmului Mergesort.

Algorithm 1 Pseudocod mergesort

```
1:  function MERGE(V1[1..n], V2[1..m])
2:      Let R[1..n + m]
3:      i = 0, j = 0
4:      while i < n and j < m do
5:          if V1[i] < V2[j] then R[i + j] = V1[i + +]
6:          elseR[i + j] = V2[j + +]
7:          end if
8:      end while
9:      while i < n do
10:         R[i + j] = V1[i + +]
11:     end while
12:     while j < m do
13:         R[i + j] = V2[j + +]
14:     end while
15: end function
```

```
1: function MergeSort(V[1..n], start, end)
2:     if start ≥ end then return
3:     end if
4:     middle = (start + end)/2
5:     MergeSort(V, start, middle)
6:     MergeSort(V, middle, end)
7:     return Merge(V[start..middle], V[middle..end])
8: end function
```

Se poate observa că algoritmul *Merge* parcurge o singură dată elementele celor doi vectori, având complexitatea $\Theta(n)$. Pentru a calcula complexitatea algoritmului *MergeSort*, vom determina întâi o funcție caracteristică acestuia.

$$T(n) = 2T(n/2) + \Theta(n)$$

În această formulă, n reprezintă dimensiunea vectorului primit la intrarea. Complexitatea algoritmului depinde de numărul de apeluri recursive (în cazul nostru, 2) și de complexitatea fiecărui apel (în cazul nostru dată de apelul funcției *Merge*, având complexitatea $\Theta(n)$).

1 Metoda iterației

$$T(n) = 2T(n/2) + \Theta(n)$$

$$2T(n/2) = 4T(n/4) + 2\Theta(n/2)$$

$$4T(n/4) = 8T(n/8) + 4\Theta(n/4)$$

$$\dots$$

$$2^kT(n/2^k) = 2^{k+1}T(n/2^{k+1}) + 2^k\Theta(n/2^k)$$

$$\dots$$

$$2^hT(1) = 2^h\Theta(n/2^h)$$

Putem considera că algoritmul se oprește la $T(1)$, de unde nu se mai fac alte apeluri recursive. De asemenea, se poate observa că toți termenii liberi sunt egali cu $\Theta(n)$. Adunând toate aceste egalități, după simplificări, obținem următoarea formă pentru $T(n)$:

$$T(n) = (h+1)\Theta(n)$$

Pentru a afla cât este h, trebuie egalat $T(1)$ cu forma generală găsită, $T(n/2^h)$:

$$n/2^h = 1 \Rightarrow h = log(n) \Rightarrow T(n) \in \Theta(nlog(n))$$

2 Metoda arborelui de recurență

În fig. 1 este exemplificată această metodă pentru *MergeSort*. În fiecare nod al arborelui punem complexitatea termenului liber. În partea dreaptă, sunt adunate valorile din nodurile de pe acel nivel. Suma valorilor de pe fiecare nivel va reprezenta complexitatea întregului algoritm.

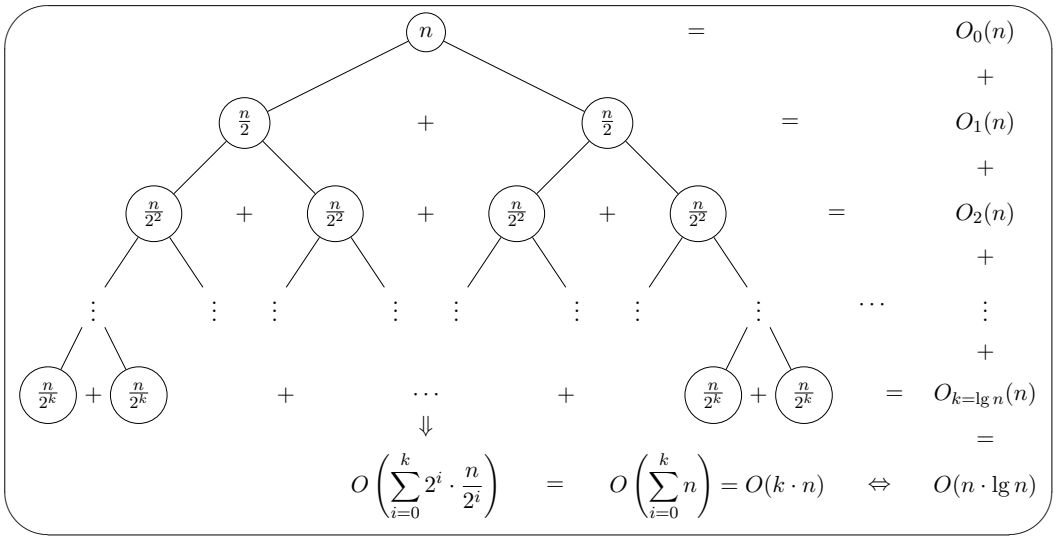

Figure 1: Arborele de recurență pentru MergeSort

Această metodă este foarte asemănătoare cu metoda iterației, în ambele cazuri fiind nevoiți să „ghicim" forma generală a sumei. O demonstrație formală ar necesita și o demonstrație prin inducție a aceste forme. Această reprezentare grafică poate fi mai intuitivă în unele cazuri, mai ales când metoda iterației este greu de aplicat.

3 Exerciții

1. $T(n) = 2T(n-1) + 1$
2. $T(n) = T(n-2) + \Theta(n)$
3. $T(n) = T(n-k) + \Theta(n)$, k const
4. $T(n) = T(n-a) + T(a) + \Theta(cn), a \geq 1 \text{ și } c > 0$ const
5. $T(n) = 2T(n/3) + nlogn$
6. $T(n) = 7T(n/2) + n^2$

7. $T(n) = 2T(\sqrt{n}) + 1$
8. $T(n) = 2T(\sqrt{n}) + logn$
9. $T(n) = 2T(\sqrt{n}) + log(log(n))$
10. $T(n) = T(n-1) + T(n-2)$
11. $T(n) = T(n/5) + T(4n/5) + \Theta(n)$
12. $T(n) = T(n/5) + T(7n/10 + 6) + \Theta(n)$
13. $T(n) = T(n/2) + T(n/4) + n^2$
14. $T(n) = T(n/2 + log(n)) + n$
15. $T(n) = 4T(n/2) + n^2/logn$