Fiche 05 :
Notion de complexité
une correction

Licence 3 informatique
2022 / 2023

Exercice 1 : Calcul de complexité d'algorithmes itératifs

Algorithme machin(T : tableau d'entier, n : entier)
begin
for i from 0 to $n-1$ **do**
  $x = T[i]$
  $T[i] = T[n-1-i]$
  $T[n-1-i] = x$
end for
end

Algorithme truc(M : matrice d'entier, n : entier)
begin
for i from 1 to $n-1$ **do**
  for j from 0 to $i-1$ **do**
    $x = M[i][j]$
    $M[i][j] = M[j][i]$
    $M[j][i] = x$
  end for
end for
end

1.a. En fait, l'algorithme `machin` laisse inchangé le tableau après avoir inverser deux fois l'ordre des éléments du tableau.
L'algorithme `truc` calcule la transposée de la matrice.

1.b. Pour l'algorithme `machin`.

Pour chaque itération de la boucle `for`, 3 affectations, 6 accès mémoires et 4 soustractions sont exécutées. On peut considérer une complexité de 3 unités par itération. Il y a n itérations, donc la complexité temporelle est $C_1(n) = 3n$. Donc $C_1(n) = \mathcal{O}(n)$, la classe de complexité de l'algorithme est linéaire.

Pour l'algorithme `truc`.

Pour chaque itération de la boucle `for` itérant sur j, on peut de nouveau considérer une complexité de 3 unités. Il y a i itérations, la complexité de cette boucle est donc $3i$ unités. La seconde boucle `for` itérant sur i comporte n itérations. Ainsi la complexité de l'algorithme est $C_2(n) = \sum_{i=1}^{n} 3i$. En réduisant la somme, nous obtenons $C_2(n) = 3\sum_{i=1}^{n} i = 3\frac{n(n-1)}{2}$. Ainsi, $C_2(n) = \mathcal{O}(n^2)$, la complexité de classe de l'algorithme est quadratique.

Exercice 2 : Calcul de complexité d'algorithme récursif

Une version de l'algorithme quicksort :

Algorithme quicksort(T : tableau d'entier, a, b : entier)
begin
if $a < b$ **then**
  pivot $\leftarrow$ partition(T, a, b)
  quicksort(T, a, pivot - 1)
  quicksort(T, pivot + 1, b)
end if
end

Algorithme partition(T : tableau d'entier, a, b : entier)
begin
pivot $\leftarrow T[b]$
$i \leftarrow a - 1$
for j from a to $b - 1$ **do**
  if $T[j] <$ pivot **then**
    $i \leftarrow i + 1$
    swap$(T[i], T[j])$
  end if
end for
if $T[b] < T[i + 1]$ **then**
  swap$(T[i + 1], T[b])$
end if
return $i + 1$
end

Complexité de l'algorithme `partition` :

Posons $n = b - a$, la taille du tableau traitée par l'algorithme. Avant et après la boucle `for`, seulement quelques instructions élémentaires sont exécutées. La boucle `for` comporte 3 instructions qui sont répétées $n - 1$ fois. La complexité de l'algorithme partition est donc $C_{part}(n) = 5 + 3n = \mathcal{O}(n)$. La complexité est linéaire.

Complexité de l'algorithme `quicksort` :

Posons $n = b-a$, la taille du tableau traitée par l'algorithme. Dans le pire des cas, la condition $a < b$ est exécutée. Ainsi la complexité de l'algorithme répond à la relation de récurrence $C(n) = C_{part}(n) + 2C(\frac{n}{2})$. Si on approxime la complexité de `partition` par $C_{part}(n) = n$, on obtient :

$$C(n) = n + 2C(\frac{n}{2}) \tag{1}$$

Pour simplifier les calculs, supposons que $n = 2^k$, c'est-à-dire que $k = \log_2(n)$. Une remarque : si n n'est pas une puissance de 2, le raisonnement reste valide en supposant que k est l'entier supérieur à $\log_2(n)$, soit $k = \lceil \log_2(n) \rceil$.

$C(n) = n + 2C(\frac{n}{2})$, or en utilisant la formule de récurrence (1), $C(\frac{n}{2}) = \frac{n}{2} + 2C(\frac{n}{4})$.
Donc, $C(n) = n + 2(\frac{n}{2} + 2C(\frac{n}{4})) = n + n + 4C(\frac{n}{4})$
En répétant k fois, le même calcul, on obtient :
$C(n) = n + n + \ldots + n + 2^k C(1)$
Or, $C(1) = 1$, $2^k = n$ et la somme contient k termes.
D'où le résultat final $C(n) = n\log_2(n) + n = n(\log_2(n) + 1) = \mathcal{O}(n\log(n))$. La complexité du quicksort est quasi-linéaire.