TD de Algorithmique n° 2

13 octobre 2011
(Correction)

I) La relation O

La notation $f(x) = O(g(x))$ signifie qu'il existe $c > 0$ tel que $f(x) \leq c \cdot g(x)$ dès que x est suffisamment grand. Noter qu'elle ne s'applique qu'à des fonctions positives à partir d'une certaine valeur n_0. On a :

réflexivité $f(x) = O(f(x))$

transitivité si $f(x) = O(g(x))$ et $g(x) = O(h(x))$, alors $f(x) = O(h(x))$

linéarité si $f_1(x) = O(f_2(x))$ et $g_1(x) = O(g_2(x))$, alors $f_1(x) + a \cdot g_1(x) = O(f_2(x) + a \cdot g_2(x))$.

$f(x) = O(a \cdot f(x))$ pour tout réel $a > 0$;

si $\lim_{x \to +\infty} f(x) = +\infty$, alors $f(x) + b = O(f(x))$ pour toute constante réelle b ;

si f, g sont strictement positives pour x assez grand, alors $f(x) = O(g(x))$ équivaut à $\frac{f}{g}$ bornée ;

Exercice 1 :

1. Comparer 10^{100} et 100^{10}.
2. Comparer $10^{(10^{10})}$ et $(10^{10})^{10}$.

Correction :

1. $10^{100} >> 100^{10} = 10^{20}$.
2. $10^{10000000000} = 10^{(10^{10})} << (10^{10})^{10} = 10^{100}$.

Exercice 2 :

Dire si les affirmations suivantes sont vrais ou fausses :

1. $2n + 3 = O(n)$,
2. $\log^{145} n = O(\log n)$,
3. $2^n + \log n = O(n^2)$,
4. $2^n = O(n^4 + 4n^2 + \log n)$,
5. $2n^7 + 4n^6 + 5n^5 + 2n^3 + 2n = O(n^7)$.

Correction : 1. vrai ; 2. vrai ; 3. faux ; 4. faux ; 5. vrai.

II) Complexité d'une boucle

Exercice 3 :

Considérez la fonction suivante :

FunA(n)
1 $x \leftarrow 0$
2 **for** $(k \leftarrow 1)$ **to** n
3 **do** $x \leftarrow x + k$
4 **return** x

1. Qu'est ce que calcule la fonction FunA(n) ?
2. Calculer la complexité de FunA(n) en fonction de son entrée $n \in \mathbb{N}$.

Correction :

1. FunA(n) $= \sum_{k=0}^{n} k = \frac{n(n+1)}{2}$.
2. La ligne 1 est exécutée seulement une fois. Dès qu'on a une boucle *while* ou *for*, la condition est testée une fois de plus par rapport aux instructions dans son corps, donc la ligne 2 est exécutée n+1 fois et la ligne 3 exactement n fois. Finalement, la ligne 4 est exécutée 1 fois. En résumant le coût de FunA(n) est $1 + (n+1) + n + 1 = 2n + 3 = O(n)$.

Exercice 4 :

Considérez les quatres bouts de code suivants :

a)
```
for (i=1; i<=n; i++) {
    for (j=i; j<=n; j++) {
      printf("salut");
    }
  }
```

b)
```
i = 1;
while (i<=n) {
    printf("salut");
    i = 2*i;
}
```

c)
```
i  = 1;
while (i<=n) {
  for (j=1; j<=i; j++) {
    printf("salut");
  }
  i = 2*i;
}
```

d)
```
for (j=1; j<=n; j++) {
  i  = 1;
  while (i<=j) {
    printf("salut");
    i = 2*i;
  }
}
```

Pour chacun, calculez (comme fonction de n) le nombre exact de fois que la chaine "salut" est affichée. Vous pourrez tester des petites valeurs de n.

Si f_b, f_c et f_d sont les trois fonctions de complexité des trois derniers programmes, lesquelles des affirmations suivantes sont vraies ? Justifiez.

$f_d \in O(f_b)$ $\quad$ $f_d \in O(f_c)$ $\quad$ $f_b \in O(f_d)$
$f_b \in O(f_c)$ $\quad$ $f_c \in O(f_d)$ $\quad$ $f_c \in O(f_b)$

Correction : a) Il y a $\sum_{i=1}^{n} \sum_{j=i}^{n} 1 = \sum_{i=1}^{n} n - i = \frac{n(n-1)}{2}$ `saluts` par changement de variable ou calcul brutal.

b) Si on aime le formalisme, on pose $i = 2^j$. Le programme devient `for (`$j = 0$`; `$j <= \lfloor \log_2 n \rfloor$`; `$j++$`) printf("salut");` donc $\lfloor \log_2 n \rfloor + 1$

c) $\sum_{i=1}^{\lfloor \log_2 n \rfloor} 2^i = \frac{2^{\lfloor \log_2 n \rfloor + 1} - 1}{2-1} = \Theta n$ car $\log_2 n \leq \lfloor \log_2 n \rfloor \leq \log_2 n + 1$ et l'expenentielle est croissante.

d) $t(n) = \sum_{j=1}^{n} \lfloor \log_2 j \rfloor + 1$ or $\sum_{j=1}^{n} \log_2 j = \log_2 n!$ donc $\log_2 n! \leq t(n) \leq \log_2 n! + n$;

De plus, pour tout i de $[1..n]$ $i(n-i) \geq n$ et $i \leq n$ donc $\lceil \frac{n}{2} \rceil \log_2 n \leq \log_2 n! \leq n \log_2 n$

Ainsi, $t(n) = \Theta(n \log n)$.

III) Trouver l'élément minimum/maximum dans un tableau

Exercice 5 :

Soit A un tableau de entiers non négatifs.

1. Ecrire la fonction $\textsc{Min}(A)$ qui implémente un algorithme pour trouver le plus petit élément de A.
2. Ecrire la fonction $\textsc{Max}(A)$ qui implémente un algorithme pour trouver le plus grand élément de A.
3. Quel est le coût du code qui suive ? Comment est ce qu'on peut l'améliorer ?

```
1  max ← MAX(A)
2  min ← MIN(A)
```

Correction :

1. La fonction suivante implémente un algorithme pour trouver le plus petit élément de A.

```
MIN(A)
1  min ← A[1]
2  for (i ← 2) to LENGTH(A)
3        do if (A[i] < min)
4              then min ← A[i]
5  return min
```

2. Comme 1 (renverser min et max).
3. On peut l'améliorer en ecrivant une procedure qui trouve min et max au meme temps.

IV) L'élément médian

Soit A un tableau de entiers (tous différents) tel que $\textsc{Length}(A) = n \geq 1$. *L'élément médian de* A, est un élément de A tel qu'il est plus grand que $\lfloor (n-1)/2 \rfloor$ éléments de A et plus petit (ou égal) que $\lceil (n-1)/2 \rceil$ éléments de A.

Exercice 6 :

1. Comment peut on déterminer l'élément médian d'un tableau A en utilisant un algorithme de tri ? Est-ce que c'est la façon la plus efficace ?
2. Décrire un algorithme naïf (qui n'utilise pas d'algorithmes de tri) pour trouver l'élément médian d'un tableau A. Est-ce qu'il peut être amélioré ?

Correction :

1. Vu la définition de "élément médian", il est évident qu'une fois qu'on a trié le tableau A, on le trouve en position $\lceil \textsc{Length}(A)/2 \rceil$. Evidemment, cet algorithme n'est pas le plus efficace parce que on perd du temps à trier le tableau et cette opération n'est pas nécessaire.
2. L'algorithme le plus simple pour trouver l'élément médian d'un tableau, consiste à voir, pour chaque élément x de A, si x est l'élément médian. Pour faire ca, il suffit de compter le nombre d'éléments de A qui sont plus petits que x :

```
IsMedian(A, x)
1  npetits ← 0
2  for (k ← 1) to Length(A)
3        do if (A[k] < x)
4              then npetits ← npetits + 1
5  if npetits = ⌊ (Length(A) − 1)/2 ⌋
6     then return true
7     else  return false
```

Maintenant, on peut utiliser la fonction IsMedian pour implémenter notre algorithme de recherche de l'élément médian.

```
Median(A)
1  medgets0
2  for k ← 1 to Length(A)
3        do if IsMedian(A, A[k])
4              then med ← A[k]
5  return med
```

On peut l'améliorer un peu en considérant que, si à un certain moment $npetits > \lfloor (\text{Length(A)} - 1)/2 \rfloor$, alors x ne peut pas être l'élément médian.