Travaux Dirigés Algorithmique n°3

Cours d'analyse, algorithmique

—Master 2 CCI—

Complexité, fonctions usuelles

Cette séance de Travaux dirigée est consacrée à la notion de complexité et a l'étude du comportement asymptotique des fonctions élémentaires. Dans une deuxième partie, nous étudions la correction et la complexité de plusieurs algorithmes simples. Pour montrer qu'un algorithme est correct, on écrit une propriété P qui est conservée à chaque étape de boucle que l'on appelle **invariant de boucle**.

▸ Exercice 1. (Nombres d'opérations de programmes élémentaires).

Pour chacun des programmes suivants, dire en fonction de n quel est le nombre d'opérations `op` que le programme effectue.

1.
```
pour i de 1 à n faire op
```
2.
```
pour i de 0 à n faire op
```
3.
```
pour i de 1 à n-1 faire op
```
4.
```
pour i de 1 à n faire
      op; op
```
5.
```
pour i de 1 à n faire
    pour j de 1 à n faire op
```
6.
```
pour i de 1 à n faire
    pour j de 1 à n faire
        pour k de 1 à n faire op
```
7.
```
pour i de 1 à n faire
    op; op
    pour j de 1 à n faire
        op
        pour k de 1 à n faire
            op
```
8.
```
pour i de 1 à n faire
    pour j de 1 à i faire
        op
```
9.
```
tant que n >= 0 faire
    op
    n := n / 2
```

▸ Exercice 2. (Ordres de grandeurs).

On dispose d'une machine qui est capable de faire 1000000 opérations par secondes. On considère des algorithmes dont les complexités sont les suivantes : $\log(n), \sqrt{n}, n, 50 * n, n\log(n), n^2, n^3, 2^n$.

1. Dessiner chacune de ces fonctions sur une échelle doublement logarithmique.
2. Quel est pour chaque algorithme, la taille des problèmes que l'on peut résoudre en une seconde ?
3. Même question si l'on dispose d'une machine 1000 fois plus rapide et de 1000s.

▸ Exercice 3. (Manipulation des Ordres de grandeurs).

Dans cet exercice f, g, h sont des fonctions positives.

1. Montrer que si $f \in O(g)$ et $g \in \Theta(h)$ alors $f \in O(h)$.
2. Montrer que si $f \in \Theta(g)$ et $g \in O(h)$ alors $f \in O(h)$.
3. Montrer par un contre-exemple que si $f \in \Theta(g)$ et $g \in O(h)$ alors f n'est pas nécessairement dans $\Theta(h)$.

▸ **Exercice 4. (Somme des entrées d'un tableau).**

On considère l'algorithme suivant :

```
Entrée : un tableau de nombres T de taille n
Sortie : un nombre s

s <- 0
i <- 0
tant que i < n faire
    s <- s + T[i]
    i <- i + 1
retourner s
```

On veut montrer que l'algorithme calcule $\sum_{i=0}^{n-1} \texttt{T}[i]$.

1. On considère une étape de boucle. On note s et s' les valeurs de la variable `s` avant et après l'étape de la boucle i et i' les valeurs de la variable `i` avant et après l'étape de la boucle. Montrer que si $s = \sum_{j=0}^{i-1} \texttt{T}[j]$ alors $s' = \sum_{j=0}^{i'-1} \texttt{T}[j]$.

On appelle l'énoncé « à la fin d'une étape de boucle, les variables `s` et `i` vérifie l'égalité $\texttt{s} = \sum_{j=0}^{\texttt{i}-1} T[j]$ » un **invariant de boucle**.

2. Montrer que avant la boucle l'invariant est vérifié.
3. En déduire que à la fin de la boucle $s = \sum_{j=0}^{n-1} T[j]$.
4. Montrer que la complexité de cet algorithme est $\Theta(n)$. Peut-on faire mieux ?

▸ **Exercice 5. (Calcul du maximum d'un tableau).**

1. En s'inspirant de l'exercice précédent, écrire un algorithme qui prend en paramètre un entier n et un tableau d'entiers T de taille n, et qui retourne le maximum m des entiers du tableau et l'indice k d'une occurence de m dans le tableau.
2. Donner un énoncé qui caractérise le fait que m et k sont correct. Un tel énoncé est appelé une **spécification**. C'est le contrat que l'utilisateur de l'algorithme passe avec le développeur.
3. À l'aide de la spécification, écrire un invariant de boucle et montrer qu'il est correct, c'est-à-dire qu'il est vrai au début de la boucle et qu'il est conservé à chaque étape de boucle.
4. En déduire, que l'algorithme fonctionne.
5. Quel est la complexité si l'on prend comme opérations élémentaires les comparaisons.
6. Quel est la complexité si l'on prend comme opérations élémentaires les affectations.
7. Quel est la complexité si l'on prend comme opérations élémentaires les comparaisons et les affectations.
8. Qu'en pensez-vous ?

▸ **Exercice 6. (Écriture d'un nombre en base 2).**

1. Décrire deux méthodes pour convertir un nombre n en base 2.

On considère l'algorithme suivant.

```
Entrée : un entier n et un tableau T alloué.
Sortie : un tableau T et un entier i

n2  = n
i   = 0
tant que n2 > 0 faire
    T[i] = n2 mod 2
    n2   = n2 / 2
    i    = i + 1
retourner T, i
```

2. Montrer que pour tout i, à la fin de la i-ème étape de boucle
$$n = \texttt{n2} \times 2^i + \sum_{j=0}^{i-1} \texttt{T}[j] \times 2^j \,.$$
3. En déduire que à la fin `T` contient l'écriture de n en base 2 c'est-à-dire que
$$n = \sum_{j=0}^{i-1} \texttt{T}[j] \times 2^j \,.$$

▸ **Exercice 7. (Calcul de a^n).**

1. Écrire un algorithme qui prend en paramètre un nombre a et un entier positif ou nul n et qui retourne a^n.
2. À l'aide d'un invariant de boucle montrer que l'algorithme est correct.
3. Quel est la complexité de cet algorithme.

On considère maintenant l'algorithme suivant

```
Entrée : un nombre a et un entier positif n
Sortie : p

n2  = n
a2i = a
res = 1
tant que n2 > 0 faire
    si n2 mod 2 == 1 alors
        res = res*a2i
    n2 = n2 / 2
    a2i = a2i*a2i
return res
```

4. Quel est le nombre d'étapes de boucle ?
5. Montrer que à la fin de chaque étape de boucle, on a
$$\texttt{a}^{\texttt{n}} = \texttt{res} \times \texttt{a2i}^{\texttt{n2}} \,.$$
6. En déduire que l'algorithme retourne a^i.

▸ **Exercice 8. Calcul de $n!$**

On considère l'algorithme suivant

```
Entrée: n>0
Sortie: n!

res <- 1
n2  <- n
tant que n2>0 faire
   res <- res*n2
   n2  <- n2 - 1
retourner res
```

1. Montrer que l'invariant `res` $= \prod_{i=\texttt{n2}+1}^{n} i$ est vrai au départ de la boucle et conservé ensuite. En déduire que le programme retourne $n! = \prod_{i=1}^{n} i$.
2. Quel est la complexité de cet algorithme ?

▸ **Exercice 9. Calcul de $\lfloor\sqrt{n}\rfloor$**

On considère l'algorithme suivant où toutes les variables contiennent des entiers.

```
Entrée: un entier n>1
Sortie: la racine de n

inf <- 1; sup <- n
tant que inf < sup-1 faire
    mid <- (inf + sup) / 2
    si mid*mid <= n faire
        inf <- mid
    sinon
        sup <- mid
 retourner inf
```

1. Montrer que l'invariant $\texttt{inf}^2 \leq n < \texttt{sup}^2$ est vrai au départ de la boucle et conservé ensuite.
2. En déduire que si le programme termine ; il retourne $\lfloor\sqrt{n}\rfloor$.
3. Montrer que la différence `sup` $-$ `inf` diminue strictement à chaque étape de boucle. En déduire que le programme termine.
4. Quelle est la complexité ?