Université de Rennes

Licence Informatique ISTIC
Méthodes Algorithmiques (MALG) *2023-2024*

Ordres de grandeur et complexité des algorithmes

Cette séance de travaux dirigés est consacrée à l'étude du comportement asymptotique des fonctions élémentaires, et à la correction et la complexité de plusieurs algorithmes simples. Pour montrer qu'un algorithme est correct, on écrit une propriété 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 et la valeur des variables d'itération à la fin du programme.

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 (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 $\Theta(h)$.

4. Pour chacune de ces propositions, justifier si $f = O(g)$ ou $f = \Omega(g)$ ou les deux (c'est à dire $f = \Theta(g)$).

	$f(n)$	$g(n)$
1.	$n - 100$	$n - 200$
2.	$log_a(n)$	$log_b(n)$
3.	n^2	n^3
4.	$n^{0.1}$	$(log\ n)^{10}$
5.	$n2^n$	3^n
6.	2^n	2^{n+1}
7.	$n!$	2^n

Invariants de boucle

Exercice 3 (Somme des entrées d'un tableau)
On considère l'algorithme suivant :

```
Entrée : un tableau de nombres A[0..n-1] de taille n
Sortie : un nombre s
1. s = 0
2. pour i = 0 à n-1 faire
3.    s = s + A[i]
4. retourner s
```

On veut montrer que l'algorithme calcule $\sum_{i=0}^{n-1} \mathtt{A}[i]$, c'est à dire la somme des éléments du tableau.

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, et i et i' les valeurs de la variable `i` avant et après l'étape de la boucle. Montrer que si $s = \sum_{j=0}^{i-1} \mathtt{A}[j]$ alors $s' = \sum_{j=0}^{i'-1} \mathtt{A}[j]$.

2. Ecrire un invariant de boucle montrant ce que contient s après chaque étape i.

3. Montrer qu'avant la boucle l'invariant est vérifié.

4. En déduire que à la fin de la boucle $s = \sum_{j=0}^{n-1} \mathtt{A}[j]$.

5. Montrer que la complexité de cet algorithme est $\Theta(n)$.

Exercice 4 (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 A[0..n-1] alloué.
Sortie : un tableau A et un entier i
1. m  = n
2. i = 0
3. tant que m > 0 faire
4.    A[i] = m mod 2
5.    m   = m div 2
6.    i    = i + 1
7. retourner A, i
```

2. Montrer que pour tout i, à la fin de la i-ème étape de boucle $n = m \times 2^i + \sum_{j=0}^{i-1} \mathtt{A}[j] \times 2^j$.
3. Montrer que l'algorithme termine.
4. En déduire qu'à la fin `A` contient l'écriture de n en base 2 càd. $n = \sum_{j=0}^{i-1} \mathtt{A}[j] \times 2^j$.

Exercice 5 (Calcul du maximum d'un tableau)

1. En s'inspirant de l'Exercice 3, écrire un algorithme qui prend en paramètre un entier n et un tableau d'entiers `A` 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 corrects. Un tel énoncé est appelé une **spécification**[1].
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. Quelle est la complexité si l'on prend comme opération élémentaire la comparaison.
6. Quelle est la complexité si l'on prend comme opération élémentaire l'affectation.
7. Quelle est la complexité si l'on prend comme opérations élémentaires la comparaison et l'affectation.

[1] C'est le contrat que l'utilisateur de l'algorithme passe avec le développeur.