Master 1 Informatique — Ducci 2
Mercredi 12 janvier 2022

Enseignants :
Oussama Messai / Serge Miguet
Université Lumière Lyon 2

Examen de Complexité

Durée : 1h30

Documents autorisés. Connexion à Internet interdite (smartphones et ordinateurs interdits)
Justifiez précisément chacune de vos réponses !

I. Analyse d'algorithme

Un programmeur débutant écrit l'algorithme suivant, pour vérifier si deux éléments d'un tableau A à n éléments sont égaux :

```
fonction DeuxEgaux(A[1..n])
    drapeau ← faux
    pour i allant de 1 à n faire
        pour j allant de 1 à n faire
            si A[i] = A[j] alors
                drapeau ← vrai
    renvoyer drapeau
```

1. Quelle est la complexité en temps de cet algorithme ?
2. Montrez que le programmeur s'est trompé, et que cet algorithme renvoie toujours `vrai` (sauf si le tableau est vide), même si tous les éléments du tableau sont différents.
3. Apportez une première modification (limitée à 2 à 4 caractères) à ce code pour qu'il ne renvoie `vrai` que si deux éléments du tableau sont égaux. Cette première modification diminue environ de moitié le nombre d'opérations effectuées (donnez le nombre exact de comparaisons effectuées), mais ne change pas sa complexité.
4. En utilisant les résultats du cours, proposez un algorithme pour résoudre le même problème en complexité O(n.log n).
5. En supposant que les valeurs des éléments de A sont des nombres entiers compris entre 1 et n, proposez un algorithme qui résout le même problème en O(n).
6. En supposant que les valeurs des éléments de A sont des nombres entiers compris entre 1 et n-1, proposez un algorithme qui résout le même problème en O(1).

II. Comparaison de temps d'exécution

1. Quelle est la valeur maximale de n pour laquelle un algorithme dont le temps d'exécution est $2n^2$ s'exécute plus vite qu'un algorithme dont le temps d'exécution est $100n$ sur la même machine ?
2. On veut comparer les implémentations du tri par insertion et du tri par fusion sur la même machine. Pour un nombre n d'éléments à trier, le tri par insertion demande $8n^2$ étapes alors que le tri par fusion en demande 64 $n.\log_2 n$. Quelles sont les valeurs de n pour lesquelles le tri par insertion l'emporte sur le tri par fusion (vous pouvez limiter n aux puissances de 2) ?

III. Analyse d'un compteur binaire

On dispose d'un compteur binaire de k bits, représenté à l'aide d'un tableau A[0..k-1]. Le bit de poids faible est stocké dans A[0] et le bit de poids fort est stocké dans A[k-1]. Par exemple, avec k=8 bits, le nombre décimal x=19 qui se décompose en x=16 + 2 + 1 = $2^4 + 2^1 + 2^0$, est représenté à l'aide du tableau binaire suivant :

A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
0	0	0	1	0	0	1	1

On désire analyser la complexité de la fonction Incrémenter( ) dont le pseudo-code est donné ci-dessous[1] :

```
Incrémenter(A)
  i←0
  tant que i<longueur[A] et A[i]=1 faire
    A[i]←0
    i←i+1
  si i<longueur[A] alors
    A[i]←1
```

1. A partir du tableau A donné en introduction, illustrez cinq appels successifs à la fonction Incrémenter(A). Pour chacun de ces cinq appels, représentez le tableau A à la fin de l'exécution, indiquez le nombre d'itérations i_{max} effectuées dans la boucle **tant que**, et indiquez quel est le nombre décimal x représenté dans le tableau A.
2. Quelle est la complexité en fonction de k, dans le meilleur des cas et dans le pire des cas, de l'appel à la fonction Incrémenter(A) ? donnez des exemples de valeurs de x pour lesquelles ces cas favorable et défavorable se produisent effectivement.
3. Quelle est la plus grande valeur n de x, que l'on peut représenter dans le tableau A ?
4. Reprenez la question 2 et montrez que si on exprime la complexité de la fonction Incrémenter( ) en fonction de n plutôt qu'en fonction de k, on obtient O(log n) dans le pire des cas.

On désire maintenant évaluer la complexité T(n) de l'opération Énumérer(n) qui, à partir du nombre x=0, effectue n appels successifs à la fonction Incrémenter(A).

5. Montrez qu'une borne supérieure « pessimiste » permet d'affirmer que T(n) = O(n.log n).

Dans la suite de l'exercice, nous allons montrer que cette borne supérieure peut être affinée, et que l'on pourra établir que T(n) = O(n), ce qui est plus optimiste.

6. Montrer que pour estimer T(n), on peut compter le nombre total d'affectations effectuées à un élément de A[].
7. Complétez le tableau suivant, dans lequel vous représenterez les 17 premiers appels à la fonction Incrémenter( ). t(x) est le nombre d'affectations à un élément de A[] effectuées lors de l'*exécution courante* de la fonction Incrémenter(x), et T(x) est le nombre total de ces affectations effectuées depuis le début de l'exécution de la fonction Énumérer(17), lors des *appels précédents* à la fonction Incrémenter( ).

[1] la primitive *longueur*[A] renvoie en temps constant le nombre d'éléments du tableau A, c'est-à-dire k.

x	A[7] … A[0]	t(x)	T(x)
0	00000000	1	0
1	00000001	2	1
2	00000010	1	3
3	00000011	3	4
4	…	…	7
…	…	…	…
16	…	…	…

8. Vérifiez expérimentalement que T(x) semble être majoré par une fonction linéaire de x.
9. Pour quelles itérations le bit A[0] change-t-il de valeur ? Combien de fois cela se produit-il pour les n itérations ?
10. Pour quelles itérations le bit A[1] change-t-il de valeur ? Combien de fois cela se produit-il pour les n itérations ?
11. En poursuivant ce raisonnement, montrez que le bit A[j] n'est affecté qu'une fois toutes les 2^j itérations.
12. Montrez que le nombre total d'affectations à un bit de A[] dans la procédure Énumérer(n) est égal à :

$$\mathrm{T(n)} = \sum_{j=0}^{k-1} \frac{n}{2^j}$$

13. En déduire que $\mathrm{T(n)} \leq 2n$.
14. En supposant que l'ensemble des valeurs que peut prendre x sont équiprobables, donnez la complexité en moyenne de la fonction Incrémenter(x).