Université de Mons–Hainaut
FS/1/5684 – Algorithmique
Examen de première session – Partie pratique

Le 23 janvier 2009

Consignes

- Pour cette partie, vous avez le droit de consulter vos notes et tout ouvrage qui vous semble utile.
- Cette partie de l'examen dure 1 heure 45 minutes.
- Veillez à bien justifier vos réponses. Une réponse mal justifiée, même correcte, ne permet pas d'obtenir le maximum des points.
- Quand vous indiquez une complexité, veillez à bien expliquer ce que sont les parlaètres qui apparaissent dans le $\mathcal{O}$. Par exemple, $\mathcal{O}(n^2)$ n'a aucun sens si n n'apparaît pas dans l'algorithme ou dans la définition de la structure qui est traitée…

Question 1 – 4 points

Considérons une image de ℓ *pixels* de large, h *pixels* de hauteur, et dont tous les *pixels* ont une couleur représentée par un nombre compris entre 1 et 256. Une telle image peut être stockée en mémoire dans un tableau de dimensions $\ell \times h$, tel que $I[i][j]$ contient le numéro de la couleur affectée au *pixel* à la i^{e} ligne, j^{e} colonne. On voit dès lors qu'on peut appliquer des filtres (tels qu'on en trouve dans les logiciels de manipulation d'images) à cette image en calculant un nouveau tableau J à partir de I, où J représente l'image obtenue après application du filtre.

L'algorithme ci-dessous met ce principe en œuvre pour appliquer à une image représentée par un tableau I un filtre de « flou ». Chaque *pixel* de J est obtenu en prenant la moyenne des valeurs du *pixel* correspondant dans I et des 8 *pixels* qui l'entourent directement. Autrement dit, $J[i][j]$ vaut la moyenne de $I[i-1][j-1], I[i-1][j], I[i-1][j+1], I[i][j-1], I[i][j], I[i][j+1], I[i+1][j-1], I[i+1][j], I[i+1][j+1]$.

```
début
    Entier i, j, k1, k2, somme ;
    i := 2 ;
    tant que i ≤ h − 1 faire
        j := 2 ;
        tant que j ≤ ℓ − 1 faire
            somme := 0 ; k1 = i − 1 ;
            tant que k1 ≤ i + 1 faire
                k2 = j − 1 ;
                tant que k2 ≤ j + 1 faire
                    somme := somme + I[k1][k2] ;
                    k2 := k2 + 1 ;
                k1 := k1 + 1 ;
            J[i][j] := somme/9 ;
            j := j + 1 ;
        i := i + 1 ;
fin
```

On vous demande de donner la complexité (en terme de $\mathcal{O}$) de cette fonction, en la justifiant de la manière la plus précise possible. En particulier, votre argumentation devra détailler chacune des boucles.

Remarque : par souci de facilité, on ne traite pas les éléments aux « bords » de l'image (c'est-à-dire les élements dans les premières et dernières lignes et colonnes de la matrice).

Correction

Cette fonction est en $\mathcal{O}(\ell \times h)$. Les deux boucles extérieures s'exécutent respectivement $h-2$ et $\ell-2$ fois, et ne contiennent, outre les deux boucles intérieures, que des opérations en temps constant (initialisations, tests, *etc*). La complexité est donc en $\mathcal{O}$ de $(\ell-2)\times(h-2)$ *fois* la complexité des deux boucles intérieures. Or, ces deux boucles ont pour effet de calculer la moyenne d'un nombre constant de *pixels* (en l'occurrence 9). Elles sont donc en $\mathcal{O}(1)$. Plus précisément, la boucle la plus intérieure effectue toujours 3 itérations et ne contient que des opérations constantes. Elle est donc en $\mathcal{O}(3\times 1)=\mathcal{O}(3)=\mathcal{O}(1)$. La boucle qui la contient directement n'effectue elle aussi que 3 itérations et ne contient des opérations constantes (y compris la boucle la plus intérieure). Cette boucle est donc en $\mathcal{O}(1)$ également.
On obtient donc une complexité de $\mathcal{O}\big((\ell-2)\times(h-2)\times 1\big)=\mathcal{O}(\ell\times h-2\times(h+\ell)+4)=\mathcal{O}(\ell\times h)$

Erreurs fréquentes La majorité des étudiants ont trouvé la bonne complexité, mais la justification contenait parfois des erreurs. La plus fréquente consiste à justifier la complexité de la façon suivante : 1) les deux boucles les plus intérieures sont en $\mathcal{O}(1)$, 2) la boucle sur l'indice j est en $\mathcal{O}(\ell)$ et 3) *la boucle la plus extérieure est en* $\mathcal{O}(h)$. Donc la complexité est en $\mathcal{O}(h)\times\mathcal{O}(\ell)=\mathcal{O}(h\times\ell)$. L'affirmation 3) est fausse car elle est en contradiction avec la réponse qui dit que l'algorithme est en $\mathcal{O}(h\times\ell)$. En effet, si la boucle la plus extérieure était en $\mathcal{O}(h)$, alors l'algorithme lui-même serait en $\mathcal{O}(h)$, puisqu'il est composé de cette boucle et d'initialisations constantes. Dans ce cas, on applique la règle de la séquence, et on a une complexité totale de $\mathcal{O}(h+1)=\mathcal{O}(h)$. L'erreur provient de la confusion entre *le nombre d'itérations* de la boucle et sa *complexité*. Dans ce cas, le *nombre d'itérations* est de $h-2$ est il est donc correct de dire que *le nombre d'itérations de la boucle est en* $\mathcal{O}(h)$. Par contre, la *complexité* d'une boucle qui s'exécute $\mathcal{O}(h)$ fois est égale à $\mathcal{O}(h)$ *fois* la complexité d'une itération (en l'occurrence $\mathcal{O}(\ell)$). La boucle la plus extérieure est donc bien en $\mathcal{O}(h\times\ell)$.

Question 2 – 2 points

Démontrez que $f(x) = x^2 + 4$ est en $\mathcal{O}(x^3 + x^2 + 2)$.

Correction

D'après la définition, $f(x)$ est en $\mathcal{O}(g(x))$ ssi il existe k et x_0 tels que, pour tout $x \geq x_0$: $f(x) \leq k \times g(x)$. Donc il faut trouver k et x_0 tels que :

$$\begin{aligned} \forall x \geq x_0 \quad &: \quad x^2 + 4 \leq kx^3 + kx^2 + 2k \\ &\Leftrightarrow \\ \forall x \geq x_0 \quad &: \quad x^3 + (k-1)x^2 + 2k - 4 \geq 0 \end{aligned}$$

Si on choisit $k = 1$ (il nous reste alors à fixer x_0), on obtient :

$$\begin{aligned} \forall x \geq x_0 \quad &: \quad x^3 - 2 \geq 0 \\ &\Leftrightarrow \\ \forall x \geq x_0 \quad &: \quad x^3 \geq 2 \end{aligned}$$

Ce qui est vrai pour pour tout $x_0 \geq \sqrt[3]{2}$. On peut donc prendre $x_0 = 2$, par exemple :

$$\forall x \geq 2 \quad : \quad x^3 \geq 2$$

D'autres valeurs de x_0 et k étaient bien entendu possibles.

Erreurs fréquentes Plusieurs étudiants ont identifié des valeurs correctes de x_0 et k mais les ont mal utilisées dans la démonstration. Rappelons que, dans la définition, x_0 et k sont quantifiées existentiellement ($\exists x_0, k \ldots$), alors que x est quantifiée universellement ($\forall x \ldots$). Donc, pour montrer que la définition est satisfaite, il suffit de trouver un x_0 et un k qui rendent l'inégalité vraie, et ce, *pour toute valeur de* x. Beaucoup d'étudiants ont confondu x et x_0, et ont remplacé dans l'égalité la variable x par la valeur choisie pour x_0. Dans ce cas, on ne respecte pas la définition, car on montre qu'il existe k, x_0 et x qui satisfont l'inégalité ; mais cela ne signifie pas que l'inégalité est satisfaire pour *toute valeur* de x.
Comme exemple simple, on peut considérer les affirmations $\forall x : x \geq 2$ et $\exists x : x \geq 2$. Si on fixe la valeur de x à 4, par exemple, l'inégalité est vraie dans les deux cas : $4 \geq 2$. Mais, ce faisant, on n'a démontré que $\exists x : x \geq 2$, car on a réussi à identifier une valeur de x plus grande que 2, mais on ne sait pas si toutes les valeurs possibles de x sont plus grandes que 2. Et en effet, la première affirmation est fausse : 0, par exemple, n'est pas plus grand que 2.

Question 3 – 4 points

Donnez un algorithme *récursif* qui reçoit deux listes triées et sans répétition L_1 et L_2 et qui renvoie *vrai* ssi toutes les valeurs contenues dans L_2 sont également présentes dans L_1.

Correction

Dans la définition récursive, une liste est soit un élément en tête de liste, suivi d'une liste ; soit la liste vide. Comme il y a deux listes, on a donc quatre possibilités, qu'on peut résumer ainsi :

1. Si L_2 est vide, elle est trivialement incluse dans L_1, que celle-ci soit vide ou non. L'algorithme doit donc retourner *vrai*.
2. Si L_2 n'est pas vide et que L_1 est vide, l'inclusion n'est pas possible. L'algorithme doit donc retourner *faux*.
3. Si aucune des listes n'est vide, appelons i_1 l'élément en tête de L_1 et i_2 l'élément en tête de L_2. Appelons aussi L'_1 et L'_2 les listes (potentiellement vides) qui suivent respectivement i_1 et i_2. Nous avons essentiellement deux possibilités :
   (a) Si $i_1 = i_2$, on a trouvé un élément supplémentaire de L_2 qui est présent dans L_1. Comme les listes sont triées et sans répétition, on est sûr que $i_2 = i_1$ n'apparaitra plus dans L_2. Il faut maintenant tester l'inclusion de L'_2 dans L'_1.
   (b) Si $i_1 \neq i_2$, l'élément en tête de L_1 ne nous permet pas de progresser dans L_2. On doit maintenant tester l'inclusion de L_2 (car i_2 n'a pas encore été trouvé) dans L'_1 (comme les listes sont triées, i_1 n'apparaîtra pas dans L_2).

   Cette dernière étape peut être améliorée en tenant compte du fait que les listes sont triées. En effet, si la $i_2 < i_1$, on également que i_2 ne sera pas présent dans L'_1, puisque les listes sont triées de façon croissante. On a dès lors trois cas :
   (a) Si $i_1 = i_2$, on procède comme dit au-dessus.
   (b) Si $i_1 > i_2$, on renvoie faux.
   (c) Sinon ($i_1 < i_2$) on teste l'inclusion de L_2 dans L'_1

L'algorithme suivant réalise cela :

Inclusion(Elem * L_1, Elem * L_2)
début
  si $L_2 =$ **NIL alors retourner** *vrai* ;
  sinon si $L_1 =$ **NIL** *ou* $L_1.info > L_2.info$ **alors retourner** *faux* ;
  sinon si $L_1.info = L_2.info$ **alors retourner** ***Inclusion***$(L_1.next, L_2.next)$;
  sinon retourner ***Inclusion***$(L_1.next, L_2)$;
fin

Erreurs fréquentes

1. À défaut d'hypothèse dans l'énoncé, il fallait considérer tous les cas possibles. Chacune des deux listes pouvait donc être vide ou non. Au total, cela fait 4 cas, donc trois (ceux où au moins une des deux listes est vide) forment un cas de base. Plusieurs étudiants en ont oublié au moins un.
2. L'énoncé demandait bien un algorithme récursif, et plusieurs étudiants ont traité certains cas à l'aide de boucles. En l'occurrence, quand L_1.info était $< L_2$.info, plusieurs étudiants ont utilisé une boucle **tant que** pour faire avance la tête de L_1 jusqu'à obtenir $L_1.\text{info} = L_2.\text{info}$. En se contentant d'appels récursifs, on obtient un algorithme beaucoup plus compact et élégant, comme le montre la solution.
3. Enfin, il faut à tout prix éviter de modifier les listes passées en paramètre (d'autant plus que ce n'était absolument pas nécessaire). Il est hors de question de *supprimer* des éléments sous prétexte de faire un appel récursif.
