Corrigé

Examen d'algorithmique

EPITA ING1 2017 S1; A. DURET-LUTZ

Durée : 1h30

Janvier 2015

1 Dénombrement (6 pts)

Donnez vos réponses en fonction de N. (On souhaite des formules précises, pas des classes de complexité.)

1. **(2 pts)** Combien de fois le programme ci-dessous affiche-t-il `"x"` ?

```
for (int i = 2 * N; i >= 0; --i)
  for (int j = 0; j < i; ++j)
    puts("x");
```

Réponse :

$$\sum_{i=0}^{2N}\sum_{j=0}^{i-1} 1 = \sum_{i=0}^{2N} i = \frac{2N(2N+1)}{2} = N(2N+1)$$

2. **(2 pts)** Et celui-ci ?

```
for (int i = 3; i < N; ++i)
  for (int j = 1; j < i; ++j)
    puts("x");
```

Réponse :

Si $N \leq 3$ rien n'est affiché.

Si $N > 3$:

$$\sum_{i=3}^{N-1}\sum_{j=1}^{i-1} 1 = \sum_{i=3}^{N-1} (i-1) = \frac{(N-2+2)(N-3)}{2} = \frac{N(N-3)}{2}$$

3. **(2 pts)** Et celui-ci ?

```
for (int i = 0; i <= N; ++i)
  {
    puts("x");
    for (int j = 0; j < i; ++j)
      puts("x");
  }
```

Réponse :

$$\sum_{i=0}^{N}\left(1+\sum_{j=0}^{i-1}1\right)=\sum_{i=0}^{N}(1+i)=\frac{(N+1)(N+2)}{2}$$

2 Gaussons-nous ! (4 pts)

L'algorithme du *pivot de Gauss*, aussi appelé *élimination de Gauss-Jordan*, peut s'écrire comme suit. Notez qu'il n'est pas nécessaire de savoir à quoi sert cet algorithme pour répondre aux questions.

```
1   // input : A[1..n][1..m], a matrix of n rows and m columns
2   for k ← 1 to m do :
3       // Find pivot for column k
4       p ← k
5       for i ← k + 1 to m :
6           if abs(A[i][k]) > abs(A[p,k]) :
7               p ← i
8       if A[p][k] = 0 :
9           error "Matrix is singular"
10      // swap rows k and p
11      A[k] ↔ A[p]
12      // Update all elements below and the pivot
13      for i ← k + 1 to m do :
14          for j ← k to n do :
15              A[i][j] ← A[i][j] − A[k][j] × (A[i][k]/A[k][k])
16          // fill lower triangular matrix with zeroes
17          A[i][k] ← 0
```

Dans les deux questions qui suivent, on suppose que la matrice est carrée ($n = m$) et inversible (la ligne 9 n'est jamais exécutée).

1. **(2 pts)** Donnez une formule (simplifiée) indiquant exactement combien de multiplications scalaires sont effectuées par cet algorithme (ligne 15) en fonction de n.

Réponse :

La ligne 15 effectue une multiplication scalaire chaque fois qu'elle est exécutée. (Certains ont décidé de compter la division comme une multiplication. Je n'ai rien contre : il suffit de multiplier tous les calculs qui suivent par deux.) Le nombre d'exécutions de cette ligne dépend uniquement des trois boucles imbriquées contrôlées par les lignes 2, 13, et 14.
Si l'on colle au code (et que l'on suppose $n = m$ comme le demande l'énoncé), le nombre de multiplications serait donc

$$\sum_{k=1}^{n}\sum_{i=k+1}^{n}\sum_{j=k}^{n}1$$

mais attention à la boucle sur i : lorsque $k = n$ la boucle i, qui doit aller de $n+1$ à n en croissant, ne doit faire aucune itération. On peut donc limiter la première somme à $1...n-1$.

$$\begin{aligned}\sum_{k=1}^{n}\sum_{i=k+1}^{n}\sum_{j=k}^{n}1 &= \sum_{k=1}^{n-1}\sum_{i=k+1}^{n}\sum_{j=k}^{n}1 = \sum_{k=1}^{n-1}\sum_{i=k+1}^{n}(n+1-k) = \sum_{k=1}^{n-1}(n-k)(n+1-k)\\ &= \sum_{k=1}^{n-1}\left(n(n+1)-(2n+1)k+k^2\right)\end{aligned}$$

Beaucoup trop d'entre vous ont sommé ces termes comme s'ils étaient linéaires : ce n'est pourtant pas le cas à cause du k^2.

$$\begin{aligned}&= (n-1)n(n+1) - (2n+1)\sum_{k=1}^{n-1}k + \sum_{k=1}^{n-1}k^2\\ &= (n-1)n(n+1) - (2n+1)\frac{n(n-1)}{2} + \frac{(n-1)n(2n-1)}{6}\\ &= \frac{(n-1)n}{6}\left(6(n+1) - 3(2n+1) + (2n-1)\right)\\ &= \frac{(n-1)n(2n+2)}{6} = \frac{(n-1)n(n+1)}{3}\end{aligned}$$

On peut aussi écrire cela $\frac{n(n^2-1)}{3}$.
(Au passage vous pouvez en profiter pour vérifier que si $n = 1$, l'algorithme n'effectue aucune multiplication scalaire.)

2. **(2 pts)** Quelle est la complexité de cet algorithme en fonction de n (soyez précis dans votre choix de $\Theta(\ldots)$ ou $O(\ldots)$).

Réponse :

D'après la question précédente, on fait au moins $\frac{n(n^2-1)}{3}$ opération, et vu l'algorithme le nombre total d'opération ne peut pas être pire que proportionnel à cette fonction.
La complexité est donc $\Theta(n^3)$.

3 Recursion (8 pts)

On considère la fonction suivante qui retourne une liste contenant tous les anagrammes de la chaîne s :

```
1  ANAGRAMS(s) :
2    if s = "" :
3      return [s]
4    res ← []    /* liste vide */
5    for w in ANAGRAMS(s[1 :]) :
6      for pos in {0, 1, ..., |w|} :
7        res.insert(w[: pos] + s[0] + w[pos :])
8    return res
```

$T(0)$	$T(n), n > 0$
$\Theta(1)$	$\Theta(1)$
$\Theta(1)$	0
	$\Theta(1)$
	$T(n-1) + \Theta((n-1)!)$
	$\Theta(n) \times (n-1)!$
	$\Theta(n) \times n!$

Si $w =$ `"abcde"`, la notation $w[: 3]$ désigne le préfixe de w ne contenant que les lettres 0, 1 et 2, soit $w[: 3] =$ `"abc"` ; tandis que $w[3 :]$ désigne le suffixe de w à partir de la lettre 3, ici $w[3 :] =$ `"de"`. En particulier, $w[: 0] = w[5 :] =$ "".

A titre d'exemple, ANAGRAM(`"foo"`) retourne [`"foo"`, `"ofo"`, `"oof"`, `"foo"`, `"ofo"`, `"oof"`]. Les doublons viennent du fait que la chaîne `"foo"` contient deux `'o'` qui peuvent être permutés : l'algorithme ne fait aucun effort pour éviter cela.

Dans tout cet exercice, on note n la taille de la chaîne s passée à ANAGRAM.

1. **(2 pts)** Pour une chaîne de taille n, quel est le nombre d'anagrammes retournés par ANAGRAMS ? (Donnez une formule précise, en fonction de n.)

Réponse :
Il y a $n!$ façons d'ordonner n lettres si l'on ne prend pas en compte les doublons.

2. **(2 pts)** Si ANAGRAMS est appelé avec une chaîne de taille $n > 0$, combien de fois la ligne 7 est-elle exécutée lors de cet appel (c'est-à-dire sans compter les exécutions de la ligne 7 lors des appels récursifs effectués ligne 5) ?

Réponse :
ANAGRAMS($s[: 1]$) retourne $(n-1)!$ mots de taille $n-1$. La ligne 8 est alors exécutée n fois par mot. La ligne 7 est donc finalement exécutée $n!$ fois pour un appel d'ANAGRAMS.
(Mais n'est-ce pas logique ? Si le nombre d'anagrames de taille n est $n!$, il faut bien faire $n!$ appels à insert() pour remplir la liste *res*.)

3. **(2 pts)** Justifiez, **en annotant les lignes de algorithme** par leurs complexités respectives, que la complexité $T(n)$ de ANAGRAMS satisfait l'équation suivante :

$$T(n) = \begin{cases} \Theta(1) & \text{si } n = 0 \\ \Theta(n) \times n! + T(n-1) & \text{si } n > 0 \end{cases}$$

4. **(2 pts)** Quelle est la solution de l'équation ci-dessus ? (Notez que $i! \times i = i! \times (i+1) - i! = (i+1)! - i!$. Fascinant, non ?)

Réponse :

Posons $T(n) = cn \times n! + T(n-1)$. En remplaçant $T(n-1)$ par sa définition, on obtient $T(n) = cn \times n! + c(n-1) \times (n-1)! + T(n-2)$. Si l'on continue ainsi jusqu'à $T(0)$ on a

$$T(n) = \sum_{i=1}^{n}(ci \times i!) + T(0) = c\sum_{i=1}^{n}(i \times i!) + \Theta(1)$$

Il nous faut donc calculer la sommes de $i \times i!$, mais puisque $i \times i! = (i+1)! - i!$ on a

$$\begin{aligned} T(n) &= c\left(\sum_{i=1}^{n}(i+1)! - \sum_{i=1} i!\right) + \Theta(1) \\ &= c\left(\sum_{i=2}^{n+1} i! - \sum_{i=1} i!\right) + \Theta(1) \\ &= c((n+1)! - 1!) + \Theta(1) \\ &= \Theta((n+1)!) \end{aligned}$$

Attention que $\Theta((n+1)!)$ n'est pas la même classe que $\Theta(n!)$ (il y a un facteur $n+1$ entre les deux, ce n'est pas une constante).

4 Complexité récursive (6 pts)

Théorème général. Pour une récurrence du type $T(n) = aT(n/b + \mathrm{O}(1)) + f(n)$ avec $a \geq 1,\ b > 1$:

— si $f(n) = \mathrm{O}(n^{(\log_b a)-\varepsilon})$ pour un $\varepsilon > 0$, alors $T(n) = \Theta(n^{\log_b a})$;
— si $f(n) = \Theta(n^{\log_b a})$, alors $T(n) = \Theta(n^{\log_b a} \log n)$;
— si $f(n) = \Omega(n^{(\log_b a)+\varepsilon})$ pour un $\varepsilon > 0$, **et** de plus $af(n/b) \leq cf(n)$ pour un $c < 1$ et toutes les grandes valeurs de n, alors $T(n) = \Theta(f(n))$.

Pour chacune des définitions récursive de complexité qui suivent, donnez la classe de complexité à laquelle elles appartiennent.

1. $T(n) = 3T(n/3) + \Theta(1)$

Réponse :

$T(n) = \Theta(n)$ (1er cas du théorème)

2. $T(n) = 2T(n/3) + \Theta(n)$

Réponse :

$T(n) = \Theta(n)$ (3e cas du théorème)

3. $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(1)$

Réponse :

$T(n) = \Theta(n)$ (1er cas du théorème)

The End