

124

Nom : BANET
Prénom : Loïc

23
26

Examen d'algorithmique

EPITA ING1 2018 S1; A. DURET-LUTZ

Durée : 2h

Janvier 2016

Consignes

- Cette épreuve se déroule **sans document et sans calculatrice**.
- Répondez sur le sujet dans les cadres prévus à cet effet.
- Soignez votre écriture, et ne donnez pas plus de détails que ceux nécessaires à vous justifier.
- Il y a **6 pages** d'énoncé, assurez-vous de l'avoir en entier.
- Le barème, indicatif, correspond à une note sur **26** points.

1 Dénombrement (4 pts)

4

Donnez des formules précises en fonction de N , pas des classes de complexité.

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

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

Réponse :

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

2. (2 pts) Et celui-ci ?

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

Réponse :

$$\sum_{i=0}^N 1 + \sum_{i=0}^N i = \sum_{i=0}^N i + (i+1) = \sum_{i=0}^N (i+2) = \frac{(N+1)(N+2+2)}{2} = \frac{(N+1)(N+4)}{2}$$

2 Complexité récursive (6 pts)

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

- 6
- si $f(n) = O(n^{(\log_b a)-\varepsilon})$ 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. $T(n) = 2T(n/2) + \Theta(n)$

Réponse: $a = 2$ et $b = 2$ donc $n^{\log_b a} = n$

en utilisant le théorème général (seconde condition)

Nous obtiendrons que $T(n) = \Theta(n \log n)$

car $f(n) = \Theta(n^{\log_b a}) = \Theta(n)$

2. $T(n) = 2T(n/4) + \Theta(\sqrt{n})$

Réponse: $a = 2$ $b = 4$ donc $\log_b a < 1$ or $\log_4 2 = 0,5$

Donc $n^{\log_b a} = \sqrt{n} = n^{0,5}$

Donc $f(n) = \Theta(n^{\log_b a}) = \Theta(\sqrt{n})$

En appliquant le théorème général (seconde condition)

Nous obtiendrons $T(n) = \Theta(n^{0,5} \log n) = \Theta(\sqrt{n} \log n)$

3. $T(n) = 4T(\lfloor n/2 \rfloor) + \Theta(1)$

Réponse: $a = 4$ $b = 2$ $\log_b a = \log_2 4 = 2$

Si n pair $T(n) = 4T(n/2) + \Theta(1)$

Si n impair $T(n) = 4T((n-1)/2) + \Theta(1) = 4T(\frac{n}{2} - \underbrace{\frac{1}{2}}_{\Theta(1)}) + \Theta(1)$

or $f(n) \in \begin{cases} O(n^{2-\varepsilon}), \\ \Theta(n^2), \\ \Omega(n^{2+\varepsilon}) \end{cases}$

Selon le théorème général
(première condition)
on obtient donc $T(n) = \Theta(n^2)$

3 Courbe de Hilbert (5 pts)

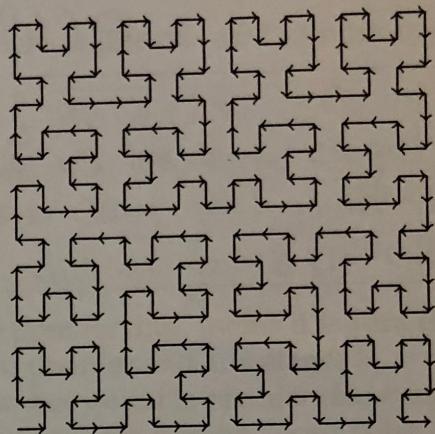
On considère la fonction recursive suivante pour dessiner une *courbe de Hilbert*. On suppose que chaque appel à `line()` trace un segment dans la direction indiquée, en temps constant.

```

void
hilbert(int n, int dir)
{
    if (n == 0)          O(1)
        return;          O(1)
    --n;                O(1)
    if (dir == UP)       O(1)
    {
        hilbert(n, RIGHT); T(n-1)
        line(UP);          O(1)
        hilbert(n, UP);   T(n-1)
        line(RIGHT);      O(1)
        hilbert(n, UP);   T(n-1)
        line(DOWN);       O(1)
        hilbert(n, LEFT); T(n-1)
    }                   O(1)+4T(n-1)
    else if (dir == LEFT)
    {
        hilbert(n, DOWN);
        line(LEFT);
        hilbert(n, LEFT);
        line(DOWN);
        hilbert(n, LEFT);
        line(RIGHT);
        hilbert(n, UP);
    }
}
}

```

Par exemple `hilbert(4, UP)` dessine la courbe suivante :



- Si j'exécute `hilbert(n, UP)` pour un n donné, combien de fois la fonction `hilbert` sera-t-elle appellée ? Donnez votre réponse en fonction de n . (L'appel initial doit être compté.)

Réponse : que ce soit `RIGHT`, `LEFT`, `UP` ou `DOWN`, un appel de fonction `hilbert(n)` appelle 4 fois `hilbert(n-1)`

Donc le nombre d'appel total sera (en posant $hilber = T$ pour l'application)

$$N = 1 + 4T(n-1) = 1 + 4^2 \cdot T(n-2) = \dots = 1 + 4^{n-1} T(1) = 1 + 4^n T(0) = 1 + 4^n$$

Le nombre d'appel total de `hilbert` est donc $1 + 4^n$

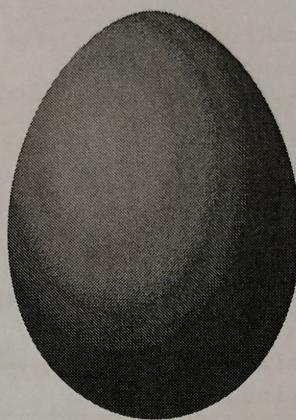
- Si j'exécute `hilbert(n, UP)` pour un n donné, combien de fois la fonction `line()` sera-t-elle appellée ? Donnez votre réponse en fonction de n .

Réponse : La fonction `line` est appellée 3 fois par appel à `hilbert`. Nous avons montré ci-dessus que le nombre d'appels à `hilbert` de `hilbert(n, up)` est $1 + 4^n$.

$$\text{Donc le nombre d'appels à } line \text{ est } 3 \times (1 + 4^n) = 3 + 3 \cdot 4^n$$

3. (1 pt) Quelle est complexité de $\text{hilbert}(n, \text{UP})$ en fonction de n ?

Réponse :
$$\begin{aligned} T(n) &= 50(1) + 4 \cdot T(n-1) + \Theta(1) = 4T(n-1) + \Theta(1) \\ &= 4(T(n-2) + \Theta(1)) + \Theta(1) \\ &= 4^n + (3 + 3 \cdot 4^{n-1})\Theta(1) + \Theta(1) \\ &= \Theta(4^n) \end{aligned}$$



4 Omelette (11 points)

Vous êtes au pied d'un grand immeuble avec en poche plusieurs œufs identiques et difficiles à casser (poules élevées en plein air dans une cimenterie). Votre but est de trouver le plus haut étage duquel on puisse lâcher un œuf sans qu'il ne se casse. On cherche une stratégie pour trouver cet étage en minimisant le nombre de lâchers d'essais. (Attention, le but n'est pas de casser le moins d'œufs possible, mais bien d'en *lâcher* le moins possible.)

On note H le nombre de niveaux de l'immeuble, et N le nombre d'œufs en poche. On numérottera les étages de 1 (rez-de-chaussée) à H (dernier étage).

On fait plusieurs hypothèses :

- Un œuf qui survit à une chute peut être utilisé à nouveau.
- Tous les œufs du lot se comportent de la même façon.
- Si un œuf lâché de l'étage i se casse, alors tout œuf lâché d'un étage $j \geq i$ se cassera.
- Si un œuf lâché de l'étage i ne se casse pas, alors aucun œuf lâché d'un étage $j \leq i$ ne se cassera.
- Il n'est pas exclu que les œufs puissent se casser dès le rez-de-chaussée, et il n'est pas exclu non plus que les œufs puissent survivre à une chute du dernier étage.

1 œuf

Si $N = 1$ (un seul œuf en poche) il n'y a qu'un stratégie possible : on lâche l'œuf depuis l'étage 1, puis depuis l'étage 2, puis 3, etc. jusqu'à ce que soit l'œuf casse, soit il survive à une chute du dernier étage H . Dans le pire des cas, cela demande H lâchers.

2 œufs

Avec $N = 2$ œufs en poche, on peut considérer plusieurs stratégies.

Stratégie 1 : On tente de lâcher le premier œuf tous les \sqrt{H} étages, puis on utilise le second œuf pour trouver l'étage précis avec une recherche linéaire comme précédemment. Par exemple si $H = 36$, on lâche le premier œuf à l'étage 6, puis aux étages 12, 18, etc. jusqu'à ce qu'il casse. Si par exemple il casse à l'étage 24, on utilise le second œuf pour tester les étages 19 à 23 l'un après l'autre.

1. (1pt) Combien de lâchers d'œufs la stratégie 1 demande-t-elle dans le pire des cas ? Donnez une formule en fonction de H .

Réponse : Le pire des cas est si le premier œuf ne se casse pas au dernier étage et si l'étage recherché est le dernier étage. Dans ce cas là, on fera \sqrt{H} lancers du premier œuf puis $\sqrt{H} - 1$ lancers du second œuf. Donc dans le pire des cas on fait $2\sqrt{H} - 1$ lancers.

Stratégie 2 : On fait en sorte que plus on a fait de tentatives avec le premier œuf, moins il faudra en faire avec le second. Par exemple si $H = 36$, on effectue les lâchers du premier œuf aux étages $h_1 = 8, h_2 = 15, h_3 = 21, h_4 = 26, h_5 = 30, h_6 = 33, h_7 = 35, h_8 = 36$, de façon à ce que si le premier œuf casse à l'étage h_i , il ne reste que $h_1 - i$ tests à réaliser avec le second œuf (remarquez comme l'écart entre h_i et h_{i+1} diminue de un à chaque fois que i augmente).

2. (2pt) Donnez une formule qui calcule h_1 en fonction de H .

Réponse : $\sum_{i=1}^{h_1} i = H$ car on remarque que $H = h_8 + h_7 + h_6 + h_5 + h_4 \dots$ et $h_1 - h_1 - 1 = 1 \quad 2 \quad 3 \quad 4 \quad 5 \dots$

$$\text{Donc } \frac{h_1(h_1+1)}{2} = H \Rightarrow h_1^2 + h_1 - 2H = 0 \quad \left| \begin{array}{l} \Delta = 1 + 8H \\ h_{1,2} = \frac{-1 - \sqrt{1+8H}}{2} \end{array} \right.$$

$$h_{1,b} = \frac{-1 + \sqrt{1+8H}}{2} \text{ est solution}$$

Donc la formule de calcul de h_1 en fonction de H est $h_1 = \frac{-1 + \sqrt{1+8H}}{2}$

$$\text{Vérifions avec } H=36 \quad h_1 = \frac{-1 + \sqrt{1+288}}{2} = \frac{-1 + 17}{2} = 8$$

3. (1pt) Combien de lâchers d'œufs la stratégie 2 demande-t-elle dans le pire des cas ? Donnez une formule en fonction de h_1 .

Réponse : Si l'œuf casse au premier lancer, nous aurons h_1 lancers dans le pire des cas. Si l'œuf casse au second lancer nous aurons $h_1 - 2 + 2 = h_1$ lancers. Si l'œuf casse au dernier lancer nous aurons h_1 lancers également. Donc Dans le pire des cas, il y aura h_1 lancers.

Programmation dynamique pour N œufs

On généralise maintenant le problème à N œufs, et on cherche le nombre minimum de lâchers nécessaires pour couvrir tous les cas. C'est-à-dire le nombre de lâchers dont la stratégie optimale a besoin dans son pire cas.

On note $W(n, h)$ le nombre minimum de lâchers qu'il faut faire dans le pire cas lorsqu'on a $n \in [0, N]$ œufs en poche, et encore $h \in [0, H]$ étages consécutifs à tester.

On a alors :

$$\begin{aligned} W(n, 0) &= 0 \\ \forall h > 0, \quad W(0, h) &= \infty \\ \forall n > 0, \forall h > 0, \quad W(n, h) &= 1 + \min_{x \in [1, h]} \max(W(n-1, x-1), W(n, h-x)) \end{aligned}$$

1. (3pts) Justifiez la définition de $W(n, h)$ ci-dessus. (À quoi correspondent les arguments du max ? Pourquoi un max, pourquoi un min ? Que représente x ?)

Réponse : $W(n, 0) = 0$ car lorsque il n'y a pas d'œufs la solution est déjà connue.
 $W(0, h) = \infty$ car sans pouvoir lancer d'œuf on a pas de résultat au problème.

On cherche le nombre minimum de lâchers dans le pire des cas. Le pire des cas est représenté par le max.

Les deux arguments du max représentent les deux cas où l'œuf c'est cassé ou l'œuf est intact après le 1^{er} lâcher.

pour un x , hauteur donnée le max vas explorer tous les cas possibles par récursion en partant du haut et du bas. Simultanément, il donnera donc le pire cas pour une hauteur donnée.

Le min vas choisir le nombre minimum de lâchers parmi tous les cas de toutes

2. (3pts) Completez le tableau suivant avec les valeurs de $W(n, h)$ pour tout $n \in [0, 3]$ et $h \in [0, 10]$.

$h = 0$	1	2	3	4	5	6	7	8	9	10
$n = 0$	0	∞								
$n = 1$	0	1	2	3	4	5	6	7	8	9
$n = 2$	0	1	2	2	3	3	3	4	4	4
$n = 3$	0	1	2	2	3	3	3	3	4	4

Le 1 exprime la nécessité de faire au moins 1 lâcher pour tester.

3. (1pts) Quelle est la complexité du calcul de $W(N, H)$, en fonction de N et H ?

Réponse :

$$W(N, H) = 1 + 2H \cdot W(N)$$

Selon le théorème général $W(N, H) = \Theta(N \log^2 H)$

The End