Nom :

Prénom :

Examen d'algorithmique

EPITA ING1 2016 S1; A. DURET-LUTZ

Durée : 1h30

Janvier 2013

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 cinq pages d'énoncé et une page d'annexe.
— Le barème, indicatif, correspond à une note sur 27.

1 Dénombrement (5 pts)

Donnez vos réponses en fonction de N.

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

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

Réponse :

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

```
for (int i = 1; i <= N; ++i)
  for (int j = 1; j < i; ++j)
    for (int k = N; k > N - 2; --k)
      puts("x");
```

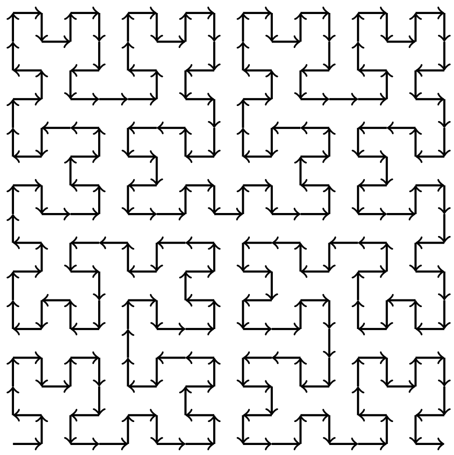
Réponse :

2 Complexité d'une fonction recursive (7 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)
    return;
  --n;
  if (dir == UP)
    {
      hilbert(n, RIGHT);
      line(UP);
      hilbert(n, UP);
      line(RIGHT);
      hilbert(n, UP);
      line(DOWN);
      hilbert(n, LEFT);
    }
  else if (dir == LEFT)
    {
      hilbert(n, DOWN);
      line(LEFT);
      hilbert(n, LEFT);
      line(DOWN);
      hilbert(n, LEFT);
      line(RIGHT);
      hilbert(n, UP);
    }
  else if (dir == RIGHT)
    {
      hilbert(n, UP);
      line(RIGHT);
      hilbert(n, RIGHT);
      line(UP);
      hilbert(n, RIGHT);
      line(LEFT);
      hilbert(n, DOWN);
    }
  else // if (dir == DOWN)
    {
      hilbert(n, LEFT);
      line(DOWN);
      hilbert(n, DOWN);
      line(LEFT);
      hilbert(n, DOWN);
      line(UP);
      hilbert(n, RIGHT);
    }
}
```

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



1. **(3 pts)** Si j'exécute `hilbert(`n`, UP)` pour un n donné, combien de fois la fonction `hilbert` sera-t-elle appelée ? Donnez votre réponse en fonction de n. (L'appel initial doit être compté.)

Réponse :

2. **(2 pts)** Si j'exécute `hilbert(`n`, UP)` pour un n donné, combien de fois la fonction `line()` sera-t-elle appelée ? Donnez votre réponse en fonction de n.

Réponse :

3. **(2 pts)** Quelle est complexité de `hilbert(`n`, UP)` en fonction de n ?

Réponse :

3 Tri postal (8 pts)

On considère une série de codes postaux de 5 chiffres, représentés dans un tableau (0). L'objectif est de trier ce tableau en plusieurs étapes. Dans la première étape (1), nous trions le tableau par rapport au dernier chiffre de chaque code postal. Dans la seconde étape (2), le tableau est trié par rapport à son avant dernier chiffre. Etc. On effectue cinq étapes en triant à chaque fois par rapport au chiffre précédent. Après la cinquième étape (5), on espère que le tableau est trié.

(0)	(1)	(2)	(3)	(4)	(5)
58170	$5817\underline{0}$	$791\underline{0}0$	$75\underline{0}13$	$7\underline{5}013$	$\underline{3}8700$
79100	$7910\underline{0}$	$468\underline{0}0$	$79\underline{1}00$	$6\underline{5}120$	$\underline{4}5800$
87160	$8716\underline{0}$	$458\underline{0}0$	$47\underline{1}20$	$4\underline{5}800$	$\underline{4}6800$
47250	$4725\underline{0}$	$387\underline{0}0$	$65\underline{1}20$	$4\underline{6}800$	$\underline{4}7120$
49530	$4953\underline{0}$	$750\underline{1}3$	$87\underline{1}60$	$4\underline{7}120$	$\underline{4}7250$
75013	$4680\underline{0}$	$471\underline{2}0$	$58\underline{1}70$	$8\underline{7}160$	$\underline{4}9530$
46800	$4712\underline{0}$	$651\underline{2}0$	$47\underline{2}50$	$4\underline{7}250$	$\underline{5}8170$
47120	$4580\underline{0}$	$495\underline{3}0$	$49\underline{5}30$	$5\underline{8}170$	$\underline{6}5120$
45800	$6512\underline{0}$	$472\underline{5}0$	$38\underline{7}00$	$3\underline{8}700$	$\underline{7}5013$
65120	$3870\underline{0}$	$871\underline{6}0$	$46\underline{8}00$	$7\underline{9}100$	$\underline{7}9100$
38700	$7501\underline{3}$	$581\underline{7}0$	$45\underline{8}00$	$4\underline{9}530$	$\underline{8}7160$

1. Quel algorithme puis-je utiliser pour réaliser le tri de chaque étape et être sûr que le tableau sera trié après la cinquième étape ? (Cochez toutes les réponses correctes.)

   ☐ le tri rapide (quick sort)

   ☐ le tri par insertion

   ☐ le tri fusion (merge sort)

   ☐ le tri par sélection

   ☐ le tri par tas (heap sort)

2. Indépendemment du fait que le résultat soit effectivement trié ou non, supposons que ce soit le tri par insertion qui soit utilisé à chaque étape. S'il y a n codes postaux à trier, quelle est la complexité totale du tri (les 5 étapes) ?

Réponse :

3. On considère maintenant un facteur qui cherche à trier une pile d'enveloppes en fonction de leur code postal, cela à l'aide de dix casiers numérotés de 0 à 9. Le facteur prend chaque enveloppe en commençant par le haut de la pile, et la dépose, face vers le bas, dans le casier étiqueté par le dernier numéro du code postal. Une fois que toutes les enveloppes ont été réparties dans les

casiers, ils forme une nouvelle pile en prenant le contenu de chaque casier dans l'ordre (casier 0 en haut, casier 9 en bas). Il a alors une pile d'enveloppes triées par rapport au dernier chiffre du code postal comme après l'étape (1) de l'exemple. Le facteur procède alors à 4 nouveaux tris, comme précédemment, pour trier sa pile par rapport aux 4 autres chiffres (toujours de la droite vers la gauche). Après ces cinq itérations la pile est triée.

Cet algorithme se transpose naturellement sur une machine pour trier un ensemble d'entiers.

Quelle structure de donnée vous semble la plus appropriée pour représenter l'un des 10 casiers.

- ☐ une chaîne de caractères
- ☐ un tableau
- ☐ une liste chaînée
- ☐ un table de hachage
- ☐ un tas

4. Quelle est la complexité de ce tri par casiers pour trier n codes postaux ?

Réponse :

4 Chaîne de multiplication de matrices (7 points)

Soient $A_1, A_2, \dots A_n$, n matrices de dimensions respectives $p_0 \times p_1$, $p_1 \times p_2$, $\dots$, $p_{n-1} \times p_n$.

On souhaite calculer le produit $A_1 \cdot A_{i+1} \cdots A_n$ avec des multiplication matricielles classiques, mais en posant les parenthèses (pour fixer l'ordre d'évaluation des produits) de façon à minimiser le nombre de multiplications scalaires.

Notons $m[i,j]$ le nombre minimum de multiplications scalaires nécessaires pour multiplier $A_i \cdot A_{i+1} \cdots A_j$.
Comme nous l'avons vu en cours, on a :

$$m[i,j] = \begin{cases} 0 & \text{si } i = j \\ \min \left\{ m[i,k] + m[k+1,j] + p_{i-1}p_k p_j \mid k \in \{i, \dots, j-1\} \right\} & \text{si } i < j \end{cases}$$

1. **(2 pts)** Que représente k dans la formule ci-dessus ?

Réponse :

2. **(2 pts)** Donnez, en fonction de n, la complexité de calculer $m[1,n]$ avec un algorithme de programmation dynamique qui implémente la définition ci-dessus.

Réponse :

3. **(3 pts)** Dans le cas particulier où quatre matrices A_1, A_2, A_3 et A_4 sont de tailles respectives 10×20, 20×5, 5×50 et 50×4, complétez le tableau m :

	j=1	j=2	j=3	j=4
i=1	0			
i=2		0		
i=3			0	
i=4				0

Notations asymptotiques

$$\mathrm{O}(g(n)) = \{f(n) \mid \exists c \in \mathbb{R}^{+\star}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$$
$$\Omega(g(n)) = \{f(n) \mid \exists c \in \mathbb{R}^{+\star}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq cg(n) \leq f(n)\}$$
$$\Theta(g(n)) = \{f(n) \mid \exists c_1 \in \mathbb{R}^{+\star}, \exists c_2 \in \mathbb{R}^{+\star}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \iff g(n) \in \mathrm{O}(f(n)) \text{ et } f(n) \notin \mathrm{O}(g(n)) \qquad f(n) \in \mathrm{O}(g(n)) \iff g(n) \in \Omega(f(n))$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \iff f(n) \in \mathrm{O}(g(n)) \text{ et } g(n) \notin \mathrm{O}(f(n)) \qquad f(n) \in \Theta(g(n)) \iff \begin{cases} f(n) \in \Omega(g(n)) \\ g(n) \in \Omega(f(n)) \end{cases}$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \in \mathbb{R}^{+\star} \iff f(n) \in \Theta(g(n))$$

Ordres de grandeurs

constante	$\Theta(1)$	
logarithmique	$\Theta(\log n)$	
polylogarith.	$\Theta((\log n)^c)$	$c > 1$
	$\Theta(\sqrt{n})$	
linéaire	$\Theta(n)$	
	$\Theta(n \log n)$	
quadratique	$\Theta(n^2)$	
	$\Theta(n^c)$	$c > 2$
exponentielle	$\Theta(c^n)$	$c > 1$
factorielle	$\Theta(n!)$	
	$\Theta(n^n)$	

Identités utiles

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$
$$\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1} \quad \text{si } x \neq 1$$
$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad \text{si } |x| < 1$$
$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2} \quad \text{si } |x| < 1$$
$$\sum_{k=1}^{n} \frac{1}{k} = \Theta(\log n)$$
$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

Définitions diverses

La complexité d'un problème est celle de l'algorithme le plus efficace pour le résoudre.

Un tri stable préserve l'ordre relatif de deux éléments égaux (au sens de la relation de comparaison utilisée pour le tri).

Un tri en place utilise une mémoire temporaire de taille constante (indépendante de n).

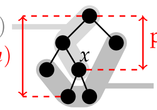
Théorème général

Soit à résoudre $T(n) = aT(n/b) + f(n)$ 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 si $af(n/b) \leq cf(n)$ pour un $c < 1$ et tous les n suffisamment grands, alors $T(n) = \Theta(f(n))$.

(Note : il est possible de n'être dans aucun de ces trois cas.)

Arbres



(nœuds $n = ni + f$)

Pour tout arbre binaire :

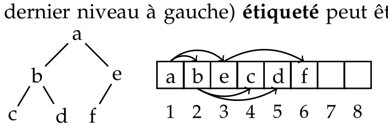
$n \leq 2^{h+1} - 1$ $\qquad h \geq \lceil \log_2(n+1) - 1 \rceil = \lfloor \log_2 n \rfloor$ si $n > 0$

$f \leq 2^h$ $\qquad h \geq \lceil \log_2 f \rceil$ si $f > 0$

$f = ni + 1$ (si l'arbre est *complet* = les nœud internes ont tous 2 fils)

Dans un arbre binaire équilibré une feuille est soit à la profondeur $\lfloor \log_2(n+1) - 1 \rfloor$ soit à la profondeur $\lceil \log_2(n+1) - 1 \rceil = \lfloor \log_2 n \rfloor$. Pour ces arbres $h = \lfloor \log_2 n \rfloor$.

Un arbre parfait (= complet, équilibré, avec toutes les feuilles du dernier niveau à gauche) **étiqueté** peut être représenté par un tableau.



Les indices sont reliés par :

Père$(y) = \lfloor y/2 \rfloor$
FilsG$(y) = y \times 2$
FilsD$(y) = y \times 2 + 1$

Rappels de probabilités

Espérance d'une variable aléatoire X **:** C'est sa valeur attendue, ou moyenne. $\mathrm{E}[X] = \sum_x \Pr\{X = x\}$

Variance : $\mathrm{Var}[X] = \mathrm{E}[(X - \mathrm{E}[X])^2] = \mathrm{E}[X^2] - \mathrm{E}^2[X]$

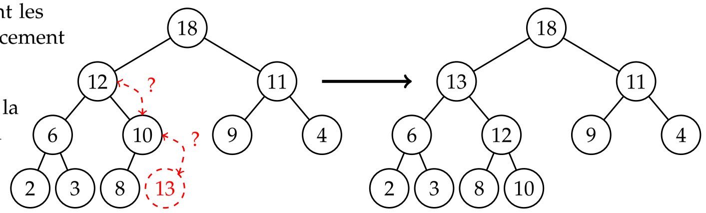
Loi binomiale : On lance n ballons dans r paniers. Les chutes dans les paniers sont équiprobables ($p = 1/r$). On note X_i le nombre de ballons dans le panier i. On a $\Pr\{X_i = k\} = \mathrm{C}_n^k p^k (1-p)^{n-k}$. On peut montrer $E[X_i] = np$ et $\mathrm{Var}[X_i] = np(1-p)$.

Tas

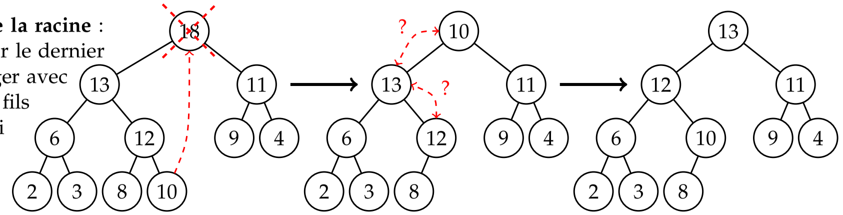
Un tas est un arbre parfait partiellement ordonné : l'étiquette d'un nœud est supérieure à celles de ses fils.

Dans les opérations qui suivent les arbres parfaits sont plus efficacement représentés par des tableaux.

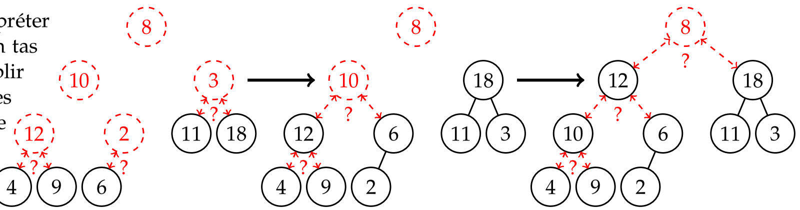
Insertion : ajouter l'élément à la fin du tas, l'échanger avec son père tant qu'il lui est inférieur (en remontant vers la racine). $T_{\text{insert}} = \mathrm{O}(\log n)$



Suppression de la racine : la remplacer par le dernier nœud ; l'échanger avec son plus grand fils tant que celui-ci est plus grand. $T_{\text{rem}} = \mathrm{O}(\log n)$



Construction : interpréter le tableau comme un tas (incorrect) puis rétablir l'ordre en partant des feuilles (vues comme des tas corrects). $T_{\text{build}} = \Theta(n)$

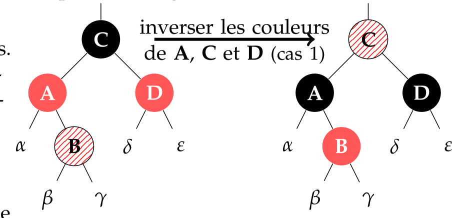


Arbres Rouge et Noir

Les ARN sont des arbres binaires de recherche dans lesquels : (1) un nœud est rouge ou noir, (2) racine et feuilles (NIL) sont noires, (3) les fils d'un nœud rouge sont noirs, et (4) tous les chemins reliant un nœud à une feuille (de ses descendants) contiennent le même nombre de nœuds noirs (= la *hauteur noire*). Ces propriétés interdisent un trop fort déséquilibre de l'arbre, sa hauteur reste en $\Theta(\log n)$.

Insertion d'une valeur : insérer le nœud avec la couleur rouge à la position qu'il aurait dans un arbre binaire de recherche classique, puis, si le père est rouge, considérer les trois cas suivants dans l'ordre.

Cas 1 : Le père et l'oncle du nœud considéré sont tous les deux rouges. **Répéter** cette transformation à partir du grand-père si l'arrière grand-père est aussi rouge.



(Dans tous ces cas, les lettres grecques représentent des sous-arbres avec la même hauteur noire.)

Cas 2 : Si le père est rouge, l'oncle noir, et que le nœud courant n'est pas dans l'axe père–grand-père, une rotation permet d'aligner fils, père, et grand-père.

Cas 3 : Si le père est rouge, l'oncle noir, et que le nœud courant est dans l'axe père–grand-père, un rotation et une inversion de couleurs rétablissent les propriétés des ARN.