Nom :

Prénom :

Examen d'algorithmique

EPITA ING1 2012 S1; A. DURET-LUTZ

Durée : 1 heure 30

Janvier 2010

Consignes

- Cet examen se déroule **sans document** et **sans calculatrice**.
- Répondez sur le sujet dans les cadres prévus à cet effet.
- Il y a six pages d'énoncé, et une page d'annexe.
  Rappelez votre nom en haut de chaque feuille au cas où elles se mélangeraient.
- Ne donnez pas trop de détails. Lorsqu'on vous demande des algorithmes, on se moque des points-virgules de fin de commande etc. Écrivez simplement et lisiblement. Des spécifications claires et implémentables sont préférées à du code C ou C++ verbeux.
- Le barème est indicatif et correspond à une note sur 25.

1 Notation Θ (3 points)

1. **(2 points)** Prouvez rigoureusement que quelles que soient trois constantes a, b, c strictements positives, on a $a\log(bn+c) = \Theta(\log(n))$

Réponse :

2. **(1 point)** Laquelle ou lesquelles des constantes a, b et c peuvent être prises nulles sans invalider l'égalité ci-dessus ?

Réponse :

2 Programmation Dynamique (13 points)

Soient $m_1 < m_2 < \cdots < m_n$ des entiers stockés dans un arbre binaire de recherche. Chaque entier m_i est associé à un poids w_i qui représente la fréquence avec lequel il va être recherché dans l'arbre (plus w_i est grand, plus m_i est recherché souvent par le programme). Notre objectif va être de construire l'arbre de recherche le plus efficace en fonction de ces poids qui sont connus à l'avance.

Pour formaliser la notion d'arbre efficace, définissons le coût d'accès pondéré C d'un arbre binaire de recherche pour n entiers comme

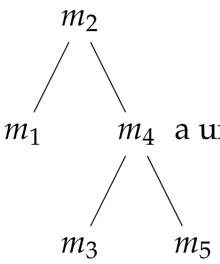
$$C = \sum_{i=1}^{n}(d_i + 1)w_i$$

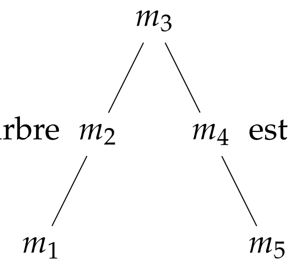
où d_i représente la profondeur du nœud représentant la valeur m_i dans l'arbre (la racine étant à la profondeur 0). Intuitivement $d_i + 1$ correspond au nombre de comparaisons à faire avant de trouver m_i dans l'arbre.

Par exemple considérons deux arbres binaires de recherches pour les valeurs

i	1	2	3	4	5
m_i	2	5	6	8	9
w_i	10	3	15	2	7

.

L'arbre  a un coût d'accès pondéré de $C = 2w_1 + 1w_2 + 3w_3 + 2w_4 + 3w_5 = 93$.

Tandis que le coût d'accès pondéré de l'arbre  est $C = 3w_1 + 2w_2 + 1w_3 + 2w_4 + 3w_5 = 76$.

Si l'on ne devait choisir qu'entre ces deux arbres, on garderait le second, de coût plus faible. On veut (malheureusement pour vous) trouver l'arbre de coût minimal parmi **tous** les arbres binaires de recherche possibles pour les valeurs $m_1, \ldots, m_n$.

Il est possible de trouver cet arbre par programmation dynamique.

Notons $C(i,j)$ le coût d'accès pondéré **minimal** des arbres binaires de recherche pour les valeurs $m_i, \ldots, m_j$. On a

$$C(i,j) = \begin{cases} 0 & \text{si } i > j \\ w_i & \text{si } i = j \\ \min\limits_{k \in [\![i,j]\!]} \left(C(i,k-1) + C(k+1,j) + \sum\limits_{m=i}^{j} w_m \right) & \text{si } i < j \end{cases}$$

1. **(2 points)** Justifiez la dernière ligne (cas $i < j$) de la définition récursive de $C(i,j)$.

Réponse :

2. **(4 points)** Completez le tableau des $C(i,j)$ suivant pour les valeurs de l'exemple.

	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$					
$i=2$	0	3			41
$i=3$	0	0	15		35
$i=4$	0	0	0	2	11
$i=5$	0	0	0	0	7

3. **(2 points)** Pour n valeurs dans l'arbre, quelle est la complexité de l'algorithme qui remplit ce tableau (justifiez votre réponse sans écrire l'algorithme).

Réponse :

4. **(3 points)** La personne qui a écrit l'algorithme avait suivi les cours d'algo, donc elle a aussi pris la peine de sauvegarder dans un tableau séparé la valeur du k qui donnait le coût minimal dans le calcul de $C(i,j)$. Le tableau obtenu est le suivant :

	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	1	3	3	3
$i=2$		3	3	3
$i=3$			3	3
$i=4$				5

Déduisez-en et dessinez l'arbre binaire de recherche de coût d'accès pondéré minimal pour les valeurs de l'exemple.

Réponse :

5. **(2 points)** Pour 5 valeurs distinctes, comme dans l'exemple, combien peut-on créer d'arbres binaires de recherche différents ? (Sans prendre en compte les poids. C'est juste du dénombrement.)

Réponse :

3 File de priorité (4 points)

Dans cette question on considère une file de priorité S implémentée à l'aide d'un tas « max », c'est-à-dire avec la valeur maximale à la racine du tas.

S est initialement vide. Donnez l'état du tableau représentant S après y avoir effectué chacune des opérations suivantes :

– `insert(5)`

5

– `insert(2)`

5	2

– `insert(7)`

– `insert(6)`

– `insert(4)`

– `extract_max()`

– `extract_max()`

– `insert(6)`

– `extract_max()`

– `extract_max()`

4 Recherche par interpolation (5 points)

Voici un algorithme de recherche dans un tableau d'entiers. Le principe est similaire à la recherche dichotomique, sauf qu'au lieu de sonder le tableau *en son milieu* avant d'explorer l'un des côtés, le point de coupe est choisi par interpolation en fonction des valeurs des extrémités et de la valeur recherchée.

Entrées :
- A : un tableau d'entiers, trié ;
- l : le plus petit indice du tableau ;
- r : le plus grand indice du tableau ;
- v : la valeur à rechercher parmi $A[l..r]$.

Sortie :
- l'indice de v dans A s'il existe, 0 sinon.

```
INTERPOLATIONSEARCH(A, l, r, v)
if l ≤ r then
    m ← ⌊ (r − l) / (A[r] − A[l]) · (v − A[l]) + l ⌋
    if v = A[m] then
        return m
    else
        if v < A[m] then
            return INTERPOLATIONSEARCH(A, l, m − 1, v)
        else
            return INTERPOLATIONSEARCH(A, m + 1, r, v)
else
    return 0
```

1. **(1 point)** Les appels récursifs de cet algorithme sont-ils terminaux ?

Réponse :

2. **(2 point)** Donnez une définition *récursive* de la complexité en pire cas de cet algorithme, en fonction de la taille du tableau $n = r - l + 1$.

Réponse :

3. **(1 point)** Quelle est la solution de cette équation ?

Réponse :

4. **(1 point)** Quelle est la complexité de l'algorithme dans le meilleur cas ?

Réponse :

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

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

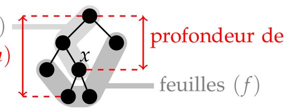
Théorème général

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 :

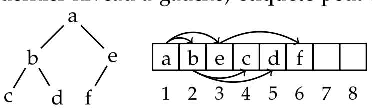
$n \leq 2^{h+1} - 1$ $\qquad h \geq \lceil \log_2(n+1) - 1 \rceil = \lfloor \log_2 n \rfloor$ si $n > 0$

$f \leq 2^h$ $\qquad h \geq \lceil \log_2 f \rceil$ 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 :

$\text{Père}(y) = \lfloor y/2 \rfloor$

$\text{FilsG}(y) = y \times 2$

$\text{FilsD}(y) = y \times 2 + 1$

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).

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

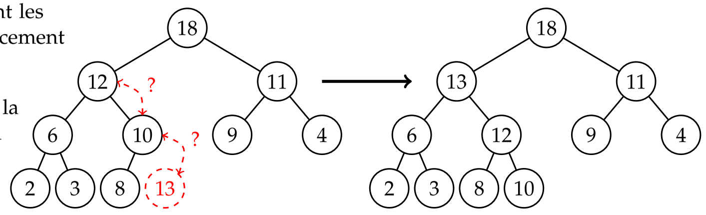
Loi binomiale : On lance n ballons dans r paniers. Les chutes dans les paniers sont équiprobables ($p = 1/r$). On note X_i le nombre de ballons dans le panier i. On a $\Pr\{X_i = k\} = \mathrm{C}_n^k p^k (1-p)^{n-k}$. On peut montrer $E[X_i] = np$ et $\mathrm{Var}[X_i] = np(1-p)$.

Tas

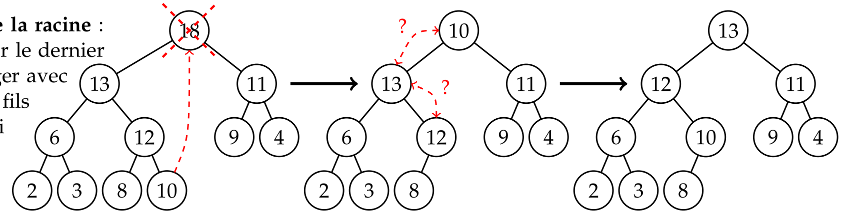
Un tas est un arbre parfait partiellement ordonné : l'étiquette d'un nœud est supérieure à celles de ses fils.

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

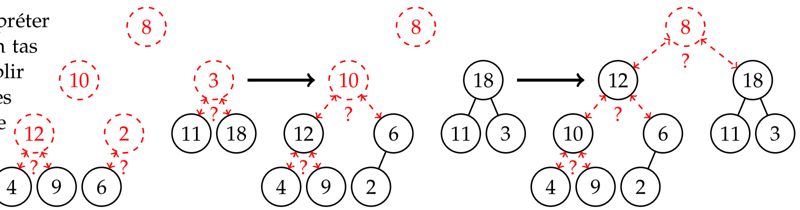
Insertion : ajouter l'élément à la fin du tas, l'échanger avec son père tant qu'il lui est inférieur (en remontant vers la racine). $T_{\text{insert}} = \mathrm{O}(\log n)$



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). $T_{\text{build}} = \Theta(n)$

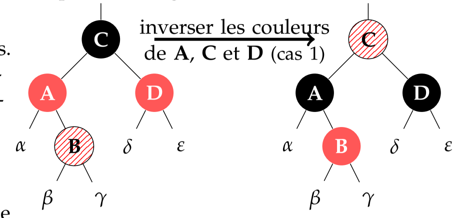


Arbres Rouge et Noir

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, une rotation et une inversion de couleurs rétablissent les propriétés des ARN.