Correction

Examen d'algorithmique

EPITA ING1 2011 S1 RATTRAPAGE; A. DURET-LUTZ

Durée : 1 heure 30

Rattrapage

Consignes

– Cet examen se déroule **sans document** et **sans calculatrice**.
– Répondez sur le sujet dans les cadres, ou en cochant des cases pour une question à choix multiple.
– Il y a 7 pages d'énoncé, et 1 page d'annexe dont vous pourriez avoir besoin.
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 **40**.
– **Dans les questions à choix multiples**, une bonne réponse rapporte 2 points tandis qu'**une mauvaise réponse retire** 1 **point**.

1 Stupides éléphants (6 points)

Dans un zoo, on a pesé tous les éléphants, et on leur a fait passer des tests pour mesurer leur intelligence. On dispose maintenant d'un ensemble de paires $(poids, QI)$. Par exemple

$$\mathcal{D} = \{(3890, 39); (3750, 40); (3920, 72); (3840, 42); (4004, 61); (3633, 37); (3647, 52); (3792, 48); (3870, 63); (3549, 53); (2950, 33)\}$$

Certains employés du zoo pensent que plus un éléphant est gros, plus il est intelligent. Nous souhaitons écrire un algorithme pour les convaincre du contraire. De l'ensemble des paires ci-dessus, nous voulons extraire le plus grand sous-ensemble tel que lorsque les poids sont placés dans l'ordre croissant, alors les QI sont décroissants. Pour une entrée $\mathcal{E}$, nous noterons ce plus grand sous-ensemble $C(\mathcal{E})$.

Avec l'exemple ci-dessus ce plus grand sous-ensemble serait :

$$C(\mathcal{D}) = \{(3549, 53); (3647, 52); (3792, 48); (3840, 42); (3890, 39)\}$$

1. **(3 points)** Expliquez comment ce problème peut se ramener a un calcul de plus longue sous-séquence croissante.

Réponse :

On commence par ordonner les paires par poids décroissants. On cherche alors dans ces paires la plus longue sous-séquence dans laquelle les QI sous croissants.

2. **(3 points)** Sachant que la plus longue sous-séquence croissante peut-être calculée en $\mathrm{O}(n^2)$, calculez et justifiez la complexité de l'algorithme d'extraction de $C(\mathcal{E})$ lorsque $\mathcal{E}$ possède n éléments.

Réponse :

Tous les tri comparatifs vus en cours sont en $\mathrm{O}(n^2)$. Enchaîner l'un de ces tris avec un calcul de plus longue sous-séquence croissante sera donc en $\mathrm{O}(n^2) + \mathrm{O}(n^2) = \mathrm{O}(n^2)$.

2 Multiplications matricielles (12 points)

On considère plusieurs algorithmes effectuant la multiplication de deux matrices $A \times B = C$ de taille $n \times n$. Ces algorithmes seront décrits à travers des définitions mathématiques, c'est à vous d'imaginer le code nécessaire pour les mettre en œuvre afin de pouvoir répondre aux questions.

Chaque algorithme de multiplication prend en entrée les tableaux des coefficients de A et B et doit produire en sortie le tableau des coefficient de C.

Le nombre total d'opérations effectuées pour obtenir le produit est proportionnel au nombre de multiplications scalaires effectuées par l'algorithme : vous pouvez donc **négliger le coût des opérations d'addition et de soustraction scalaires ou matricielles effectuées par ces algorithmes**.

2.1 Algorithme 1 (2 points)

Si l'on note $a_{i,j}$, $b_{i,j}$ et $c_{i,j}$ les coefficients respectifs des matrices A, B et C, on a

$$\forall i \in [\![1, n]\!], \forall j \in [\![1, n]\!], \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k} \cdot b_{k,j}$$

1. Qu'elle est la complexité de la multiplication de deux matrices de taille $n \times n$ en utilisant cette définition ?

☐ $\Theta(n)$

☐ $\Theta(n \log n)$

☐ $\Theta(n^2)$

☒ $\Theta(n^3)$

☐ $\Theta(n^4)$

2.2 Algorithme 2 (4 points)

Pour cet algorithme et le suivant, on suppose que n est une puissance de 2 (c'est-à-dire $n = 2^m$) ; il est toujours possible de s'y ramener en complétant les matrices avec des lignes et des colonnes remplies de zéros.
Découpons chacune de ces matrices en quatre blocs de taille $\frac{n}{2} \times \frac{n}{2}$:

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}, \quad B = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}, \quad C = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

On a alors :

$$\begin{aligned} C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\ C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\ C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{aligned}$$

Ceci suggère un algorithme récursif de calcul des coefficients de C.

1. Si l'on note $T(n)$ la complexité du calcul des coefficients de C lorsque les matrices A et B sont de taille $n \times n$, quelle définition récursive de T suggère cet algorithme ?

   ☐ $T(n) = 4T(n/2)$
   ☐ $T(n) = 4T(n/4) + O(n^2)$
   ☐ $T(n) = 4T(n/4) + O(n^3)$
   ☐ $T(n) = 7T(n/2)$
   ☒ $T(n) = 8T(n/2)$

2. Quelle est la solution de cette équation ?

   ☐ $T(n) = \Theta(n \log_2 n)$
   ☐ $T(n) = \Theta(n^2)$
   ☐ $T(n) = \Theta(n^{\log_2 7})$
   ☒ $T(n) = \Theta(n^3)$
   ☐ $T(n) = \Theta(n^{\log_2 12} \log_2 n)$

2.3 Algorithme 3 (6 points)

Comme dans l'algorithme précédent, on suppose les matrices découpées en quatre blocs de taille $\frac{n}{2} \times \frac{n}{2}$. Posons

$$\begin{aligned}
M_1 =& (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}) \\
M_2 =& (A_{2,1} + A_{2,2})B_{1,1} \\
M_3 =& A_{1,1}(B_{1,2} - B_{2,2}) \\
M_4 =& A_{2,2}(B_{2,1} - B_{1,1}) \\
M_5 =& (A_{1,1} + A_{1,2})B_{2,2} \\
M_6 =& (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2}) \\
M_7 =& (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})
\end{aligned}$$

on a alors

$$\begin{aligned}
C_{1,1} =& M_1 + M_4 - M_5 + M_7 \\
C_{1,2} =& M_3 + M_5 \\
C_{2,1} =& M_2 + M_4 \\
C_{2,2} =& M_1 - M_2 + M_3 + M_6
\end{aligned}$$

Ceci suggère un deuxième algorithme de calcul récursif des sous-matrices $C_{i,j}$.

1. Si l'on note $T(n)$ la complexité du calcul des coefficients de C lorsque les matrices A et B sont de taille $n \times n$, quelle définition récursive de T suggère l'algorithme 3 ?

   ☐ $T(n) = 12T(n/2)$
   ☐ $T(n) = 12T(n/2) + \mathrm{O}(n^2)$
   ☐ $T(n) = 12T(n/4) + \mathrm{O}(n^2)$
   ☒ $T(n) = 7T(n/2)$
   ☐ $T(n) = 8T(n/3)$

2. Quelle est la solution de cette équation ?

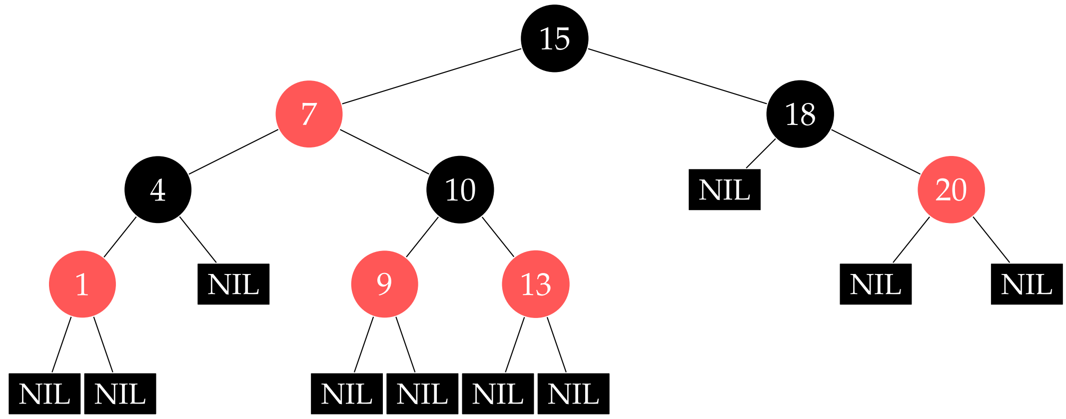
   ☐ $T(n) = \Theta(n \log_2 n)$
   ☐ $T(n) = \Theta(n^2)$
   ☒ $T(n) = \Theta(n^{\log_2 7})$
   ☐ $T(n) = \Theta(n^3)$
   ☐ $T(n) = \Theta(n^{\log_2 12} \log_2 n)$

3. Diriez-vous de ce troisième algorithme que c'est (cochez toutes les réponses correctes) :

   ☐ un algorithme glouton
   ☒ un algorithme diviser pour régner
   ☐ un algorithme de programmation dynamique
   ☐ un algorithme dichotomique
   ☐ un algorithme de tri

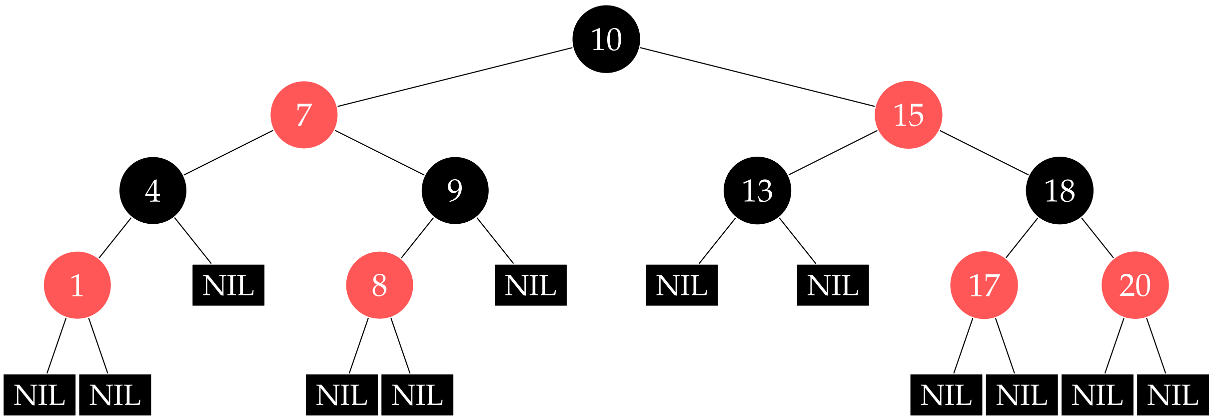
3 Arbre rouge et noir (12 points)

1. **(4 points)** On considère l'arbre rouge et noir suivant :



Dessinez l'arbre rouge et noir résultant de l'insertion des valeurs 8 et 17 dans l'arbre ci-dessus. (Si vous n'utilisez pas de couleur, merci d'indiquer clairement quelle convention vous avez choisie pour distinguer les nœuds rouges des noirs.)

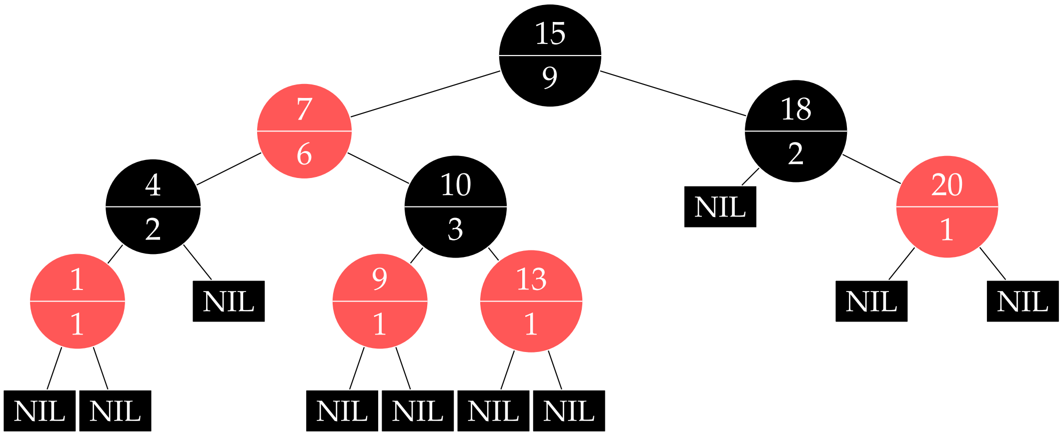
Réponse :



2. **(2 points)** Quelle est la complexité de rechercher la k^e plus grande valeur dans un arbre rouge et noir de n nœuds ? (Indiquez la classe de complexité la plus précise possible.)

- ☐ $\Theta(1)$ opérations
- ☐ $O(\log_2 n)$ opérations
- ☒ $O(n)$ opérations
- ☐ $O(n \log_2 n)$ opérations
- ☐ $O(n^2)$ opérations

3. **(6 points)** On modifie maintenant la définition des arbres rouge et noir pour inclure dans chaque nœud la taille du sous-arbre qui y est enraciné (sans compter les feuilles NIL). Cette information supplémentaire est ici représentée dans la partie basse du nœud.



Donnez, pour cette nouvelle structure, les complexités des algorithmes (que vous devrez imaginer) pour

L'insertion d'une valeur :

- ☐ $\Theta(1)$ opérations
- ☒ $O(\log_2 n)$ opérations
- ☐ $O(n)$ opérations
- ☐ $O(n \log_2 n)$ opérations
- ☐ $O(n^2)$ opérations

La suppression de la racine :

- ☐ $\Theta(1)$ opérations
- ☒ $O(\log_2 n)$ opérations
- ☐ $O(n)$ opérations
- ☐ $O(n \log_2 n)$ opérations
- ☐ $O(n^2)$ opérations

La recherche de la n^e plus grande valeur :

- ☐ $\Theta(1)$ opérations
- ☒ $O(\log_2 n)$ opérations
- ☐ $O(n)$ opérations
- ☐ $O(n \log_2 n)$ opérations
- ☐ $O(n^2)$ opérations

4 Divers (10 points)

1. **(3 points)** Démontrez rigoureusement que $\lfloor n \rfloor + \lceil n \rceil \in \Theta(n)$.

Réponse :

Pour $n \geq 1$ on a $\lfloor n \rfloor + \lceil n \rceil \geq 2\lfloor n \rfloor > n$. D'autre part pour $n \geq 2$ on a $\lfloor n \rfloor + \lceil n \rceil \leq 2\lceil n \rceil < 3n$. Ainsi $\forall n \geq 2$, $n < \lfloor n \rfloor + \lceil n \rceil < 3n$ et on a prouvé que $\lfloor n \rfloor + \lceil n \rceil \in \Theta(n)$.

2. **(4 points)** Indiquez une façon de rendre stable n'importe quel algorithme de tri. Quelle complexité (en temps et espace) cette modification ajoute-t-elle à un algorithme de tri lorsqu'il faut trier n valeurs ?

Réponse :

Il suffit de modifier les clefs des valeurs à trier pour y inclure leur indice d'origine. La fonction de comparaison doit alors être adaptée pour comparer ces indices lorsque les parties originales des clefs sont identiques.
Ce changement demande $\Theta(n)$ mémoire supplémentaire (on ne peut donc plus prétendre que le tri soit *en place*, car l'occupation mémoire dépend de n), en revanche il ne modifie pas la complexité du temps d'exécution.

3. **(3 points)** Soit E un ensemble contenant $2n+1$ éléments (autrement dit un nombre impair d'éléments). Combien existe-t-il de sous-ensembles de E qui possèdent un nombre pair d'éléments ?

Réponse :

E possède $2^{2n+1} = 2 \times 4^n$ sous-ensembles mais on ne veut que ceux de taille paire.
Pour chaque sous-ensemble $F \subset E$ de taille paire, le l'ensemble $E \setminus F$ est un sous-ensemble de E de taille impaire. Il y a donc autant de sous-ensembles de taille paire que de sous-ensembles de taille impaire.
E possède donc $\frac{2\times 4^n}{2} = 4^n$ sous-ensembles de taille paire, et autant de taille impaire.

FIN

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).

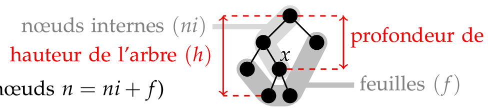
Théorème général

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 :

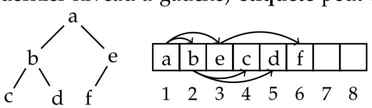
$n \leq 2^{h+1} - 1$ $\qquad h \geq \lceil \log_2(n+1) - 1 \rceil = \lfloor \log_2 n \rfloor$ si $n > 0$

$f \leq 2^h$ $\qquad h \geq \lceil \log_2 f \rceil$ 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 :
$\text{Père}(y) = \lfloor y/2 \rfloor$
$\text{FilsG}(y) = y \times 2$
$\text{FilsD}(y) = y \times 2 + 1$

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

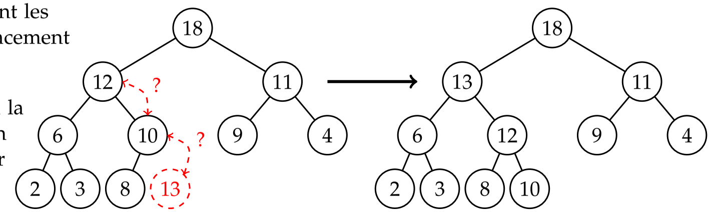
Loi binomiale : On lance n ballons dans r paniers. Les chutes dans les paniers sont équiprobables ($p = 1/r$). On note X_i le nombre de ballons dans le panier i. On a $\Pr\{X_i = k\} = \mathrm{C}_n^k p^k (1-p)^{n-k}$. On peut montrer $E[X_i] = np$ et $\mathrm{Var}[X_i] = np(1-p)$.

Tas

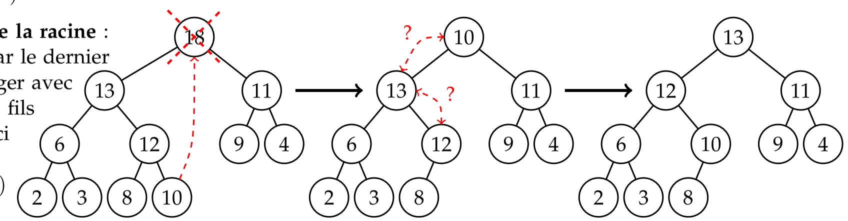
Un tas est un arbre parfait partiellement ordonné : l'étiquette d'un nœud est supérieure à celles de ses fils.

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

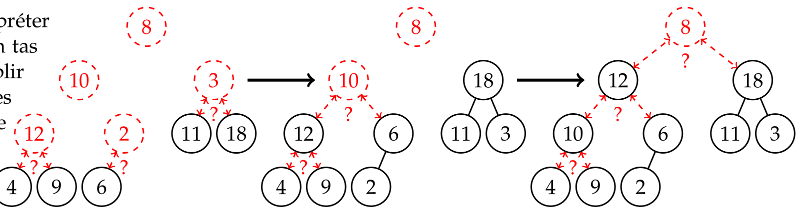
Insertion : ajouter l'élément à la fin du tas, l'échanger avec son père tant qu'il lui est inférieur (en remontant vers la racine). $T_{\text{insert}} = \mathrm{O}(\log n)$



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

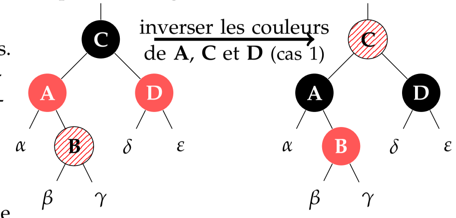


Arbres Rouge et Noir

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.