Institut Galilée — Algorithmique, arbres et graphes
Année 2008-2009 — L2

Devoir à rendre le 25 mai 2009

Le barême est uniquement indicatif.

Vous pouvez écrire les algorithmes en C ou en pseudo-code. Si cela vous pose trop de difficultés n'hésitez pas à répondre en décrivant un algorithme par des phrases ! Vous pouvez faire appel à des fonctions auxilliaires vues en cours (comparaison, tris, sous-tableau, etc.).

Première partie — 10 points

Exercice 1 (Partiel mai 2008)**.**
Rappeler les définitions utilisées et justifier (démontrer) vos réponses.

1. Est-ce que $n^2 - 2n + 1 = O(n^2)$? — 1 pt, 9 min
2. Est-ce que $\sum_{i=1}^{n} \log i = \Omega(n)$? — 1 pt, 9 min
3. Est-il vrai que si $f = O(g)$ et $f = \Omega(h)$ alors $g = \Omega(h)$? — 1 pt, 9 min

Exercice 2 (Invariant de boucle)**.**
Étant donné un tableau T de $N > 0$ entiers, l'algorithme Maximum renvoie l'indice de l'élément maximum de T. Démontrer le à l'aide d'un invariant de boucle. — 1.5 pt, 13 min

Fonction Maximum(T)
```
m = 0;
pour i ← 1 à Taille(T) − 1 faire
    si T[i] ≥ T[m] alors
        m = i;
retourner m;
```

Exercice 3 (Piles, files)**.**
Partant d'une pile vide, on ajoute (empile) 10, puis 20, 30, 40, 50 combien doit-on retirer (dépiler) d'éléments pour que le prochain élément qui sera retiré soit 40 ? (Facile) Même question en utilisant une file (initialement vide). — 0,5 pt, 4 min

Exercice 4 (tas)**.**

1. Former un tas max en insérant successivement les éléments $5, 7, 3, 9, 2, 0, 6, 4, 8, 1$, répondre en représentant le tas obtenu. — 1 pt, 9 min
2. Dans le polycopié du cours vous trouverez une autre façon de former un tas. Essayez là sur l'exemple précédant (vous considérerez que les éléments de la question précédente sont rangés dans le même ordre dans un tableau). Cette question ne compte pas pour le devoir !
3. Supprimer l'élément maximum du tas, répondre en représentant le nouveau tas. — 0.5 pt, 4 min

Exercice 5 (Insertion / suppression ABR)**.**
Former un arbre binaire de recherche en insérant successivement et dans cet ordre les éléments : $5, 7, 3, 9, 2, 0, 6, 4, 8, 1$ (répondre en représentant l'arbre obtenu). Supprimer l'élément 5. (Répondre en représentant le nouvel arbre. Il y a deux réponses correctes possibles, selon la variante choisie pour l'algorithme de suppression). — 1 pt, 9 min

Exercice 6.
Écrire une fonction récursive Somme(x) prenant un arbre binaire dont les éléments sont des entiers et rendant la somme de ses éléments. — 1 pt, 9 min

Exercice 7.
Classer les fonctions de complexité $n \log n, \log n, n^2, n$ par ordre croissant. Pour les complexités $\log n$ et n donner l'exemple d'un algorithme (du cours ou des TD) qui a asymptotiquement cette complexité en pire cas, en temps. — 1.5 pt, 13 min

Problème du rang (cinq exercices)

tot: 10 pt

Dans cette partie on s'intéresse au problème suivant : étant donné un ensemble A de N éléments deux à deux comparables, déterminer quel est l'élément de rang k (le k-ième plus petit élément), c'est à dire l'élément $x \in A$ tel que exactement $k-1$ éléments de A sont plus petits que x. Bien entendu, on peut supposer que k est choisi tel que $1 \leq k \leq N$. On pourra considérer que les éléments de A sont distincts (la comparaison ne les déclare jamais égaux). Si on a par exemple les éléments 23, 62, 67, 56, 34, 90, 17 (N vaut 7) alors l'élément de rang 3 est 34.

Exercice 8.
Dans la liste 21, 90, 30, 18, 35, 45, 77, quel est l'élément de rang 4 ?

0,5 pt
4 min

Les exercices suivants portent sur l'écriture d'un algorithme réalisant la fonction de recherche de l'élément de rang k dans un ensemble A de N éléments : Sélection(S, k), où S est la structure de donnée contenant les éléments de A.

Rang dans un tableau

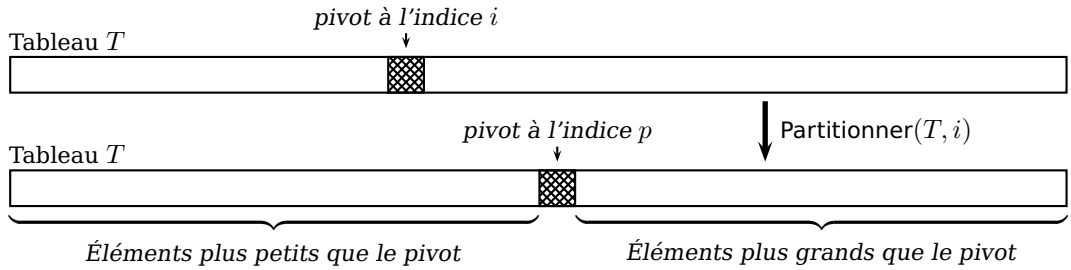
On suppose que les éléments de A sont donnés en entrée dans un tableau T non trié de N éléments.

Exercice 9.
Quel est le moyen *le plus simple* de réaliser la fonction Sélection(T, k) ? La réponse doit tenir en 3-4 lignes de pseudo-code ou de C en faisant appel (comme fonction auxiliaire) à un algorithme du cours dont vous préciserez le nom. Donner un minorant asymptotique en fonction de N en pire cas du temps d'exécution de Sélection(T, k).

1 pt
9 min

Rappel sur le tri rapide. On rappelle le fonctionnement du tri rapide (*quicksort*), algorithme permettant de trier un tableau d'éléments deux à deux comparables (le résultat, c'est à dire le tableau contenant les éléments dans l'ordre est rendu en place). Le principe de fonctionnement est le suivant. Si le tableau contient un ou zéro élément, il n'y a rien à faire. Si le tableau contient plus d'un élément :

Partition on choisit un élément du tableau, d'indice i, appelé *pivot* et on partitionne le tableau, en place, autour de cet élément. Après partition, le pivot est à un certain indice p du tableau, les éléments plus petits que le pivot sont (éventuellement dans le désordre) aux indices inférieurs à p et les éléments plus grands que le pivot sont (éventuellement dans le désordre) aux indices supérieurs à p. Le pivot est donc à sa place définitive.



Appels récursifs On relance le tri sur chacun des deux sous-tableaux obtenus par partition : le tableau des éléments d'indices inférieurs à p et le tableau des éléments d'indices supérieurs à p.

Exercice 10.
Dans cet exercice, on va mettre en œuvre un algorithme inspiré du tri rapide pour réaliser la fonction Sélection(T, k). L'idée est qu'il n'est pas besoin de trier tout le tableau pour trouver l'élément de rang k.

On suppose que les indices du tableau commencent à 0. La remarque importante est que le pivot après partitionnement est l'élément de rang $p+1$. Si après partition on trouve un p égal à $k-1$, alors l'élément de rang k est le pivot.

On suppose que l'on a une fonction Partitionner(T, i) qui effectue la partition autour d'un élément d'indice i (pivot) et renvoie l'indice p du pivot après partition.

1. Après partition autour d'un pivot où chercher l'élément de rang k lorsque $k < p+1$? Et lorsque $k > p+1$? — 0.5 pt, 4 min
2. Décrire simplement le principe d'un algorithme récursif, basé sur la partition, réalisant Sélection(T, k). (A t-on besoin de faire deux appels récursifs comme dans le tri rapide ?) — 1 pt, 9 min
3. Écrire en pseudo-code ou en C, l'algorithme Sélection(T, k) sous forme itérative. — 2 pt, 18 min
4. Écrire l'algorithme de partitionnement Partitionner(T, i). On pourra commencer par échanger l'élément d'indice i avec l'élément d'indice 0 de manière à se ramener toujours au cas où le premier élément du tableau a été choisi comme pivot. — 1.5 pt, 13 min

Rang dans un arbre binaire de recherche avec comptage des descendants

On enrichit la structure de données *arbre binaire de recherche* (ABR) en ajoutant à chaque nœud x un entier Total(x) donnant le nombre d'éléments stockés dans le sous-arbre dont le nœud x est racine, l'élément stocké dans x inclus. Exemple figure 1.

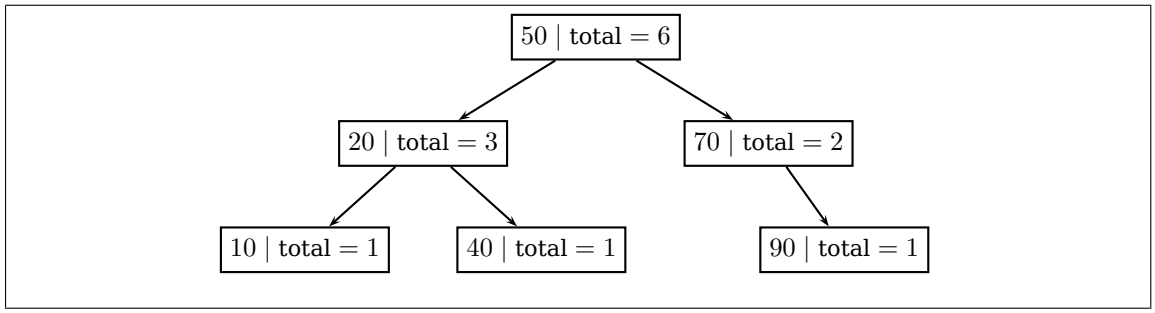


Fig. 1: Un exemple d'arbre binaire de recherche avec comptage des descendants

Dans la suite, on suppose que les éléments de A sont stockés dans un ABR (de racine r) enrichie par ce comptage.

Exercice 11.
Écrire un algorithme (C ou pseudo-code) réalisant la fonction Sélection(r, k). Donner en fonction de la hauteur h de l'arbre, un majorant asymptotique serré de son temps d'exécution. — 1.5 pt, 13 min

En guise d'aide, on rappelle, en pseudo-code, l'algorithme de recherche dans un ABR.

Fonction Rechercher(r, c)

Entrées : Le nœud racine r d'un arbre binaire de recherche et une clé c.
Sorties : Le nœud de l'arbre contenant l'élément de clé c s'il en existe un, ou le nœud vide N sinon.

```
si EstVide(r) alors
    retourner N ;
sinon
    si c = Clé(r) alors
        retourner r ;
    sinon
        si c < Clé(r) alors
            retourner Rechercher(Gauche(r), c) ;
        sinon
            retourner Rechercher(Droite(r), c) ;
```

Exercice 12.
Pour réaliser les ABRs avec comptage des descendants, il faut adapter les fonctions d'ajout et de suppression d'un élément.

1. Rappeler (en C ou en pseudo-code) l'algorithme d'ajout d'un élément dans un ABR et indiquer les modifications à lui apporter. 1 pt 9 min
2. Pour la suppression d'un élément, il ne vous est pas demandé de rappeler le code de la fonction la réalisant. Mais supposons que cette fonction renvoie une référence sur le parent p du nœud qui a été *réellement* supprimé. Donnez dans ce cas une méthode (une fonction prenant le paramètre p) pour mettre à jour le comptage à travers l'arbre. 1 pt 9 min

Corrigé

Vous pouvez écrire les algorithmes en C ou en pseudo-code. Si cela vous pose trop de difficultés n'hésitez pas à répondre en décrivant un algorithme par des phrases ! Vous pouvez faire appel à des fonctions auxilliaires vues en cours (comparaison, tris, sous-tableau, etc.).

Première partie — 10 points

Correction de l'exercice 1.

Rappel des définitions (c'était demandé) :

- $f = O(g)$ ssi
$$\exists c_0 > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq f(n) \leq c_0 g(n).$$
- $f = \Omega(g)$ ssi
$$\exists c_0 > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq c_0 g(n) \leq f(n).$$
- $f = \Theta(g)$ (f ssi $f = O(g)$ et $f = \Omega(g)$

1. Oui $n^2 - 2n + 1 = O(n^2)$. En effet, $n^2 - 2n + 1 = (n-1)^2 \leq 1 \times n^2 (\forall n \geq 1)$ ainsi prendre $c = 1$ et $n_0 = 1$ convient.
2. Oui. Pour $i \geq 2$, on a $\log i \geq 1$. Donc $\sum_{i=1}^{n} \log i \geq \log 1 + \sum_{i=2}^{n} 1 = n - 1$. Lorsque $n \geq 2$ on a $\frac{n}{2} - 1 \geq 0$ donc (en ajoutant $\frac{n}{2}$ de part et d'autre) on a encore $n - 1 \geq \frac{1}{2}n$. Ainsi, prendre $c = \frac{1}{2}$ et $n_0 = 2$ convient.
3. Oui. Puisque $f = O(g)$, il existe $n_0 \in \mathbb{N}$ et $c_0 > 0$ tels que $f(n) \leq c_0 g(n), \forall n \geq n_0$. Et puisque $f = \Omega(h)$, il existe $n_1 \in \mathbb{N}$ et $c_1 > 0$ tels que $f(n) \geq c_1 h(n), \forall n \geq n_1$. Donc, à partir du rang $n_2 = \max(n_0, n_1) \in \mathbb{N}$, $c_1 h(n) \leq c_0 g(n)$, ce qui donne, en posant $c_2 = \frac{c_1}{c_0}$, $g(n) \geq c_2 h(n)$ avec $c_2 > 0$. Ce qui montre bien que $g = \Omega(h)$.

Correction de l'exercice 2.

On pose l'invariant de boucle : $\forall 0 \leq j \leq i-1$, $T[j] \leq T[m]$.

À l'initialisation de la boucle $i = 1$ et l'invariant est bien vérifié puisque m vaut 0 et le seul j pour lequel on doit vérifier $T[j] \leq T[m]$ est 0.

Conservation. On suppose que l'invariant est vérifié jusqu'au début de la i-ème étape de boucle et on montre qu'il est encore vérifié au début de l'étape suivante. À l'étape i, $T[m]$ est comparé à $T[i]$, il y a alors deux possibilités :

- si $T[i]$ est supérieur ou égal à $T[m]$ donc $T[i]$ est supérieur ou égal à $T[j]$ pour tout $0 \leq j \leq i-1$ (par hypothèse) et bien entendu supérieur ou égal à lui même. Et dans ce cas m prend la valeur i donc l'invariant est vérifié au début de l'étape suivante (pour i incrémenté de 1).
- sinon $T[i]$ est plus petit que $T[m]$, m ne change pas et l'invariant est vérifié au début de l'étape suivante (pour i incrémenté de 1) : pour la valeur de i correspondant à l'étape actuel on a, par hypothèse, que $T[j] \leq T[m]$ pour tout $0 \leq j \leq i-1$ et on l'a encore pour $j = i$.

Terminaison. L'invariant est vérifié au début de l'étape $i = \mathsf{Taille}(T)$ pour laquelle la condition d'exécution de la boucle devient fausse. Ainsi on a $T[j] \leq T[m]$ pour tout $0 \leq j \leq \mathsf{Taille}(T) - 1$, c'est à dire que m est bien l'indice du plus grand élément du tableau. Par ailleurs le tableau n'est pas modifié par cet algorithme.

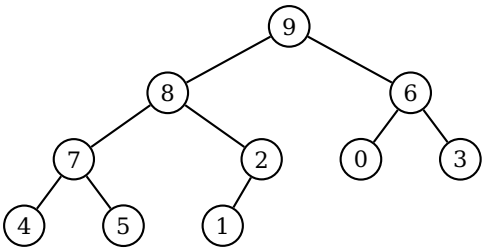
Correction de l'exercice 3.

Après empilement des éléments il faut dépiler un élément (50) pour que 40 se trouve en haut de la pile (et soit ainsi le prochain à sortir par dépilement).

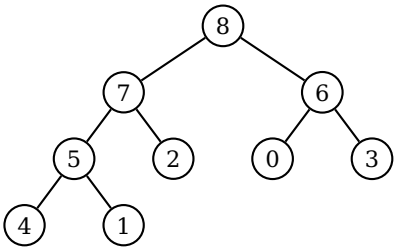
Après ajouts des mêmes éléments dans une file il faut retirer trois éléments (10, 20 puis 30) pour que la tête de pile (le prochain à sortir par retrait) puisse être le 40.

Correction de l'exercice 4.

1. Le tas obtenu par insertion successives est le suivant :

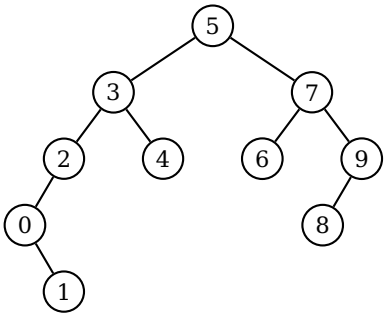


2. Curieusement on obtient le même tas par la méthode alternative mais ça n'est pas le cas en général.
3. Le tas obtenu après retrait du maximum :

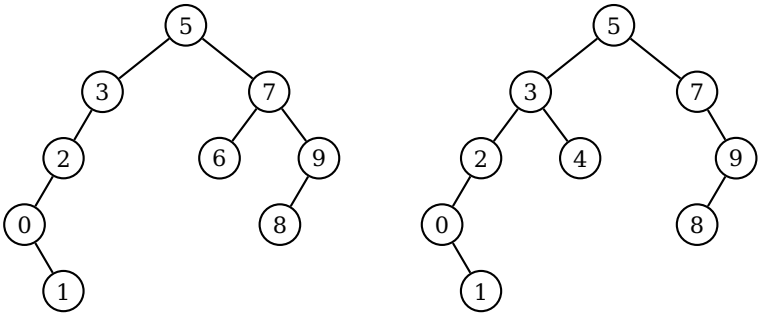


Correction de l'exercice 5.

1. L'arbre de recherche obtenu :



2. Deux possibilités en utilisant le prédécesseur (à gauche) et en utilisant le successeur (à droite) :



Correction de l'exercice 6.

```
int somme(ab_t x) {
  if (estVide(x)) {
```

```
    return 0;
  }
  return somme(gauche(x)) + somme(droite(x)) + x->e;
}
```

Correction de l'exercice 7.

Par ordre croissant : $\log n, n, n \log n, n^2$.

Exemples de réponses correctes :

La recherche dichotomique d'un élément dans un tableau ordonné de n éléments est asymtpotiquement en $\log n$ en pire cas.

La recherche du maximum dans un tableau de n éléments est asymptotiquement en n dans tous les cas, en particulier en pire cas.

Correction de l'exercice 8.

La même liste triée est : 18, 21, 30, 35, 45, 77, 90, de quatrième élément 35, 35 est donc l'élément de rang 4.

Correction de l'exercice 9.

Le plus simple est de trier le tableau avec un bon algorithme de tri (en $N \log N$ en pire cas), par exemple le tri par tas, et de renvoyer le k-ième élément du tableau trié (celui d'indice $k-1$ si le tableau commence à l'indice zéro comme en C). En pire cas le temps d'exécution sera asymtpotiquement équivalent à (donc en particulier minoré par) $N \log N$ puisque renvoyer le k-ième élément coûte un temps constant après le tri en $N \log N$.

Fonction Sélection(T, k)

TriParTas (T);
retourner $T[k-1]$;

Correction de l'exercice 10.

1. Si $k < p+1$ alors l'élément de rang k est plus petit que le pivot, cet élément se trouve dans la première partie du tableau (les éléments plus petits que le pivot), et si $k > p+1$ l'élément de rang k est plus grand que le pivot et se trouve dans la seconde partie du tableau (les éléments plus grands que le pivot).
2. Par ce qui précède pour rechercher l'élément de rang k (avec $1 \leq 0 \leq N$) il suffit de partitionner le tableau autour d'un pivot choisi aléatoirement et de comparer k à l'indice p du pivot :
   - lorsque $k = p+1$ l'algorithme termine et renvoie $T[p]$;
   - lorsque $k < p+1$ il faut recommencer récursivement à chercher l'élément de rang k dans le tableau $T[0 \ldots p-1]$;
   - lorsque $k > p+1$ il faut recommencer récursivement à chercher l'élément de rang $k-p-1$ dans le tableau $T[p+1 \ldots N-1]$.
3. En pseudo-code ou en C :

Fonction Sélection(T, k)

$p =$ Partitionner$(T, 0)$;
si $p + 1 = k$ **alors**
  retourner $T[p]$;
si $k < p + 1$ **alors**
  retourner Sélection ($T[0 \dots p - 1]$, k);
si $k > p + 1$ **alors**
  retourner Sélection ($T[p + 1 \dots$ Taille$(T) - 1]$, $k - p - 1$);

```
element_t selection(element_t t[], int taille, int k) {
  int p;
  p = partitionner(t, taille, 0);
  if (k == p + 1) return t[p];
  if (k < p + 1) return selection(t, p, k);
  if (k > p + 1) return selection(t + p + 1, taille - p - 1, k - p - 1);
}
```

4. En C :

```
int partitionner(element_t t[], int taille, int k) {
  int p = 0;
  int i;
  element_t tmp; /* pour les echanges */
  if (k != 0) { /* echange de t[k] et t[0] */
     tmp = t[k];
     t[k] = t[0];
     t[0] = tmp;
  }
  /* on partitionne. Invariant de boucle :
     - de 1 à p inclu les éléments sont plus petits que le pivot
     - de p + 1 à k - 1 inclu ils sont plus grands
     - au dela c'est l'inconnu */
  for (i = 1; i < taille; i++) {
    if (t[0] > t[i]) {
      p++;
      /* echange de t[i] et t[p] */
      tmp = t[i];
      t[i] = t[p];
      t[p] = tmp;
    }
  }
  /* on met le pivot a sa place (echange entre t[0] et t[p]) */
  tmp = t[0];
  t[0] = t[p];
  t[p] = tmp;
  return p;
}
```

Correction de l'exercice 11.

Fonction Selection(r, k)

si EstVide(r) **alors**
  retourner N ;
si EstVide(Gauche(r)) **alors**
  $p = 0$
sinon
  $p =$ Total(Gauche(r)) ;
si $k = p + 1$ **alors**
  retourner r ;
si $k < p + 1$ **alors**
  retourner Sélection(Gauche(r), k) ;
si $k < p + 1$ **alors**
  retourner Sélection(Droite(r), $k - p - 1$) ;

Correction de l'exercice 12.

1. L'ajout d'un élément dans un ABR fonctionne comme la recherche de cet élément, avec une descente le long d'une branche de l'arbre, sauf que lorsqu'il n'est pas trouvé il faut créer un nœud pour l'ajouter. Si l'élément est déjà dans l'arbre, on peut choisir de ne pas l'ajouter (pour éviter les répétitions) ou de l'ajouter tout de même n'importe où. En cohérence avec le cours nous utilisons la première solution, mais alors pour la mise à jour du compteur des descendants (le total) il faut attendre de savoir si l'élément est réellement ajouté ou non. Ainsi il y a plusieurs solutions correctes à cet exercice : soit ajouter systématiquement le nouveau nœud, quitte à produire une répétition, soit attendre de savoir si le nœud a été réellement ajouté pour mettre à jour les compteurs (la solution adoptée ici), soit encore mettre à jour à la descente en supposant que seul des nouveaux éléments sont ajoutés dans l'arbre.

   Solution itérative en C (les nœuds possèdent un champ `total` pour le comptage).

```
/* inserer(&x, a) insere un élément a dans un arbre binaire de
   recherche x avec comptage des descendants.
   S'il existe déjà un élément b égal à a dans l'arbre
   aucun noeud n'est créé (pas de répétition).  La fonction inserer()
   ne renvoie rien, l'arbre x est passé par adresse (px = adresse de
   l'arbre) pour être directement modifié (argument-résultat). */

void inserer(abr_t *px, element_t e) {
  abr_t x, parent;
  int cote;
  if ( !(*px) ) {
    *px = nouveau_noeud(e);
    return;
  }
  x = *px;
  /* Recherche de la place de a (parent et côté où le mettre) */
  while ( x ) {
    parent = x;
    switch ( cote = comparer(e, x->e) ) {
    case 1:
      x = x->gauche;
      break;
    case -1:
```

```
      x = x->droite;
      break;
    case 0:
      return; /* pas de répétition dans l'arbre,
                 et on ne met pas à jour total */
    }
  }
  /* parent sera le parent du nouveau noeud, la valeur de cote dit de
       quel côté l'attacher à son parent.  */
  x = nouveau_noeud(e);
  x->parent = parent;
  if (cote == 1) {
    parent->gauche = x;
  }
  else {
    parent->droite = x;
  }
  /* mise à jour du total */
  x->total = 1;
  while ( x = x ->parent ) { /* tant que le parent existe */
    x->total++;
  }
}
```

2. La donnée est l'adresse du parent du nœud qui vient d'être supprimé. Le plus simple est d'appliquer une remontée vers la racine avec décrement de 1 du total pour chaque nœud traversé :

```
void maj_suppression(abr_t x) {
  while ( x ) {
    x->total--;
    x = x->parent;
  }
}
```