Institut Galilée — Algorithmique, arbres et graphes
Année 2005-2006 — L2

Examen du jeudi 8 juin 2006

Aucun document autorisé – pas de calculatrice

Le barème (uniquement indicatif) n'est pas directement proportionnel à la difficulté des questions ou au temps nécessaire pour y répondre.

Première partie : questions de cours

10 pt

Comme dans le cours, on considère des éléments qui sont fait d'une clé et de données satellites. On considère ici que les clés sont des entiers. Dans les arbres et dans les tableaux que l'on représente on se contente de donner les clés des éléments sans faire apparaître les données satellites associées.

1 Notation asymptotique, tris

Exercice 1.

1. *Si* $f(n) = 10n + 100$*, est-ce que* $f(n) = O(n)$ *?* $f(n) = O(n \log n)$ *?* $f(n) = \Theta(n^2)$ *?* $f(n) = \Omega(\log n)$ *? (Répondre par oui ou par non, sans justification.)* (0.5 pt) 5
2. *En notation asymptotique quelle est la borne minimale en temps des tris par comparaison, en pire cas et en moyenne ?* (0.5 pt) 10

2 Piles, files, tas

Exercice 2.

1. *Si, partant d'une pile p vide, on ajoute (en empilant), les entiers* 1*, puis* 2*, puis* 3*, puis* 4*, puis* 5 *et que, ensuite, on supprime (par dépilement) deux éléments, quels entiers contient la pile ?* (0.5 pt) 15
2. *Même question avec une file (utiliser les fonctions d'ajout et de suppression des files à la place de l'empilement et du dépilement).* (0.5 pt) 20

Exercice 3 (Insertion / suppression)**.** *On se donne le tas max (ou maximier) de la figure 1. En utilisant les algorithmes vus en cours :*

1. *Insérer un élément de clé* 44 *dans ce tas. Répondre en représentant le nouveau tas.* (0.5 pt) 25
2. *En repartant du tas initial, supprimer l'élément de clé maximale. Répondre en représentant le nouveau tas.* (0.5 pt) 30

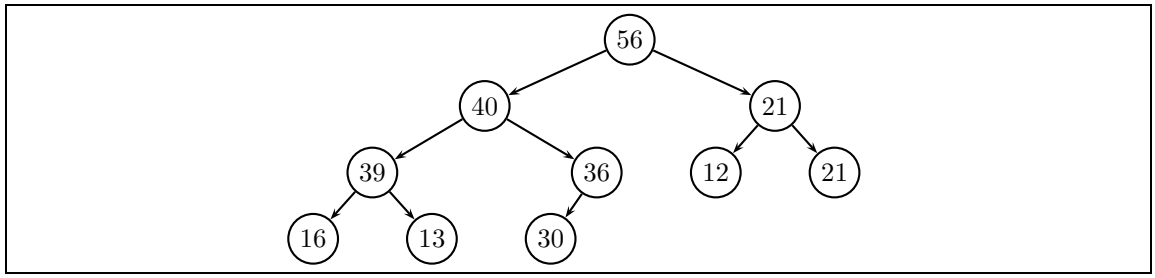


Fig. 1: Tas

3 Arbres binaires de recherche (ABR)

Exercice 4 (Insertion)**.** *Décrire en quelques lignes le principe de l'algorithme d'insertion d'un élément de clé c dans un arbre binaire de recherche.* Attention : lorsque la clé c est déjà présente dans l'arbre, il y a le choix entre deux possibilités. Dans cet exercice, fixer votre choix (toujours à gauche ou bien toujours à droite) et conserver cette convention pour tout l'examen. (1 pt) 40

Exercice 5 (Commutativité de l'insertion)**.** *L'opération d'insertion d'éléments dans un arbre binaire de recherche est-elle une opération « commutative » au sens où l'insertion de x puis de y dans un arbre binaire de recherche produit le même arbre que l'insertion de y puis de x ? Si oui, dire pourquoi, si non donner un contre exemple.* (1 pt) 50

Exercice 6 (Insertion)**.** *Dessiner l'arbre binaire de recherche obtenu par insertions successives des éléments de clés* $24, 16, 32, 45, 11, 12, 28, 26, 32$ *en partant de l'arbre vide.* (1 pt) 60

Exercice 7. *Les opérations de recherche d'insertion et de suppression dans les ABR se font en temps $O(h)$ où h est la hauteur de l'arbre c'est à dire le nombre maximum de nœuds sur une branche. En pire cas, que vaut cette hauteur en fonction du nombre n d'éléments de l'arbre ? Donner, pour tout n, une suite d'insertions (juste les clés) telle que l'arbre binaire de recherche obtenu réalise ce pire cas.* (1 pt) 70

4 Rotations, arbres rouge noir

Exercice 8. *Rappel : une rotation doit préserver la propriété des arbres binaires de recherche.*

1. *Compléter la partie droite de la figure 2, avec les noms des différents éléments (x, y) et sous-arbres (A, B, C).* Vous pouvez répondre sur le sujet. (0.5 pt) 75
2. *Dans la figure 3, quelles série de rotations (centre et sens) peut-on utiliser pour faire remonter à la racine le nœud 32, fils droit de 24 ?* (0.5 pt) 80

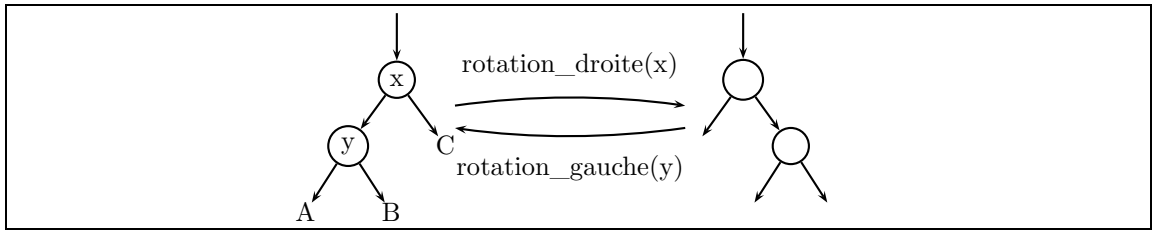


Fig. 2: Rotations gauche et droite

Exercice 9. *Colorier tous les nœuds de l'arbre binaire de recherche de la figure 3 pour en faire un arbre rouge noir.* Vous pouvez répondre sur le sujet. (1 pt) 90

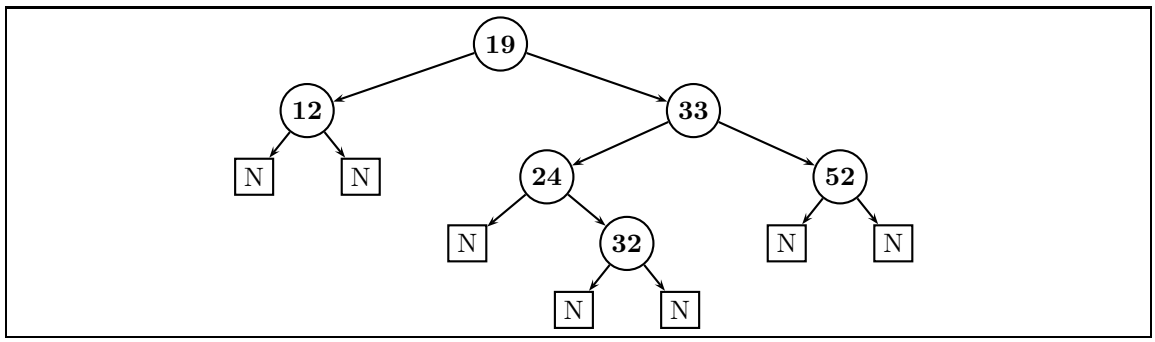


Fig. 3: Coloriage

Exercice 10. *Pour chaque arbre de la figure 4, dire s'il s'agit d'un arbre rouge noir. Si non, pourquoi ?* (1 pt) 100

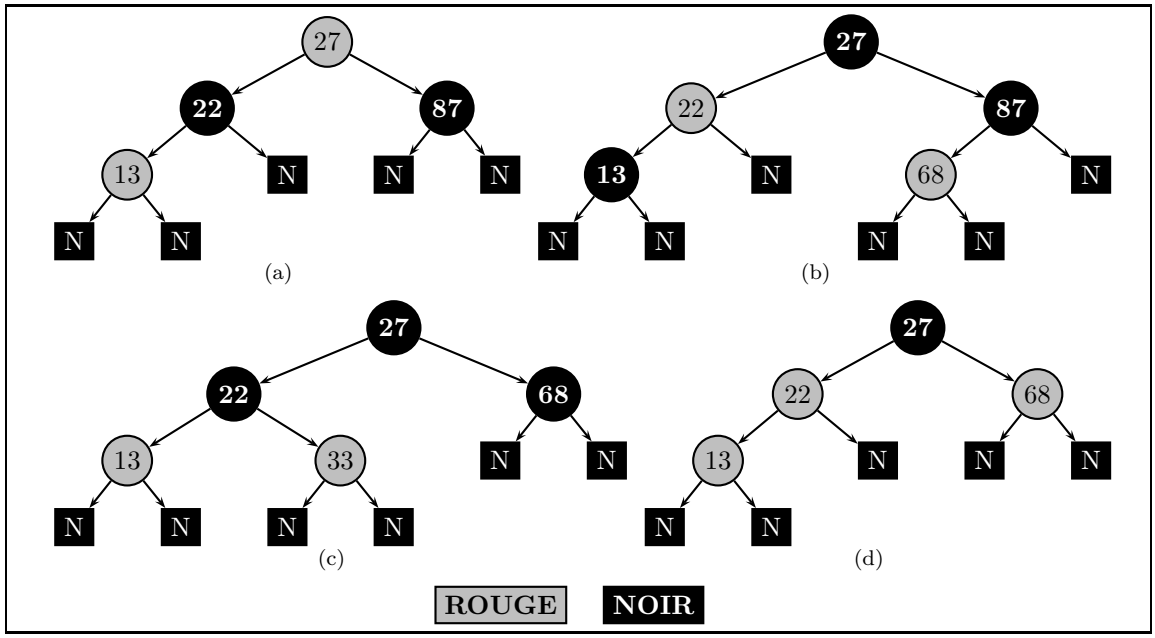


Fig. 4: Rouge noir ?

Seconde partie

10 pt

On pourra considérer que le type des éléments est donnée par le code :

```
typedef struct {   /* Un objet de type element_t est donné par :    */
  int cle;         /* - une clé (= un entier)                       */
  data_t donnee;   /* - et une donnée satellite (type non détaillé) */
} element_t;
```

Pour représenter les arbres binaires de recherche ont peut utiliser le type :

```
typedef struct noeud_s {  /* Un noeud est la donnée :                        */
  element_t e;            /*  - d'un élément                                 */
  struct noeud_s *parent; /*  - d'un pointeur vers un noeud parent           */
  struct noeud_s *gauche; /*  - d'un pointeur vers un noeud fils gauche      */
  struct noeud_s *droite; /*  - d'un pointeur vers un noeud fils droit       */
} noeud_t, * abr_t;       /* Un ABR est un pointeur sur un noeud (la racine) */
```

Exercice 11. *Rappel : la fonction de recherche des arbre binaire de recherche prend en argument une clé c et un pointeur vers la racine d'un arbre binaire de recherche A et rend un des éléments stockés dans A dont la clé est c.*

1. *Écrire le code de la fonction de recherche en C (algorithme vu en cours).* (1 pt) 110

Lorsque qu'il y a plusieurs éléments de clé c dans l'arbre, la fonction renvoie un seul de ces éléments, e.

2. *En supposant qu'on n'a effectué que des insertions et des suppressions, est-ce que l'élément e renvoyé est le dernier élément de clé c a avoir été inséré dans A, le premier élément de clé c a avoir été inséré dans A, ou ni l'un ni l'autre ? Justifier votre réponse (par un raisonnement ou un contre-exemple).* (2 pt) 130

3. *Écrire une fonction de recherche qui prend en entrée un arbre binaire de recherche A et une clé c et affiche tous les éléments de A de clé c. Pour l'affichage d'un élément, on se donne une fonction* `void elt_affiche(element_t e)`. (1 pt) 140

Exercice 12. *Même question qu'à l'exercice 5 pour les commutations entre la suppression et l'insertion. L'insertion et la suppression dans les ABR sont elles des opérations qui « commutent » au sens où la suppression de x puis l'insertion de y dans un arbre binaire de recherche produit le même arbre que l'insertion de y puis la suppression de x ? Si oui, dire pourquoi, si non donner un contre exemple.* (2 pt) 160

Exercice 13 (À la fois ABR et tas ?)**.** *Un tas est nécessairement un arbre binaire quasi-complet. Est-il toujours possible d'organiser un ensemble de n clés en tas max de manière à ce que cet arbre binaire soit aussi un arbre binaire de recherche ? (Justifier par un raisonnement ou un contre-exemple).* (1 pt) 170

Exercice 14 (Partition d'un ABR – difficile ! –)**.** *On veut écrire une fonction de partition d'un arbre binaire de recherche autour d'un élément. Plus précisément, étant donné*

- *un arbre binaire de recherche A*
- *et un élément x de clé c,*

on veut obtenir deux ABR, G et D, tels que

- *G contient tous les éléments de A de clé inférieure à c*
- *et D tous les éléments de A de clé supérieure à c.*

Un élément de A de clé c sera placé indifféremment soit dans G soit dans D (dans un premier temps on peut considérer qu'il n'y a pas deux clés semblables).

On veut que la fonction de partition fasse le moins d'opérations possibles. On évitera en particulier l'algorithme consistant à parcourir la liste de tous les éléments de A.

Pour simplifier l'écriture en C, on peut considérer que la fonction reçoit A ainsi qu'un arbre binaire B à un seul nœud contenant l'élément x et rend son résultat en faisant de G le sous arbre gauche de B et de D le sous-arbre droit de B (remarque : l'arbre B obtenu est un arbre binaire de recherche dont les éléments sont ceux de A plus l'élément x, placé à la racine).

1. *Décrire un algorithme de partition (on pourra l'expliquer sur un exemple), et si possible écrire la fonction C (on pourra utiliser des fonctions intermédiaires).* (2.5 pt) 195
2. *Donner une majoration asymptotique de son temps d'exécution en fonction de la hauteur de l'arbre A.* (0.5 pt) 200

Correction de la première partie

Correction 1.

1. *On a* $f(n) = O(n)$, $f(n) = O(n \log n)$, $f(n) \neq \Theta(n^2)$ *et* $f(n) = \Omega(\log n)$. (0.5 pt) 5
2. *En pire cas, comme en moyenne les tris par comparaison font (au moins)* $\Omega(n \log n)$ *comparaisons.* (0.5 pt) 10

Correction 2.

1. *La pile contient* 1, 2, 3 *(de la base au sommet).* (0.5 pt) 15
2. *La file contient* 3, 4, 5 *(dans cet ordre).* (0.5 pt) 20

Correction 3.

1. *Voir figure 5(a).* (1 pt) 30
2. *Voir figure 5(b).* (1 pt) 40

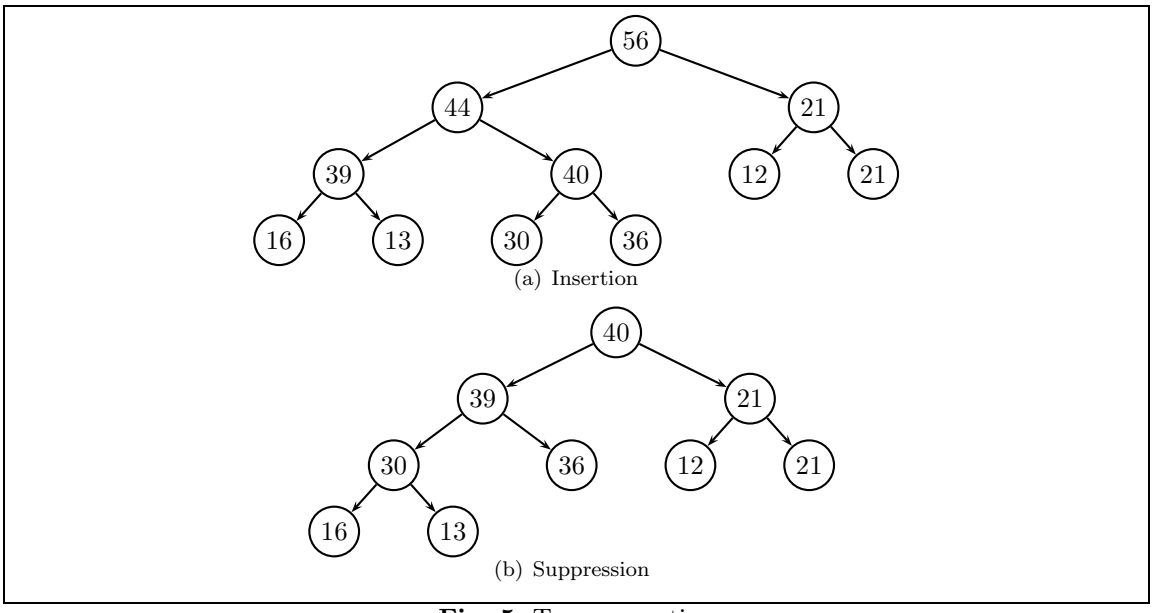


Fig. 5: Tas : correction

Correction 4. *Pour insérer un élément e de clé x dans un ABR A : si A est vide alors on rend l'ABR à un seul nœud contenant e ; sinon on compare x avec y la clé de la racine de A, si x <= y, respectivement si x > y, on insère x dans le sous-arbre de gauche de A, respectivement le sous-arbre de droite de A, et on rend le nouveau A obtenu.* (1 pt) 50

Correction 5. *L'insertion n'est pas commutative :considérons l'insertion de* 1 *puis* 2 *dans un arbre vide et l'insertion de* 2 *puis* 1 *on obtient un arbre binaire de recherche différent dans chaque cas (figure).* (1 pt) 60

Correction 6. *Voir figure 7.* (1 pt) 70

Correction 7. *En pire cas la hauteur est linéaire en n (réponse suffisante) et, plus précisément, elle est égale à n. En effet, quel que soit n, l'arbre binaire de recherche obtenu par insertions successives des n premiers entiers est un peigne à droite de hauteur n.* (1 pt) 80

Correction 8.

1. *Correction en bleu sur la figure 8.* (0.5 pt) 85
2. *Rotation gauche de centre 24 ; rotation droite de centre 33 ; rotation gauche de centre 19.* (0.5 pt) 90

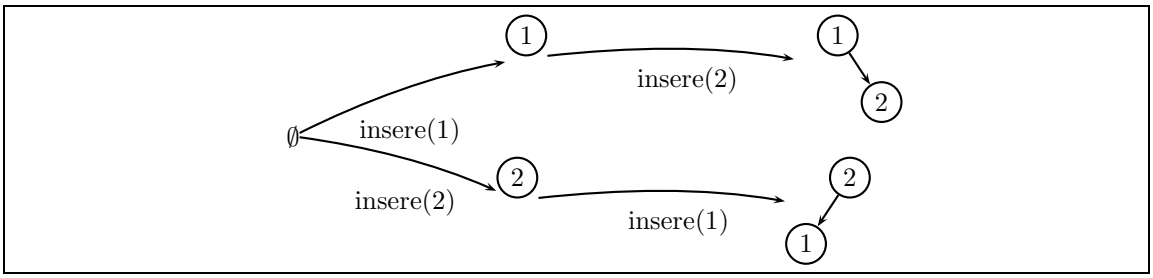


Fig. 6: Contre-exemple à la commutativité de l'insertion dans les ABR

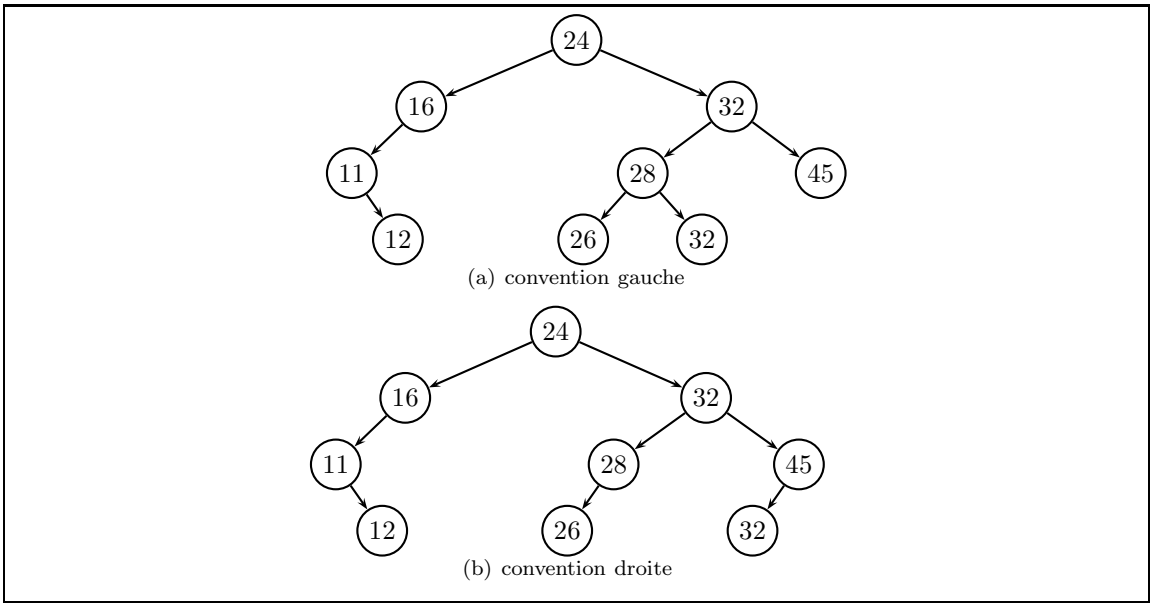


Fig. 7: Corrigé arbre binaire de recherche (insertion)

Correction 9. *Une seule solution, celle de la figure 9.* (1 pt) 100

Correction 10. *Aucun de ces arbres n'est un arbre rouge noir : (a) n'a pas la racine noire ; dans (b) le nombre de nœuds noirs sur chaque branche de la racine aux feuilles n'est pas constant, il est de* 3 *sur la première branche (la plus à gauche) et de* 2 *sur la troisième branche ; (c) n'est pas un arbre binaire de recherche (33 est plus grand que 27) ; dans (d) un nœud rouge n'a pas ses deux fils noirs.* (2 pt) 120

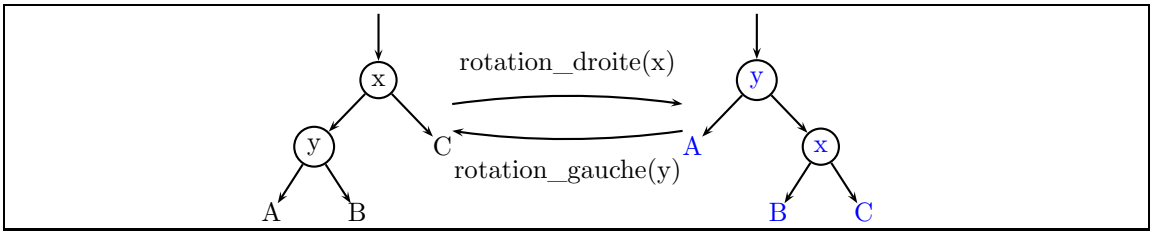


Fig. 8: Rotations gauche et droite

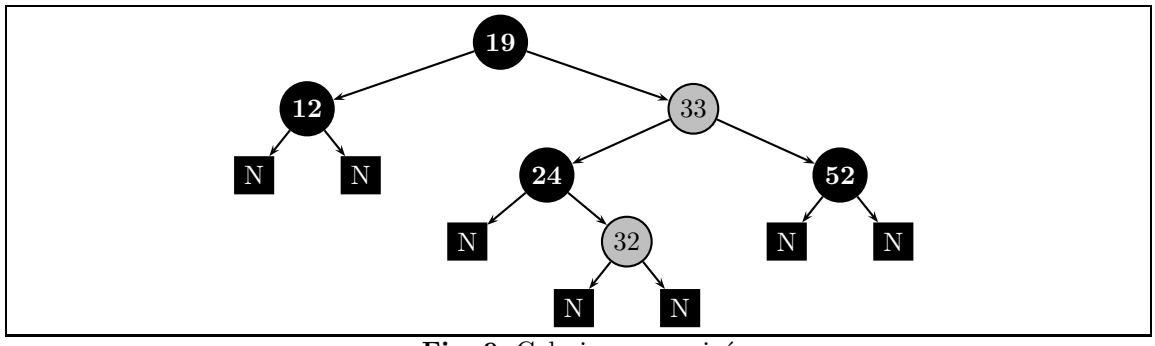


Fig. 9: Coloriage – corrigé

Correction de la seconde partie

Correction 11.

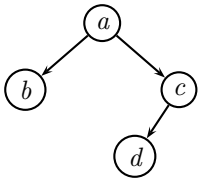
1. *Code :* (1 pt) 130

```
noeud_t * rechercheAbr(abr_t A, int c){
/* on rend le noeud contenant l'element de cle c...          */
  if ( A == NULL) return NULL; /* ou NULL s'il n'y en a pas  */
  if ( c == (A->e)->cle ) return A;
  if ( c < (A->e)->cle ) return rechercheAbr(A->gauche, c);
  if ( c > (A->e)->cle ) return rechercheAbr(A->droite, c);
}
```

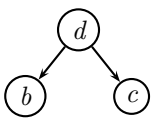
2. On a changé l'énoncé en début d'examen pour séparer la question en deux : (a) on considère uniquement les insertions, (b) on considère aussi les suppressions.

   a. *Si on ne fait que des insertions, l'élément rendu est toujours le premier élément de clé c inséré dans l'arbre. Considérons la première fois qu'on a inséré un élément de clé c et appelons cet élément x. Toute les insertions suivantes d'éléments de clé c se font sous le nœud contenant x. L'élément x est donc le premier élément de clé c rencontré lors de la recherche.* (1.5 pt) 145

   b. *Si on accepte aussi les suppressions, la réponse est ni l'un ni l'autre. Voici un contre-exemple, en prenant la convention de l'insertion à gauche (il s'adapte facilement par symétrie à un contre-exemple pour la convention à droite). Considérons l'insertion dans l'arbre vide d'un élément a de clé 1 puis d'un élément b de clé 0 et de deux éléments c, puis d de clé 2. On obtient l'arbre*



   Lorsqu'on supprime a comme il a deux fils on prend soit son successeur d soit son prédecesseur b et on l'échange avec a avant de pouvoir supprimer a. Dans le cas on c'est le successeur, on obtient alors l'arbre :



   dans lequel la recherche d'un élément de clé 2 donnera d et non c. (1.5 pt) 160

3. *Code :* (1.5 pt) 175

```
void rechercheAbr2(abr_t A, int c){
  if ( A == NULL) return;
  /* si on tient un element de cle c on l'affiche
     et on relance la procédure sur chacun de ses sous-arbres */
  if ( c == (A->e)->cle ) {
    elt_affiche(A->e);
    rechercheAbr2(A->gauche, c);
    rechercheAbr2(A->droite, c);
  }
  /* sinon on part du "bon" cote ! */
  if ( c < (A->e)->cle ) rechercheAbr2(A->gauche, c);
  if ( c > (A->e)->cle ) rechercheAbr2(A->droite, c);
}
```

Correction 12. *Réponse : non. Voici, dans la figure 10 un contre exemple, où on insère* 2 *et on retire* 3 *(on utilise le prédecesseur pour la suppression, si on utilisait le successeur on aurait un contre-exemple similaire).* (2 pt) 195

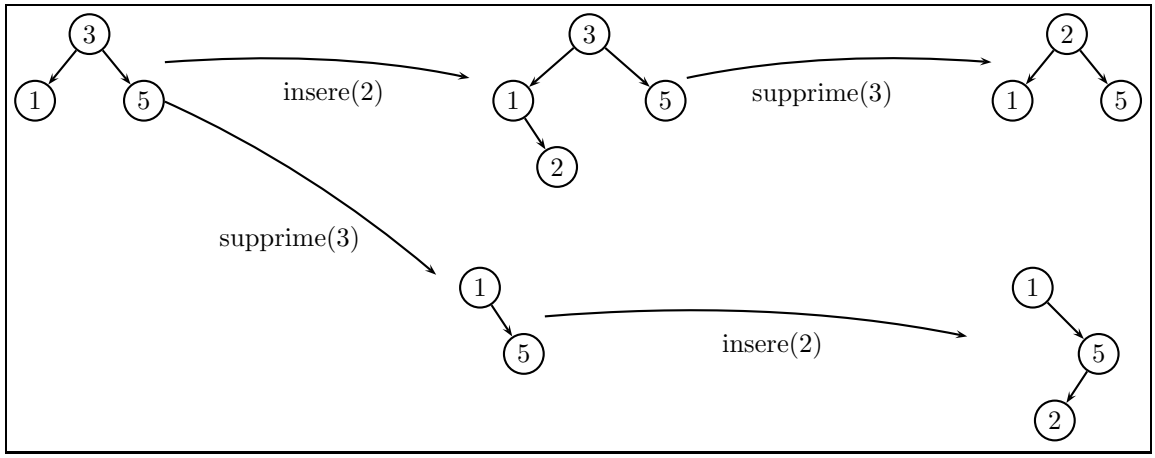


Fig. 10: Contre exemple commutation insertion / suppression

Correction 13. *Réponse non. Il suffit de prendre trois clés différentes. Un arbre binaire quasi-complet les contenant est un arbre A fait d'une racine a d'un fils gauche b et d'un fils droit c. Puisque cet arbre doit être un tas max on doit avoir $a \geq b$ et $a \geq c$. Par ailleurs, A devant être un arbre binaire de recherche on doit avoir $b \leq a$ et $a \leq c$. D'où $a = c$ ce qui est impossible (on a choisi trois clés différentes).* (1.5 pt) 210

Correction 14.

1. *Le plus simple est d'effectuer une insertion de x dans A, puis de faire remonter x à la racine par une série de rotations.* (3 pt) 240

```
void partition1(abr_t A, abr_t B){
  noeud_t *p = NULL;
  int c;

  /* On se debarrasse du cas trivial */
  if (A == NULL) return B;

  /***********************************************/
  /* Insertion de x (l'unique element de B) dans A */
  /***********************************************/
```

```
  c = (B->e)->cle; /* c = la cle de x */
  while ( A != NULL ){
    p = A;
    if ( c <= (A->e)->cle ) A = A->gauche;
    else A = A->droit;
  }
  if ( c <= (p->e)->cle ) p->gauche = B;
  else p->droit = B;
  B->parent = p;

  /*****************************/
  /* Remontee de x a la racine  */
  /*****************************/
  while ( p != NULL ){
    /* Attention : on considere que la rotation echange les contenus
       des noeuds, pas leurs adresses, sinon il faut adapter */
    if ( B == p->gauche ){
      rotation_droite(p); /* p reste le noeud du haut, son contenu change */
    }
    else rotation_gauche(p); /* idem, p reste le noeud du haut */
    B = p;
    p = p->parent;
  }
}
```

2. *Le temps d'exécution est de l'ordre de deux fois le parcours de la racine de A à une feuille, il est donc en $O(h)$.* (0.5 pt) 245