Algorithmique
Correction Partiel n$^{\mathrm{o}}$ 2

Info-Sup – Epita

8 juin 2011 - 09 :00

Solution 1 **(Arbre 234 - Propriétés et ...-** ***10 points*****)**

1. Propriétés d'un arbre 2.3.4 :
   - Chaque nœud a exactement 2, 3 ou 4 fils.
   - Chaque nœud contient 1, 2 ou 3 clés ordonnées telles que $x_1 < x_2 < x_3$.
   - Tous les éléments du premier sous-arbre sont inférieurs à x_1
   - Tous les éléments du i^{ième} sous-arbre (i=2,3) sont strictement supérieurs à x_{i-1} et inférieurs à x_i.
   - Tous les éléments du dernier sous-arbre sont strictement supérieurs à x_3.
   - Toutes les feuilles sont au même niveau.

2. L'insertion d'une clé se fait aux feuilles. L'insertion pose un problème si le nœud concerné est un 4-nœud.

3. La technique pour résoudre ce problème est l'éclatement (voir Fig. **??**). La procédure consiste à créer un nouveau nœud (`nœud_234`) et de transférer la dernière clé avec ses deux fils gauche et droit vers ce nouveau nœud. L'ancien nœud garde uniquement la première clé et ces deux fils gauche et droit. La clé médiane est insérée dans le nœud père avec l'ancien nœud comme fils gauche et le nouveau nœud comme fils droit de cette clé. Cette transformation ne peut se faire que si le nœud père n'est pas plein (nombre de clés < 3) : dans ce cas, il suffit de réorganiser le nœud pour insérer la nouvelle clé au bon endroit .

4. Deux modes d'utilisation :

   a) Insertion avec éclatement à la remontée :

   Dans cette version, on cherche la feuille concernée par l'insertion de la clé. Si la feuille dans laquelle on veut insérer est un 4-nœud, on l'éclate. Un problème se pose lorsque le nœud père est déjà plein (un 4-nœud) : dans ce cas, la procédure d'éclatement est appliquée au nœud père pour créer de la place pour la nouvelle clé. On effectue les éclatement des nœuds à la remontée tant qu'on rencontre des 4-nœuds. Les éclatements peuvent ainsi se propager jusqu'à la racine !

   b) Insertion avec éclatement à la descente :

   Dans cette version, on est plus prévoyant. En descendant dans l'arbre à la recherche de la feuille concernée par l'insertion, on éclate tous les nœuds pleins (nombre de clés = 3) situés sur notre chemin. De cette façon, quand on est sur un nœud, on est sûr que son père contient de la place pour une éventuelle insertion d'une clé médiane issue de l'éclatement du nœud courant. Cette prévoyance engendre des éclatements qui s'avèrent inutiles pour l'insertion de certaines clés.
   Cette méthode est la plus pratique à implémenter.

   Notons que l'éclatement de la racine ne se comporte pas tout à fait comme un éclatement d'un nœud interne et doit donc être traité à part. De plus, le cas où l'arbre initial est nul est aussi un cas à part.

5. Avec éclatement à la remontée :
   Pour cette méthode, nous avons pour l'ajout des valeurs 4, 11, 9 et 18 les arbres successifs des figures 1, 2, 3 et 4.

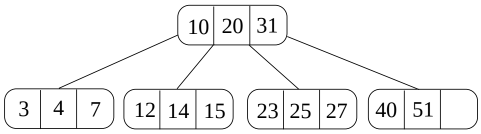

FIG. 1 – Insertion de 4 avec éclatement à la remontée

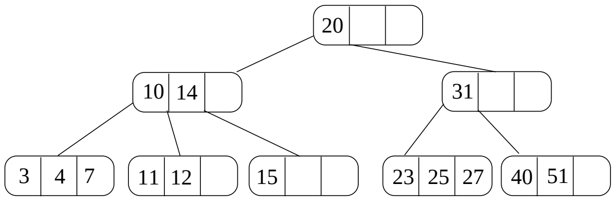

FIG. 2 – Insertion de 11 avec éclatement à la remontée

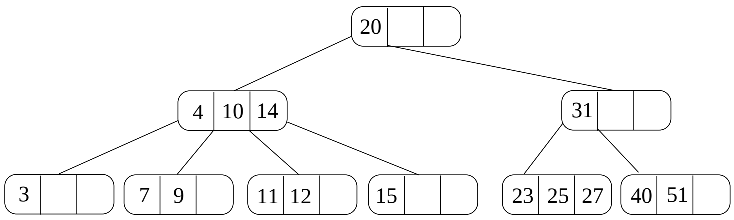

FIG. 3 – Insertion de 9 avec éclatement à la remontée

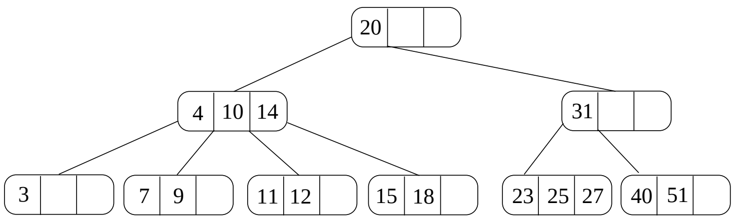

FIG. 4 – Insertion de 18 avec éclatement à la remontée

6. Avec éclatement à la descente :
   Pour cette méthode, nous avons pour l'ajout des valeurs 4, 11, 9 et 18 les arbres successifs des figures 5, 6, 7 et 8.

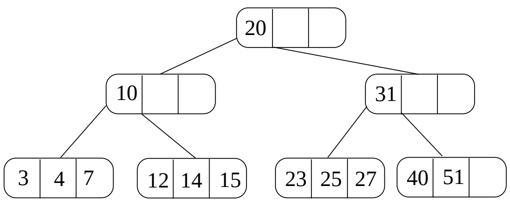

FIG. 5 – Insertion de 4 avec éclatement à la descente

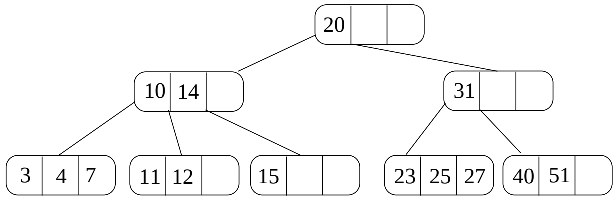

Fig. 6 – Insertion de 11 avec éclatement à la descente

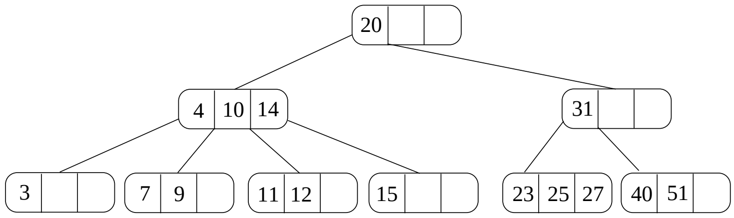

Fig. 7 – Insertion de 9 avec éclatement à la descente

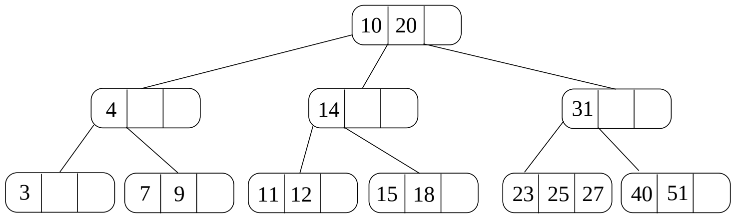

Fig. 8 – Insertion de 18 avec éclatement à la descente

7. Représentation bicolore de l'arbre234 :
   En considérant les 3-nœuds penchés à droite, on obtient l'arbre bicolore de la figures 9.

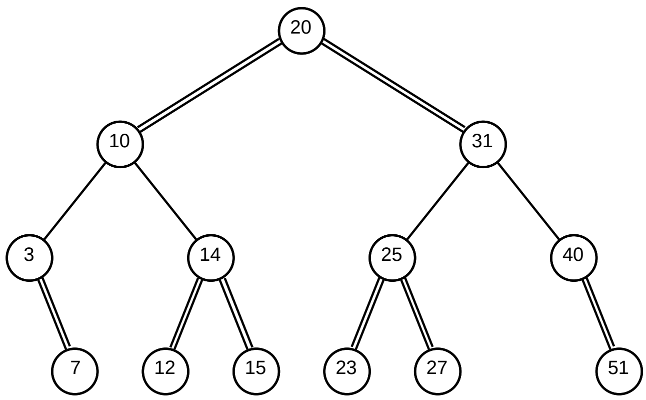

Fig. 9 – Arbre bicolore

8. Les propriétés d'un arbre bicolore sont les suivantes :

   – c'est un ABR.
   – Un nœud est soit rouge soit noir.
   – La racine est noire.
   – Si un nœud est rouge, ses deux descendants sont noirs.
   – Le père d'un nœud rouge est noir.
   – Chaque branche contient le même nombre de nœuds noirs.
   – sa hauteur est majorée par Log_2^n pour n éléments

Solution 2 **(La taille en plus –** ***6 points*****)**

Spécifications :

La fonction `copie (t_arbreBinaire` B`, t_AB_taille` A`) : entier` construit A, copie de B avec les tailles renseignées, et retourne la taille de l'arbre construit.

Principe :

- Si B est vide, alors l'arbre A est vide, sa taille est 0.
- Sinon,
  - on crée un nouveau nœud qui sera la racine de A, ayant pour valeur celle de la racine de B ;
  - le rappel de la fonction sur le fils gauche de B permet d'obtenir le fils gauche de A, copie du premier, ainsi que sa taille ; le fils droit et sa taille sont obtenus de la même manière ;
  - les deux tailles ainsi obtenues permettent de donner à A sa taille : la somme des deux + 1.

```
algorithme fonction transf_taille : entier
    parametres locaux
        t_AB_taille    B
    parametres globaux
        t_AB_taille    A
debut
    si B = NUL alors
        A ← NUL
        retourne (0)
    sinon
        allouer (A)
        A↑.cle ← B↑.cle
        A↑.taille ← 1 + transf_taille (B↑.fg, A↑.fg)
                      + transf_taille (B↑.fd, A↑.fd)
        retourne (A↑.taille)
    fin si
fin algorithme fonction transf_taille
```

Solution 3 **(Ajout avec mise à jour de la taille –** ***7 points*****)**

1. (a) **Principe :**
   Pour ajouter l'élément x à l'ABR B :
   – Si B est vide, alors on le remplace par un arbre dont la racine contient x et les fils sont vides ;
   – sinon, on compare x à la valeur contenue dans la racine de B : si cette dernière est strictement inférieure à x, alors l'ajout se fera dans le fils droit de B, dans le fils gauche dans le cas contraire.

   (b) **Les nœuds à mettre à jour :** sont ceux de la branche contenant la nouvelle feuille (chaque nœud rencontré sur le chemin jusqu'à la nouvelle feuille voit sa taille incrémentée).

2. **Spécifications :**
   La fonction `ajout_feuille (t_element` x`, t_AB_taille` B`)` ajoute x en feuille de l'ABR B et retourne le nouvel arbre.

```
algorithme fonction ajout_feuille : t_AB_taille
    parametres locaux
        t_element    x
        t_AB_taille  B

debut
    si B = NUL alors
        allouer (B)
        B↑.cle ← x
        B↑.fg ← NUL
        B↑.fd ← NUL
        B↑.taille ← 1
    sinon
        B↑.taille ← B↑.taille + 1       /* un nouvel élément dans B */
        si x <= B↑.cle alors
            B↑.fg ← ajout_feuille (x, B↑.fg)
        sinon
            B↑.fd ← ajout_feuille (x, B↑.fd)
        fin si
    fin si
    retourne B
fin algorithme fonction ajout_feuille
```

Solution 4 **(Médian – *7 points*)**

1. **Méthode :**

(a) *Définition abstraite de l'opération taille :*

Opérations
taille : ArbreBinaire → Entier

Axiomes
taille (arbre-vide) = 0
taille (<o, G, D>) = 1 + *taille* (G) + *taille* (D)

(b) *Définition abstraite des opérations kieme et Médian :*

Opérations
kieme : ArbreBinaire × Entier → Nœud
médian : ArbreBinaire → Nœud

Préconditions
kieme (A, k) est-défini-ssi $1 \leq k \leq$ *taille*(A)
médian (A) est-défini-ssi A ≠ arbre-vide

Axiomes
$k = taille(G)+1 \Rightarrow kieme(<o, G, D>, k) = o$
{si G contient k-1 élément alors le $k^{ème}$ est la racine}
$1 \leq k \leq taille(G) \Rightarrow kieme(<o, G, D>, k) = kieme(G, k)$
$taille(G)+1 < k \leq taille(<o, G, D>) \Rightarrow kieme(<o, G, D>, k) = kieme(D, k-taille(G)-1)$
$A \neq \text{arbre-vide} \Rightarrow médian(A) = kieme(A, (taille(A)+1) \text{ div } 2)$

Avec
A, G, D : ArbreBinaire
o : Nœud
k : entier

2. **Algorithme :**

Spécifications :

La fonction `kieme` prend en paramètres un ABR B (de type `t_AB_taille`) non vide et un entier k tel que $1 \leq k \leq taille(B)$. Elle retourne le pointeur sur le k^{ième} élément (dans l'ordre croissant) de B.

```
algorithme fonction kieme : t_AB_taille
    parametres locaux
        t_AB_taille   B
        entier        k

    variables
        entier        tailleG
debut
    si B↑.fg = NUL alors
        tailleG ← 0
    sinon
        tailleG ← B↑.fg↑.taille
    fin si

    si tailleG = k-1 alors
        retourne B
    sinon
        si k <= tailleG alors
            retourne kieme (B↑.fg, k)
        sinon
            retourne kieme (B↑.fd, k-tailleG-1)
        fin si
    fin si
fin algorithme fonction kieme
```