Algorithmique
Correction Partiel n° 2 (P2)

Info-sup (s2) – Epita

9 juin 2015 - 10:00

Solution 1 **(Arbre 234 - Propriétés et insertions -** ***4 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^{e}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.

4. 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.

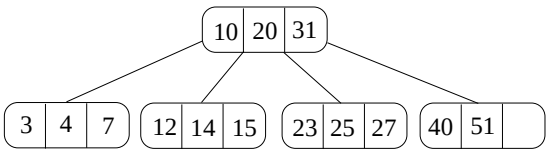


Figure 1 – Insertion de 4 avec éclatement à la remontée

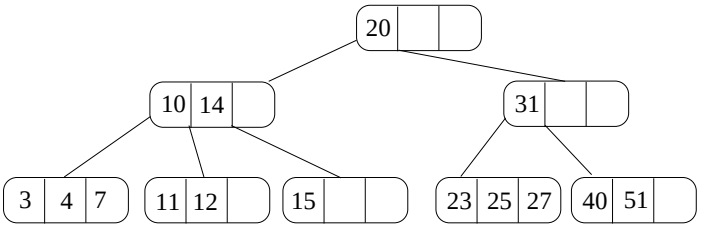


Figure 2 – Insertion de 11 avec éclatement à la remontée

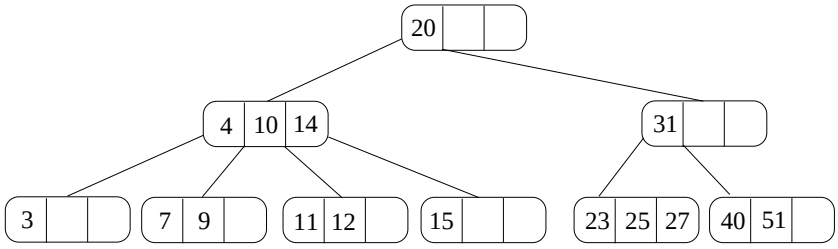


Figure 3 – Insertion de 9 avec éclatement à la remontée

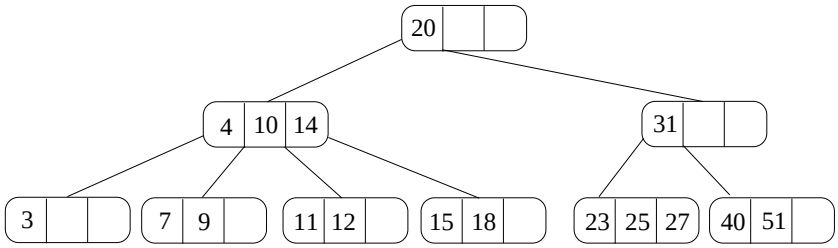


FIGURE 4 – Insertion de 18 avec éclatement à la remontée

Solution 2 **(Arbre binaire et code préfixe –** ***7 points*****)**

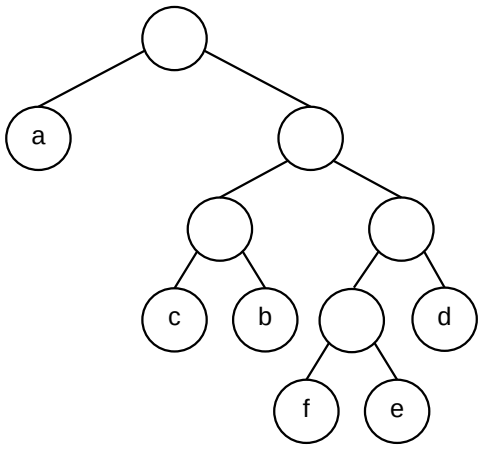
1. Code :

lettre	a	b	c	d	e	f
code	0	101	100	111	1101	1100

   1101110011000100 1101 décodé donne : `e f f a c e`

2. Le code d'une lettre correspond à l'**occurrence de la feuille** la contenant.

3. L'arbre correspondant au code de la question 1 :



4. **Spécifications :**
   La procédure `call_print_code (t_arbreBinaire` B`, caractere` c`)` affiche le code correspondant au caractère c dans l'arbre B s'il existe. Elle affiche "no code found" sinon.

```
algorithme fonction print_code : booleen
    parametres locaux
        t_arbreBinaire  B
        caractere   c
        chaine      code

debut
    si B↑.fg = B↑.fd alors      /* leaf */
        si B↑.cle = c alors
            ecrire (code)
            retourne vrai
        sinon
            retourne faux
        fin si
    sinon
        si print_code (B↑.fg, c, code + '0') alors
            retourne vrai
        sinon
            retourne print_code (B↑.fd, c, code + '1')
        fin si
    fin si
fin algorithme fonction print_code

algorithme procedure call_print_code
    parametres locaux
        t_arbreBinaire B
        caractere   c
debut
    si B = NUL ou non print_code (B, c, "") alors
        print ("no code found")
    fin si
fin algorithme procedure call_print_code
```

---

version "moche", sans optimisation : procédure

```
algorithme procedure print_code_moche
    parametres locaux
        t_arbreBinaire  B
        caractere   c
        entier      code

debut
    si B↑.fg = B↑.fd alors      /* leaf */
        si c = B↑.cle alors
            ecrire (B↑.cle, " : ", code)
        fin si
    sinon
        print_code_moche (B↑.fg, c, code+'0')
        print_code_moche(B↑.fd, c, code+'1')
    fin si
fin algorithme procedure print_code_moche

algorithme procedure call_print_code_moche
    parametres locaux
        t_arbreBinaire B
debut
    si B <> NUL alors
        print_code_moche (B, c, "")
    fin si
fin algorithme procedure call_print_code_moche
```

Solution 3 **(Tas – *2 points*)**

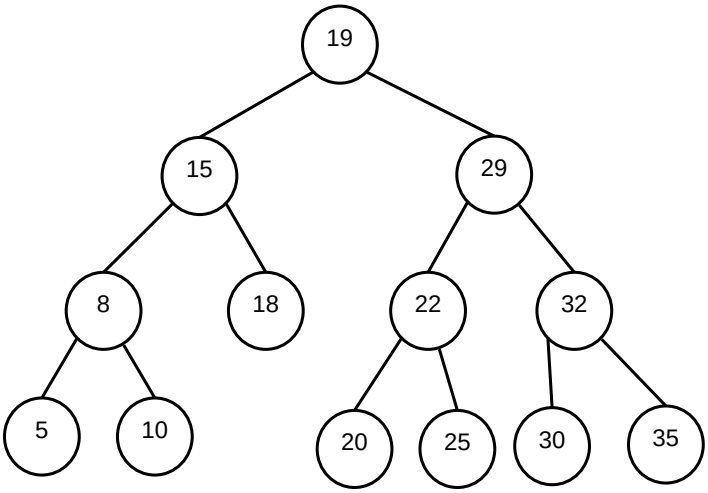
1. Les 3 opérations :
   — suppression du minimum
   — ajout de 7
   — ajout de 10
2. Dans quel ordre ?
   — ajout de 7
   — ajout de 10
   — suppression du minimum

Solution 4 **(AVL - Suppression du minimum – *8 points*)**

1. Rotations et changements de hauteur après suppression du minimum :

deseq(racine)	*deseq(fils droit)*	rotation	Δh
-2	-1	rg	1
	0		0
	1	rdg	1

2. Arbre après suppression du minimum :



3. **Spécifications :** La fonction `supp_min_avl (`A`, `min`)` effectue la suppression du nœud contenant la valeur minimale de l'AVL A non vide, valeur affectée à min et retourne un booléen indiquant si l'arbre a changé de hauteur.

```
algorithme fonction supp_min_avl : booleen
    parametres globaux
        t_avl        A      /* A ≠ NUL */
        t_element    min

    variables
        t_avl    T
debut
    si A↑.fg = NUL alors
        min ← A↑.cle
        T ← A
        A ← A↑.fd
        liberer (T)
        retourne vrai
    sinon
        si non supp_max_avl (A↑.fg, min) alors
            retourne faux
        sinon
            A↑.deseq ← A↑.deseq - 1
            si A↑.deseq = -2 alors
                si A↑.fd↑.deseq = 1 alors
                    rdg (A)
                sinon
                    rg (A)
                fin si
            fin si
            retourne (A↑.deseq = 0)
        fin si
    fin si
fin algorithme fonction supp_min_avl
```

Version "longue" :

```
algorithme fonction supp_min_avl : booleen
    parametres globaux
        t_avl        A      /* A ≠ NUL */
        t_element    min

    variables
        t_avl    T
debut
    si A↑.fg = NUL alors
        min ← A↑.cle
        T ← A
        A ← A↑.fd
        liberer (T)
        retourne vrai
    sinon
        si non supp_min_avl (A↑.fg, min) alors
            retourne faux
        sinon
            A↑.deseq ← A↑.deseq - 1
            selon A↑.deseq faire
                 0 : retourne vrai
                -1 : retourne faux
                -2 : selon A↑.fd↑.deseq faire
                        -1 : rg (A)
                             retourne vrai
                         0 : rg (A)
                             retourne faux
                         1 : rdg (A)
                             retourne vrai
                     fin selon
            fin selon
        fin si
    fin si
fin algorithme fonction supp_min_avl
```