Algorithmique
Correction Contrôle n° 2

Info-Sup – Epita

24 mar 2011 - 10 :00

Solution 1 **(ABR : chemin de recherche –** ***2 points*****)**

Les séquences ② et ③ sont impossibles :

① 18, on descend à droite - 27, on descend à droite - 60 on descend à gauche - 47, on descend à gauche - 29, on descend à droite - **42**

② 57, on descend à gauche - 18, on descend à droite - 53, on descend à droite - **55, on descend à gauche - 48, n'est pas supérieur à 53**

③ 30, on descend à droite - 38, on descend à droite - 40, on descend à droite - 48, on descend à droite - **50, on descend à gauche - 42 n'est pas supérieur à 48**

④ 36, on descend à droite - 46, on descend à gauche - 38, on descend à droite - 44, on descend à gauche - 40, on descend à droite - **42**

Solution 2 **(ABR : insertions –** ***3 points*****)**

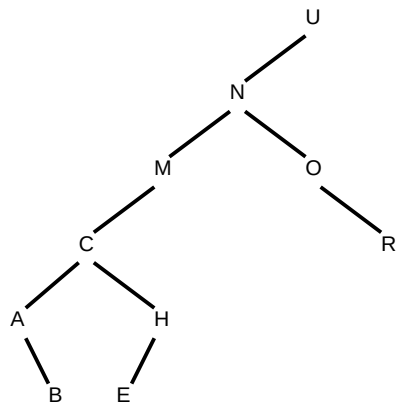

Fig. 1 – Un moche ABR (en feuille)

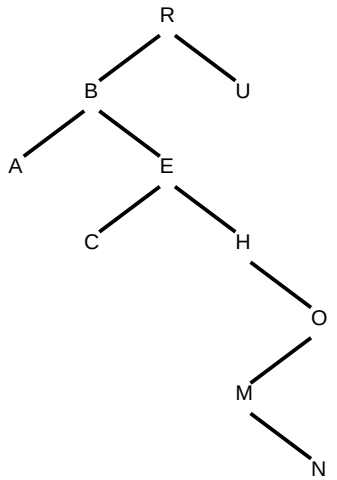

Fig. 2 – Un moche ABR (en racine)

Solution 3 **(ABR : recherche du minimum –** ***4 points*****)**

1. *La valeur minimum d'un ABR se trouve :* en bas du bord gauche.
   Principe :
   Il suffit de parcourir le bord gauche : tant que le fils gauche n'est pas vide, on descend à gauche. Lorsqu'on arrive à un noeud sans fils gauche, il contient la valeur minimale de l'ABR.

2. *Version itérative :*

```
algorithme fonction cherche_min : Element
parametres locaux
    ArbreBinaire B      /* non vide */
debut
    tant que g(B) <> arbre-vide faire
        B ← g(B)
    fin tant que
    retourne (contenu (racine (B)))
fin algorithme fonction cherche_min
```

Version récursive :

```
algorithme fonction cherche_min : Element
parametres locaux
    ArbreBinaire B      /* non vide */
debut
    si g(B) = arbre-vide alors
        retourne (contenu (racine (B)))
    sinon
        retourne (cherche_min (g(B)))
    fin si
fin algorithme fonction cherche_min
```

Solution 4 **(Listes chaînées - Croissance – *5 points*)**

a. *Version itérative*

Spécifications :

La fonction `est_croissante (L)` détermine si la liste L, de type `t_pListe`, est strictement croissante.

Principe :

Si la liste est non vide, on parcourt la liste tant que l'élément courant est inférieur ou égal à l'élément suivant (s'il existe !). Si le parcours atteint la fin de la liste, ou si elle était vide, alors elle est croissante.

```
algorithme fonction est_croissante : booleen
    parametres locaux
        t_pListe    L

debut
    si L = NUL alors
        retourne vrai
    sinon
        tant que (L↑.suivant <> NUL) et (L↑.valeur <= L↑.suivant↑.valeur) faire
            L ← L↑.suivant
        fin tant que
        retourne (L↑.suivant = NUL)
    fin si
fin algorithme fonction est_croissante
```

b. *Version récursive*

Spécifications :

La fonction `est_croissante (L)` détermine si la liste L, de type `t_pListe`, est croissante (au sens large).

Principe :

Une liste vide ou à un seul élément est croissante. Si la liste a au moins deux éléments, elle est croissante si son premier élément est inférieur au deuxième et si la liste privée de sa tête est croissante.

```
algorithme fonction test_rec : booleen
    parametres locaux
        t_pListe    L

debut
    si L↑.suivant = NUL alors
        retourne vrai
    sinon
        retourne (L↑.valeur < L↑.suivant↑.valeur) et test_rec (L↑.suivant)
    fin si
fin algorithme fonction test_rec

algorithme fonction est_croissante : booleen
    parametres locaux
        t_pListe    L

debut
    retourne (L = NUL) ou  test_rec (L)
fin algorithme fonction est_croissante
```

Solution 5 **(Listes chaînées : suppression –** ***6 points*****)**

Spécifications :

La fonction `supprime (x, L)` supprime la première occurrence de la valeur x dans la liste triée L. Elle retourne un booléen indiquant si la suppression a eu lieu.

Principe :

- On recherche la valeur : on avance dans la liste tant qu'il reste des éléments et qu'on n'a pas atteint ou dépassé x, tout en conservant un pointeur sur l'élément précédent.
- Si on a trouvé x :
  - soit on est en tête de liste : la nouvelle tête est l'élément suivant,
  - soit on est au milieu (ou à la fin) : le suivant du précédent est le suivant de l'élément à supprimer !

```
algorithme fonction supprime : booleen
    parametres locaux
        t_element    x
    parametres globaux
        t_pListe     L

    variables
        t_pListe     p, prec

debut
   /* recherche avec conservation du précédent */
    p ← L
    tant que (p <> NUL) et (x > p↑.valeur) faire
        prec ← p
        p ← p↑.suivant
    fin tant que

   /* suppression si trouvé */
    si (p <> NUL) et  (x = p↑.valeur) alors
        si p = L alors          /* suppression de la tête */
                L ← L↑.suivant
        sinon
                prec↑.suivant ← p↑.suivant
        fin si
        liberer (p)
        retourne (vrai)
    sinon
        retourne (faux)
    fin si
fin algorithme fonction supprime
```