Algorithmique
Correction Partiel n° 2

Info-Sup – Epita

5 juin 2012

Solution 1 **(Arbres Bicolores et Arbres 2.3.4. –** ***6 points*****)**

1. L'arbre 2.3.4. est celui de la figure 1

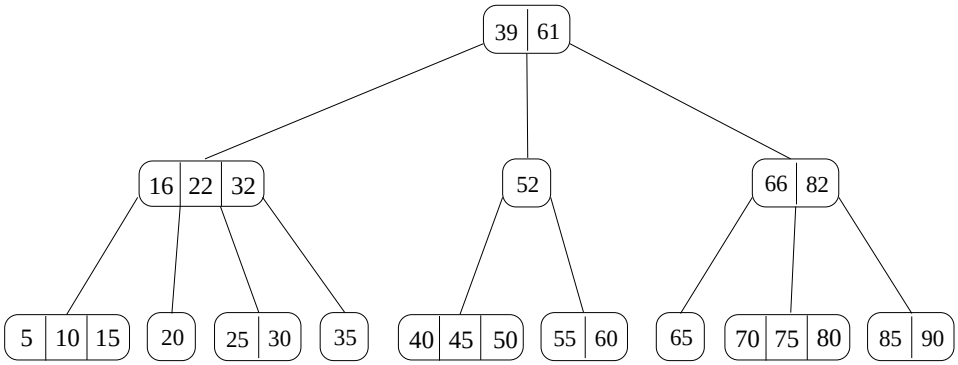

Figure 1 – Arbre 2.3.4.

2. Compter les noeuds noirs contenus dans n'importe quelle branche de l'arbre bicolore.

   En effet, on sait que l'arbre bicolore correspondant à un arbre 2.3.4. possède entre la racine et chaque feuille le même nombre de noeuds noirs. Ce nombre est égal à la hauteur de l'arbre 2.3.4. puisqu'il correspond à la notion d'ascendance dans l'arbre 2.3.4. (en opposition à la gémellité).

3. Compter tous les noeuds noirs de l'arbre bicolore.

   En effet, on sait qu'il y a une bijection noeudNoir-noeud2.3.4. puisque les noeuds noirs correspondent à la notion d'ascendance dans l'arbre 2.3.4. (en opposition à la gémellité). Il suffit donc de les compter dans l'arbre bicolore pour connaître le nombre de noeud 2.3.4. dans l'arbre 2.3.4.

Solution 2 **(AVL et mesures... -** ***9 points*****)**

1. $2^{h+1} - 1$.
   Un arbre binaire complet est un AVL (chaque noeud est d'équilibre 0). Son nombre de noeuds est le maximum pour une hauteur h, donc $2^{h+1} - 1$.

2. Les arbres T_0, T_1, T_2 et T_3 avec un nombre de noeuds minimum pourrait être ceux de de la figure 2

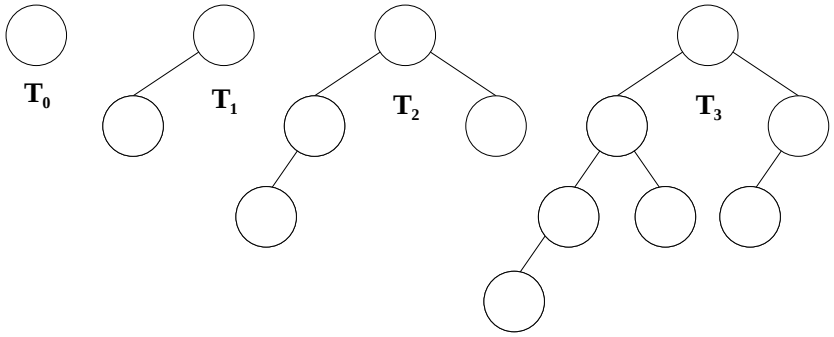

FIGURE 2 – AVL minimaux.

3. Soit A un AVL de hauteur $h > 1$ quelconque, $g(A)$ et $d(A)$ sont des AVL de hauteur$\leqslant h-1$. La condition d'équilibre implique que soit ils sont tous les deux de $h-1$, soit l'un est de hauteur $h-1$ et l'autre $h-2$. Si $A = T_h$, ses sous-arbres gauche et droit ont un nombre de noeuds minimal par rapport à leur hauteur. Maintenant, deux arbres minimaux de hauteur $h-1$ ont plus de noeuds qu'un arbre minimal de hauteur $h-1$ et un de hauteur $h-2$. Les deux sous-arbres de T_h sont donc T_{h-1} et T_{h-2}

4. On en déduit que :
$$N(T_h) = 1 + N(T_{h-1}) + N(T_{h-2})$$
D'où $F_h = F_{h-1} + F_{h-2}$ avec $F_0 = 1, F_1 = 2$. F_h est la suite de Fibonacci et les arbres T_h sont appelés arbres de Fibonacci.

Solution 3 **(En long – *5 points*)**

Ce que j'avais en tête : Voici les versions j'avais en tête en posant cet exercice.

Spécifications :
La procédure `copie (t_arbreBinaire` B`,` C`)` effectue une copie de B dans C en inversant en chaque nœud de l'arbre les fils gauche et droit. L'arbre B est détruit.

```
algorithme procedure copie
  parametres globaux
      t_arbreBinaire  B, C

debut
  si B = NUL alors
      C ← NUL
  sinon
      allouer (C)
      C↑.cle ← B↑.cle
      copie (B↑.fd, C↑.fg)
      copie (B↑.fg, C↑.fd)
      liberer (B)
      B ← NUL
  fin si
fin algorithme procedure copie
```

Version fonction :

Spécifications :
La fonction `copie (t_arbreBinaire B)` retourne une copie de B en inversant en chaque nœud de l'arbre les fils gauche et droit. L'arbre B est détruit.

```
algorithme fonction copie : t_arbreBinaire
    parametres globaux
        t_arbreBinaire  B

    variables
        t_arbreBinaire  C
debut
    si B = NUL alors
        retourne NUL
    sinon
        allouer (C)
        C↑.cle ← B↑.cle
        C↑.fg ← copie (B↑.fd)
        C↑.fd ← copie (B↑.fg)
        liberer (B)
        B ← NUL
        retourne C
    fin si
fin algorithme fonction copie
```

Beaucoup mieux : Voici une version différente (inspirée des versions trouvées sur les copies), qui ne construit pas de nouvel arbre, mais se contente d'échanger les deux fils.

Spécifications :
La fonction `swap_fils (t_arbreBinaire B)` modifie B en inversant en chaque nœud de l'arbre ses fils gauche et droit, et retourne l'arbre modifié.

```
algorithme fonction swap_fils : t_arbreBinaire
   parametres locaux
      t_arbreBinaire  B

   variables
      t_arbreBinaire T

debut
   si B <> NUL alors
      T ← B↑.fg
      B↑.fg ← swap_fils (B↑.fd)
      B↑.fd ← swap_fils (T)
   fin si
   retourne B
fin algorithme fonction swap_fils
```

Solution 4 **(En large – *5 points*)**

Schéma du parcours :

parcours en largeur, avec lequel on repère les changements de niveau :
- on commence par enfiler la racine, suivie d'une marque (NUL si l'arbre est représenté par un pointeur)
- tant que la file n'est pas vide, on défile le premier élément :
  - si c'est une marque de changement de niveau : une autre marque est enfilée si la file n'est pas vide ;
  - sinon on enfile les fils non vides du nœud défilé.

Le calcul de la hauteur se réduira donc à compter le nombre de marques de changement de niveau. Pour le nombre de feuilles, il suffit de compter, à chaque nœud défilé, si c'est une feuille.

Spécifications :

La procédure `hauteur_feuilles (t_arbreBinaires` B`, entier hauteur, nbFeuilles)` calcule la hauteur et le nombre de feuilles de l'arbre B.

```
algorithme procedure hauteur_feuilles
   parametres locaux
      t_arbreBinaire   B
   parametres globaux
      entier   hauteur, nb_feuilles

   variables
      t_file    f
debut
   nb_feuilles ← 0
   hauteur ← -1
   si B <> NUL alors
      f ← enfiler (B, file-vide ())
      f ← enfiler (NUL, f)      /* marque de changement de niveau */
      faire
         B ← defiler (f)
         si B = NUL alors
            hauteur ← hauteur + 1
            si non est_vide (f) alors
               f ← enfiler (NUL, f)
            fin si
         sinon
            si B↑.fg = B↑.fd alors
               nb_feuilles ← nb_feuilles + 1
            sinon
               si B↑.fg <> NUL alors
                  f ← enfiler (B↑.fg, f)
               fin si
               si B↑.fd <> NUL alors
                  f ← enfiler (B↑.fd, f)
               fin si
            fin si
         fin si
      tant que non est_vide (f)
   fin si
fin algorithme procedure hauteur_feuilles
```

***Solution 5* (En travers... – *5 points*)**

Opérations

nb_inter : ArbreBinaire × Entier × Entier → Entier

Axiomes

nb_inter (arbre-vide, inf, sup) = 0
r < inf ⇒ *nb_inter* (<r, G, D>, inf, sup) = *nb_inter* (G, inf, sup)
r > sup ⇒ *nb_inter* (<r, G, D>, inf, sup) = *nb_inter* (D, inf, sup)
inf ≤ r ≤ sup ⇒
nb_inter (<r, G, D>, inf, sup) = 1 + *nb_inter* (G, inf, sup) + *nb_inter* (D, inf, sup)

Spécifications :

La fonction `nb_inter (t_arbreBinaire` B`, entier` inf`,` sup) retourne le nombre d'éléments de B dans l'intervalle $[inf, sup]$.

```
algorithme fonction nb_inter : entier
   parametres locaux
      t_arbreBinaire B
      t_element   inf, sup

debut
   si B = NUL alors
      retourne 0
   sinon
      si B↑.cle < inf alors
         retourne nb_inter (B↑.fd, inf, sup)
      sinon
         si B↑.cle > sup alors
            retourne nb_inter (B↑.fg, inf, sup)
         sinon
            retourne 1 + nb_inter (B↑.fg, inf, sup) + nb_inter (B↑.fd, inf, sup)
         fin si
      fin si
   fin si
fin algorithme fonction nb_inter
```