Algorithmique
Partiel n° 2

Info-Sup – Epita

D.S. 312431.16 BW (7 juin 2011 - 09 :00)

Consignes (à lire) :

- Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - Aucune réponse au crayon de papier ne sera corrigée.
- La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 (le barème est effectivement sur 30, mais la note sera ramenée à 20) et que les points de présentation (2 au maximum) sont retirés de cette note.
- **Les algorithmes :**
  - Tout algorithme doit être écrit dans le langage Algo (pas de C#, Caml ou autre).
  - Tout code Algo non indenté ne sera pas corrigé.
  - Tout ce dont vous avez besoin (types, routines) est indiqué en **annexe** (dernière page) !
- Durée : 3h.

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

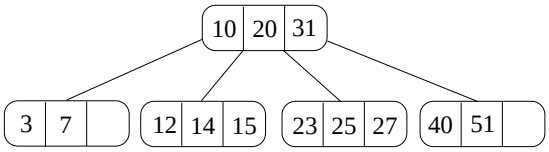

FIG. 1 – Arbre 2.3.4.

1. Rappeler trois propriétés d'un arbre 2.3.4.

2. Quel problème pose une insertion classique dans un arbre 2.3.4. ?

3. Quelle technique utilise-t-on pour le résoudre ?

4. Quelles sont les deux modes d'utilisation de cette technique ?

5. Insérer successivement les clés 4, 11, 9 et 18 sur l'arbre de la figure 1 en utilisant le premier mode *(Dessiner les trois arbres 2.3.4. (après chaque ajout) ainsi que l'arbre final).*

6. Recommencer les insertions sur l'arbre de la figure 1 avec le second mode*(Dessiner les trois arbres 2.3.4. (après chaque ajout) ainsi que l'arbre final).*

7. Représenter l'arbre de la figure 1 en bicolore. Vous considérerez les 3-noeuds penchés à droite.

8. Rappelez quatre propriétés des arbres bicolores.

Type à utiliser pour les exercices suivants

On modifie l'implémentation classique des arbres binaires (rappelée en annexe) de façon à ce que chaque nœud indique la taille[1] de l'arbre dont il est racine. Le type `t_AB_taille` correspondant est donné en annexe.

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

Après avoir donné son principe, écrire une fonction qui construit à partir d'un arbre binaire classique (de type `t_arbreBinaire`) un arbre équivalent au premier (contenant les mêmes valeurs aux mêmes places) mais avec le champ *taille* renseigné en chaque nœud (de type `t_AB_taille`). La fonction retournera de plus la taille de l'arbre.

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

1. (a) Donner le principe général (indépendant de toute représentation) de la version récursive de l'ajout en feuille dans un arbre binaire de recherche (avec redondance).
   (b) Dans le cas d'un arbre *avec la taille en plus*, quels sont les nœuds dont la taille doit être mise à jour ?

2. Écrire un algorithme **récursif** qui ajoute en feuille un élément dans un arbre binaire de recherche représenté par le type `t_AB_taille`. L'algorithme doit de plus mettre à jour, lorsque nécessaire, le champ *taille* en chaque nœud de l'arbre.

   Remarque : si vous ne voyez pas comment mettre à jour la taille, écrivez au moins la version "normale" de l'ajout en feuille dans un ABR !

[1] Taille désignant, comme chacun le sait, le nombre de nœuds de l'arbre.

Exercice 4 (Médian – *7 points*)

On s'intéresse à la recherche du nœud médian d'un arbre binaire de recherche B, c'est à dire celui qui, dans la liste des éléments en ordre croissant, se trouve à la place $(taille(B) + 1)\ DIV\ 2$.

Pour cela, on va écrire une fonction `kieme (B, k)` qui retourne le nœud contenant le $k^{ième}$ élément de l'ABR B (dans l'ordre des éléments croissants). Par exemple, l'appel à `kieme(B, 3)` avec B l'arbre de la figure 2 nous retournera le nœud contenant la valeur 5.

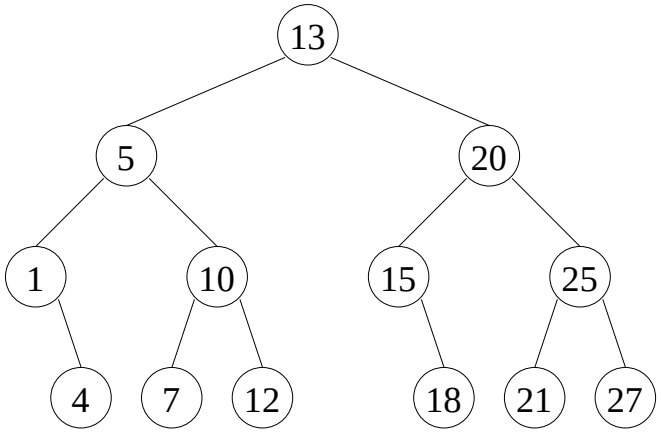

FIG. 2 – Arbre Binaire de Recherche

1. **Méthode :** Le but ici est de déterminer le principe de la fonction `kieme (B, k)` (avec $1 \leq k \leq taille(B)$), qui sera à écrire à la question 2.

   (a) On ajoute à la définition abstraite des arbres binaires (donnée en annexe) l'opération *taille*. Donner la définition abstraite de cette opération.

   (b) Donner la définition abstraire de l'opération *kieme* (utilisant obligatoirement l'opération *taille*) : compléter les définitions abstraites données.
   Indice : comparer k à la taille du fils gauche...

2. **Algorithme :**
   On donne la fonction suivante :

   Spécifications :
   La fonction `median` prend en paramètre un ABR B (de type `t_AB_taille`) et retourne le pointeur sur l'élément médian de B, la valeur NUL si ce dernier est vide.

```
algorithme fonction median : t_AB_taille
    parametres locaux
        t_AB_taille    B
debut
    si B = NUL alors
        retourne (NUL)
    sinon
        retourne (kieme (B, (B↑.taille + 1) div 2))
    fin si
fin algorithme fonction median
```

   Écrire la fonction `kieme` en utilisant le principe de la question 1.

Annexe

Type algébrique abstrait d'un arbre binaire :

SORTE
ArbreBinaire

UTILISE
Nœud, Élément

OPÉRATIONS

arbre-vide	:	→ ArbreBinaire
<–, –, –>	:	Nœud × ArbreBinaire × ArbreBinaire → ArbreBinaire
racine	:	ArbreBinaire → Nœud
contenu	:	Nœud → Élément
g	:	ArbreBinaire → ArbreBinaire
d	:	ArbreBinaire → ArbreBinaire

PRÉCONDITIONS
racine(B_1) **est-défini-ssi** $B_1 \neq$ *arbre-vide*
g(B_1) **est-défini-ssi** $B_1 \neq$ *arbre-vide*
d(B_1) **est-défini-ssi** $B_1 \neq$ *arbre-vide*

AXIOMES
racine(<o, B_1, B_2>) = o
g(<o, B_1, B_2>) = B_1
d(<o, B_1, B_2>) = B_2

AVEC
B_1, B_2 : ArbreBinaire
o : Nœud

Implémentation dynamique des arbres binaires

Classique :

```
types
    /* déclaration du type t_element */
    t_arbreBinaire = ↑ t_noeudBinaire
    t_noeudBinaire = enregistrement
        t_element        cle
        t_arbreBinaire   fg, fd
    fin enregistrement t_noeudBinaire
```

Avec le champ *taille* :

```
types
    t_AB_taille = ↑ t_noeud_bin_taille
    t_noeud_bin_taille = enregistrement
        t_element    cle
        t_AB_taille  fg, fd
        entier       taille
    fin enregistrement t_noeud_bin_taille
```