

Algorithmique 1

Feuille 4 : Arbres binaires

On considère le type abstrait `arbreBinaire` d'objet avec les primitives suivantes :

```
arbreBinaire= sommet
```

```
fonction créerArbreBinaire(val Racine:objet):sommet;
```

```
fonction getValeur(val S:sommet):objet;  
fonction filsGauche(val S:sommet):sommet;  
fonction filsDroit(val S:sommet):sommet;  
fonction pere(val S:sommet):sommet;
```

```
fonction setValeur(ref S:sommet, val x:objet):vide;  
fonction ajouterFilsGauche(ref S:sommet, val x:objet):vide;  
fonction ajouterFilsDroit(ref S:sommet, x:objet):vide;  
fonction supprimerFilsGauche(ref S:sommet):vide;  
fonction supprimerFilsDroit(ref S:sommet):vide;  
fonction detruireSommet(ref S:sommet):vide;
```

Exercice 4.1 *Parcours*

Soit l'arbre binaire dont les feuilles sont étiquetées avec les nombres naturels illustré sur Fig.1.

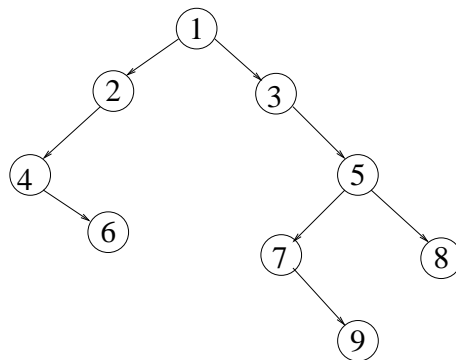


FIG. 1 – Arbre binaire

1. Donner les mots correspondants resp. à son parcours préfixe, infixé et postfixé.
2. Existe-t-il un arbre binaire à 8 sommets dont le mot préfixe est 12345678 et le mot suffixe est
 - (a) 53247681 ?
 - (b) 43527681 ?

3. Donner le principe d'un algorithme pour reconstruire un arbre binaire à partir de ces mots préfixe et suffixe.

Exercice 4.2 *Parcours hiérarchique*

Soit le tableau d'entiers `Tab={0, 2, 3, 4, 6, 7, 9, 10, 12, 13, 14, 16, 20}` et la fonction `remplirTableauArbre` vue en cours. Dessiner l'arbre binaire produit par l'appel à `remplirTableauArbre(Tab)`.

Exercice 4.3 *Parcours en profondeur*

Utiliser un parcours en profondeur pour écrire les fonctions :

1. `compter (val A : arbreBinaire): entier` qui compte le nombre de feuilles dans A.
2. `hauteur (val A : arbreBinaire): entier` qui calcule la hauteur de A.
3. `valeur_minimum(val A : arbreBinaire): entier` qui retourne le minimum des valeurs contenues dans A.

Exercice 4.4 *Appartenance, égalité*

1. Écrire une fonction qui teste si un élément appartient à un arbre binaire.
2. Écrire une fonction qui teste l'égalité de deux arbres binaires.
3. Écrire une fonction qui teste si un arbre binaire est un sous arbre d'un autre arbre binaire.

Exercice 4.5 *Arbre binaire complet*

On appelle arbre binaire complet un arbre binaire tel que chaque sommet possède 0 ou 2 fils.

1. Donner des exemples d'arbres binaires complets.
2. Écrire une fonction qui teste si un arbre binaire est complet
`arbreBinaire_complet(val A:arbreBinaire): boolean;`

Exercice 4.6 *Arbre binaire parfait*

On appelle arbre binaire parfait un arbre binaire (complet) tel que chaque sommet soit père de deux sous arbres de même hauteur.

1. Donner des exemples d'arbres binaires parfaits.
2. Écrire une fonction qui teste si un arbre binaire est parfait.
`arbreBinaire_parfait(val A:arbreBinaire): boolean;`

Exercice 4.7 *Arbre binaire quasi-parfait*

On appelle arbre binaire quasi-parfait un arbre binaire parfait éventuellement grignoté d'un étage en bas à droite.

1. Donner des exemples d'arbres binaires quasi-parfaits.
2. Écrire une fonction qui teste si un arbre binaire est quasi-parfait
`arbreBinaire_quasi_parfait(val A:arbreBinaire): boolean;`

Problème récurrent (notre fil d'ariane)

Exercice 4.8 Gestion d'une piste d'atterrissage des avions

Un avion est un enregistrement contenant :

- l'indicatif (6 caractères)
- la destination (30 caractères)
- l'autonomie résiduelle de carburant comptée en heures de vol (entier)
- deux booléens indiquant s'il y a un pirate à bord et s'il y a le feu.

1. Définir les structures de données nécessaires.
2. Ecrire la fonction `Priorité` ainsi que la gestion complète de la piste.
3. Envisager le cas de suppression d'un élément quelconque de la file lorsque le pirate a mis sa menace de détournement à exécution.

Quelles sont les notions que vous venez de voir qui peuvent permettre d'amorcer le fil d'ariane ?

Annexe A : Rappel de la sémantique des primitives du type abstrait arbre binaire.

```
arbreBinaire= sommet
fonction getValeur(val S:sommet):objet;
/* vaut NIL si le sommet n'existe pas */
fonction filsGauche(val S:sommet):sommet;
/* vaut NIL si S n'a pas de fils gauche */
fonction filsDroit(val S:sommet):sommet;
/* vaut NIL si S n'a pas de fils droit */
fonction pere(val S:sommet):sommet;
/* vaut NIL si S est la racine de l'arbre */
fonction setValeur(ref S:sommet;val x:objet):vide;
/* affecte au sommet S la valeur x */
fonction ajouterFilsGauche(ref S:sommet,val x:objet):vide;
/* filsGauche(S)==NIL doit être vérifié */
fonction ajouterFilsDroit(ref S:sommet,x:objet):vide;
/* filsDroit(S)==NIL doit être vérifié */
fonction supprimerFilsGauche(ref S:sommet):vide;
/* filsGauche(S) est une feuille */
fonction supprimerFilsDroit(ref S:sommet):vide;
/* filsDroit(S) est une feuille */
fonction detruireSommet(ref S:sommet):vide;
/* S est une feuille */
fonction créerArbreBinaire(val Racine:objet):sommet;
```

Annexe B : Construction d'un arbre binaire à partir d'un tableau d'entiers.

```
fonction remplirTableauArbre(ref T:tableau[1..N] d'entier):
    arbreBinaire d'entier;
var A:arbreBinaire d'entier;
var F: file de sommet;
var s:sommet;
var i:entier;
début
    créerFile(F);
    A= créerArbreBinaire(T[1]);
    enfiler(F, A);
    tmp= 2;
    tantque 2*tmp-1 <= N faire
        pour i= tmp à 2*tmp-1 par pas de 2 faire
            s= valeur(F);
            defiler(F);
            ajouterFilsGauche(s, T[i]);
            enfiler(F, filsGauche(s));
            ajouterFilsDroit(s, T[i+1]);
            enfiler(F, filsDroit(s));
        finpour;
        tmp= tmp*2;
    fintantque
    pour i=tmp à N par pas de 2 faire
        s= valeur(F);
        defiler(F);
        ajouterFilsGauche(s, T[i]);
        ajouterFilsDroit(s, T[i+1]);
    finpour;
    si N mod 2 != 0 alors
        ajouterFilsGauche(valeur(F), T[N])
    fin
    détruireFile(F);
    retourner(A);
fin
```