Arbres binaires et arbres lexicographiques

Dans ce TD, outre les arbres binaires, on s'intéressera également aux arbres lexicographiques.

1. **Vérification**
   Écrire une fonction `int estDeRecherche(Arbre a)` qui renverra `1` si l'arbre est arbre binaire de recherche, et `0` sinon.

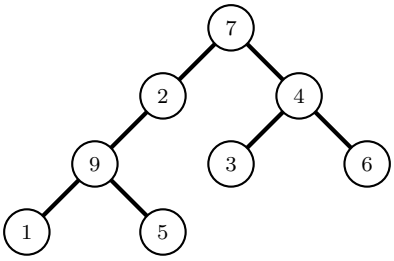

FIGURE 1 – Arbre binaire

2. **Chemins dans un arbre binaire**
   On considère un arbre binaire de hauteur inférieure à une constante `MAX`. Écrire une fonction `void afficheChemin(Arbre T)` qui permet l'affichage de tous les chemins menant de la racine aux feuilles dans l'ordre d'un parcours en profondeur préfixe.
   Par exemple, pour l'arbre dessiné en Figure 1, on affichera

```
7 2 9 1
7 2 9 5
7 2 3
7 4 6
```

   Pour ce faire, on appellera une fonction auxiliaire `void affCheminAux(Arbre a,int buffer[], int indice)`, où `buffer` est un tableau de taille `MAX` déclaré dans la fonction `afficheChemin`, et où `indice` indique la première case vide de `buffer`.

3. **Arbres ternaires lexicographiques**

   a. Dessiner l'arbre ternaire lexicographique obtenu après avoir inséré successivement les mots suivants (on a omis les accents pour plus de simplicité) :

   — bavard
   — bavette
   — bavaroise
   — truc
   — animal
   — billard
   — biere
   — bar
   — triche

Dans la suite de cet exercice, on utilisera les types C suivants :

```c
typedef struct noeud{
   char lettre;          /* étiquette */
   struct noeud *fg;     /* fils gauche */
   struct noeud *fd;     /* fils droit */
   struct noeud *fils;   /* fils principal */
} Tnoeud, *Tarbre;
```

Le caractère `'0'` sera utilisé pour indiquer les fins de mots.

b. Écrire une fonction `int recherche(Tarbre a, char *m)` qui renvoie `1` si le mot `m` est stocké dans l'arbre `a`, et qui renvoie `0` sinon.

c. Écrire une fonction `int nombreMot(Tarbre a)` qui renvoie le nombre de mots stockés dans l'arbre `a`.

d. Écrire une fonction `int ajouterMot(Tarbre *a, char *m)` qui ajoute le mot `m` à l'arbre `*a`. La fonction devra renvoyer `0` en cas d'erreur, `1` en cas d'exécution correcte.