DS n° 2 Informatique III

Calculatrice et documents non autorisés

Lorsqu'aucune consigne n'est précisée, les réponses peuvent être données en C ou en pseudo-code.

Exercice 1 (*Parcours d'ABR*) (5.5 pts)

On considère un ABR dont les valeurs ont été ajoutées dans l'ordre suivant :
— 20, 10, 15, 40, 5, 35, 32, 7, 17, 9, 25, 6, 42

1. Dessiner l'arbre obtenu.
2. Donner les valeurs de son parcours préfixe
3. Donner les valeurs de son parcours infixe
4. Le parcours postfixe obéit à la règle suivante : traiter(fils gauche), traiter(fils droit), traiter racine.
   Donner les valeurs du parcours postfixe
5. Ecrire la fonction permettant d'effectuer le parcours postfixe.
6. Ecrire la fonction `Arbre* suppMin(Arbre* pA, int* pElt)` qui est utilisée pour la suppréssion dans les ABR.
   Cette fonction sert à rechercher la valeur minimum de l'arbre passé en paramètre, la renvoyer à la fonction appelante, libérer l'espace mémoire du noeud trouvé, et refaire les liens avec les enfants du noeud supprimé

Exercice 2 (*Arbre équilibré*) (3.5 pts)

On rappelle que l'équilibre d'un arbre est la différence de hauteur entre son sous-arbre droit et son sous-arbre gauche. Un arbre est dit déséquilibré lorsque la valeur absolue de cet équilibre est strictement supérieure à 1.

1. Ecrire une fonction qui va calculer la hauteur d'un arbre
2. Ecrire une fonction qui retourne 1 si l'arbre passé en paramètre est équilibré, et retourne 0 dans le cas contraire

Note : les fonctions demandées dans cet exercice ne doivent pas être nécessairement optimales d'un point de vue complexité temporelle.

Exercice 3 (*arbre complet*) (2 pts)

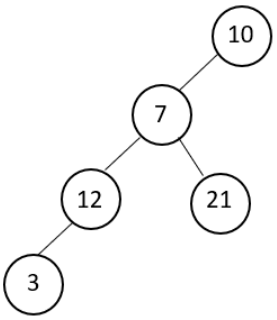
Un arbre binaire est dit complet si chaque nœud a soit 0 soit 2 fils. Ecrire une fonction qui retourne 1 si un arbre est complet, 0 sinon.

Exercice 4 (*Arbre miroir*) (4 pts)

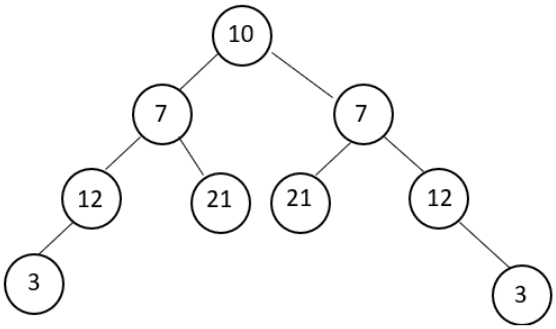
On suppose un arbre binaire classique dont la racine n'a qu'un fils gauche (pas de fils droit).
Ecrire une fonction miroir qui va modifier l'arbre afin d'ajouter une partie à droite qui sera le miroir de la partie à gauche
Le choix des paramètres et de la valeur de retour est libre

Exemple :

Avant l'appel à la fonction miroir

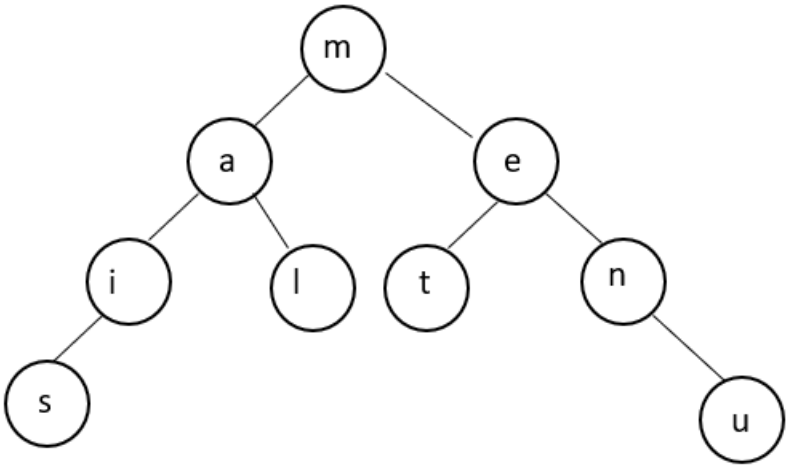


Après l'appel à la fonction miroir



Exercice 5 (*Recherche d'un mot dans un arbre*) **(5 pts)**

— On considère qu'un arbre tel que celui présent sur la figure suivante, peut permettre de stocker des mots. La première lettre d'un mot sera un noeud (pas forcément la racine) et les lettres suivantes seront accessibles en parcourant les enfants successifs.
— Dans cet arbre, il n'y a que des lettres distinctes. C'est à dire que les mots présents dans cet arbre ne contiennent jamais plusieurs fois la même lettre
— Par exemple, l'arbre suivant contient les mots : "mal", "mais", "ais", "me", "met", "et", "menu".



https ://fr.overleaf.com/project/6369009ecab9c992d05e9827

1. Ecrire la structure permettant de construire un arbre contenant un caractère.
2. Ecrire une fonction `int motExistant (Arbre *a, char *mot)` qui va retourner 1 si la chaîne de caractères `mot` est dans l'arbre, et qui va retourner 0 sinon.

Note : Toutes les fonctions jugées nécessaires et qui seraient appelées par la fonction `motExistant` doivent être écrites.