Programme et Algorithme III
TD/TP 5: Arbre

CYU & ZUST

25 novembre, 2021

1 Préparation

Créez des entêtes arbre.h [1] contenant les fonctions/procédures nécessaires à l'implantation.
Du type abstrait arbre à savoir :

- Le type **TreeNode** (typedef)
- **typeElem**: peut varier selon la valeur contenue dans le nœud : int, char, struct ... (pour ce TP on choisira int pour typeElem)

Opération de base

- typedef typeElem int;
- Les types : `arbre_empty, bool` etc.
- TreeNode consa(typeElem , arbre , arbre) ;
- TreeNode left (arbre) ;
- arbre right (arbre) ;
- booleen isEmptyTree (arbre) ;
- typeElem root (arbre) ;
- préfixe (arbre), infixe (arbre), postfixe (arbre);
- bool (afficher null pour les "enfants vides" des nœuds externes)
  Exemple: pour l'arbre démontré dans Figure 2

  la fonction de parcours préfixe doit afficher

  ```
  préfix:  5, 1, "null", "null", 4, 3, "null", "null", 6, "null", null"
  ```

Pour valider votre implantation, vous pouvez créer un jeu des données arbres par relier plusieurs nœuds avec ordre défini.

[1] Vous pouvez prendre les codes mis en œuvre au cours/TP du dernier semestre.

2 Valider un arbre binaire de recherche (ABR)

Étant donné la racine `root` d'un arbre binaire, déterminer s'il s'agit d'un arbre binaire de recherche (ABR) valide.

Une ABR valide est définie comme suit :

1. La sous-arborescence gauche d'un nœud ne contient que des nœuds dont la clé est inférieure à la clé du nœud.
2. Le sous-arbre droit d'un nœud ne contient que les nœuds dont la clé est supérieure à la clé du nœud.
3. Les deux sous-arbres, gauche et droit, doivent également être des arbres de recherche binaires.

Exemple 1:

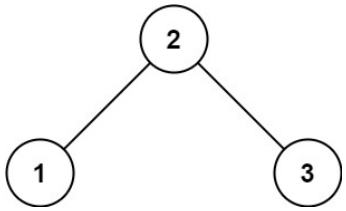

Figure 1: Exemple 1 de Exo 2

```
Entrée: root = [2,1,3]
Sortie: true
```

Exemple 2:

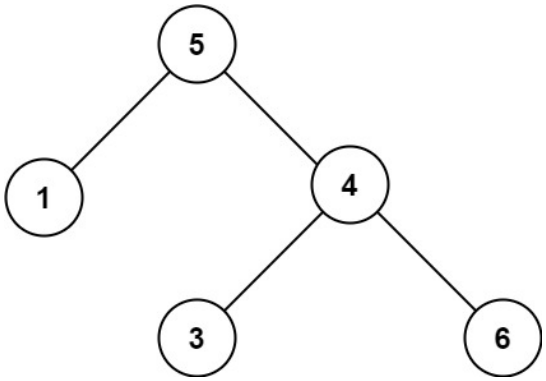

Figure 2: Exemple 2 de Exo 2

```
Entrée: root = [5,1,4,null,null,3,6]
Sortie: false
```

Explication: Le valeur du nœud racine est 5 avec enfant droit valorisé 4. $5 > 4$ *Pour simplifier l'implantation, les contraints supplémentaires sont ajoutés*

- Le nombre de nœuds de l'arbre est limité, donc pas besoin de considérer les problème d'échelle.

Question: Écrire le pseudo-code et implanter en langage C.

```
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};
bool isValidBST(struct TreeNode* root){
}
```

3 Récupérer l'arbre binaire de recherche

On vous donne la racine d'un arbre binaire de recherche (ABR), où les valeurs d'**exactement** deux nœuds de l'arbre ont été échangées par erreur. Récupérez l'arbre sans modifier sa structure.

Exemple:

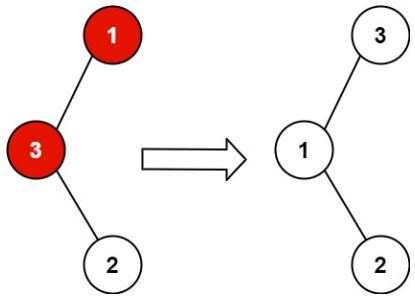

Figure 3: Exemple 1 de Exo 3

```
Input: root = [1,3,null,null,2]
Output: [3,1,null,null,2]
```

Explication : 3 ne peut pas être un enfant gauche de 1 car 3 ¿ 1. L'échange de 1 et 3 rend la ABR valide.

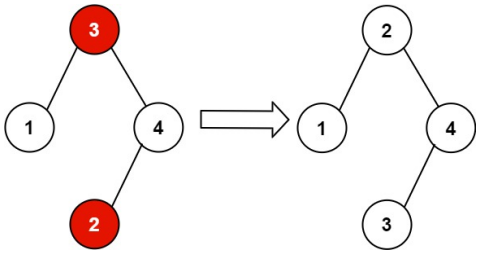

Figure 4: Exemple 2 de Exo 3

```
Entrée : root = [3,1,4,null,null,2]
Sortie : [2,1,4,null,null,3]
```

Explication : 2 ne peut pas être dans le sous-arbre droit de 3 car 2 ¡ 3. L'échange de 2 et 3 rend la ABR valide.

Question: Écrire le pseudo-code et implémenter en langage C.

```
void recoverTree(struct TreeNode* root){
}
```

4 Construction d'un arbre binaire à partir des parcours en préfixe et en infixe

Étant donné deux tableaux d'entiers, `preorder` et `inorder`, où `preorder` est le parcours de préfix d'un arbre binaire et `inorder` est le parcours de infixe du même arbre, construire et retourner l'arbre binaire.

Exemple:

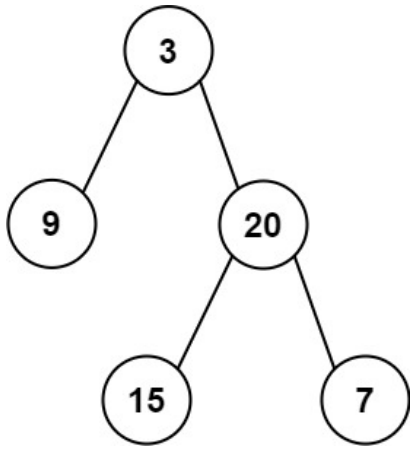

Figure 5: Exemple 1 de Exo 4

```
Entrées: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
Sortie: [3,9,20,null,null,15,7]
```

Pour simplifier l'implantation, les assomptions suivantes sont acceptées:

- `inorder.length == preorder.length`
- `preorder` et `inorder` consistent en des valeurs uniques.
- Chaque valeur de `inorder` apparaît également dans `preorder`.

Question: Écrire le pseudo-code.