TD/TP sur les arbres binaires

1 TD : Rappel d'algorithmes vus en cours et implémentations.

En cours, vous avez vu une structure de liste chaînée où chaque cellule contient deux suivants. Cette structure nous permet d'implanter des arbres binaires. À partir de maintenant, nous appellerons les cellules des **nœuds** et les deux références seront nommées les fils **gauche** et **droit**. La « tête » se nomme la **racine**, et les nœuds qui n'ont pas de fils sont des **feuilles**.

Dans un premier temps, vous allez travailler sur des algorithmes simples travaillant sur les arbres, en vous basant sur la version récursive de l'algorithme de parcours vu en cours. Il vous est demandé de rédiger ces algorithmes en langage algorithmique (vous pouvez ajouter des schémas, des analyses de complexité et des propositions et preuves d'invariants). Dans un second temps, il vous est demandé de traduire (implémenter) ces algorithmes dans un langage de programmation particulier (en C pour les parcours informatique et math-info, en Python pour le parcours mathématiques).

La structure de données conseillée en langage C pour les arbres d'entiers est la suivante :

```
struct noeud_s;
typedef struct noeud_s noeud;

typedef noeud* arbre;

struct noeud_s {
    int valeur;
    arbre gauche;
    arbre droit;
};
```

La structure de données conseillée en Python est la suivante :

```
class Arbre:
    def __init__(self, val=None, g=None, d=None):
        self.valeur = val
        self.gauche = g
        self.droit = d
```

Exercice 1.1. Parcours en profondeur d'un arbre binaire [Schéma de base]. Rappeler l'algorithme récursif pour parcourir (en profondeur) un arbre binaire puis fournir une implémentation de cet algorithme.

Profil de la fonction (en C) :

```
void parcours_arbre(arbre a);
```

Exercice 1.2. Hauteur d'un arbre binaire [Schéma de base]. Donner un algorithme pour calculer la hauteur d'un arbre binaire (longueur du plus long chemin de la racine à une feuille) puis fournir l'implémentation de cet algorithme.

Profil de la fonction (en C) :

```
int hauteur_arbre(arbre a);
```

2 TD : D'autres algorithmes et implémentations.

Exercice 2.1. Recherche d'un élément. Donner un algorithme et son implémentation pour déterminer la référence d'un nœud qui comporte une valeur nulle (0) dans un arbre. Dans le cas où l'arbre ne comporte pas de valeur nulle (0), retourner la référence nulle (NULL en C ou None en Python).

Profil de la fonction (en C) :

```
arbre rechercher_zero(arbre a);
```

Remarque : Un arbre peut contenir plusieurs fois la valeur nulle, nous recherchons à retourner la référence du premier nœud que l'on trouve.

Exercice 2.2. Recherche et calcul de la profondeur d'un élément. Écrire un algorithme qui recherche une valeur entière dans un arbre. Si la valeur est dans l'arbre, l'algorithme retourne `vrai` **et** calcule la profondeur du nœud qui contient la valeur. Sinon, l'algorithme retourne `faux`.

Profil de la fonction (en C) :

```
int rechercher_arbre(arbre a, int* profondeur);
```

Exercice 2.3. Ajout d'un nœud dans un arbre binaire de recherche. Un arbre binaire de recherche est un arbre binaire dans lequel chaque nœud du sous-arbre gauche a un valeur inférieure à la valeur de la racine, et chaque nœud du sous-arbre droit possède une valeur supérieure à celle-ci. Ecrire un algorithme qui rajoute une valeur dans un arbre. Si la valeur est déjà dans l'arbre, l'algorithme retourne `faux`. Sinon, l'algorithme modifie la structure de l'arbre en respectant les contraintes d'un arbre binaire de recherche (en créant le nœud là où il faut) et retourne `vrai`.

Attention ! Pour cette fonction, la racine est potentiellement modifiée ! En Python, la fonction doit donc renvoyer un couple (booléen, arbre).

En C, on passe donc **un pointeur** vers l'arbre (pour potentiellement le faire pointer vers une nouvelle racine).

Profil de la fonction (en C) :

```
int ajouter_abr(arbre* a, int valeur);
```

3 TP : Implémentations et test.

Le TP sur les arbres comporte quelques exercices immédiatement issus du TD et quelques exercices supplémentaires. Ces exercices sont disponibles sur Caseine.

Exercice 3.1. Somme des valeurs des nœuds [Schéma de parcours de base]. Implémenter et tester un algorithme qui calcule la somme des nœuds d'un arbre binaire contenant des entiers. La saisie (à partir d'un fichier) et l'affichage d'un arbre binaire sont fournis.

Exercice 3.2. Recherche d'un élément [exercice de TD]. Implémenter et tester un algorithme qui qui vérifie (retourne vrai ou faux) si un arbre contient un nœud à valeur zéro.

Exercice 3.3. Ajout d'un nœud dans un arbre binaire de recherche [exercice de TD]. Implémenter et tester l'algorithme qui ajoute une valeur à un arbre binaire de recherche.

Exercice 3.4. Suppression de nœud d'un arbre binaire de recherche. Proposer, implémenter et tester un algorithme qui supprime une valeur d'un arbre binaire de recherche.