TP n°5 : Parcours et Traitement des Arbres en Python

Dr. Kaoutar SENHAJI

January 26, 2026

Introduction

Les structures d'arbres jouent un rôle fondamental en informatique. Elles permettent de représenter et de manipuler des données hiérarchiques efficacement. Ce TP a pour but de vous familiariser avec la conception et l'implémentation des arbres, ainsi qu'avec les algorithmes de parcours et de traitement associés. Vous utiliserez le langage Python pour réaliser ces exercices.

Durée estimée : 3 heures

Prérequis :

- Notions de base sur les structures de données (listes, dictionnaires, etc.).
- Connaissances préalables sur les fonctions récursives.

Objectifs pédagogiques :

- Comprendre les différentes représentations d'arbres en Python.
- Implémenter des algorithmes pour parcourir un arbre en profondeur (DFS) et en largeur (BFS).
- Appliquer des traitements spécifiques sur les nœuds d'un arbre.
- Analyser la complexité algorithmique des parcours d'arbres.

Exercices

Partie 1 : Représentation d'un arbre en Python

Objectif : Comprendre comment représenter un arbre en Python.

Exercice 1 : Définition d'un nœud et d'un arbre

a. Définir une classe `Node` permettant de représenter un nœud avec une valeur et une liste d'enfants.

b. Créer une classe `Tree` pour représenter un arbre n-aire.

c. Construire un exemple d'arbre contenant 5 nœuds avec une structure hiérarchique de votre choix.

Exercice 2 : Affichage de la structure de l'arbre

a. Implémenter une méthode permettant d'afficher l'arbre sous forme textuelle (par exemple, en utilisant un parcours récursif).

b. Discuter de la complexité mémoire pour représenter un arbre n-aire.

Partie 2 : Parcours d'arbre

Objectif : Implémenter et analyser les algorithmes de parcours d'arbre.

Exercice 3 : Parcours en profondeur (DFS)

a. Écrire une fonction récursive pour parcourir un arbre en profondeur.

b. Tester votre fonction sur l'exemple d'arbre défini dans l'exercice précédent.

c. Décrire la complexité en temps de cet algorithme.

Exercice 4 : Parcours en largeur (BFS)

a. Implémenter un algorithme de parcours en largeur à l'aide d'une file (queue).

b. Tester votre fonction sur l'exemple d'arbre défini dans l'exercice précédent.

c. Décrire la complexité en temps et en espace de cet algorithme.

Partie 3 : Traitement d'arbres

Objectif : Appliquer des traitements spécifiques sur les nœuds d'un arbre.

Exercice 5 : Somme des valeurs d'un arbre

a. Implémenter une fonction qui calcule la somme des valeurs de tous les nœuds de l'arbre.

b. Tester votre fonction sur l'exemple d'arbre défini précédemment.

Exercice 6 : Recherche d'une valeur spécifique

a. Écrire une fonction qui vérifie si une valeur donnée existe dans l'arbre.

b. Tester votre fonction avec différentes valeurs (présentes et absentes de l'arbre).

Exercice 7 (Bonus) : Problèmes supplémentaires

- Trouver le plus grand nœud dans un arbre.
- Calculer la profondeur maximale de l'arbre.

Analyse et Conclusion

Objectif : Réfléchir sur les performances des algorithmes et synthétiser les apprentissages.

Exercice 8 : Analyse des parcours

a. Comparez la complexité en temps des parcours DFS et BFS.

b. Discutez de leur efficacité respective selon le type de problème traité.

Exercice 9 : Synthèse

a. Résumez les concepts appris lors de ce TP.

b. Proposez des pistes pour approfondir l'étude des arbres (par exemple, introduction aux arbres binaires de recherche ou AVL).