UNIVERSITE TOULOUSE III PAUL SABATIER
Faculté des Sciences et d'Ingénierie

Arbres binaires

CTD 8 : Arbres binaires et Récursivité

Algorithmique

Semestre 2 - 2019-2020

Introduction
Motivation et rappels
Arbres binaires
Description du type
Abstraction : Opérations
Structure de données (implémentations)
Récursivité
Implémentation avec tuple
Principe
Exemples
Applications
Exemples à résultat en dehors du type
Exemples à résultat dans le type
Exercices
Appartenance
Somme des valeurs
Hauteur d'un arbre
Feuille gauche
Sous arbre par valeur
Miroir
Substitution

Introduction - Motivation et rappels

Introduction
Motivation et rappels
Arbres binaires
Implémentation avec tuple
Applications
Exercices

Motivation : la structure de données **arbre binaire** (on dira simplement arbre dans la suite) est omniprésente en algorithmique et nombres d'algorithmes les concernant sont récursifs. Éventuellement munis de propriétés supplémentaires, ce sont de puissants outils de structuration des données (arbres de recherche, tas ou heap, arbretas ou treap, AVL, arbres rouge-noirs, B-arbres, arbres splay,...)

Abstraction de données

Comme précédemment, on va définir d'abord l'**abstraction de données** en décrivant les **opérations qu'il est possible de lui appliquer**, indépendamment de la manière dont cela sera programmé.

La **description de chaque opération** précise :

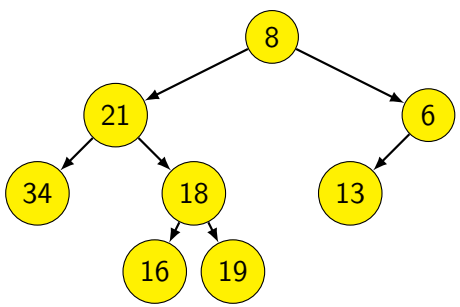
- Ses entrées.
- Les préconditions qui doivent être vérifiées pour que l'opération s'exécute correctement.
- Ses sorties.
- Les post conditions vraies après exécution de l'opération.

Définition informelle

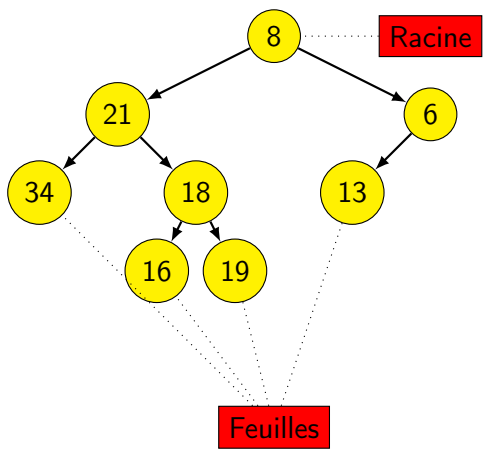
- Un arbre binaire peut être vu comme une hiérarchie dont chaque élément est un nœud, chaque nœud étant le parent d'au plus deux autres nœuds.
- Chaque noeud peut être porteur d'une information (un entier, une chaîne, une liste, un arbre, etc.).
- Le nœud parent a donc au plus deux enfants, s'il n'en a pas, c'est une feuille.
- L'unique nœud qui n'a pas de parent est la racine.

Définition informelle

Exemple visuel :



Définition informelle



Introduction

Arbres binaires
Description du type
Abstraction : Opérations
Structure de données (implémentations)
Récursivité

Implémentation avec tuple

Applications

Exercices

Arbres binaires - Description du type

- nombre d'éléments **variable** (évolution dynamique)
- éléments de même type
- structures **récursives** : on distingue l'élément **racine** et le reste (son gauche et son sous-arbre droit).
  Dans la suite, on désignera le sous-arbre gauche par SAG et le droit par SAD.

Arbres binaires - Abstraction : Opérations

- opérations de construction
  - création (d'un arbre vide)
  - jumelage de deux arbres et d'un nœud
- opérations d'accès
  - accès à la valeur associée à la racine
  - accès au SAG et au SAD
  - test d'arbre vide

Arbres binaires - Structure de données (implémentations)

- pas de module standard en Python (mais on trouve divers modules à installer comme `tree`)
- tuple ou objet en Python, objet en JAVA, ...
- struct plus pointeurs en C
- tableaux dynamiques

La récursivité est particulièrement adaptée pour traiter des structures de données récursives comme les arbres

Démarche pour écrire une fonction récursive sur les arbres

- Cas d'arrêt :
  - Le plus souvent arbre vide.
  - Éventuellement feuille (ses deux sous-arbres sont vides).
- Cas récursif : rappeler la fonction récursive sur les sous-problèmes
  - les deux sous-arbres
  - la valeur à la racine

"Vérifier/Constater" que la fonction termine, c'est-à-dire qu'on atteint au moins un cas d'arrêt :

- argument général : on a diminué la taille de l'arbre en passant aux sous-arbres.

Introduction

Arbres binaires

Implémentation avec tuple
Principe
Exemples

Applications

Exercices

Implémentation avec tuple - Principe

- à l'aide du type `tuple` (rappel : on ne peut modifier un tuple $\Rightarrow$ sécurisation du type)
- un arbre vide est représenté par la valeur `None` (autre option : un tuple vide)
- un arbre non vide par un tuple de trois éléments :
  - le premier est la valeur associée à la racine
  - le deuxième est un tuple représentant le SAG
  - le troisième est une tuple représentant le SAD
  - accès à la valeur associée à la racine par `arbre[0]`
  - accès au SAG par `arbre[1]`
  - accès au SAD par `arbre[2]`

Afin de s'affranchir de l'implémentation, on commence par écrire les fonctions d'accès et de construction de base.
On n'utilisera plus qu'elles ensuite.

Fonctions d'accès :

- valeur associée à la racine :

```
def valeur_racine(arbre):
    assert not est_vide(arbre), "acces a element d'un arbre vide"
    #fonction est_vide a venir
    return arbre[0]

```

- accès au SAG / au SAD :

```
def sag(arbre):
    assert not est_vide(arbre), "acces au SAG d'un arbre vide"
    return arbre[1]

def sad(arbre):
    assert not est_vide(arbre), "acces au SAD d'un arbre vide"
    return arbre[2]

```


Fonctions de construction :

- créer arbre vide (convention None) :

```
def arbre_vide():
    return None

```

On pourra changer de convention plus tard, cela ne devrait en aucun cas changer quoique ce soit des autres fonctions qui utiliseront nos arbres !

- jumeler arbres :

```
def jumeler(valeur, gauche, droit):
    return (valeur,gauche,droit)

```


Écrire une fonction `vide` qui teste si un arbre est vide.

- avec la convention `None` :
  - `if arbre==None:`
  - `if arbre is None:`
  - `if not arbre:` (`None` $\sim$ `False` lors des tests)

On va s'en tenir à la première option :

```
def est_vide(arbre):
    return arbre==None
```

Introduction

Arbres binaires

Implémentation avec tuple

Applications
Exemples à résultat en dehors du type
Exemples à résultat dans le type

Exercices

Applications - Exemples à résultat en dehors du type

Écrire la fonction récursive `taille` qui renvoie le nombre de nœuds d'un arbre (sa taille).

2 cas à traiter :

- Cas d'arrêt : l'arbre est vide
  → On retourne 0.
- Cas récursif : il n'est pas vide
  → On retourne la somme de la taille des deux sous-arbres + 1.

Applications - Exemples à résultat en dehors du type

Fonction récursive taille

```
1 def taille(arbre):
2   if est_vide(arbre):
3     return 0
4   else:
5     return (1\
6             + taille(sag(arbre))\
7             + taille(droite(arbre)))
```


Applications - Exemples à résultat dans le type

Écrire la fonction récursive `ajout_a_droite` qui ajoute un nœud, contenant une valeur x donnée, "le plus à droite" de l'arbre. Dans l'arbre vu au début, on ajouterait un fils droit au nœud contenant la valeur 6.

2 cas à traiter :

- Cas d'arrêt : l'arbre est vide
  → On retourne une feuille contenant x.
- Cas récursif : il n'est pas vide
  → on ajoute le nœud dans le SAD et on jumelle le résultat avec le SAG et la valeur racine (inchangés).

Applications - Exemples à résultat dans le type

Fonction récursive ajout_a_droite

```
def ajout_a_droite(arbre,x):
  if est_vide(arbre):
    return jumeler(x,arbre_vide,arbre_vide)
  else:
    return jumeler(valeur_arbre(arbre),\
                   sag(arbre),\
                   ajout_a_droite(sad(arbre),x))
```


Introduction

Arbres binaires

Implémentation avec tuple

Applications

Exercices
- Appartenance
- Somme des valeurs
- Hauteur d'un arbre
- Feuille gauche
- Sous arbre par valeur
- Miroir
- Substitution
- Parcours Gauche-Racine-Droite
- Cheminement
- Accès par chemin

Exercices - Appartenance

Exercice 1

Écrire la fonction récursive `appartient` qui, étant donnés un élément et un arbre, retourne `True` si l'élément est présent dans l'arbre et `False` sinon.

Exercices - Somme des valeurs

Exercice 2

Écrire la fonction récursive `somme_arbre` qui, étant donnée un arbre d'entiers, calcule la somme des valeurs présentes dans l'arbre.

Par convention, la somme des éléments d'un arbre vide sera 0.

Exercices - Hauteur d'un arbre

Exercice 3

Écrire la fonction récursive `hauteur_arbre` qui, étant donnée un arbre, calcule la longueur du plus long chemin entre la racine et une feuille.

Par convention, la hauteur d'un arbre vide est 0.

Exercices - Feuille gauche

Exercice 4

Écrire la fonction récursive `feuille_gauche` qui, étant donnée un arbre non vide, retourne la valeur associée à la feuille la plus à gauche.

Exercices - Sous arbre par valeur

Exercice 5

Écrire une fonction récursive `sous_arbre` qui renvoie le sous-arbre dont la racine contient une valeur v, s'il y en a plusieurs, il faut renvoyer le plus à gauche. S'il n'y en a pas, il on renvoie un arbre vide.

Exercices - Miroir

Exercice 6

Écrire une fonction récursive `miroir` qui, étant donnée un arbre, retourne l'arbre obtenu par symétrie verticale.

Exercices - Substitution

Exercice 7

Écrire la fonction récursive `substituer` qui, étant donnés deux valeurs et un arbre, construit un nouvel arbre dans lequel toute occurrence de la première valeur a été remplacée par une occurence de la deuxième (l'arbre initial ne sera pas modifié).

Exercices - Parcours Gauche-Racine-Droite

Exercice 8

Écrire la fonction récursive `parcours_GRD` qui, étant donnés un arbre, renvoie la liste des éléments s'y trouvant dans l'ordre GRD. Dans l'arbre vu au début, ça donnerait `[34,21,16,18,19,8,13,6]`.

Exercices - Accès par chemin

Exercice 9

Écrire une fonction récursive `acces_par_chemin` qui accède à l'élément d'un arbre décrit par un chemin sous forme d'une chaîne constituée de "g" et de "d". L'élément désigné par "gdg" dans l'arbre du début est 16.

Exercices - Calcul de chemin

Exercice 10

Écrire une fonction récursive `chemin_vers_feuille` qui calcule le chemin (chaîne constituée de "g" et de "d") à suivre depuis la racine d'un arbre pour aboutir à la feuille la plus profonde (la plus à gauche en cas d'égalité). Indice : a) s'inspirer de la fonction `hauteur`, b) c'est plus simple en cherchant le chemin vers l'arbre vide le plus profond et en le tronquant.