

Algorithmique — 1. Structures de données

1. Types abstraits de données et rappels

Bruno Grenet



<https://membres-ljk.imag.fr/Bruno.Grenet/Algorithmique.html>

Université Grenoble Alpes – IM²AG
L3 Mathématiques et Informatique

Introduction à la partie 1

Structures de données étudiées

- ▶ Rappels : tableau, liste (chaînée), arbre binaire
- ▶ Structures *linéaires* : pile, file, file à priorité, tas, tableau dynamique
- ▶ Dictionnaire : arbre binaire de recherche, table de hachage

Objectifs

- ▶ Description de la structure de données
- ▶ Analyse de ses performances
- ▶ Comparaisons entre différentes structures

Méthode

1. Identifier les opérations que la structure doit fournir *type abstrait de données*
2. Construire la structure de données en se basant sur autre plus simple
3. Analyser les coûts : espace nécessaire, complexité des opérations
4. Comparer avec d'autres structures similaires

Table des matières

1. Type abstrait de données

2. Tableaux et listes chaînées

3. Arbres binaires

Table des matières

1. Type abstrait de données

2. Tableaux et listes chaînées

3. Arbres binaires

Motivation

Écriture d'algorithmes

- ▶ Manipulation de données
- ▶ Abstraction vis-à-vis du matériel, langage de programmation, etc.
- ▶ *Modèle* des opérations permises et de leur complexité

Exemples

- ▶ Tri d'un tableau d'entiers
 - ▶ Accès et modifications des éléments
 - ▶ Calcul sur les entiers
- ▶ Algorithme de multiplication d'entiers
 - ▶ Représentation des entiers
 - ▶ Calcul sur les bits

- ▶ Tout algorithme se base sur un (ou des) type(s) abstrait(s) de données...
- ▶ ... même si c'est la plupart du temps implicite

Qu'est-ce qu'un type abstrait de données ?

Ingrédients

- ▶ le **nom** du TAD et la **description** des objets représentés
- ▶ des **opérations**
 - ▶ constructeurs
 - ▶ modificateurs
 - ▶ observateurs
- ▶ Propriété importante : **statique** ou **dynamique**

créer un nouvel objet
modifier un objet
obtenir de l'information sur un objet
taille fixée ou variable

Et c'est tout !

- ▶ Pas de **représentation** des données
 - ▶ Pas d'implantation des opérations
- *Interface* d'une structure de données

Exemples

Entiers : +, -, ×, ÷, <, >, ...

Liste : ESTVIDE(), TÊTE(), QUEUE()

Chaîne : accès au $i^{\text{ème}}$ caractère, longueur, concaténation, ...

À quoi ça sert ?

Pourquoi travailler avec un type *abstrait* de données ?

Indépendance de l'implantation

- ▶ Un même algorithme fonctionne avec différentes implantations
- ▶ Implantation améliorée → performances améliorées

Se libérer l'esprit

- ▶ Fixer un nombre restreint d'opérations possibles... et s'y tenir!
- ▶ Se concentrer sur les *fonctionnalités*, pas les détails d'implantation

Complexité des structures de données

- ▶ Fixer des objectifs
- ▶ Trouver la meilleure représentation possible
- ▶ Indépendamment d'un problème particulier à résoudre

les opérations du TAD

Un exemple : les entiers

Dans des langages

- ▶ Entiers bornés (par ex. 64 bits), signés ou non signés C, C++, Java, OCaml, ...
- ▶ Entiers non bornés, en général signés Python

Un TAD « entier » possible

- ▶ Description : les entiers relatifs (\mathbb{Z})
- ▶ Opérations :
 - ▶ Addition, soustraction, multiplication, division euclidienne
 - ▶ Comparaisons entre entiers

Implantations et réalisations

- ▶ Implantations : `int` en Python, bibliothèques `BigInteger` en Java, `GMP` en C/C++, ...
- ▶ Réalisations :
 - ▶ Tableau de booléens + algorithmes de calcul TADS tableau et booléen
 - ▶ Tableau d'entiers bornés + algorithmes de calcul TADS tableau et entier borné
 - ▶ Entiers *récurifs* sous forme d'arbre binaire TAD arbre binaire
 - ▶ ...

Bilan sur les TAD

Rien de nouveau, mais formalisé !

- ▶ Tableaux, listes (chaînées), arbres binaires → TADS
- ▶ Implicite en l'absence de difficulté
- ▶ Besoin d'explicite pour des types plus complexes

ensemble, graphe, ...

Pas de TADS « officiels »

- ▶ Définition d'un TAD *en fonction des besoins*
- ▶ Quelques *classiques*
- ▶ Variations d'un cours à l'autre, d'un livre à l'autre, etc.
 - ▶ en particulier les noms varient

tableau, liste, pile, ...

Dans le cours d'algorithmique

- ▶ Étude de quelques TAD classiques
 - ▶ Types linéaires : file, pile, file à priorité
 - ▶ Ensemble et ~~ensembles disjoints~~, dictionnaire
- ▶ Utilisation (implicite ou explicite) de TAD dans les algorithmes

Table des matières

1. Type abstrait de données

2. Tableaux et listes chaînées

3. Arbres binaires

Les tableaux comme un TAD

Définition

- ▶ Ensemble de données indexées par un entier
- ▶ Opérations :
 - ▶ $T \leftarrow \text{NVTABLEAU}(n)$: nouveau tableau de taille n
 - ▶ $T[i]$: accès au $i^{\text{ème}}$ élément en lecture & écriture
 - ▶ $\text{TAILLE}(T)$: longueur du tableau

statique

Notation : $\#T$

Complexités

- ▶ Taille de la représentation : $O(n)$
- ▶ NVTABLEAU : $O(n)$
- ▶ Accès au $i^{\text{ème}}$ élément et $\text{TAILLE}(T)$: $O(1)$

Remarques

- ▶ Souvent : type des entrées à préciser
- ▶ Similarité et différences avec une implantation pratique
 - ▶ Exemple en C : `malloc, T[i]` → quasi identique!
 - ▶ Complexité : en général cohérent... mais pas toujours → questions de cache par ex.

Listes (chaînées) comme un TAD

Définition

- ▶ Ensemble de données organisé de manière séquentielle
- ▶ Opérations :
 - ▶ $L \leftarrow \text{NVLISTE}()$: nouvelle liste vide
 - ▶ $\text{ESTVIDE}(L)$
 - ▶ $\text{TÊTE}(L)$: premier élément ; $\text{QUEUE}(L)$: liste privée de la tête
 - ▶ $\text{AJOUT}(L, x)$: ajoute l'élément x en tête de L

dynamique

Complexités

- ▶ Taille proportionnelle au nombre d'éléments
- ▶ Opérations en $O(1)$!

Remarques

- ▶ Souvent : type des entrées à préciser
- ▶ Modélise deux structures classiques :
 - ▶ Impératif : structure à base de maillons et pointeurs
 - ▶ Fonctionnel : listes à la OCaml $x : : L$

Autres opérations sur les listes

Exemples d'opérations standard

- ▶ Parcours de la liste *itérateur*
- ▶ Longueur de la liste
- ▶ Recherche d'un élément par sa valeur, min / max, ...
- ▶ Insertion d'un élément en fin de liste / à un endroit donné

Complexités

- ▶ Dans le TAD précédent : toutes en $O(n)$ $n =$ taille de la liste
- ▶ On peut aussi définir des TAD enrichis, par ex.:
 - ▶ Stockage de la longueur de la liste $\rightarrow O(1)$
 - ▶ Pointeur vers la fin de la liste $\rightarrow O(1)$ pour l'insertion en fin

- ▶ Il n'y a pas **un seul** TAD possible, tout est question de **choix de modélisation**
- ▶ Les TAD enrichis sont réalisés (algorithmiquement) à partir du TAD de base

Bilan sur les tableaux et les listes

Deux structures de base en algorithmique

Tableau : représentation *statique* de n éléments ; accès en $O(1)$

Liste : représentation *dynamique* de n éléments ; accès en $O(n)$

Tableaux et listes comme TAD

- ▶ Utilisés dans ce cours pour construire d'autres TAD
- ▶ Proches des implantations pratiques en programmation

À réviser si vous n'êtes pas à l'aise !

- ▶ Exemples :
 - ▶ calculer le maximum dans un tableau ou une liste
 - ▶ recherche d'un élément dans un tableau trié ou dans une liste triée
- ▶ Noms exacts des opérations pas importants
 - ▶ $NV\text{TABLEAU}(n)$ ou « Créer un tableau de taille n » ou ...
 - ▶ $AJOUT(L, x)$ ou « Ajouter un élément x en tête de L » ou ...

→ **il faut (et il suffit) que ce soit clair !**

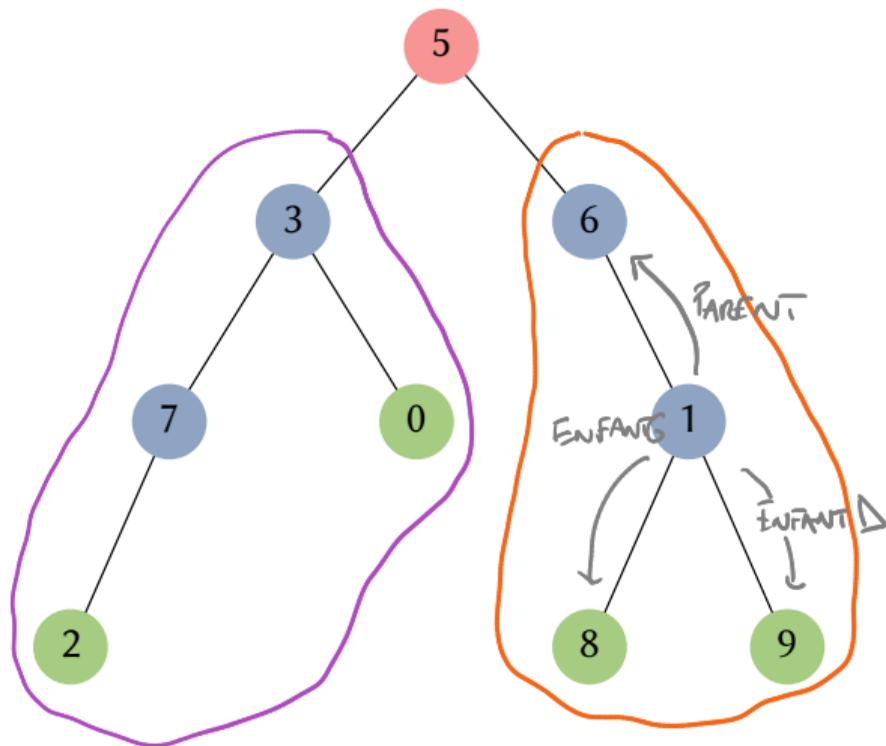
Table des matières

1. Type abstrait de données

2. Tableaux et listes chaînées

3. Arbres binaires

Vocabulaire



Définition

Un **arbre binaire** est défini récursivement :

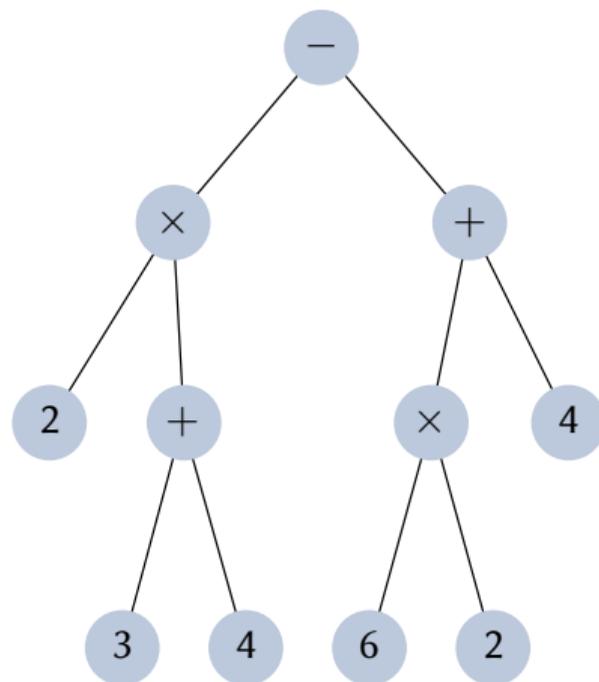
- ▶ soit l'arbre vide \emptyset
- ▶ soit constitué de 3 éléments :
 - ▶ la **racine** *donnée*
 - ▶ le **sous-arbre gauche** *arbre binaire*
 - ▶ le **sous-arbre droit** *arbre binaire*

Vocabulaire

- ▶ Nœud : **racine**, **nœud interne**, **feuille**
- ▶ **Parent**, **enfants gauche et droit**

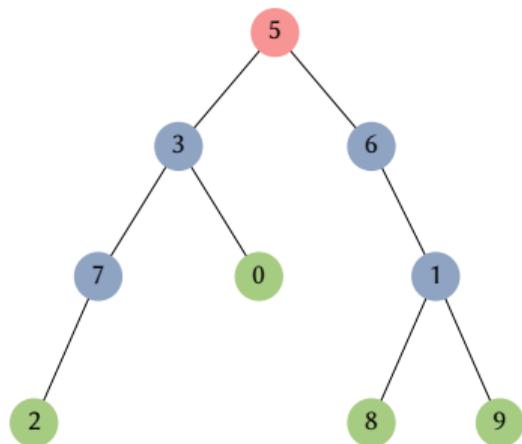
Utilité des arbres binaires

- ▶ Arbres binaires de recherche
- ▶ Tas
- ▶ Analyse syntaxique
- ▶ Bases de données
- ▶ Partition binaire de l'espace
- ▶ Tables de routage
- ▶ ...



$$2 \times (3 + 4) - (6 \times 2 + 4)$$

Hauteur et niveaux



Preuve par récurrence

$$k=0: \{x : h(x)=0\} = \{\text{racine}\} \rightsquigarrow 1 = 2^0$$

$k>0$: Supposons $\#\{x : h(x)=k-1\} \leq 2^{k-1}$. Chaque nœud a ≤ 2 enfants

$$\text{Donc } \#\{x : h(x)=k\} \leq 2 \times 2^{k-1} = 2^k \quad \square$$

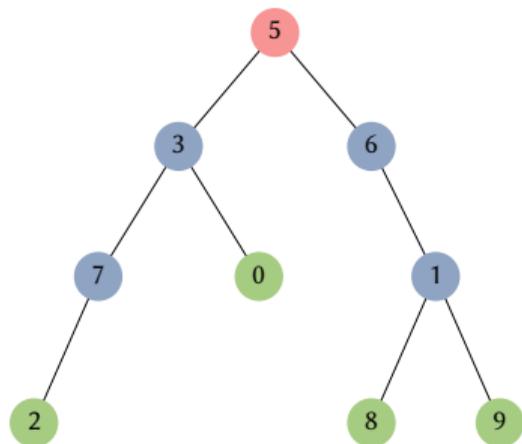
Définition

- ▶ **Hauteur** $h(x)$ d'un nœud x dans \mathcal{A} :
 - ▶ 0 si x est la racine de \mathcal{A}
 - ▶ $1 + h(p)$ sinon, où p est le parent de x
- ▶ **Hauteur** d'un arbre \mathcal{A} :
 $h(\mathcal{A}) = \max\{h(x) : x \in \mathcal{A}\}$

Lemme

$$\#\{x : h(x) = k\} \leq 2^k$$

Hauteur et niveaux



Définition

- ▶ **Hauteur** $h(x)$ d'un nœud x dans \mathcal{A} :
 - ▶ 0 si x est la racine de \mathcal{A}
 - ▶ $1 + h(p)$ sinon, où p est le parent de x
- ▶ **Hauteur** d'un arbre \mathcal{A} :

$$h(\mathcal{A}) = \max\{h(x) : x \in \mathcal{A}\}$$

Lemme

$$1 \leq n_k = \#\{x : h(x) = k\} \leq 2^k$$

Preuve

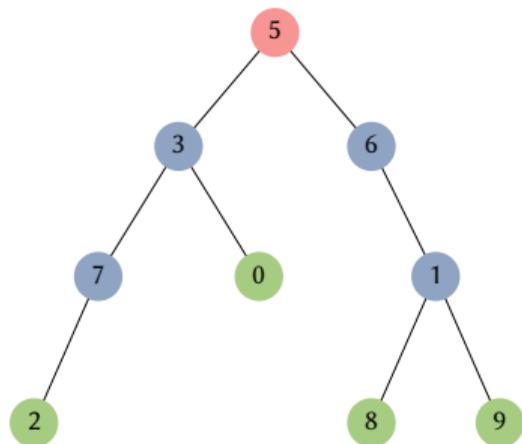
$$n = \sum_{k=0}^{h(\mathcal{A})} n_k$$

$$\Rightarrow \frac{\sum_{k=0}^{h(\mathcal{A})} 1}{1+h(\mathcal{A})} \leq n \leq \sum_{k=0}^{h(\mathcal{A})} 2^k = 1_2 + 10_2 + 100_2 + \dots + 10\dots0_2 = \underbrace{111\dots1}_{\leftarrow 1+h(\mathcal{A})}_2 = 2^{1+h(\mathcal{A})} - 1$$

Lemme

$$1 + h(\mathcal{A}) \leq n < 2^{1+h(\mathcal{A})} \quad \text{où } n = \#\mathcal{A}$$

Hauteur et niveaux



Preuve

- $n + h(\mathcal{A}) \leq n \Rightarrow h(\mathcal{A}) < n$
- $n \leq 2^{1+h(\mathcal{A})} \Rightarrow \log n < 1 + h(\mathcal{A})$
 $\Rightarrow \lfloor \log n \rfloor \leq h(\mathcal{A})$

Définition

- ▶ **Hauteur** $h(x)$ d'un nœud x dans \mathcal{A} :
 - ▶ 0 si x est la racine de \mathcal{A}
 - ▶ $1 + h(p)$ sinon, où p est le parent de x
- ▶ **Hauteur** d'un arbre \mathcal{A} :
 $h(\mathcal{A}) = \max\{h(x) : x \in \mathcal{A}\}$

Lemme

$$\#\{x : h(x) = k\} \leq 2^k$$

Lemme

$$1 + h(\mathcal{A}) \leq n < 2^{1+h(\mathcal{A})} \quad \text{où } n = \#\mathcal{A}$$

Corollaire

$$\lfloor \log(n) \rfloor \leq h(\mathcal{A}) \leq n$$

log

Un TAD « arbre binaire »

Définition

- ▶ Ensemble de données organisé de manière récursive
- ▶ Opérations :
 - ▶ $\text{NVARBRE}()$: nouvel arbre vide
 - ▶ $\text{NVARBRE}(r, G, D)$: nouvel arbre de racine r , et d'enfants G et D
 - ▶ $\text{ESTVIDE}(\mathcal{A})$: test
 - ▶ $\text{RACINE}(\mathcal{A}), \text{ENFANTG}(\mathcal{A}), \text{ENFANTD}(\mathcal{A})$: racine et enfants *uniquement si* $\neg \text{ESTVIDE}(\mathcal{A})$

dynamique

Complexités

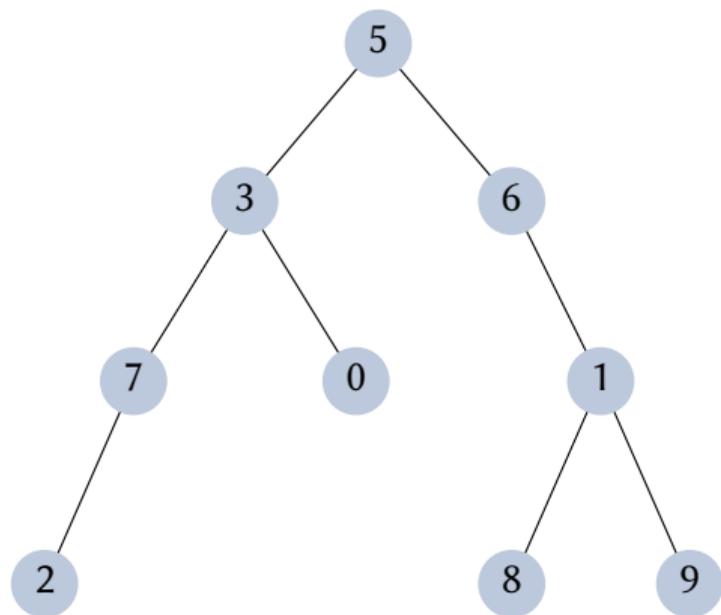
- ▶ Taille proportionnelle au nombre d'éléments
- ▶ Opérations en $O(1)$

Remarques

- ▶ Souvent : type des données à préciser
- ▶ TAD proche de l'implantation pratique
- ▶ Opération parfois ajoutée : $\text{PARENT}(\mathcal{A})$ en $O(1)$

nœuds et pointeurs

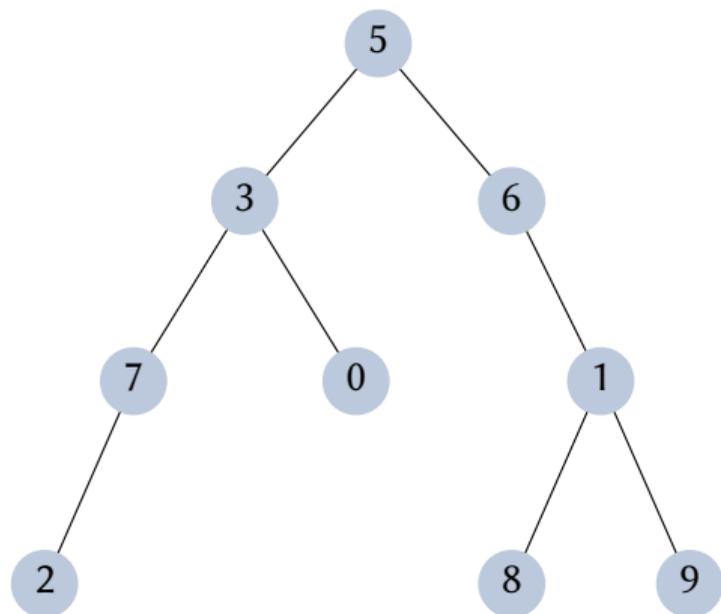
L'outil de base : le parcours en profondeur



PARCOURSINFIXE(\mathcal{A}):

1. Si NON(ESTVIDE(\mathcal{A})):
2. PARCOURSINFIXE(ENFANTG(\mathcal{A}))
3. *Traiter* RACINE(\mathcal{A})
4. PARCOURSINFIXE(ENFANTD(\mathcal{A}))

L'outil de base : le parcours en profondeur



PARCOURSINFIXE(\mathcal{A}):

1. Si NON(ESTVIDE(\mathcal{A})):
2. PARCOURSINFIXE(ENFANTG(\mathcal{A}))
3. *Traiter* RACINE(\mathcal{A})
4. PARCOURSINFIXE(ENFANTD(\mathcal{A}))

Correction

Chaque sommet est traité une fois et une seule

Complexité

$O(n)$ appels à *Traiter*

$n = \#\mathcal{A}$

Ordre de traitement

2 7 3 0 5 6 8 1 9

Deux variantes

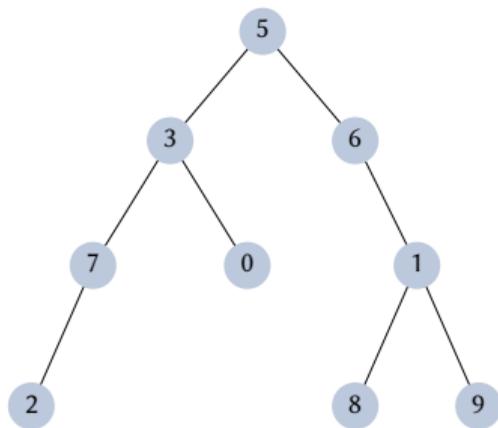
PARCOURS PRÉFIXE(\mathcal{A}):

1. Si NON(ESTVIDE(\mathcal{A})):
2. *Traiter* RACINE(\mathcal{A})
3. PARCOURS PRÉFIXE(ENFANTG(\mathcal{A}))
4. PARCOURS PRÉFIXE(ENFANTD(\mathcal{A}))

PARCOURS POSTFIXE(\mathcal{A}):

1. Si NON(ESTVIDE(\mathcal{A})):
2. PARCOURS POSTFIXE(ENFANTG(\mathcal{A}))
3. PARCOURS POSTFIXE(ENFANTD(\mathcal{A}))
4. *Traiter* RACINE(\mathcal{A})

5 3 7 2 0 6 1 8 9

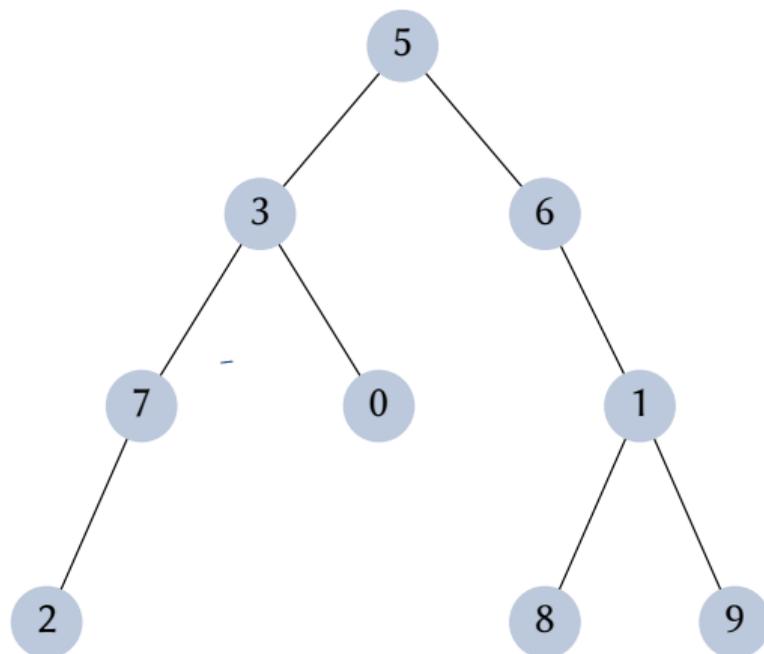


2 7 0 3 8 9 1 6 5

Exemples d'algorithmes

MINIMUM(\mathcal{A}):

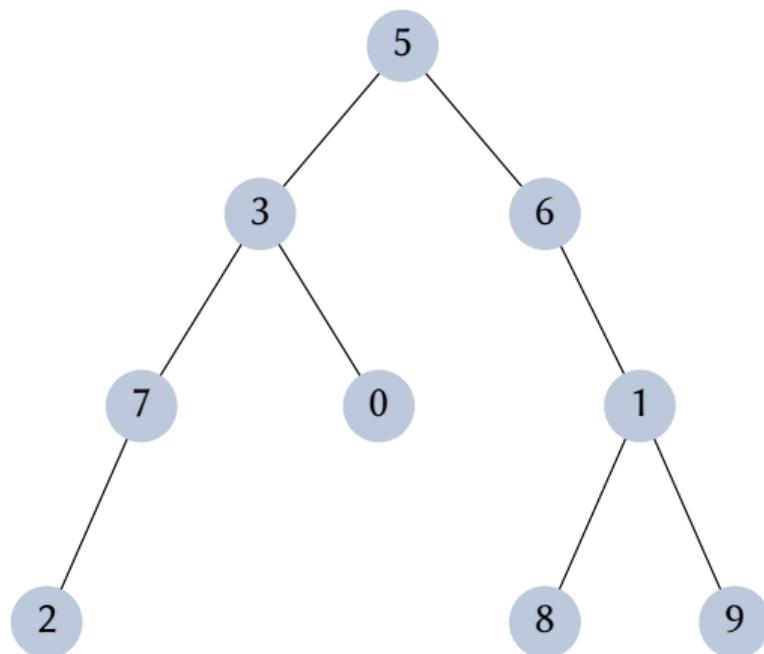
1. $m \leftarrow +\infty$
2. Si NON(ESTVIDE(\mathcal{A})):
3. $m_G \leftarrow \text{MINIMUM}(\text{ENFANTG}(\mathcal{A}))$
4. $m_D \leftarrow \text{MINIMUM}(\text{ENFANTD}(\mathcal{A}))$
5. $m \leftarrow \min(m_G, m_D, \text{RACINE}(\mathcal{A}))$
6. Renvoyer m



Exemples d'algorithmes

$\text{NBNEUDS}(\mathcal{A})$:

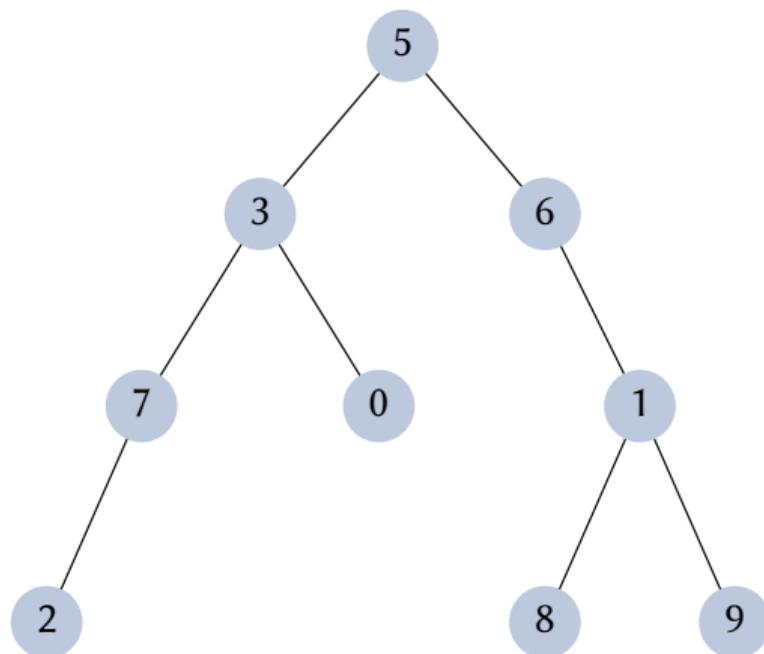
1. $n \leftarrow 0$
2. Si $\text{NON}(\text{ESTVIDE}(\mathcal{A}))$:
3. $n_G \leftarrow \text{NBNEUDS}(\text{ENFANTG}(\mathcal{A}))$
4. $n_D \leftarrow \text{NBNEUDS}(\text{ENFANTD}(\mathcal{A}))$
5. $n \leftarrow n_G + n_D + 1$
6. Renvoyer n



Exemples d'algorithmes

HAUTEUR(\mathcal{A}):

1. $h \leftarrow 0$
2. Si NON(ESTVIDE(\mathcal{A})):
3. $h_G \leftarrow$ HAUTEUR(ENFANTG(\mathcal{A}))
4. $h_D \leftarrow$ HAUTEUR(ENFANTD(\mathcal{A}))
5. $h \leftarrow 1 + \max(h_G + h_D)$
6. Renvoyer h



Bilan sur les arbres binaires

Une (autre) structure de base en algorithmique

- ▶ Représentation structurée de l'information
 - ▶ arbres binaires de recherche
 - ▶ tas
 - ▶ ...

Bilan sur les arbres binaires



Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

Teams

Q&A for work



Learn More

What are the applications of binary trees?

Applications of binary trees

▲
380



- [Binary Search Tree](#) - Used in *many* search applications where data is constantly entering/leaving, such as the `map` and `set` objects in many languages' libraries.
- [Binary Space Partition](#) - Used in almost every 3D video game to determine what objects need to be rendered.
- [Binary Tries](#) - Used in almost every high-bandwidth router for storing router-tables.
- [Hash Trees](#) - used in p2p programs and specialized image-signatures in which a hash needs to be verified, but the whole file is not available.
- [Heaps](#) - Used in implementing efficient priority-queues, which in turn are used for scheduling processes in many operating systems, Quality-of-Service in routers, and A* (*path-finding algorithm used in AI applications, including robotics and video games*). Also used in heap-sort.
- [Huffman Coding Tree \(Chip Uni\)](#) - used in compression algorithms, such as those used by the .jpeg and .mp3 file-formats.
- [GGM Trees](#) - Used in cryptographic applications to generate a tree of pseudo-random numbers.
- [Syntax Tree](#) - Constructed by compilers and (implicitly) calculators to parse expressions.
- [Treap](#) - Randomized data structure used in wireless networking and memory allocation.
- [T-tree](#) - Though most databases use some form of B-tree to store data on the drive, databases

Bilan sur les arbres binaires

Une (autre) structure de base en algorithmique

- ▶ Représentation structurée de l'information
 - ▶ arbres binaires de recherche
 - ▶ tas
 - ▶ ...

Arbres binaires comme TAD

- ▶ Utilisé dans ce cours pour construire d'autres TAD
- ▶ Proche des implantations pratiques en programmation

À réviser (aussi) si vous n'êtes pas à l'aise !

- ▶ Exemples :
 - ▶ compter le nombre de feuilles
 - ▶ recherche d'un élément particulier

Conclusion

Type abstrait de données

- ▶ Décrit un ensemble de *données* et les opérations qu'on effectue dessus
- ▶ Ne spécifie pas la représentation des données ni l'implantation des opérations

→ Focus sur les fonctionnalités

Usage en algorithmique et dans ce cours

- ▶ Description de TADS *classiques*
 - ▶ Utilisés fréquemment et implantés dans beaucoup de (bibliothèques de) langages
 - ▶ Réalisation algorithmique de TADS
- ▶ Description d'algorithmes basés sur les TADS

→ Fixe les limites des opérations autorisées dans un pseudo-code

C'est nouveau pour vous ?

Non vous en utilisez depuis longtemps sans le savoir...

Oui nouveau cadre formel d'étude des algorithmes