

Cours 4 : Les arbres binaires

Définition
Implémentation
Manipulation

Définition

- ❖ Un arbre binaire est un arbre qui possède au maximum deux sous-arbres (d'où le *binaire*)

Deux implémentations possibles

❖ Version itérative

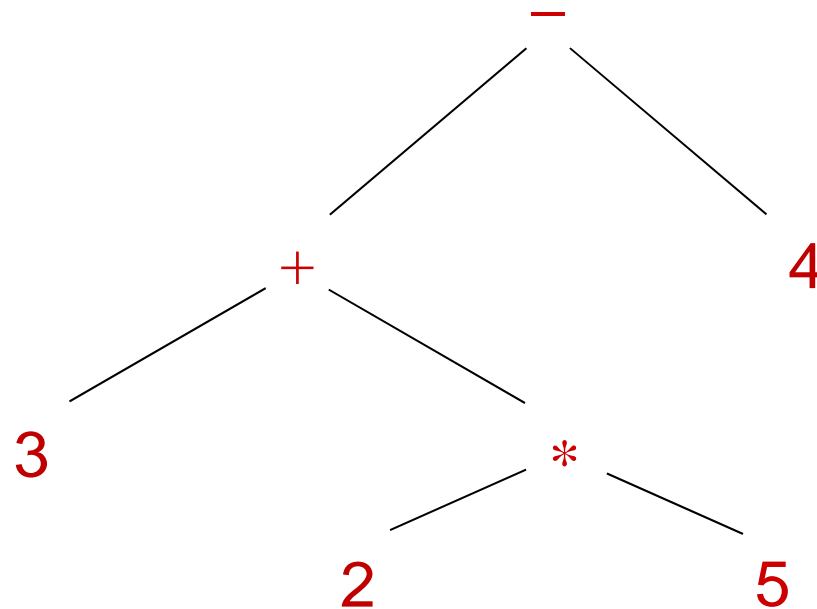
- Un arbre binaire est constitué de nœuds
- Chaque nœud « pointe » vers deux nœuds de l'étage inférieur

❖ Version récursive

- Un arbre binaire peut être vide
- Un arbre binaire possède un nœud (étiqueté ou pas)
- Un arbre binaire possède deux sous-arbres « fils »

Utilisation

- ❖ Enormément d'applications, que ce soit dans le domaine informatique ou pas :
 - Expressions mathématiques : $3 + 2 * 5 - 4$



Autres exemples

- ❖ Résultats d'un tournoi à élimination directe (tennis par exemple)
- ❖ Arborescence de *si-alors-sinon* (voir les arbres de décision pour les tris vus à la première séance)

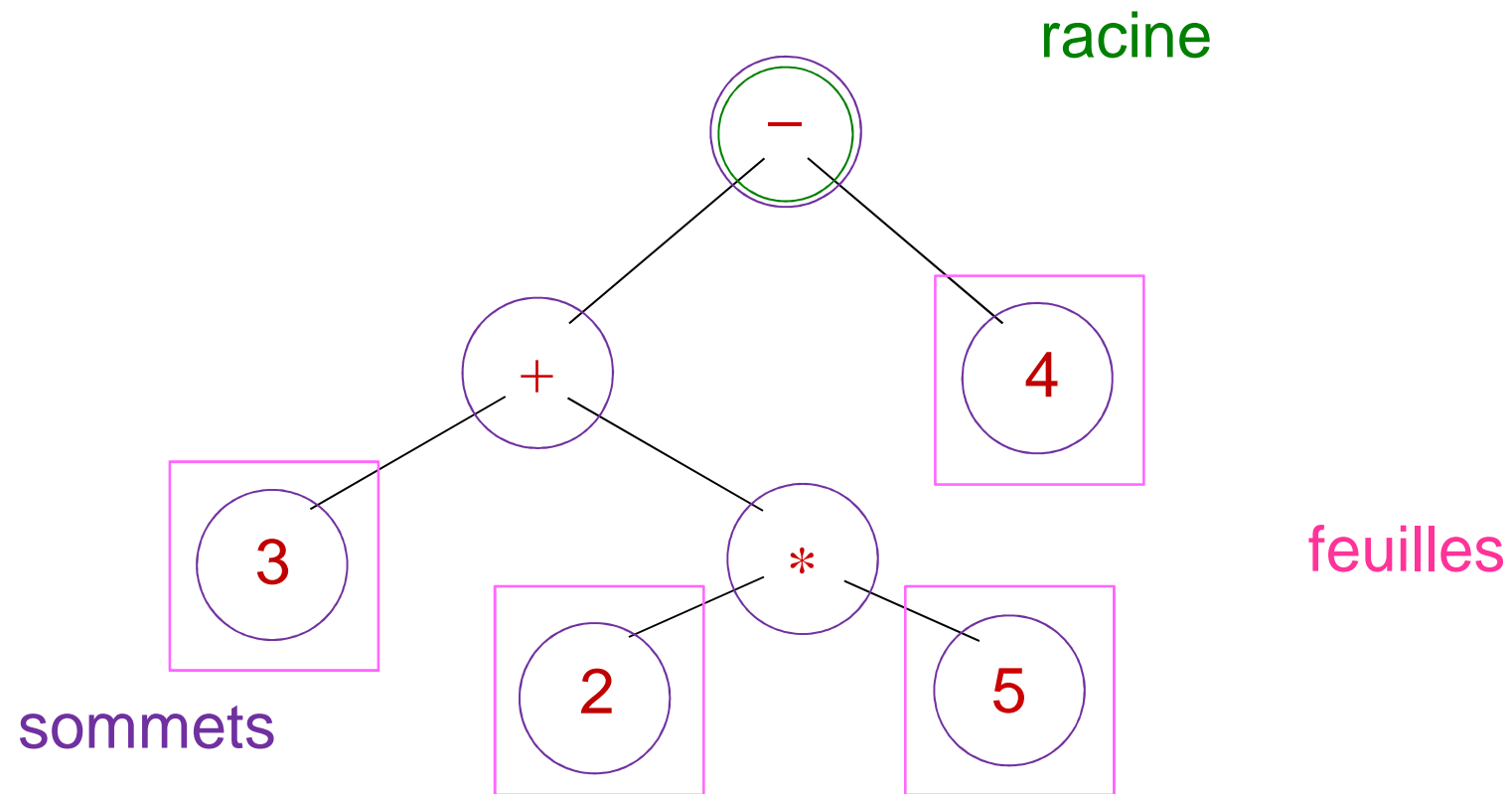
Un peu de vocabulaire

- ❖ Un arbre est constitué de *nœuds* (ou *sommets*)
- ❖ Ces sommets sont reliés par des *arcs* (ou *arêtes*) orientés : père \rightarrow fils
- ❖ Il existe dans tout arbre un nœud qui n'est le point d'arrivée d'aucun arc : c'est la *racine*

Un peu de vocabulaire (2)

- ❖ Tout autre nœud est la racine d'un sous-arbre de l'arbre principal
- ❖ Un nœud qui n'est le point de départ d'aucun arc est appelé *feuille*
- ❖ Pour les arbres binaires, on distinguera de façon visuelle le fils gauche du fils droit

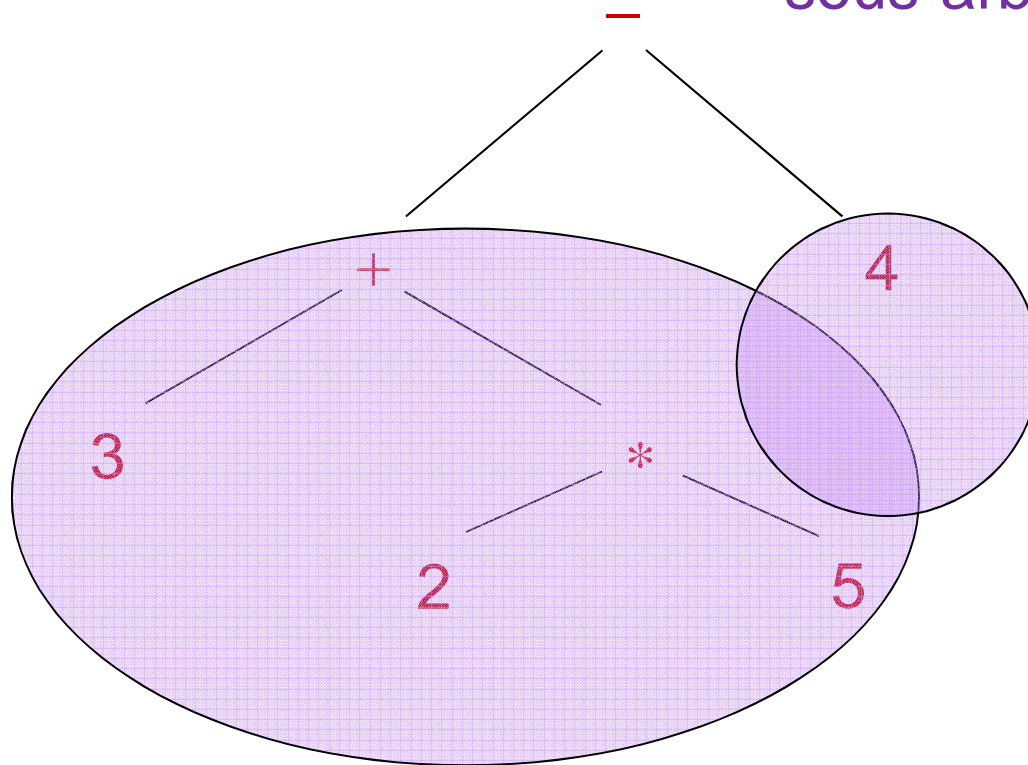
Un exemple



Un exemple

sous-arbre gauche de « - »

sous-arbre droit de « - »



Un peu de vocabulaire (3)

- ❖ La *hauteur* d'un nœud est la longueur du plus long chemin de ce nœud aux feuilles qui en dépendent plus 1
 - C'est le nombre de nœuds du chemin
 - La *hauteur d'un arbre* est la hauteur de sa racine
 - L'arbre vide a une hauteur 0
 - L'arbre réduit à une racine étiquetée a une hauteur 1

Un peu de vocabulaire (4)

- ❖ La *profondeur* d'un nœud est le nombre de nœuds du chemin qui va de la racine à ce nœud
 - La racine d'un arbre est à une profondeur 0
 - La profondeur d'un nœud est égale à la profondeur de son père plus 1
 - Si un nœud est à une profondeur p , tous ses fils sont à une profondeur $p+1$
- ❖ Tous les nœuds d'un arbre de même profondeur sont au même niveau

Premier traitement sur un arbre

❖ L'affichage

❖ Trois façons d'afficher

- Préfixe
- Infixe
- Postfixe

Affichages

❖ Préfixe :

- On affiche la racine, puis le sous-arbre gauche, puis le sous-arbre droit

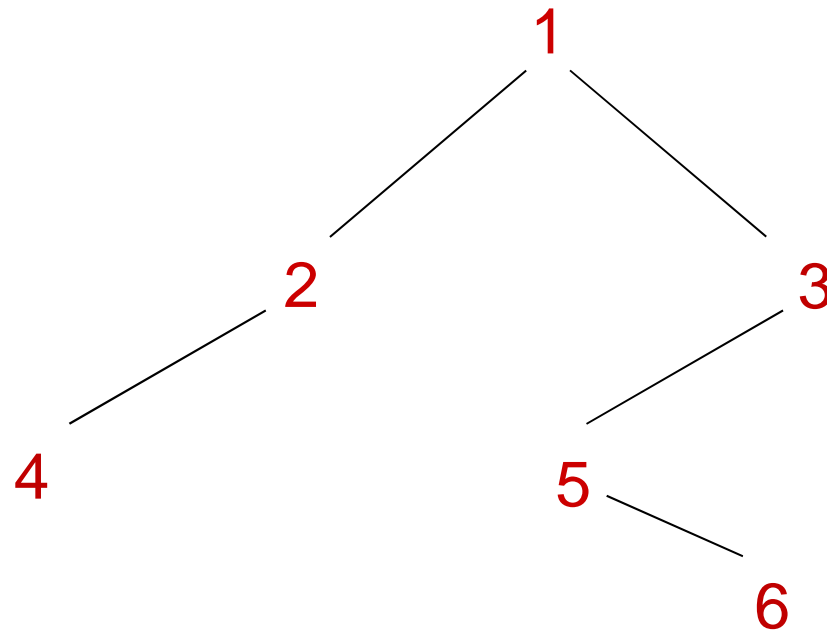
❖ Infixe :

- On affiche le sous-arbre gauche, puis la racine, puis le sous-arbre droit

❖ Postfixe :

- On affiche le sous-arbre gauche, puis le sous-arbre droit, puis la racine

Un exemple



Résultats

❖ Préfixe :

- 1 2 4 3 5 6

❖ Infixe :

- 4 2 1 5 6 3

❖ Postfixe

- 4 2 6 5 3 1

Deuxième traitement

❖ Recherche d'un élément

❖ Au moins deux façons de faire :

- DFS : *Depth-First Search* ou recherche en profondeur d'abord
- BFS : *Breadth-First Search* ou recherche en largeur d'abord

DFS en détail

❖ DFS :

- On cherche à la racine
- S'il n'y est pas :
 - On cherche dans le fils gauche
 - Puis s'il n'était pas dans le fils gauche, on cherche dans le fils droit

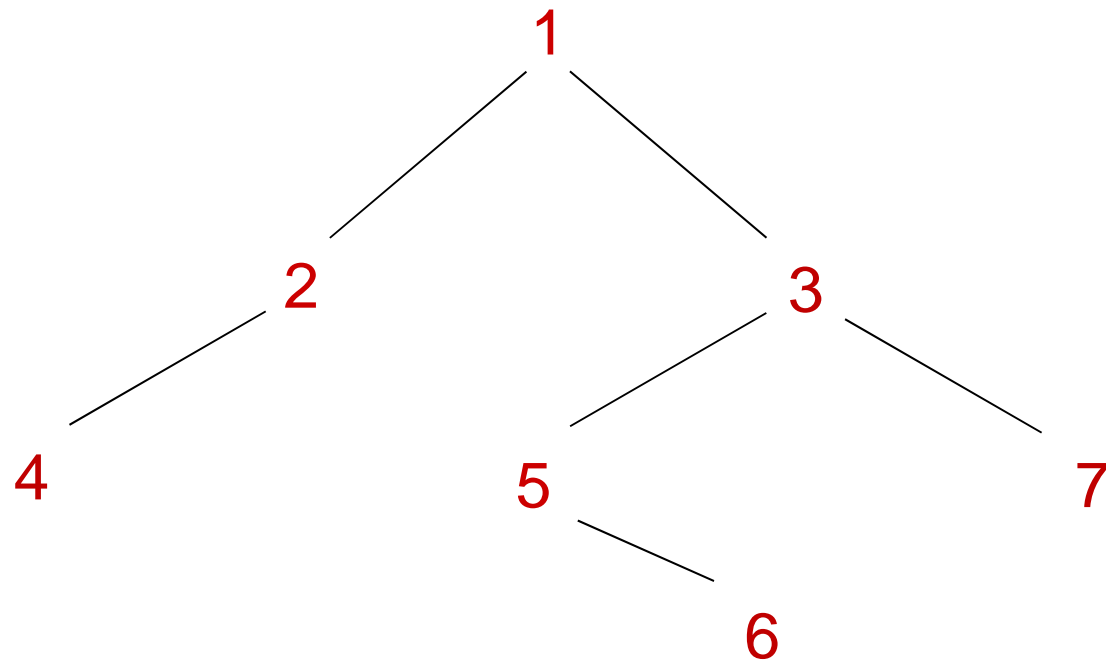
❖ Idée : explorer à fond chaque branche avant de passer à la suivante

❖ On s'arrête quand on a trouvé ou qu'il n'y a plus de branches à explorer

Réflexions

- ❖ On reconnaît encore une fois un fonctionnement récursif
- ❖ Problème : si l'élément est dans le sous arbre droit de la racine, on va quand même explorer tout le sous-arbre de gauche

Examples



Avec DFS : on cherche 4

- ❖ Racine = 1 \rightarrow *non*
 - On explore le fils gauche
 - Racine = 2 \rightarrow *non*
 - On explore le fils gauche
 - Racine = 4 \rightarrow *TROUVÉ*

Avec DFS (2) : on cherche 3

❖ Racine = 1 \rightarrow *non*

- On explore le fils gauche
 - Racine = 2 \rightarrow *non*
 - On explore le fils gauche
 - Racine = 4 \rightarrow *non*
 - Pas de fils
 - Pas de fils droit
- On explore le fils droit
 - Racine = 3 \rightarrow *TROUVÉ*

BFS en détail

❖ BFS :

- On cherche à la racine
- Si l'élément n'y est pas :
 - On cherche dans les nœuds de profondeur 1
 - S'il n'est pas dans à la profondeur 1, on cherche à la profondeur 2
 - Etc...

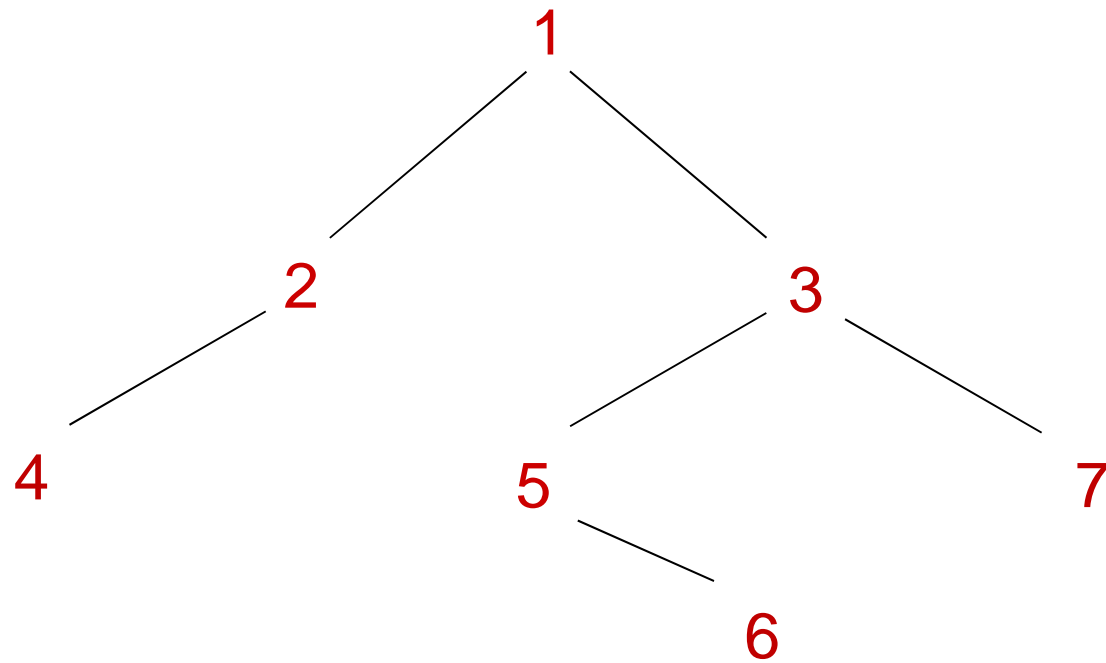
Réflexions

- ❖ Plutôt un fonctionnement itératif
- ❖ On va avoir besoin d'une file FIFO (ou autre structure équivalente) pour stocker les nœuds à explorer :
 - On ne peut pas « sauter » d'un nœud de profondeur p à un autre
 - Il faut se souvenir de la liste des nœuds à traiter

Fonctionnement de la file

- ❖ Au début, la file contient l'arbre principal
- ❖ Si la valeur n'est pas à la racine du premier arbre de la file
 - On supprime l'arbre de la file
 - On ajoute ses deux sous-arbre en fin de file s'ils ne sont pas vides
 - Et on explore l'arbre suivant, qui est le premier de la file
- ❖ On s'arrête quand on a trouvé ou que la file est vide

Examples



Avec BFS : on cherche 3

- ❖ File = [arbre « 1 »]
- ❖ Valeur de la racine = 1 → *non*
- ❖ On enlève l'arbre « 1 » et on ajoute ses deux sous-arbres
 - File = [*sous-arbre gauche de « 1 », sous-arbre droit de « 1 »*]
 - On explore le premier : racine = 2 → *non*
 - On l'enlève et on ajoute ses sous-arbres
 - File= [*sous-arbre droit de « 1 », sous-arbre gauche de « 2 »*]
 - On explore le premier : racine = 3 → TROUVÉ

Avec BFS (2) : on cherche 4

- ❖ File = [arbre « 1 »]
- ❖ Valeur de la racine = 1 → *non*
- ❖ On enlève l'arbre « 1 » et on ajoute ses deux sous-arbres
 - File = [s.-a. gauche de « 1 », s.-a. droit de « 1 »]
 - On explore le premier : racine = 2 → *non*
 - On l'enlève et on ajoute ses deux sous-arbres
 - File = [s.-a. droit de « 1 », s.-a. gauche de « 2 »]
 - On explore le premier : racine = 3 → *non*
 - On l'enlève et on ajoute ses deux sous-arbres
 - File = [s.-a. gauche de « 2 », s.-a. gauche de « 3 », s.-a. droit de « 3 »]
 - On explore le premier : racine = 4 → *TROUVÉ*

Comparatif

- ❖ Avec DFS, on n'a besoin d'aucun stockage en plus mais on peut s'attarder trop longtemps dans une branche
- ❖ Avec BFS, on fonctionne par profondeur incrémentale donc on évite le piège mais on a besoin d'un stockage dont la taille augmente à chaque étape

Les ABR

- ❖ On voit que la recherche dans les arbres binaires est peu aisée
- ❖ Autre problème : où ajouter un élément ?
- ❖ Solution : les ABR ou arbres binaires de recherche

Définition

- ❖ Un *arbre binaire de recherche* (*ABR*) est un arbre binaire dans lequel chaque nœud possède une clé telle que
- chaque nœud du sous-arbre *gauche* possède une clé inférieure ou égale à celle du nœud considéré
 - chaque nœud du sous-arbre *droit* possède une clé supérieure ou égale à celle-ci
 - (on pourra interdire ou non des clés de valeur égale)

Pourquoi les problèmes sont-ils réglés ?

- ❖ Pour la recherche : à partir de la racine, on sait si on doit explorer le fils gauche ou le fils droit
 - DFS devient très efficace
- ❖ Pour l'ajout, le nouveau nœud ajouté devient une feuille
 - Ici aussi il suffit de faire une DFS en descendant systématiquement à droite ou à gauche suivant qu'on est inférieur ou supérieur à la racine

Et si on veut supprimer un nœud ?

❖ Si c'est une feuille

- Facile, rien à faire
- On garde un ABR

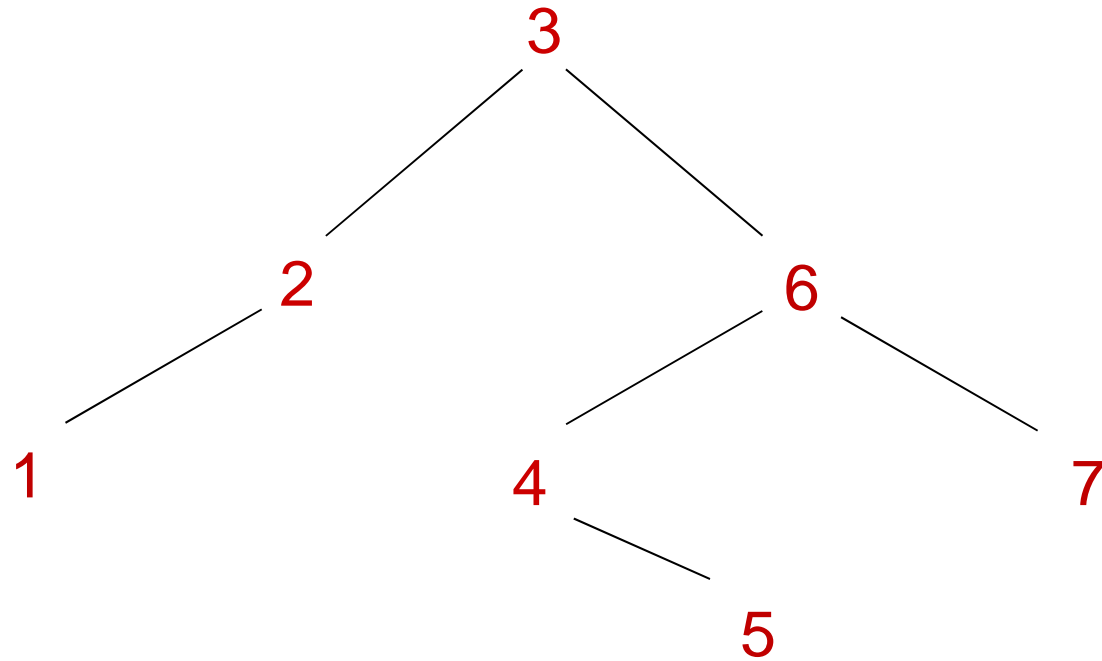
❖ Sinon si c'est un nœud avec un seul s.-a.

- On le remplace par son sous-arbre

❖ Sinon

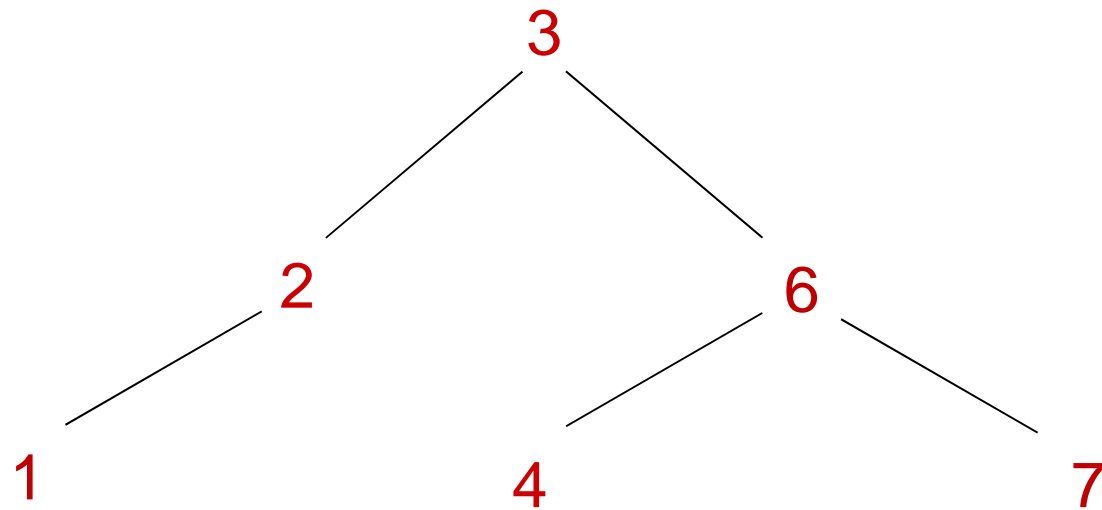
- On lui donne la valeur minimale de son sous-arbre droit et on supprime ce nœud (récursif)

Exemple

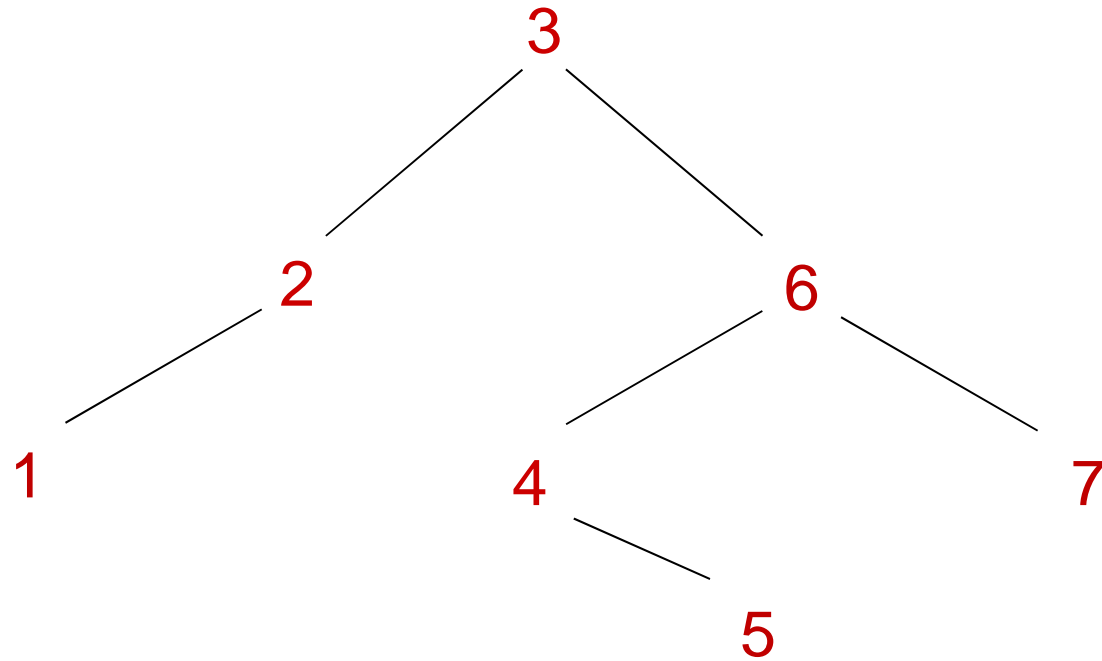


On supprime 5

On supprime 5

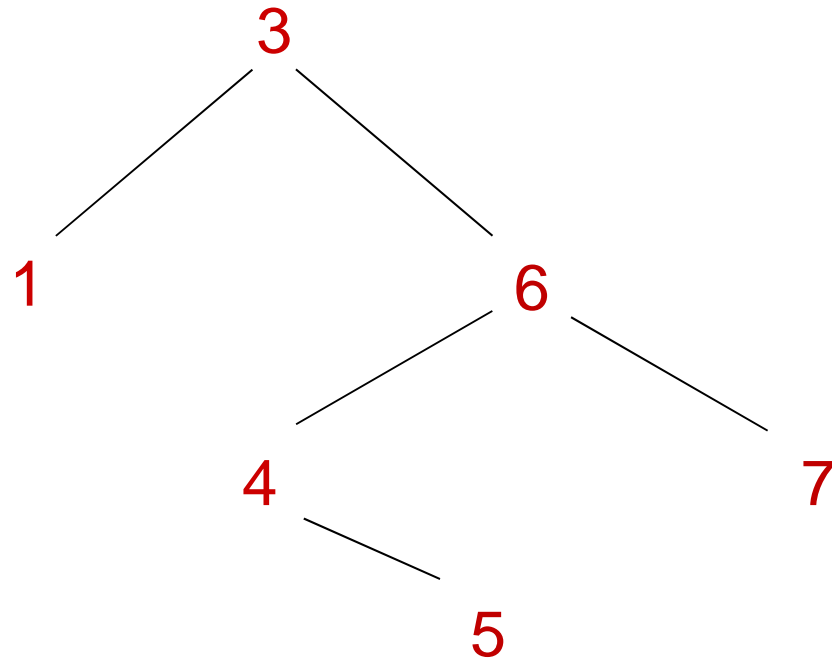


Exemple

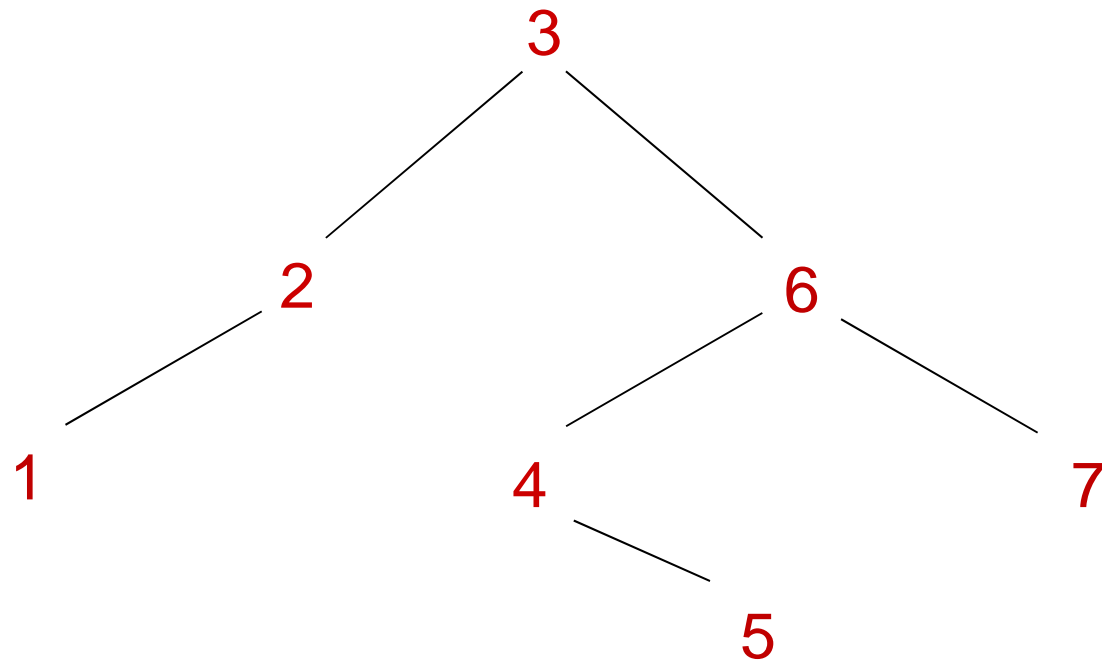


On supprime 2

On supprime 2



Exemple



On supprime 3

On supprime 3

