

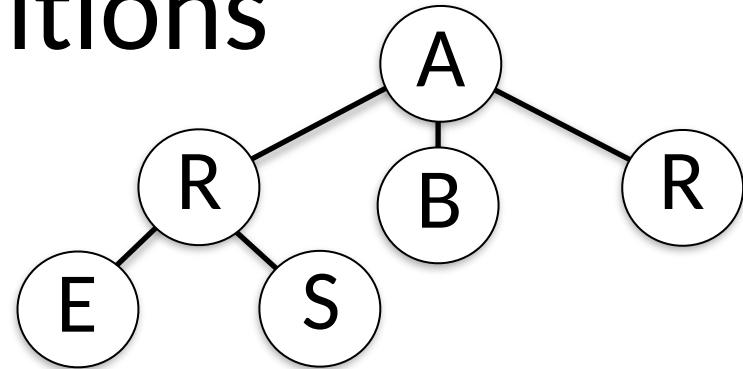
Structures de Données

Séance 3 – Arbres binaires

INFRES 12 – janvier 2020

Arbres - Définitions

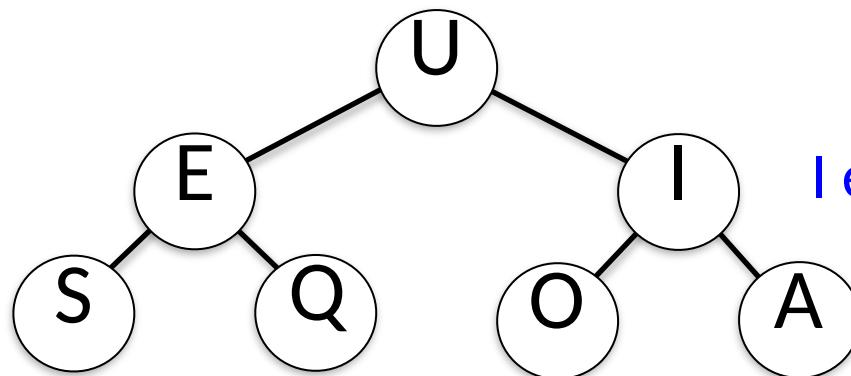
- Structure de données constituée de **nœuds** et d'**arêtes** reliant ses nœuds.
- Seuls les nœuds portent des informations.
- Un et un seul nœud joue le rôle de **racine**.
- Une **branche** ou **chemin** est une suite de nœuds à traverser pour aller d'un nœud à un autre.
- Chaque nœud est relié à la racine par une seule branche.



Arbres - Définitions

- Analogie avec un arbre généalogique : **père, fils, frères.**
- Chaque nœud sauf la racine possède un père.
- Un nœud sans fils est appelé **feuille** ou **nœud terminal**.

U est la racine



I et E sont les fils de U

S et Q sont frères

S, Q, O et A sont des feuilles

Arbres - Définitions

- Le **degré** d'un nœud est le nombre de ses fils.
- Le **niveau** d'un nœud est égal au nombre d'arêtes qui le séparent de la racine.
- La **hauteur** d'un arbre est son niveau maximal.
- La **longueur totale** est la somme des longueurs de toutes les branches = somme des niveaux des nœuds.

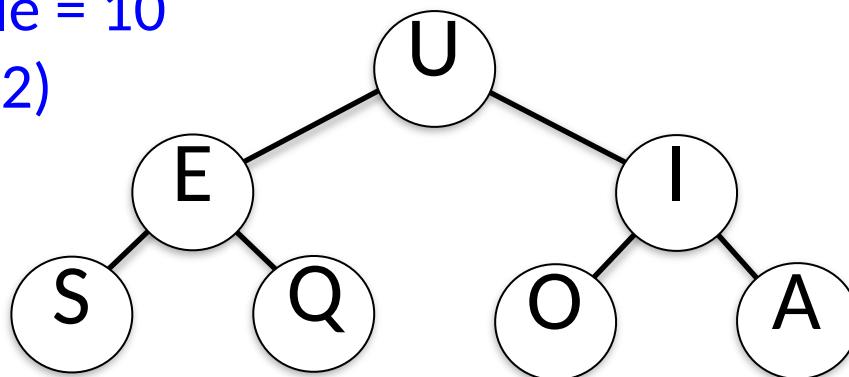
Longueur totale = 10

$$= 0 + 2 * 1 + 4 * 2$$

Degrés :

U, E et I : 2

S, Q, O et A : 0



Niveau 0

Niveau 1

Niveau 2

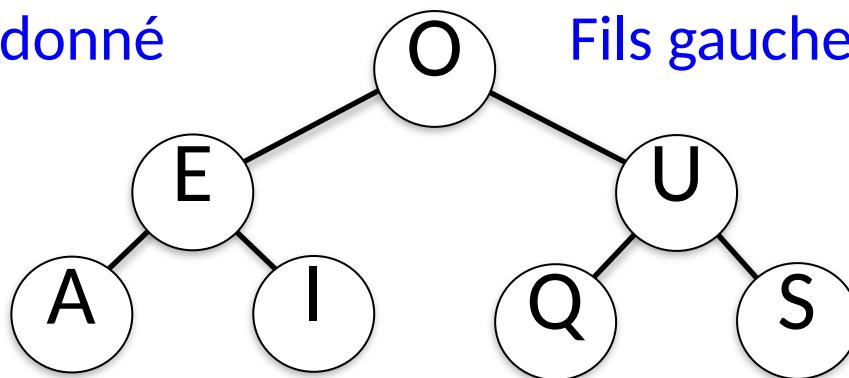
Hauteur = 2

Arbres - Définitions

- **Structure récursive** : chaque nœud est la racine d'un sous arbre.
- Un arbre est **ordonné** s'il y a une relation d'ordre entre un nœud et ses fils.

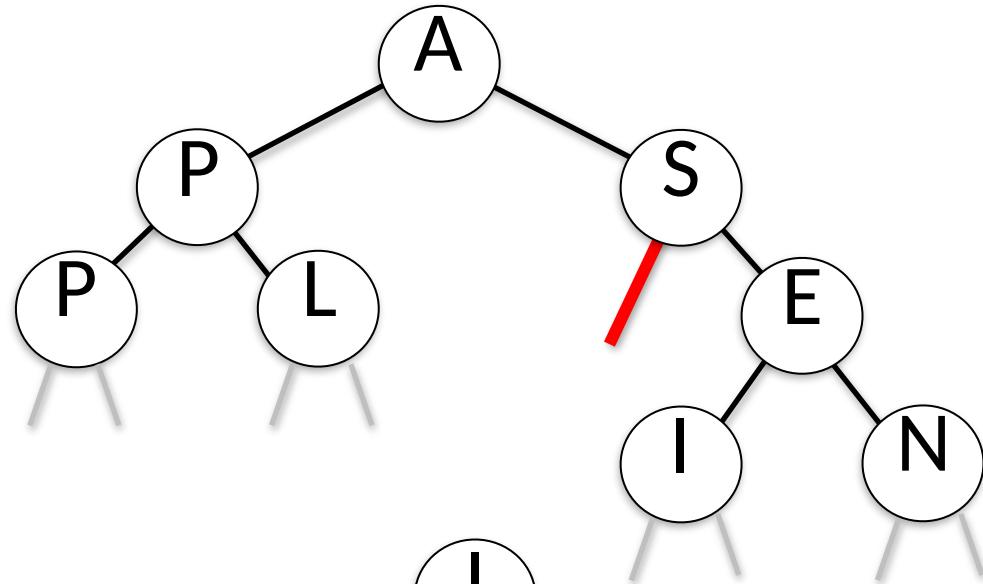
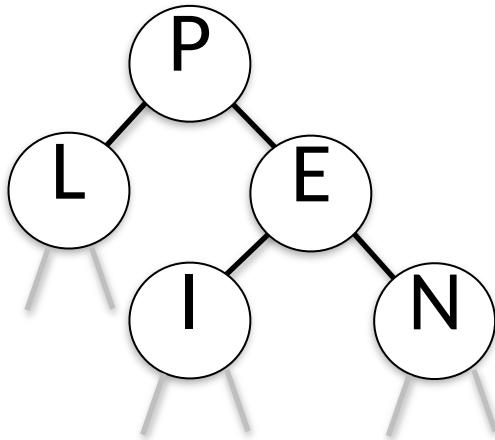
Arbre binaire ordonné

Fils gauche < nœud < fils droit

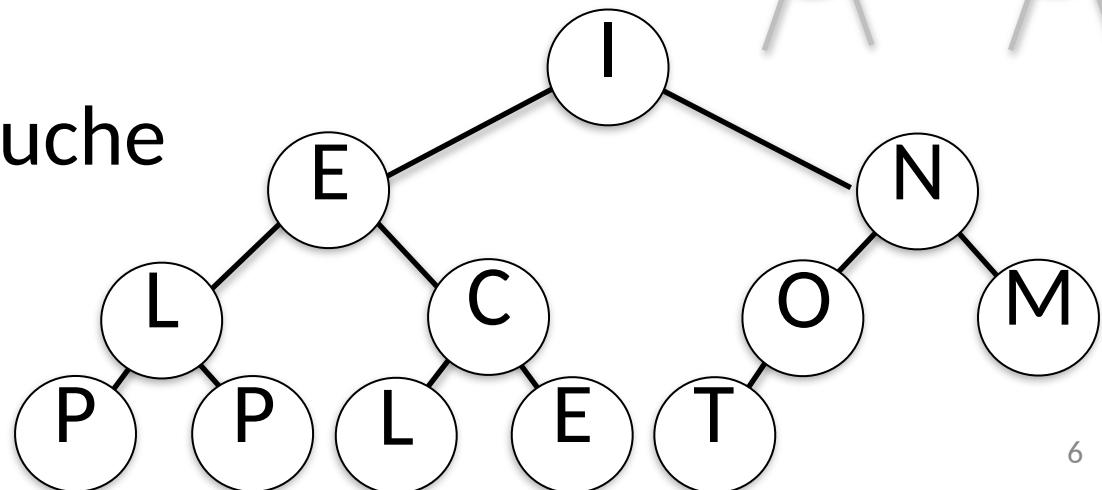


Arbres - Définitions

- Arbre **plein** : fils absents qu'au dernier niveau

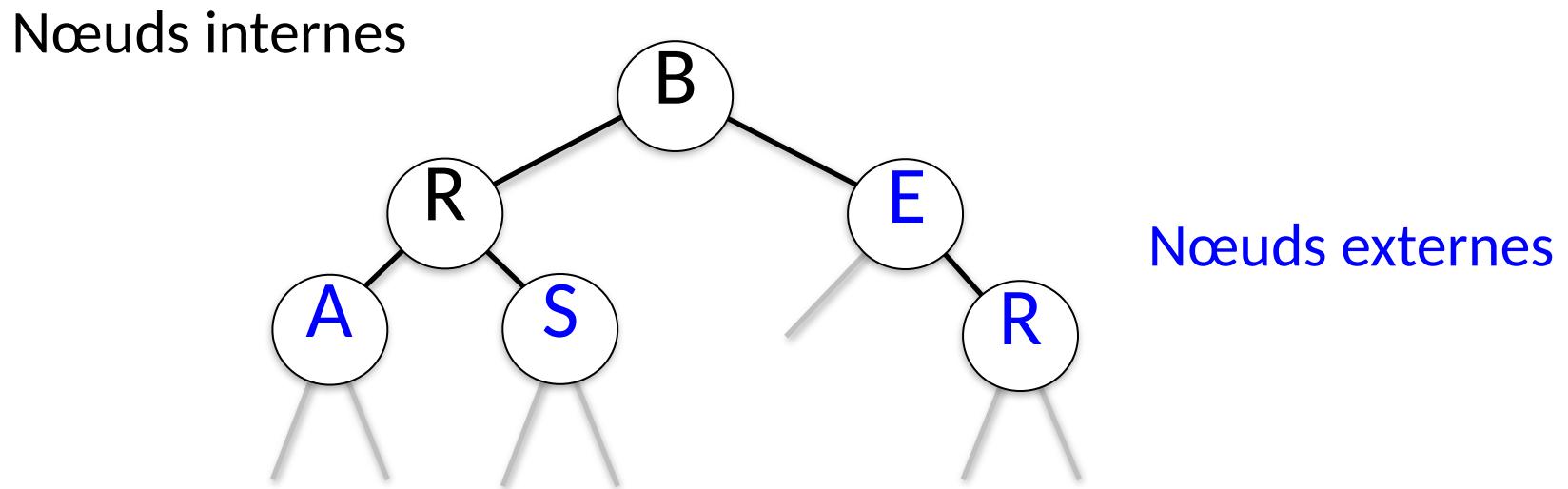


- Arbre **complet** :
plein + feuilles à gauche



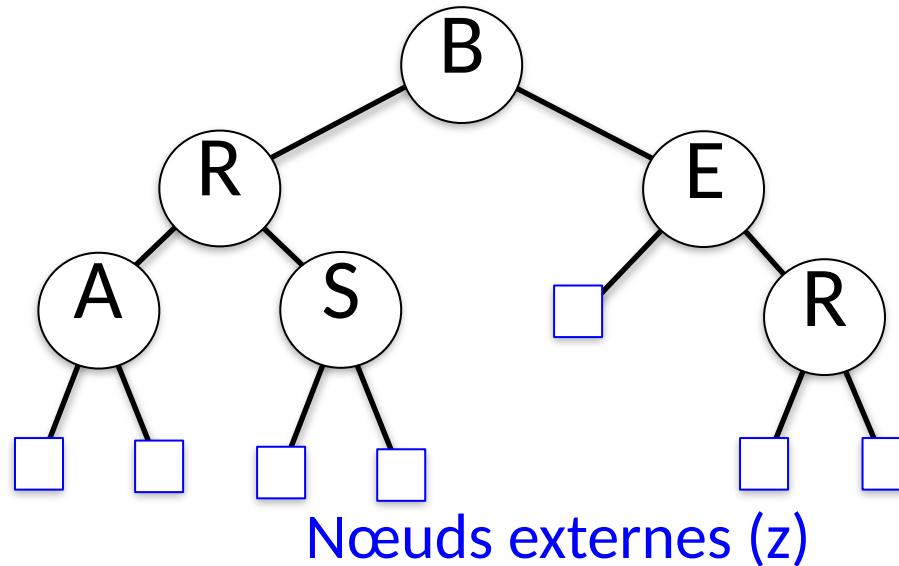
Arbres - Définitions

- **Nœud externe** : a au moins un fils non défini.
- **Nœud interne** : a tous ses fils définis



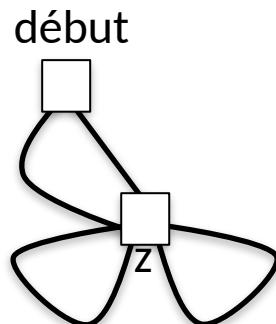
Arbres - Définitions

- **Nœud externe** : a au moins un fils non défini.
- **Nœud interne** : a tous ses fils définis
- Si l'on ajoute des nœuds externes vides (dits factices), seuls ces nœuds sont externes.

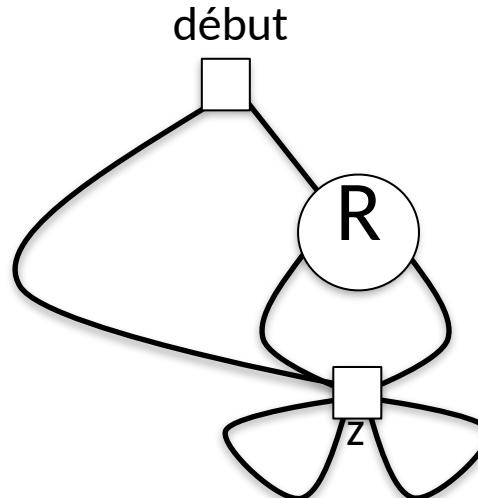


Arbre- Nœuds factices

- Un seul nœud factice pour les nœuds terminaux qui est son propre fils gauche et son propre fils droit.
- Un nœud factice de début : la racine est son fils droit.



Arbre vide



Arbre binaire

Implantation par pointeur

```
struct noeud{  
    int cle;  
    struct noeud *g;  
    struct noeud *d;  
};
```

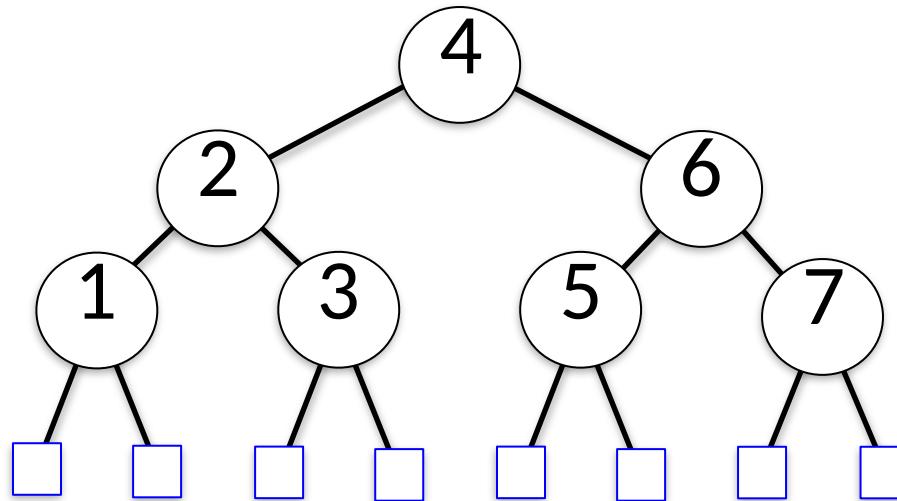
```
struct arbre {  
    noeud *debut;  
    noeud *z;  
};
```

Parcours d'arbre

- Visiter tous les nœuds une et une seule fois
- Parcours en profondeur :
 - Parcours préfixé : nœud, gauche, droit
 - Parcours infixé : gauche, nœud, droit
 - Parcours postfixé : gauche, droit, nœud
- Parcours en largeur :
 - par niveau, de gauche à droite dans chaque niveau.

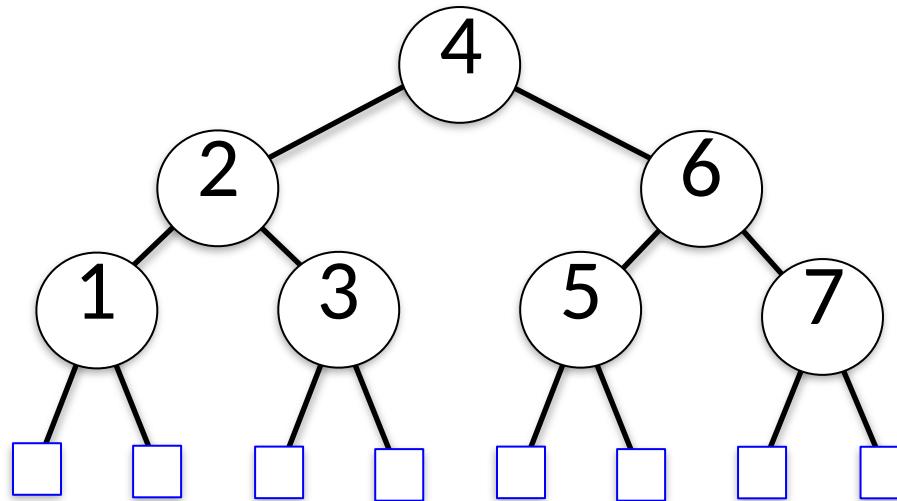
Parcours prefixé

- racine, gauche, droit



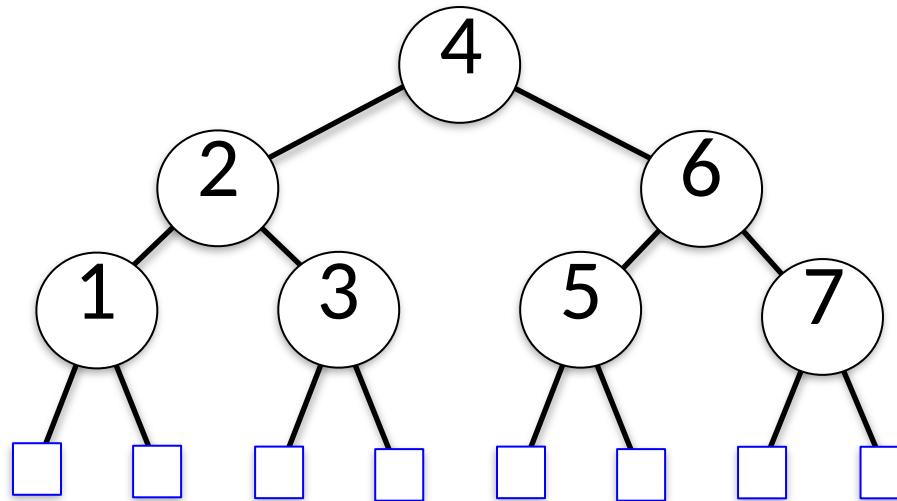
Parcours prefixé

- racine, gauche, droit : 4213657



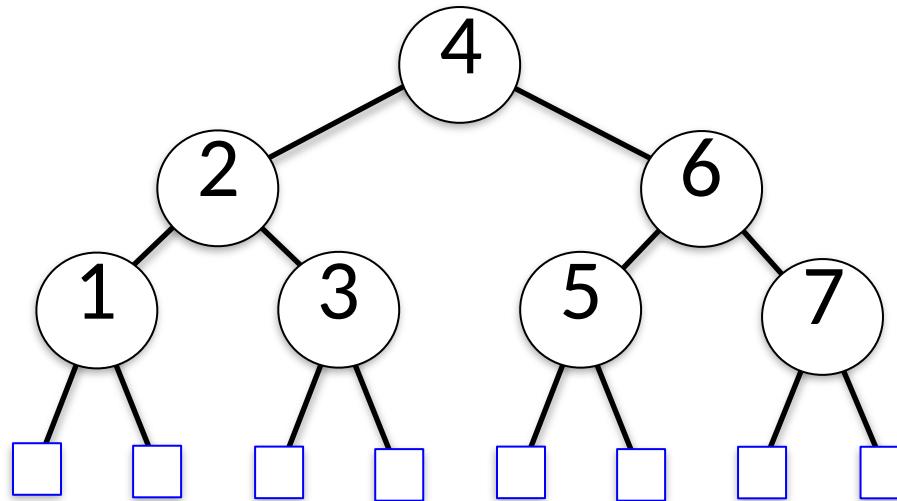
Parcours infixé

- gauche, racine, droit



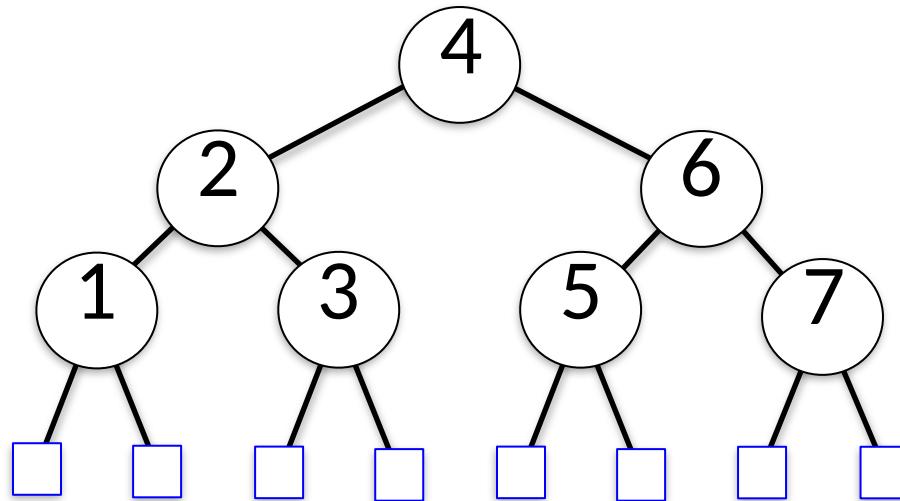
Parcours infixé

- gauche, racine, droit : 1234567



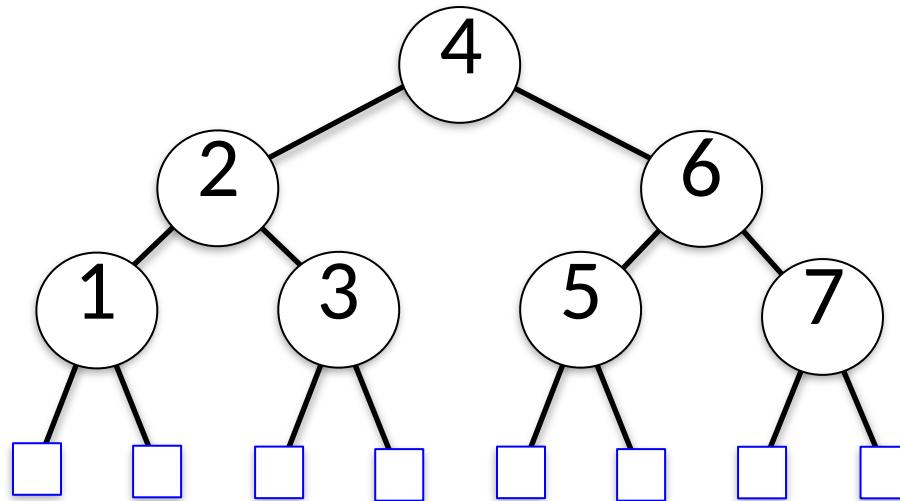
Parcours postfixé

- gauche, droit, racine



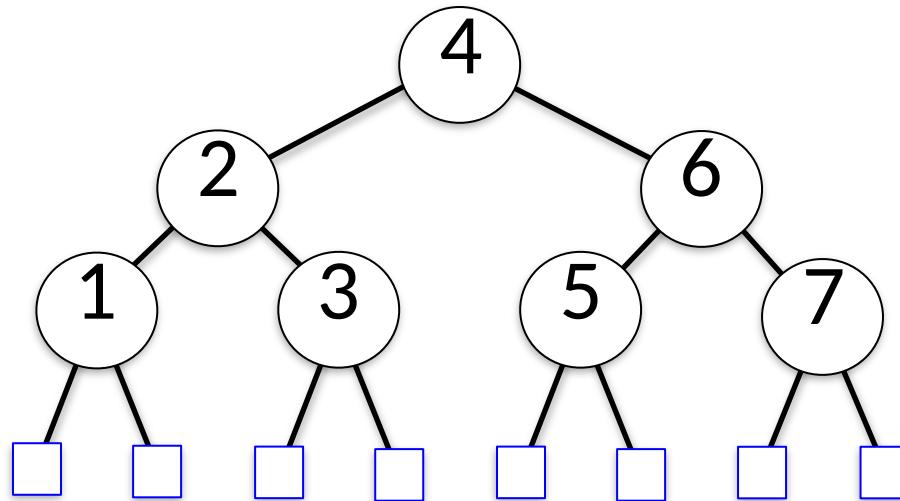
Parcours postfixé

- gauche, droit, racine : 1325764



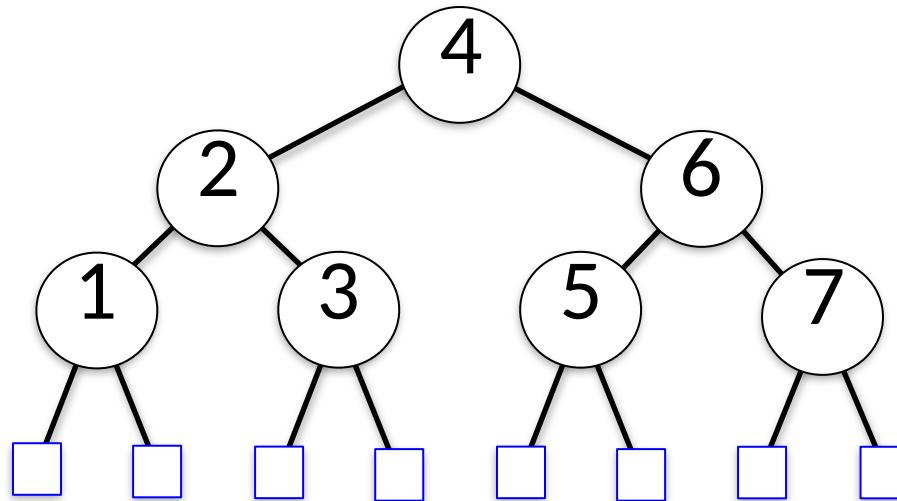
Parcours en largeur

- Par niveau, de gauche à droite



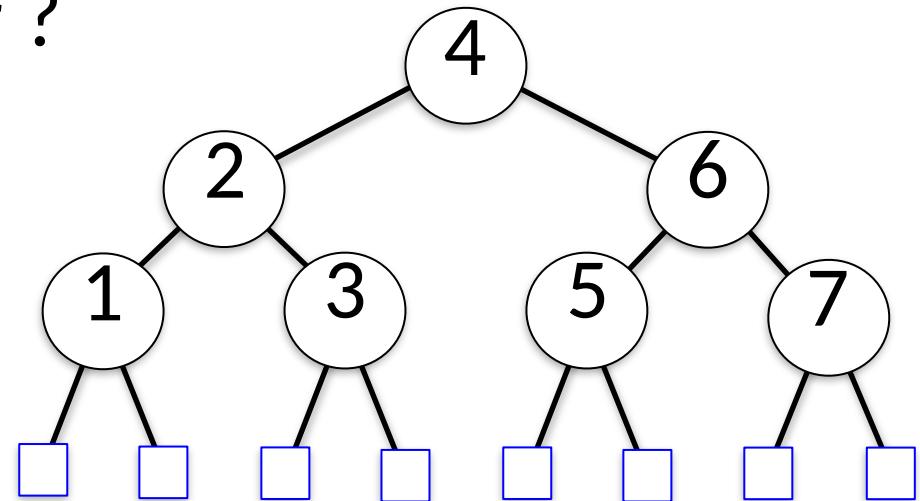
Parcours en largeur

- Par niveau, de gauche à droite : 4261357



Algorithmes de parcours

- Parcours en profondeur ?



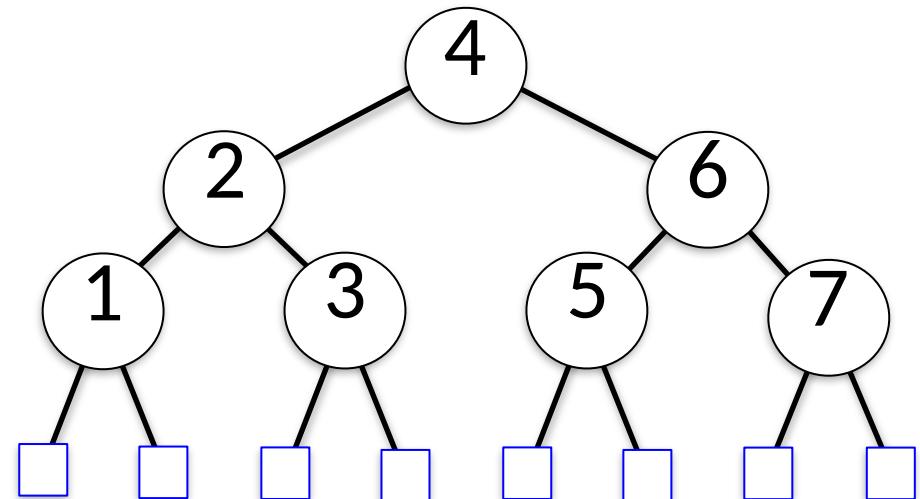
Parcours récursif

- Exemple : infixé

```
void parcourir(nœud *n) {  
    if (n->g != z) parcourir(n->g);  
    afficher(n);  
    if (n->d != z) parcourir(n->d);  
} // z est le noeud factice
```

Algorithmes de parcours

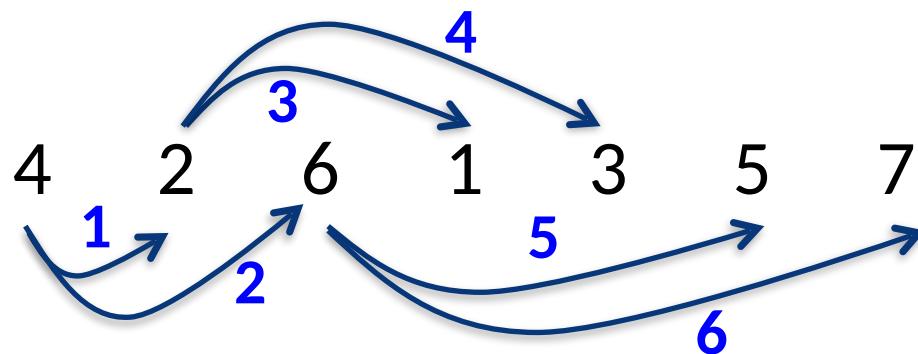
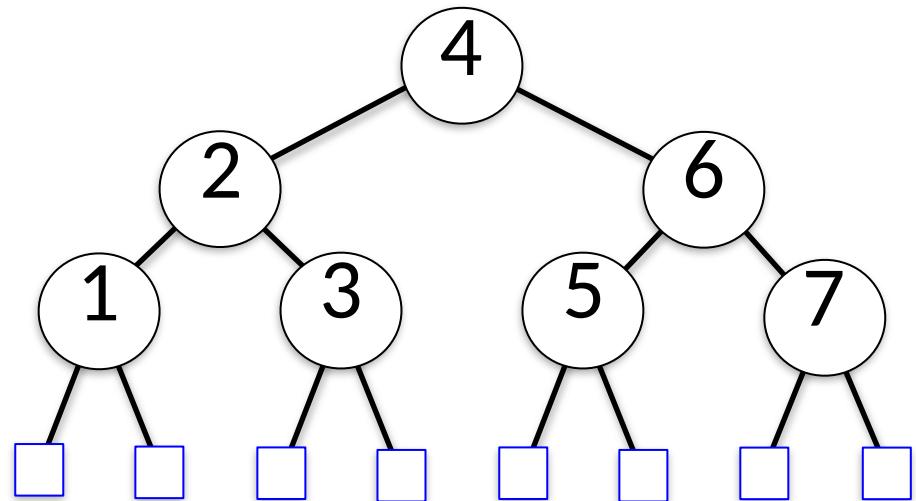
- Parcours en largeur ?



Algorithmes de parcours

- Parcours en largeur :

⇒ 4261357



Parcours avec une file

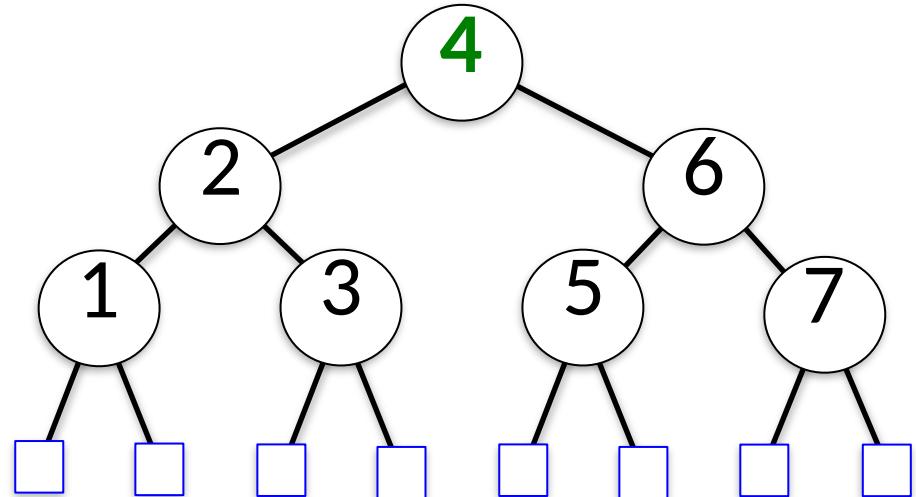
- Utilisation d'une file :

```
enfiler(racine);  
while (!estvide(file)) {  
    n = defiler();  
    afficher(n);  
    if (n->g != z) enfile(n->g);  
    if (n->d != z) enfile(n->d);  
} // z est le noeud factice
```

Parcours avec une file

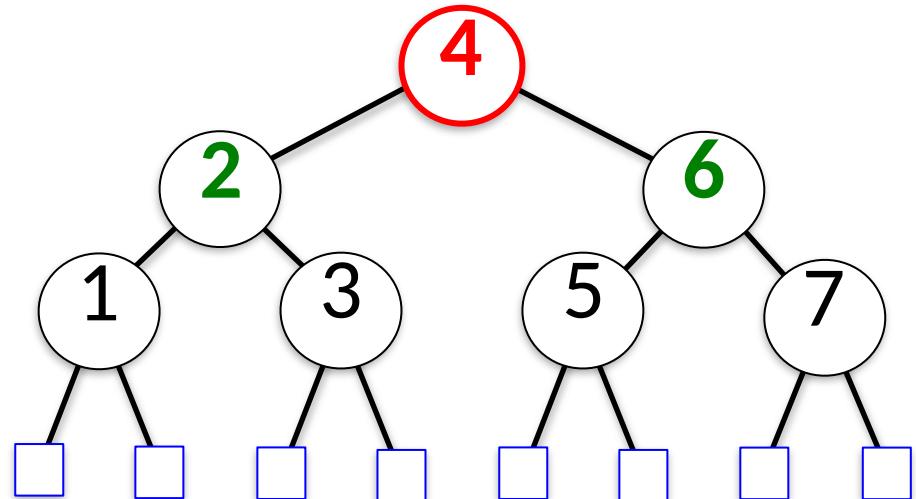
```
enfiler(racine);  
while (!estVide(file)) {  
    n = defiler();  
    afficher(n);  
    if (n->g != z)  
        enfiler(n->g);  
    if (n->d != z)  
        enfiler(n->d);  
}
```

4



Parcours avec une file

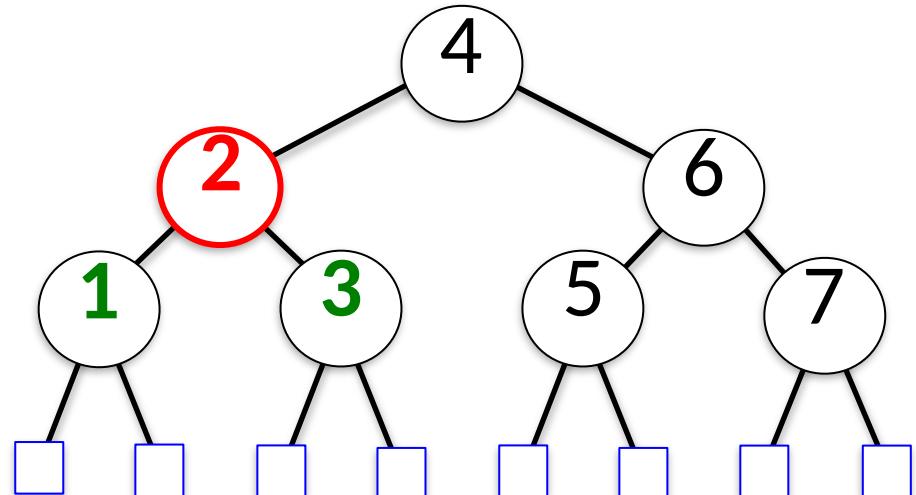
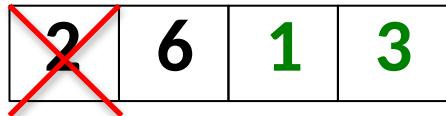
```
enfiler(n);  
while (!estVide(file)) {  
    n = defiler();  
    afficher(n);  
    if (n->g != z)  
        enfiler(n->g);  
    if (n->d != z)  
        enfiler(n->d);  
}
```



4

Parcours avec une file

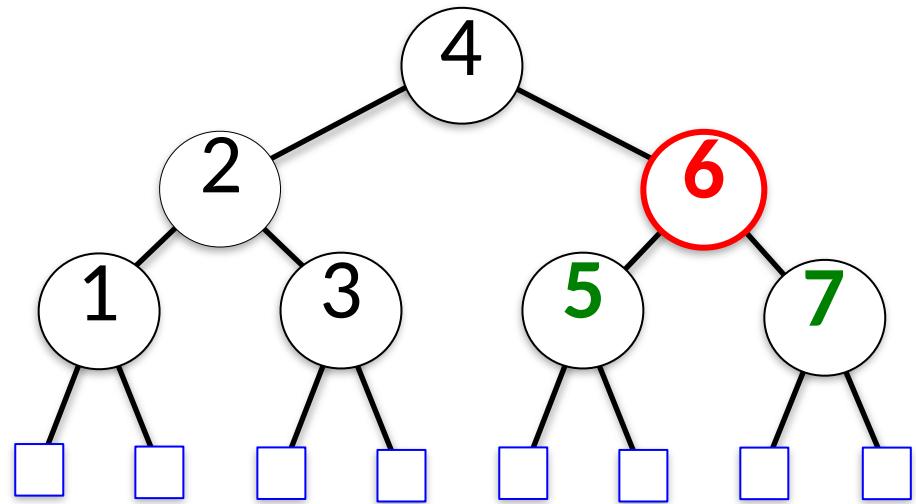
```
enfiler(n);  
while (!estVide(file)) {  
    n = defiler();  
    afficher(n);  
    if (n->g != z)  
        enfiler(n->g);  
    if (n->d != z)  
        enfiler(n->d);  
}
```



4 2

Parcours avec une file

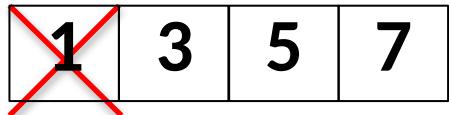
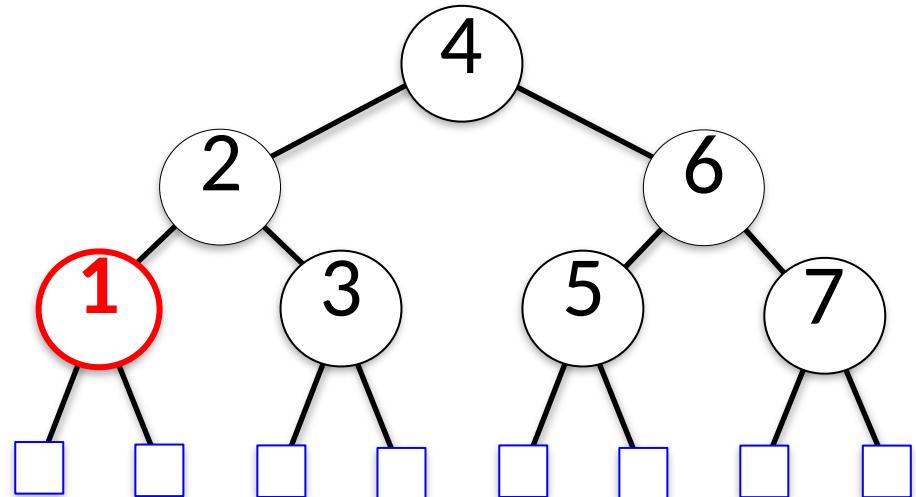
```
enfiler(n);  
while (!estVide(file)) {  
    n = defiler();  
    afficher(n);  
    if (n->g != z)  
        enfiler(n->g);  
    if (n->d != z)  
        enfiler(n->d);  
}
```



4 2 6

Parcours avec une file

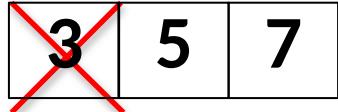
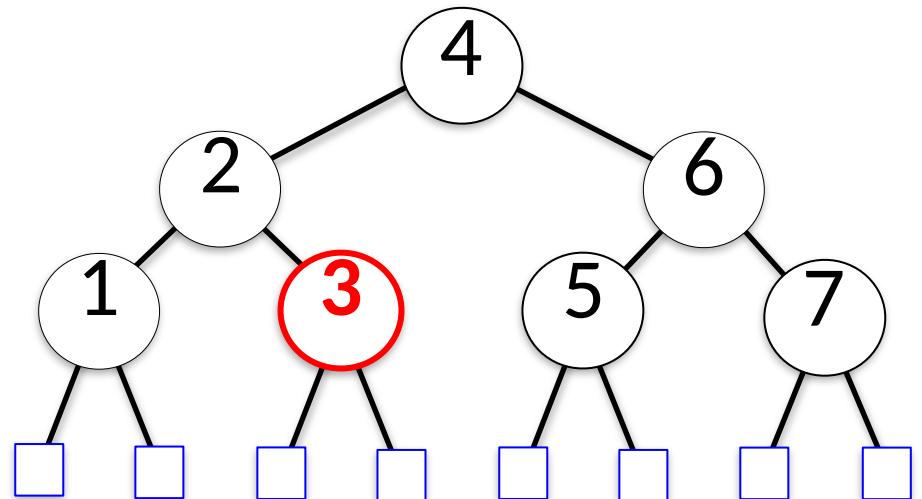
```
enfiler(n);
while (!estVide(file)) {
    n = defiler();
    afficher(n);
    if (n->g != z)
        enfiler(n->g);
    if (n->d != z)
        enfiler(n->d);
}
```



4261

Parcours avec une file

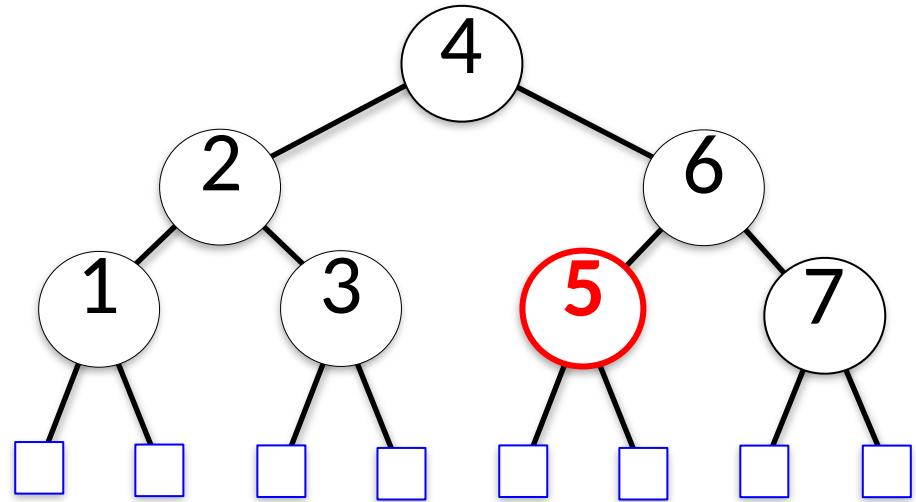
```
enfiler(n);  
while (!estVide(file)) {  
    n = defiler();  
    afficher(n);  
    if (n->g != z)  
        enfiler(n->g);  
    if (n->d != z)  
        enfiler(n->d);  
}
```



4 2 6 1 3

Parcours avec une file

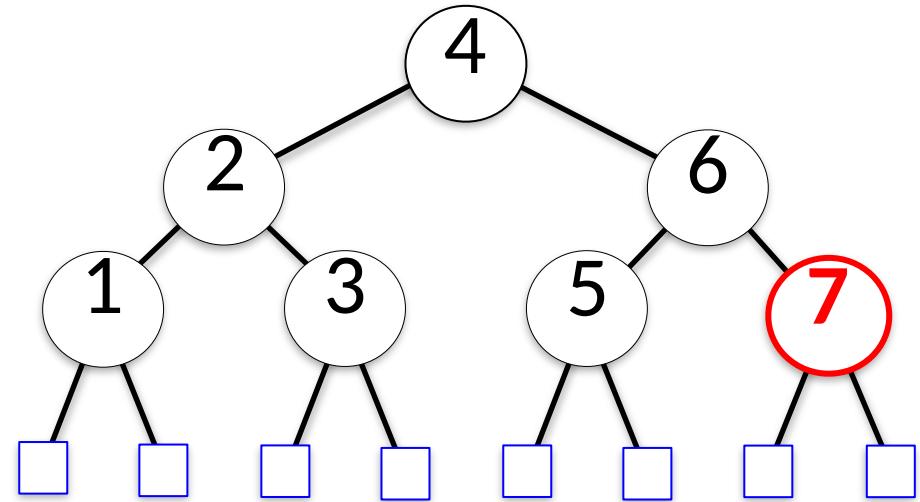
```
enfiler(n);
while (!estVide(file)) {
    n = defiler();
    afficher(n);
    if (n->g != z)
        enfiler(n->g);
    if (n->d != z)
        enfiler(n->d);
}
```



4 2 6 1 3 5

Parcours avec une file

```
enfiler(n);  
while (!estVide(file)) {  
    n = defiler();  
    afficher(n);  
    if (n->g != z)  
        enfiler(n->g);  
    if (n->d != z)  
        enfiler(n->d);  
}
```



4 2 6 1 3 5 7 => parcours en largeur

Parcours : méthodes itératives

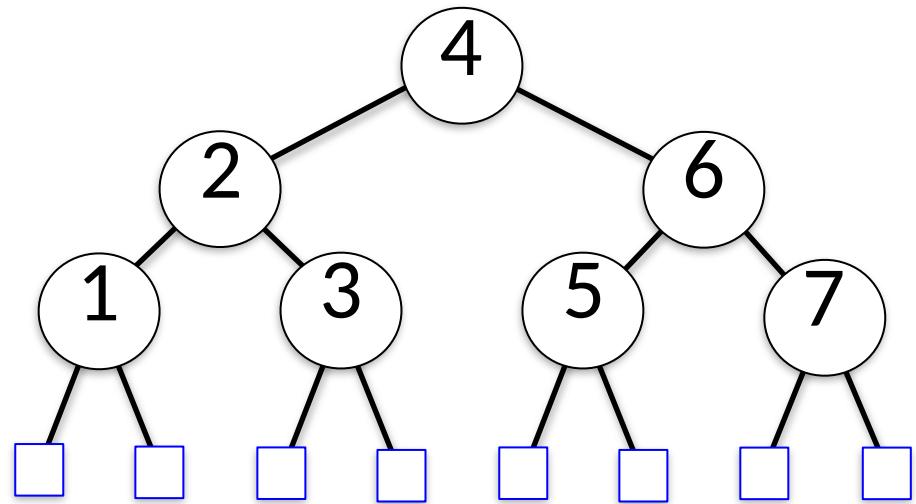
- Utilisation d'une pile :

```
empiler(n);  
while (!estVide(pile)) {  
    n = depiler();  
    afficher(n);  
    if (n->d != z) empiler(n->d);  
    if (n->g != z) empiler(n->g);  
}
```

=> Quel type de parcours ?

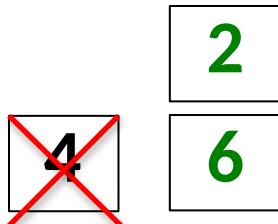
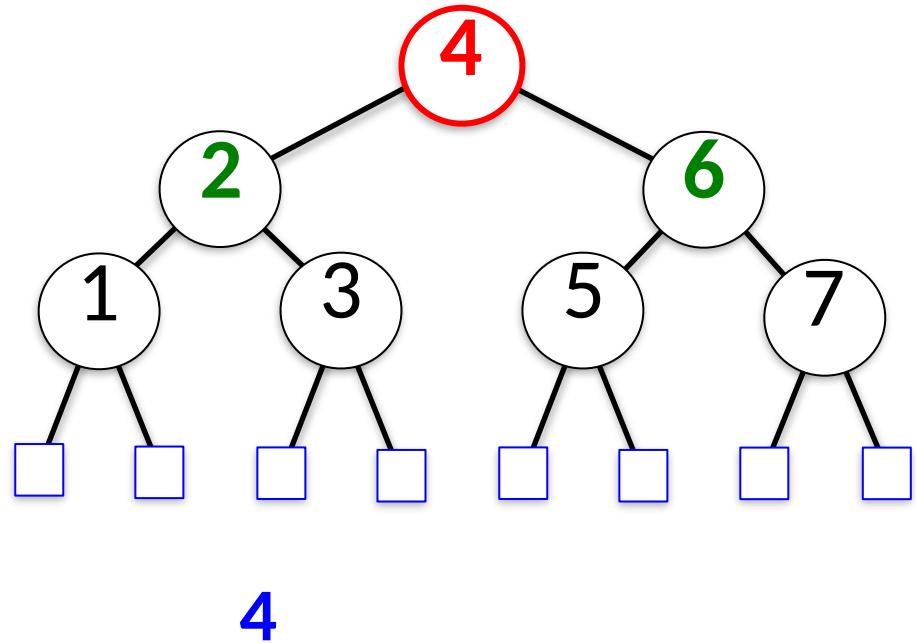
Parcours avec une pile

```
empiler(n);  
while (!estVide(pile)) {  
    n = depiler();  
    afficher(n);  
    if (n->d != z)  
        empiler(n->d);  
    if (n->g != z)  
        empiler(n->g);  
}
```



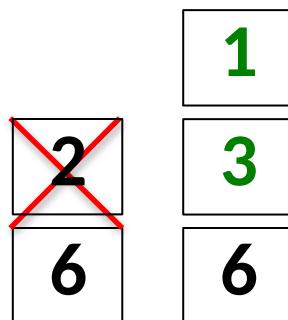
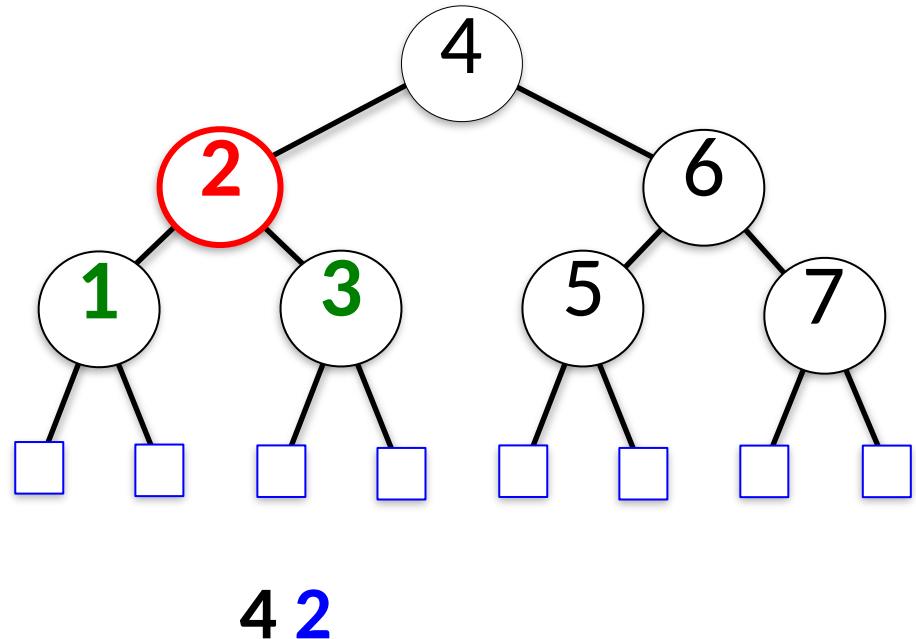
Parcours avec une pile

```
empiler(n);
while (!estVide(pile)) {
    n = depiler();
    afficher(n);
    if (n->d != z)
        empiler(n->d);
    if (n->g != z)
        empiler(n->g);
}
```



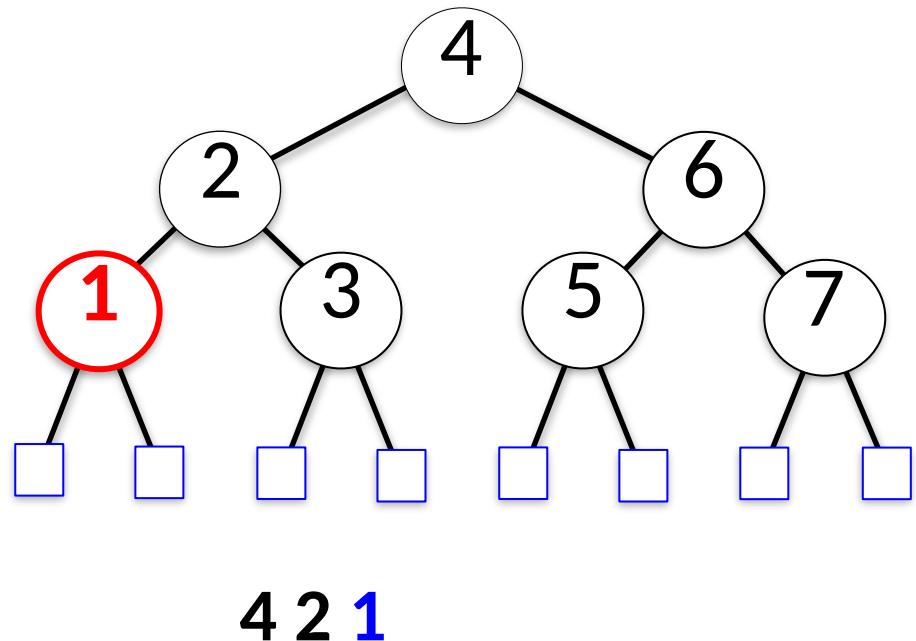
Parcours avec une pile

```
empiler(n);  
while (!estVide(pile)) {  
    n = depiler();  
    afficher(n);  
    if (n->d != z)  
        empiler(n->d);  
    if (n->g != z)  
        empiler(n->g);  
}
```



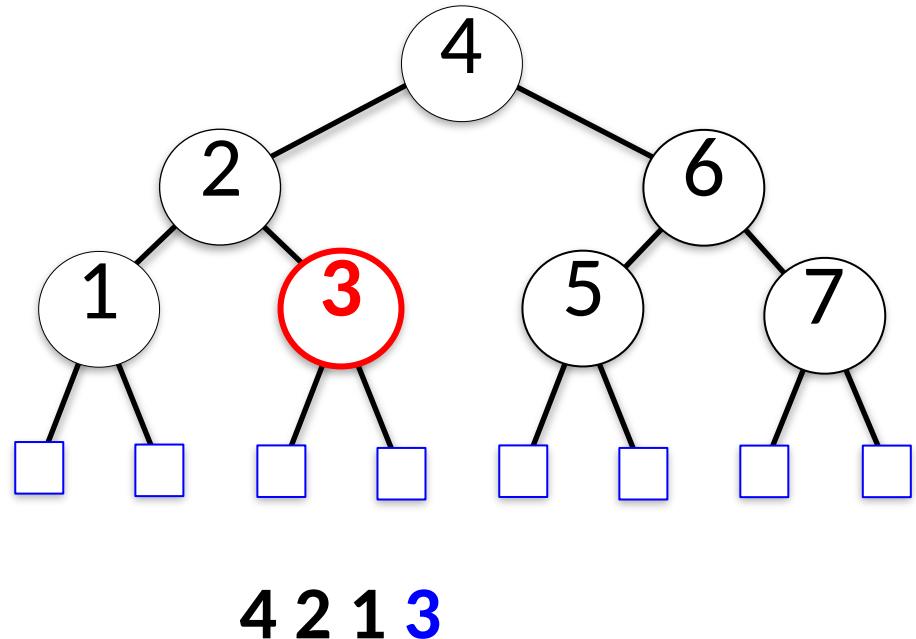
Parcours avec une pile

```
empiler(n);
while (!estVide(pile)) {
    n = depiler();
    afficher(n);
    if (n->d != z)
        empiler(n->d);
    if (n->g != z)
        empiler(n->g);
}
```



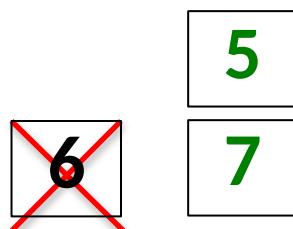
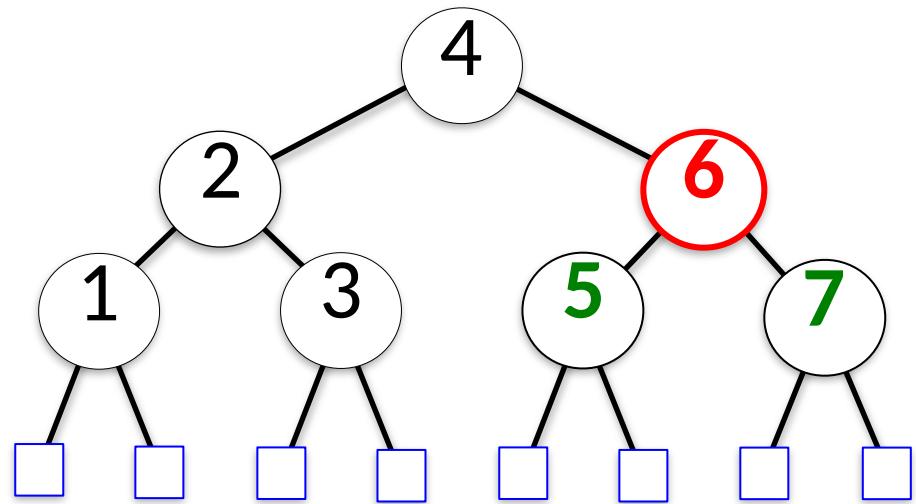
Parcours avec une pile

```
empiler(n);
while (!estVide(pile)) {
    n = depiler();
    afficher(n);
    if (n->d != z)
        empiler(n->d);
    if (n->g != z)
        empiler(n->g);
}
```



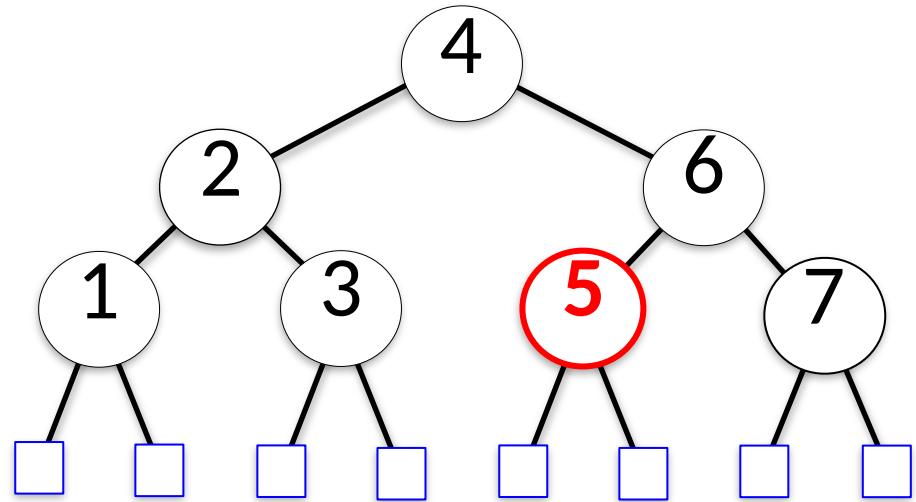
Parcours avec une pile

```
empiler(n);
while (!estVide(pile)) {
    n = depiler();
    afficher(n);
    if (n->d != z)
        empiler(n->d);
    if (n->g != z)
        empiler(n->g);
}
```



Parcours avec une pile

```
empiler(n);
while (!estVide(pile)) {
    n = depiler();
    afficher(n);
    if (n->d != z)
        empiler(n->d);
    if (n->g != z)
        empiler(n->g);
}
```

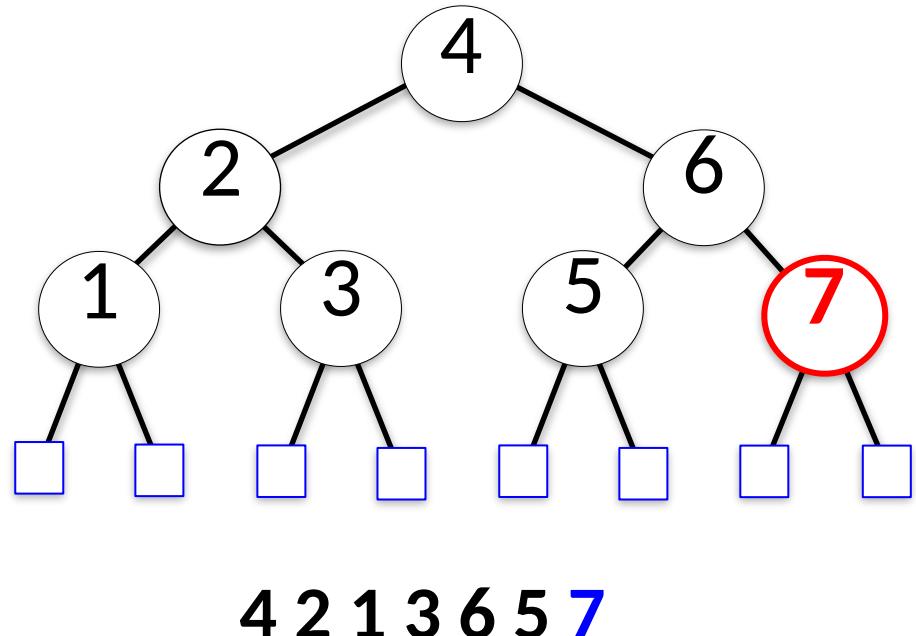


4 2 1 3 6 5



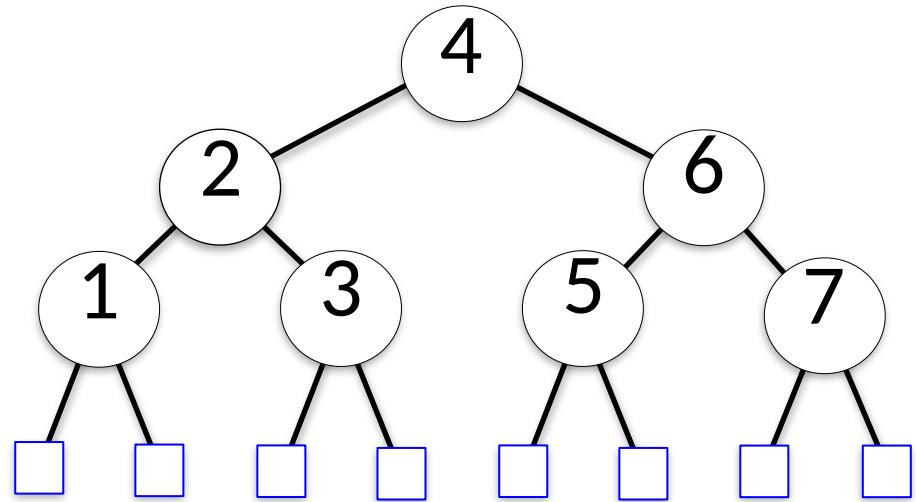
Parcours avec une pile

```
empiler(n);
while (!estVide(pile)) {
    n = depiler();
    afficher(n);
    if (n->d != z)
        empiler(n->d);
    if (n->g != z)
        empiler(n->g);
}
```



Parcours avec une pile

```
empiler(n);
while (!estVide(pile)) {
    n = depiler();
    afficher(n);
    if (n->d != z)
        empiler(n->d);
    if (n->g != z)
        empiler(n->g);
}
```



4 2 1 3 6 5 7

=> Parcours en profondeur, préfixé

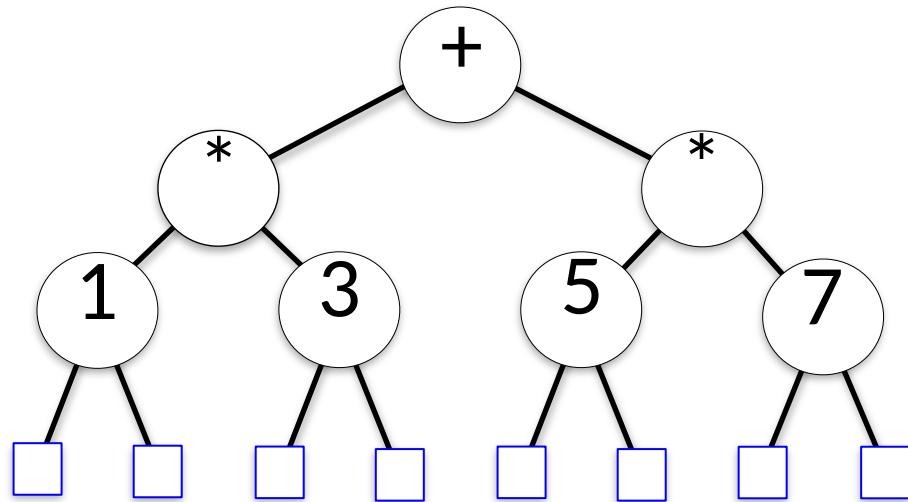
Arbre de syntaxe abstraite

- Abstract Syntax Tree en anglais (AST)

Arbre de syntaxe abstraite (AST)

Noeuds interne = opérateurs

Feuilles = opérande



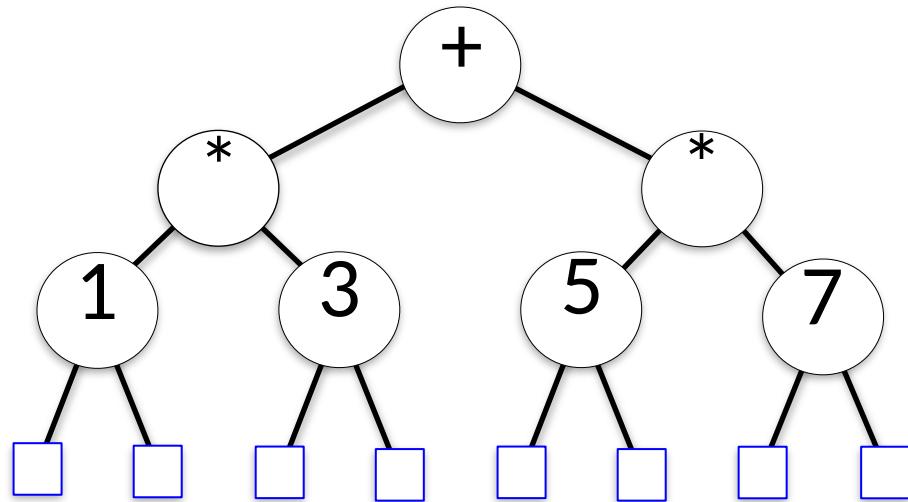
Type de parcours pour obtenir la notation standard :

$1 * 3 + 5 * 7 ?$

Arbre de syntaxe abstraite (AST)

Noeuds interne = opérateurs

Feuilles = opérande



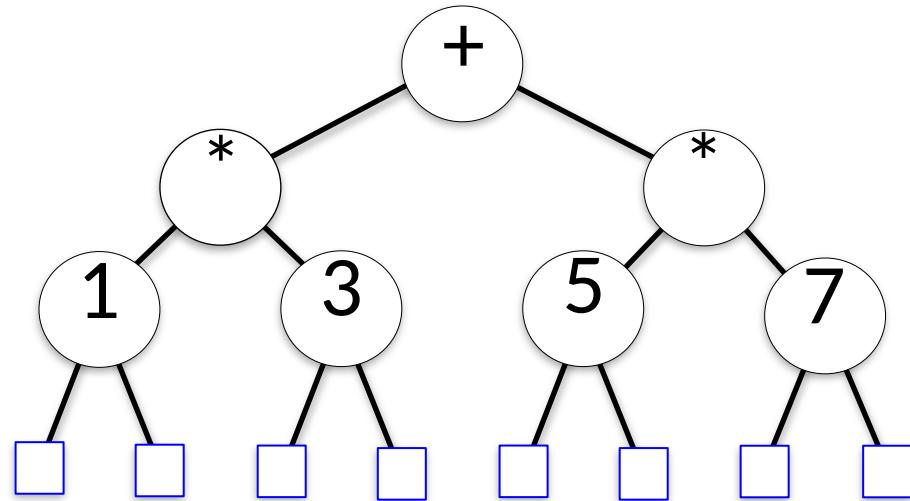
Type de parcours pour obtenir la notation standard :

$1 * 3 + 5 * 7 \Rightarrow \text{infixé}$

Arbre de syntaxe abstraite (AST)

Noeuds interne = opérateurs

Feuilles = opérande

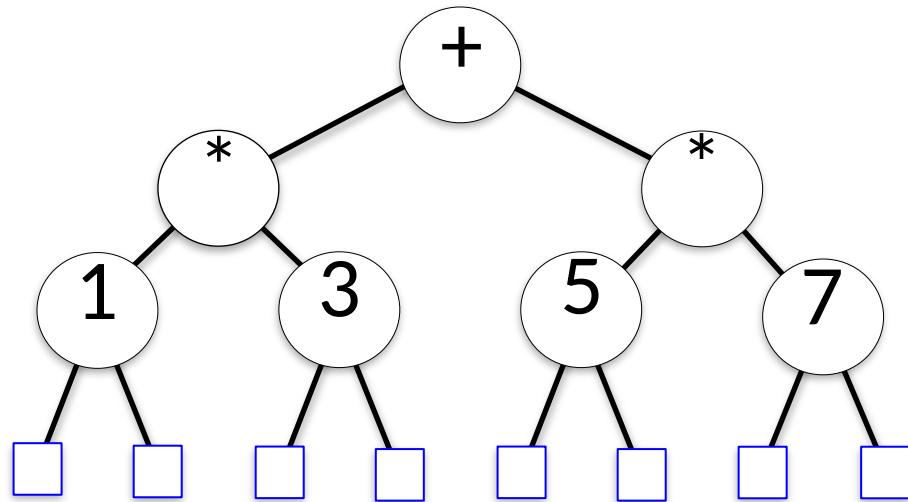


Type de parcours pour obtenir la notation polonaise
inversée : 13*57*+ ?

Arbre de syntaxe abstraite (AST)

Noeuds interne = opérateurs

Feuilles = opérande



Type de parcours pour obtenir la notation polonaise inversée : $13*57*+ \Rightarrow \text{postfixé}$

Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en
les prenant dans la pile puis le mettre dans
la pile.

Le dernier noeud restant dans la pile sera
la racine de l'arbre.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 4 | 5 | * | * | 6 | 7 | * | + | * |
|---|---|---|---|---|---|---|---|---|---|---|

Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

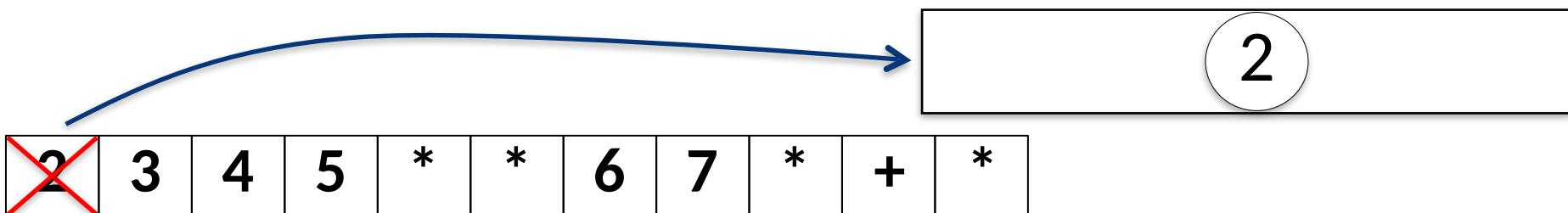
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

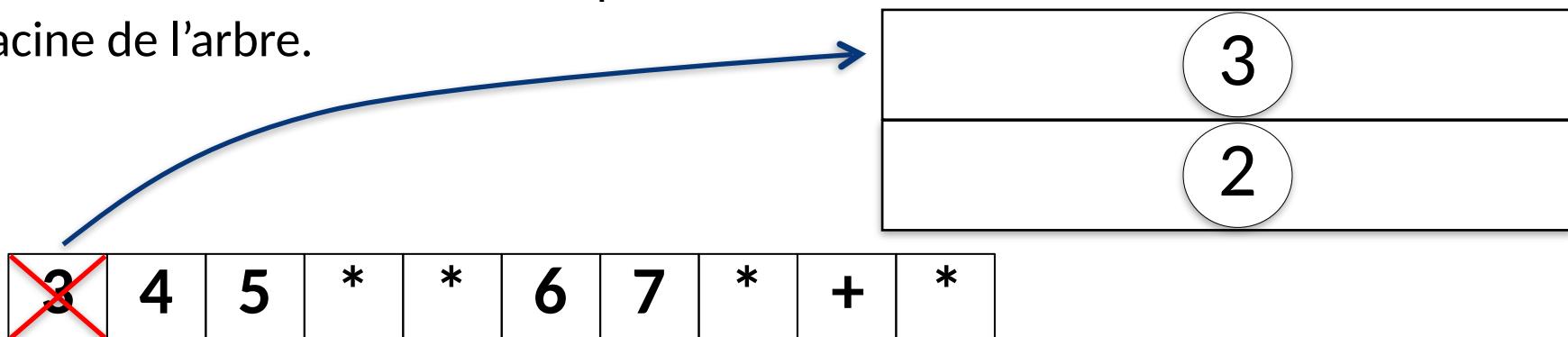
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

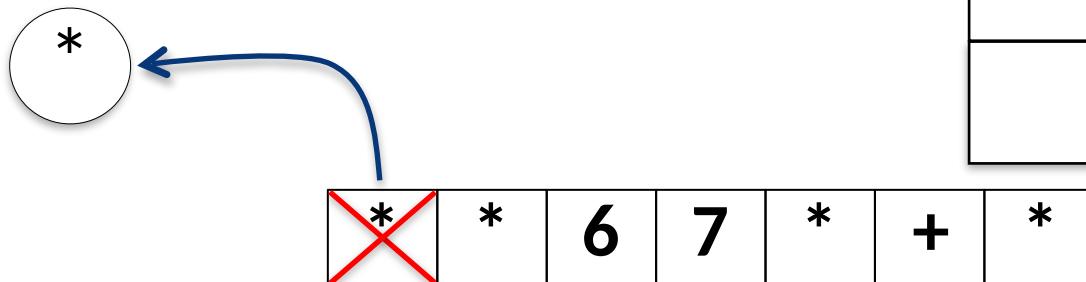
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

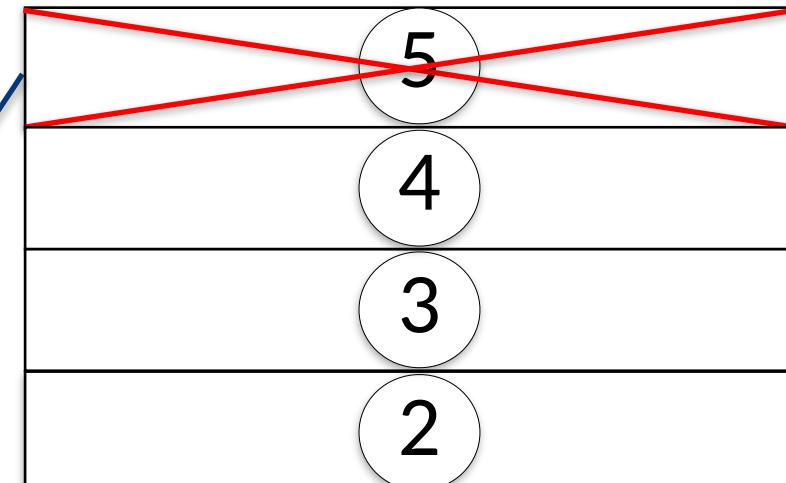
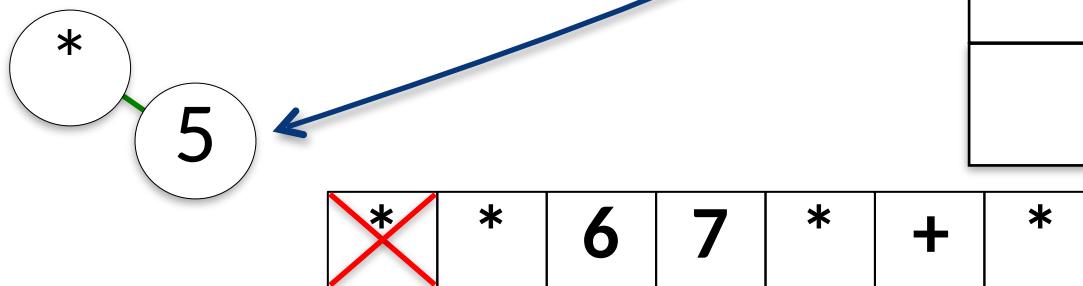
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

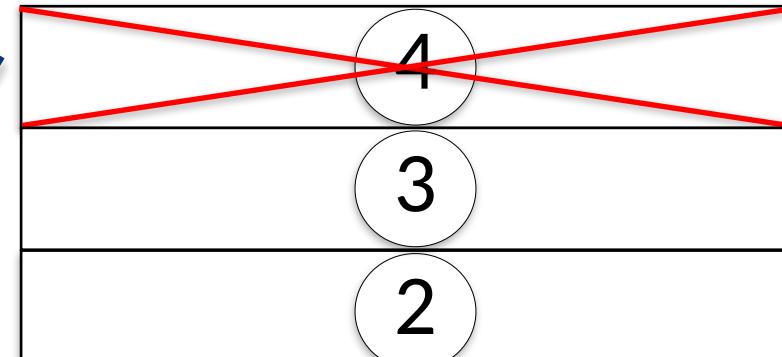
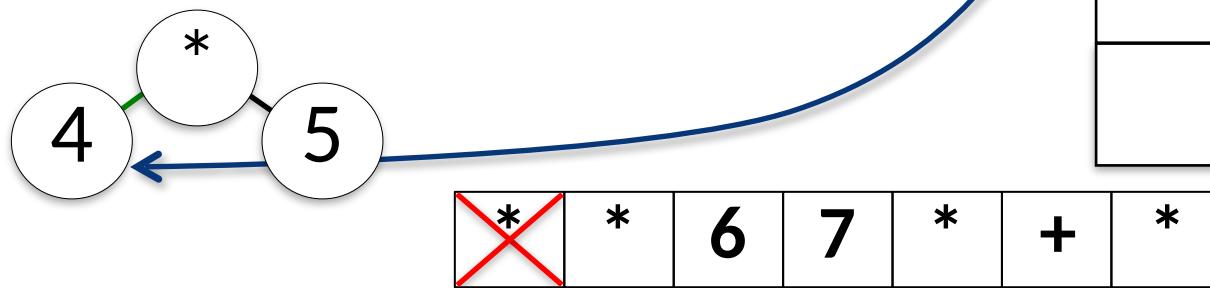
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

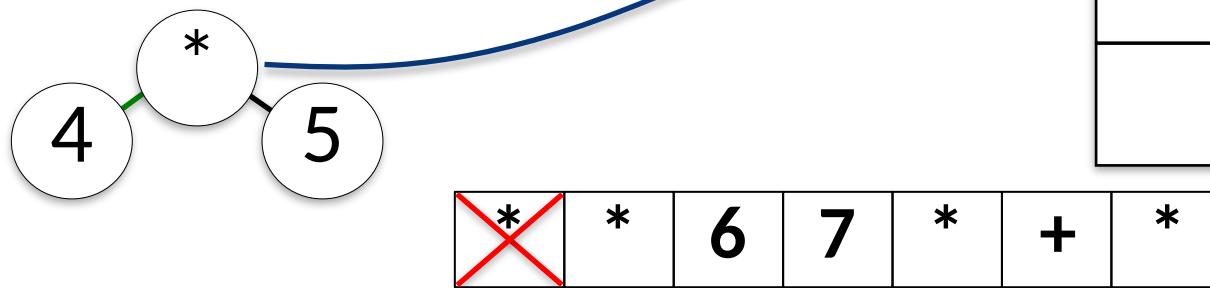
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile **puis le mettre dans la pile.**

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

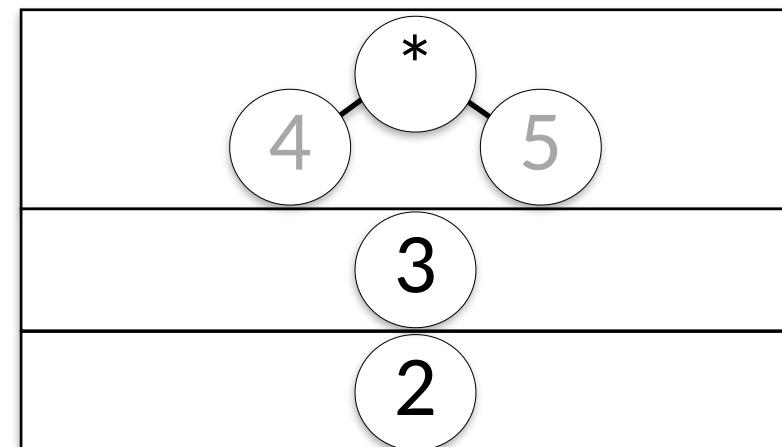
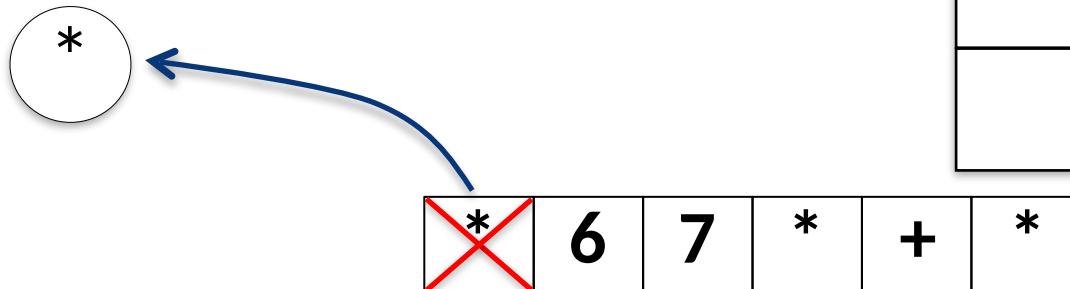
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

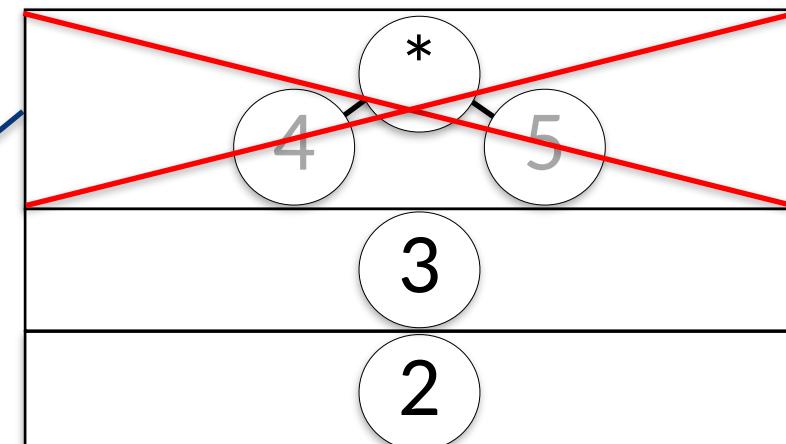
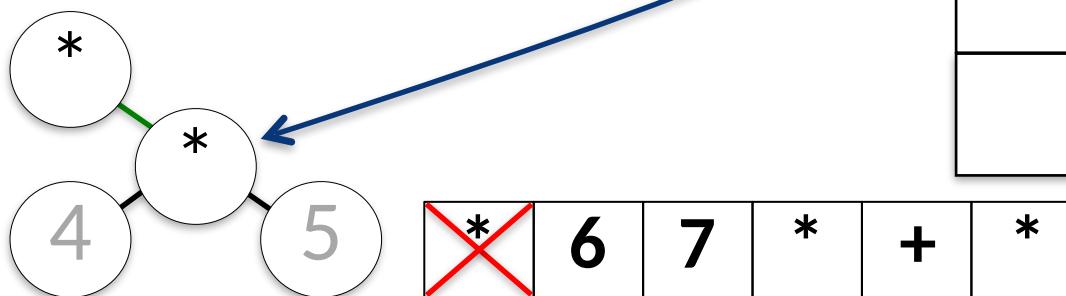
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

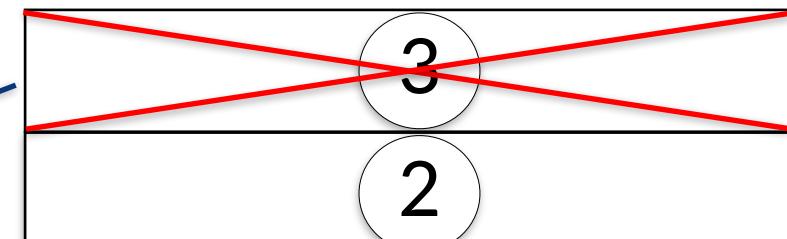
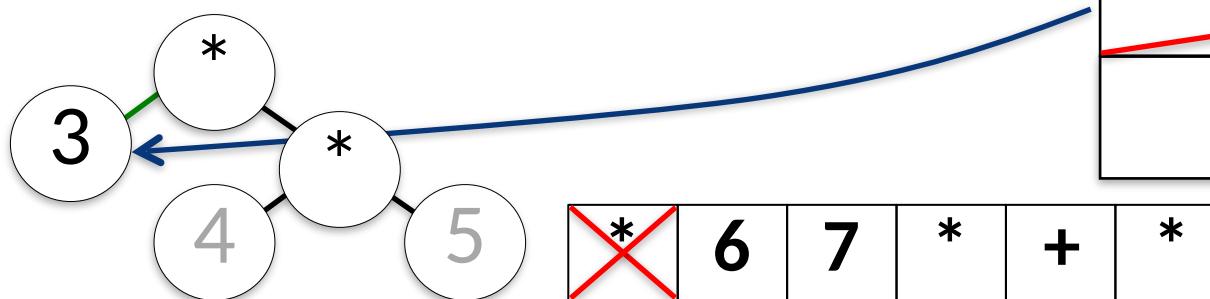
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

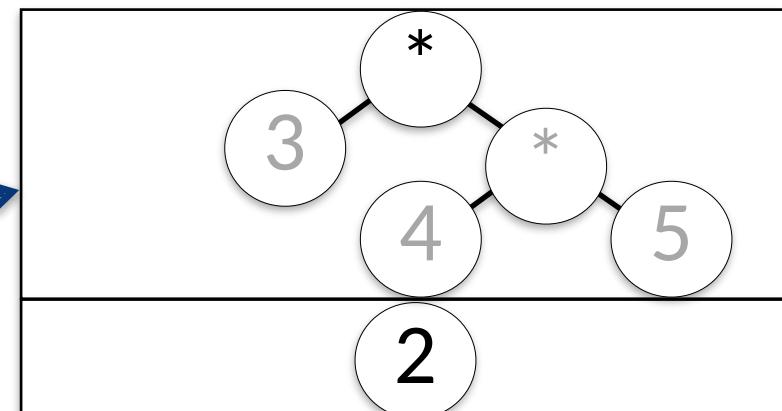
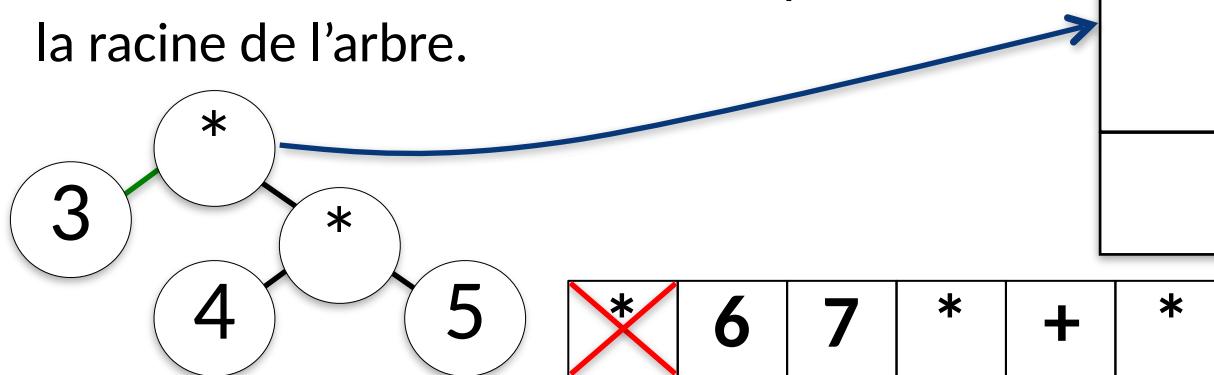
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile **puis le mettre dans la pile.**

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

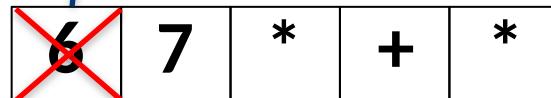
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

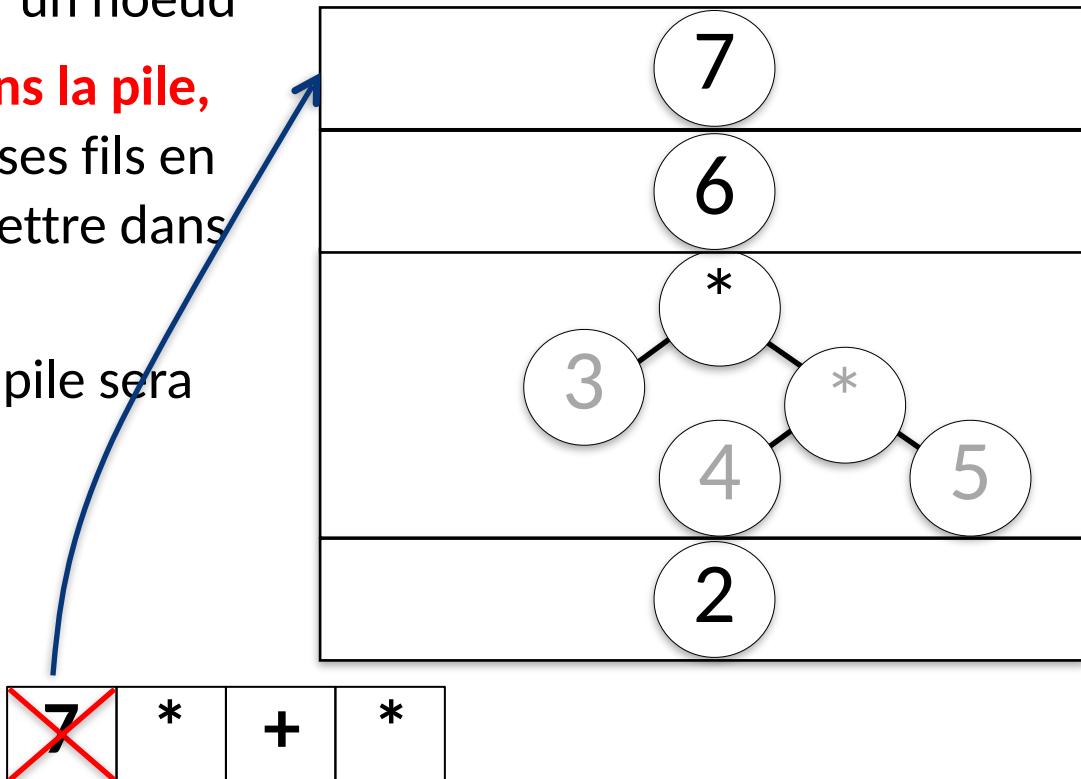
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

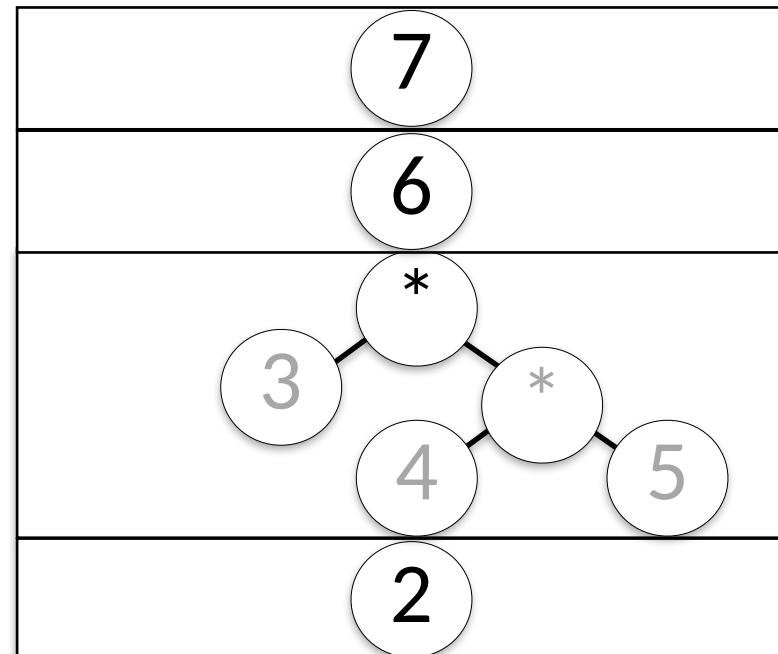
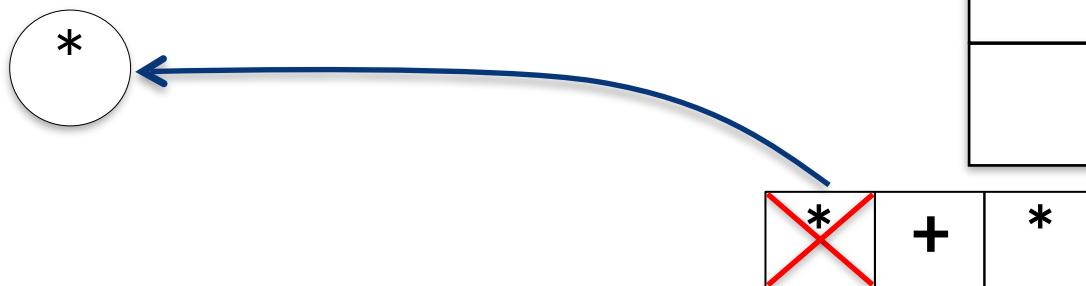
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

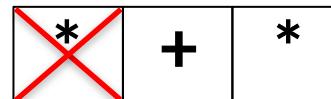
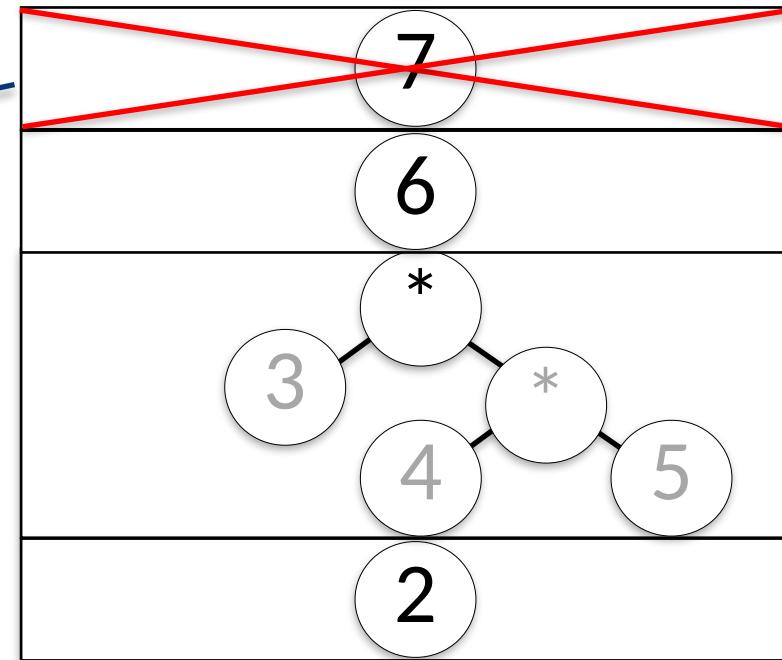
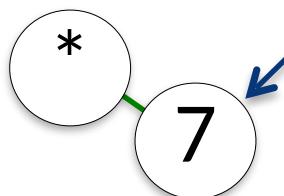
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

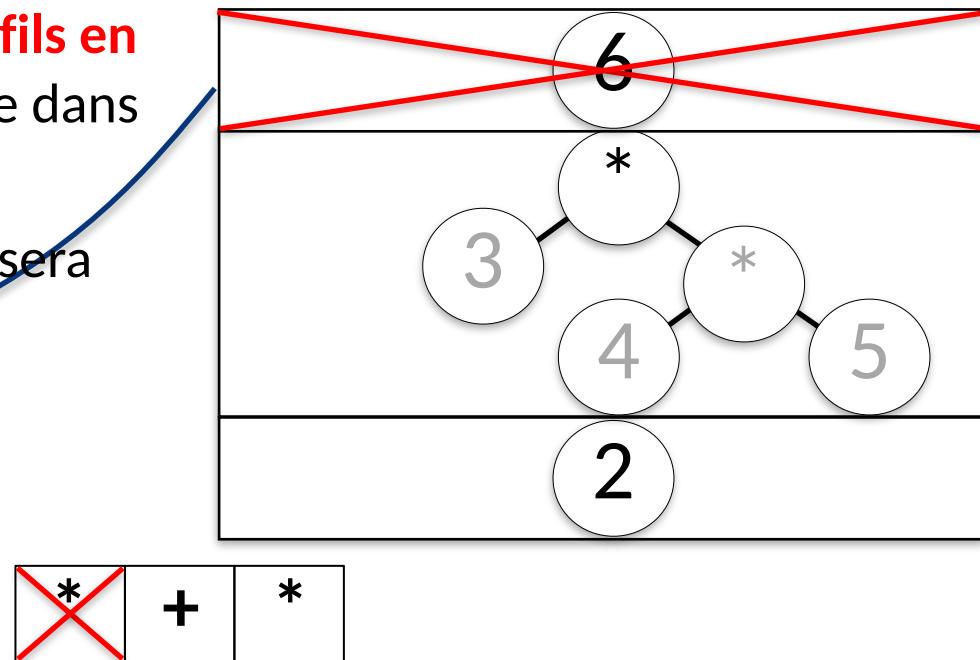
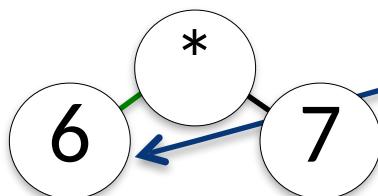
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

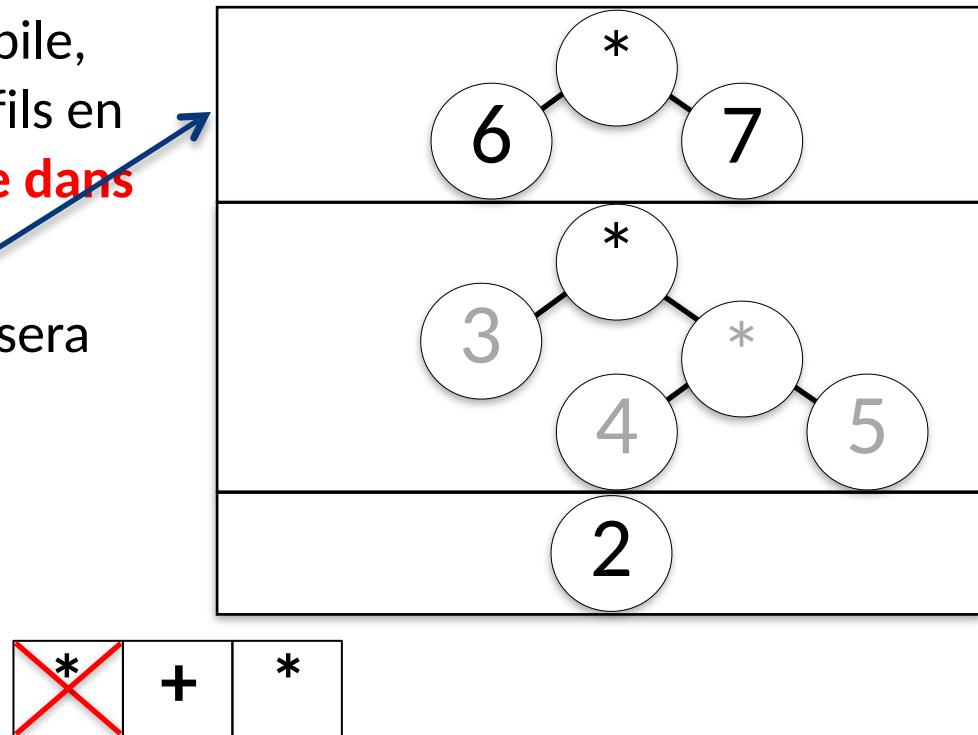
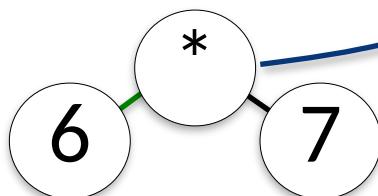
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile **puis le mettre dans la pile.**

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

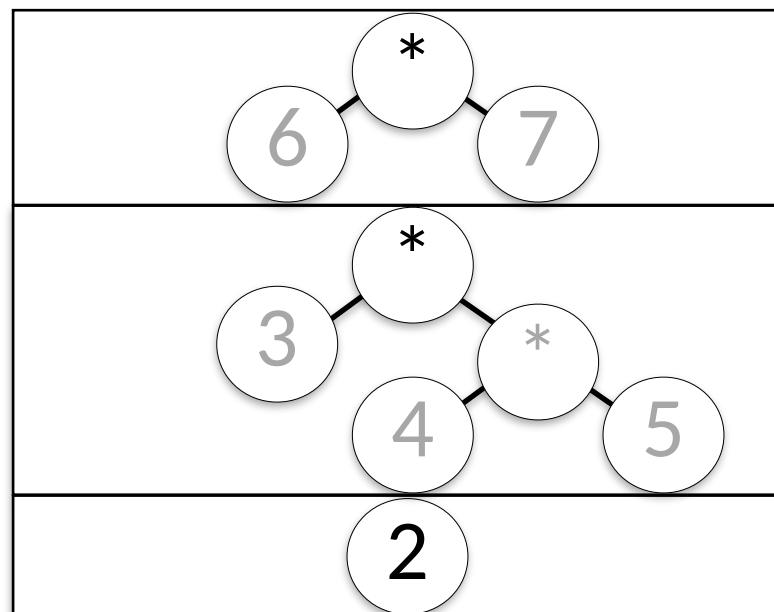
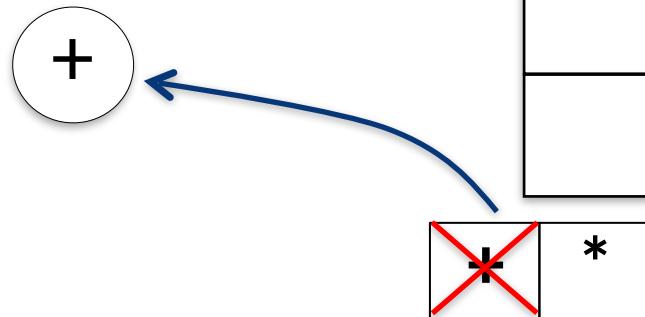
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

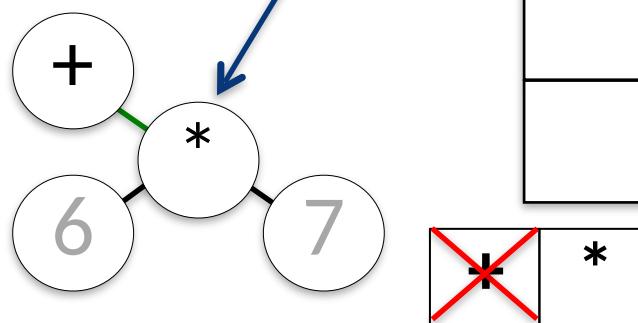
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

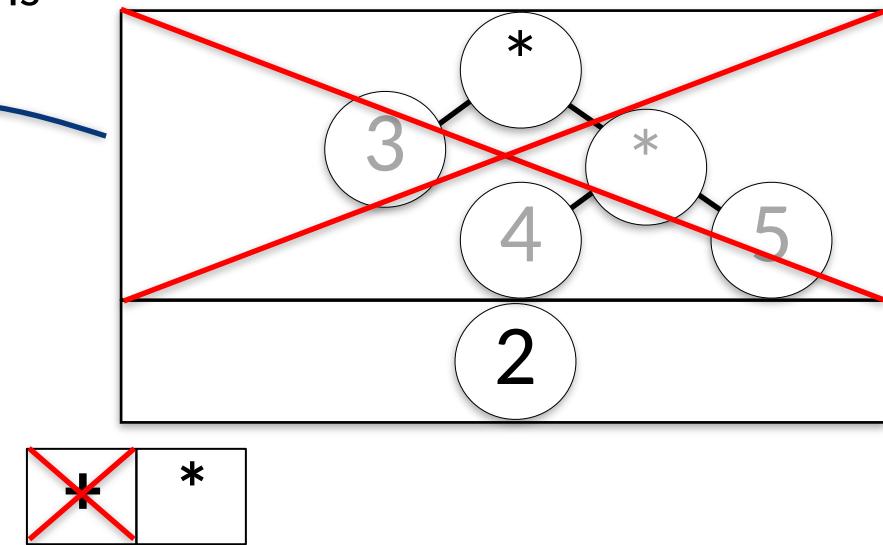
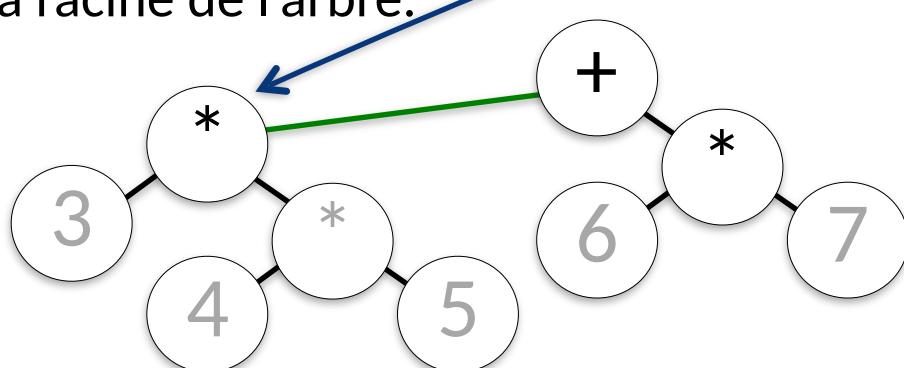
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

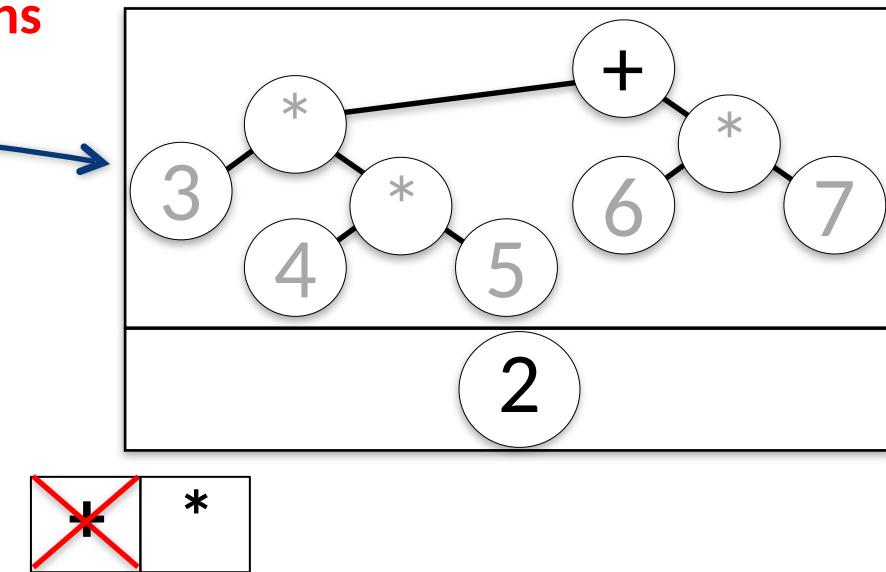
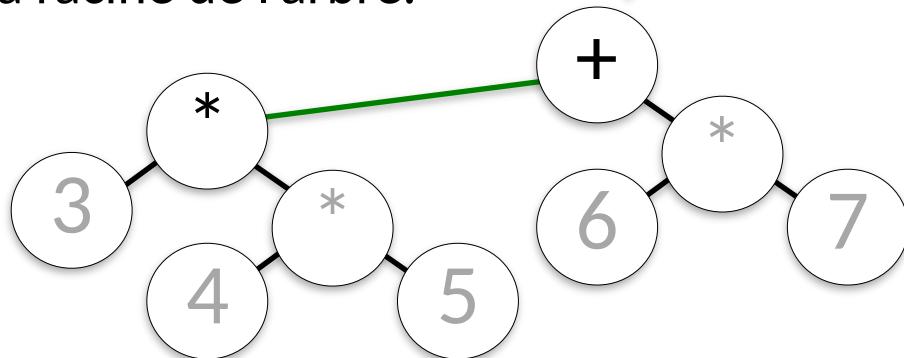
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile **puis le mettre dans la pile.**

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

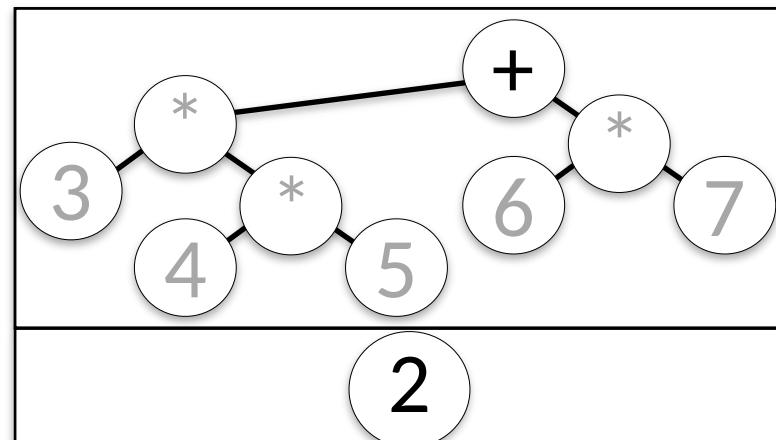
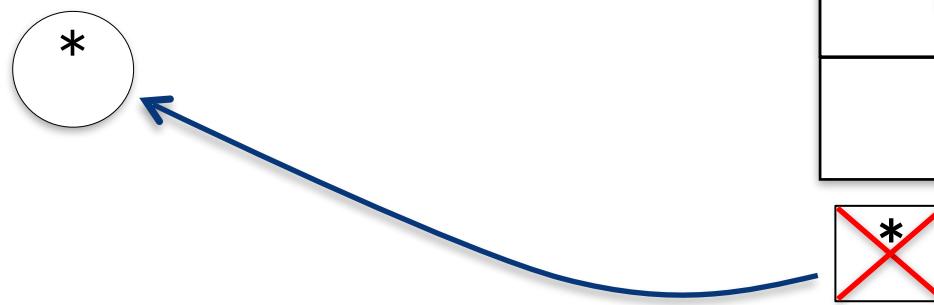
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

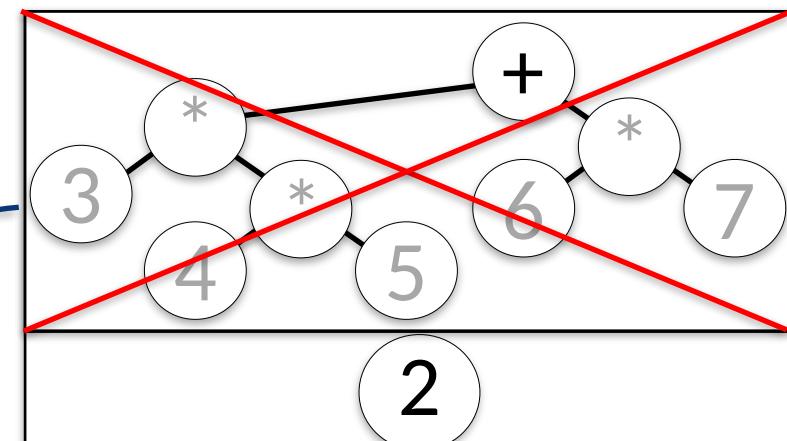
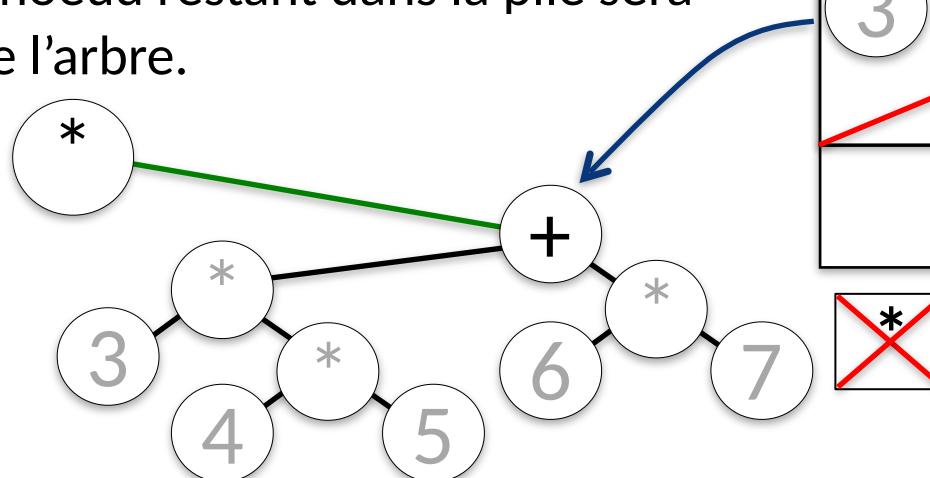
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

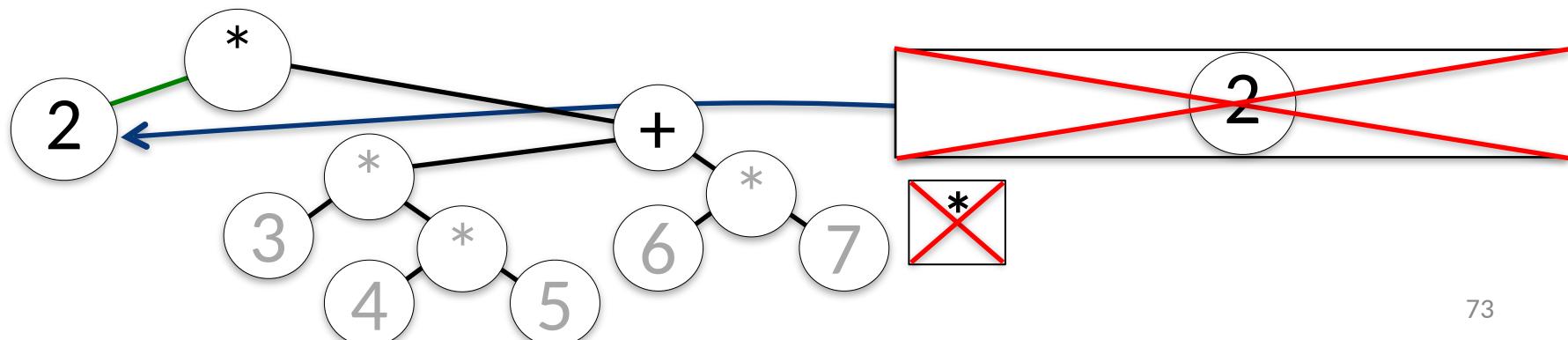
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

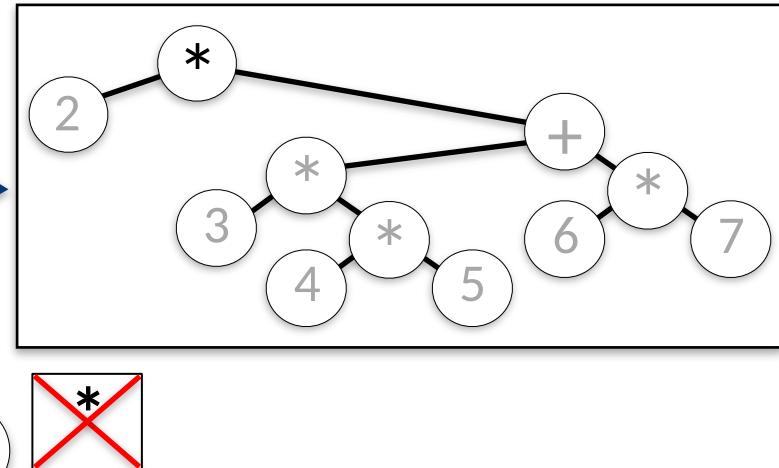
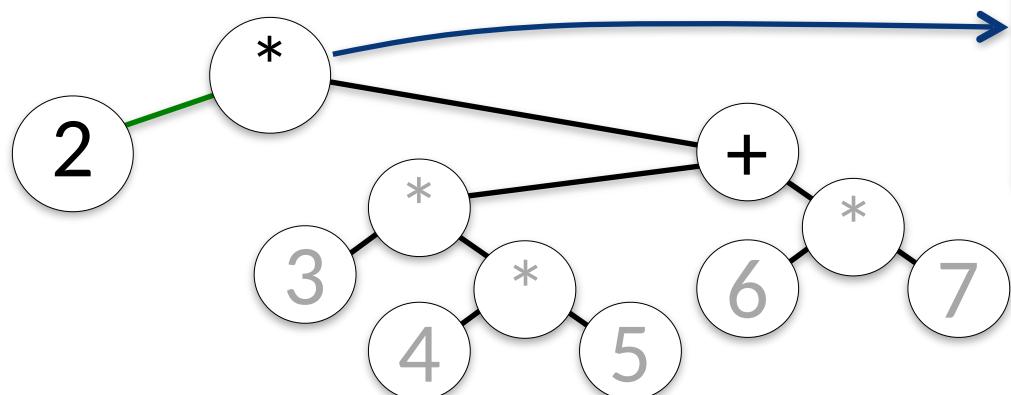
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile **puis le mettre dans la pile.**

Le dernier noeud restant dans la pile sera la racine de l'arbre.



Arbre de syntaxe abstraite (AST)

Génération de l'AST à partir de la NPI sous forme de file

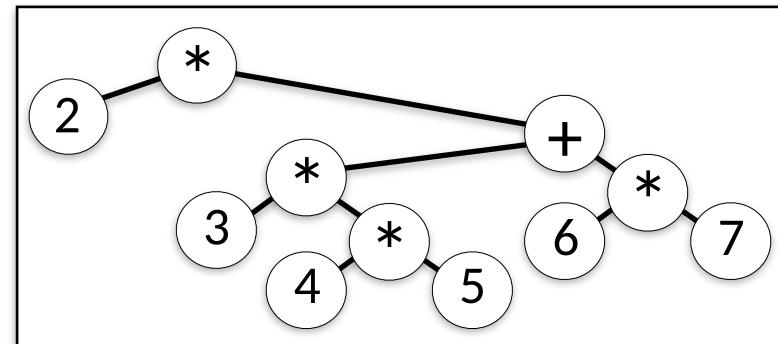
Parcours de la file :

Pour chaque valeur défilée, créer un noeud

Si c'est un nombre, le mettre dans la pile,

Si c'est un opérateur, lui affecter ses fils en les prenant dans la pile puis le mettre dans la pile.

Le dernier noeud restant dans la pile sera la racine de l'arbre.



TAS

- Heap en anglais

TAS

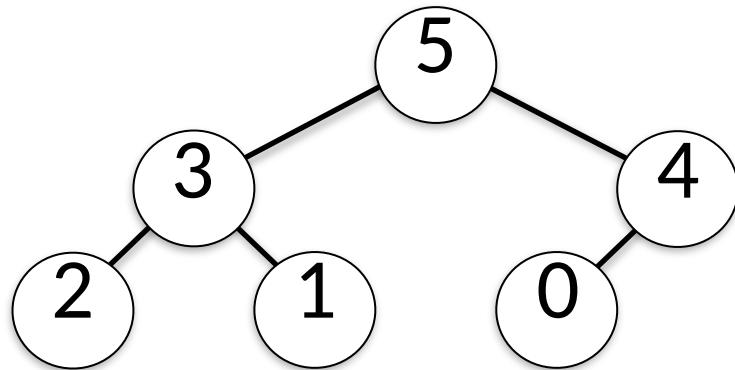
- Un tas est un arbre complet dans lequel il y a une relation d'ordre entre un nœud et ses fils
 - Tas maximier : la clé d'un nœud est supérieure ou égale à celle de ses fils
 - Tas minimier : la clé d'un nœud est inférieure ou égale à celle de ses fils.

⇒Implantation d'une **file de priorité**

⇒Tri par tas

TAS

- Implantation par pointeurs sur fils gauche/droit.
- Implantation par tableau (cf. arbre complet).



Sentinelle

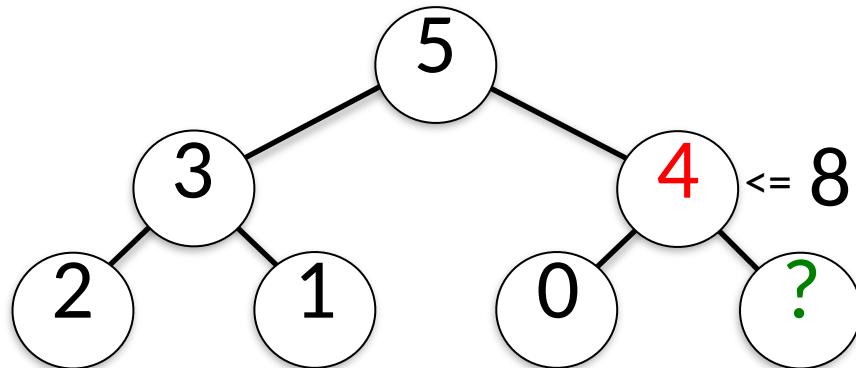
Sommet

| | | | | | | | | |
|---|---|---|---|---|---|---|---|--|
| | | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
| x | 5 | 3 | 4 | 2 | 1 | 0 | | |

2 * 1 2 * 1 + 1 2 * 2 2 * 2 + 1 2 * 3

TAS maximier - Insertion

- Insertion en fin de tableau puis tamisage : on fait remonter la valeur insérée à sa place.
- Exemple : insertion de 8 ?

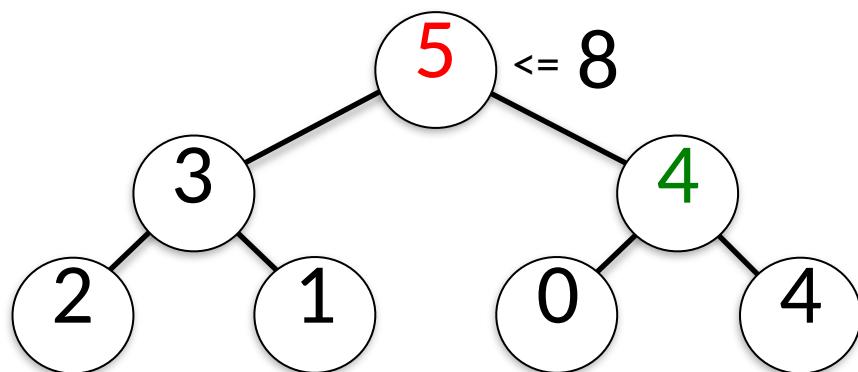


| $p(i/2)$ | i |
|----------|-----|
| 0 | 1 |
| 8 | 5 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 2 | 3 | 4 | 5 | 6 | 7 | |
| 8 | 5 | 3 | 4 | 2 | 1 | 0 |

TAS maximier - Insertion

- Insertion en fin de tableau puis tamisage : on fait remonter la valeur insérée à sa place.
- Exemple : insertion de 8 ?

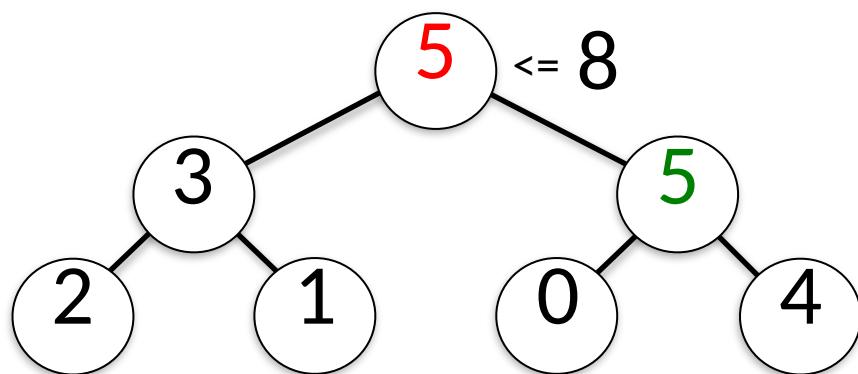


| | p | $(i/2)$ | i |
|---|-----|---------|-----|
| 0 | 1 | 2 | 3 |
| 8 | 5 | 3 | 4 |

A blue curved arrow points from the value 4 in the second row to the index 4 in the first row, indicating the insertion position.

TAS maximier - Insertion

- Insertion en fin de tableau puis tamisage : on fait remonter la valeur insérée à sa place.
- Exemple : insertion de 8 ?



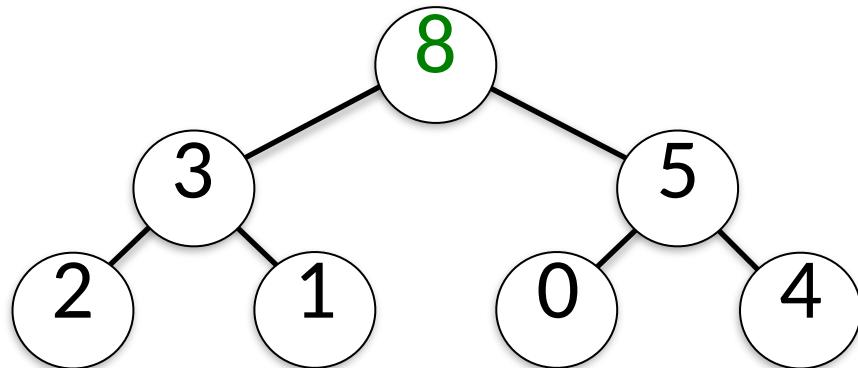
$p(i/2)$ i

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 5 | 3 | 5 | 2 | 1 | 0 | 4 |

A blue arrow points from the value 8 in the second row to the index 1 above it, indicating the insertion point.

TAS maximier - Insertion

- Insertion en fin de tableau puis tamisage : on fait remonter la valeur insérée à sa place.
- Exemple : insertion de 8 ?



| p | i | | | | | | | |
|-----|-----|---|---|---|---|---|---|--|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
| 8 | 8 | 3 | 5 | 2 | 1 | 0 | 4 | |

TAS maximier - Insertion

Mettre la valeur à ajouter dans la sentinelle

Incrémenter la taille du tas

Utiliser un pointeur n partant du nouveau noeud ajouté

Tant que val est supérieur à la valeur du parent de n

 Descendre la valeur du parent de n dans n

 Faire pointer n sur son propre parent

 // inutile de mettre val dans n ici car elle sera
 écrasée à la prochaine itération : on le donc fera
 uniquement à la fin

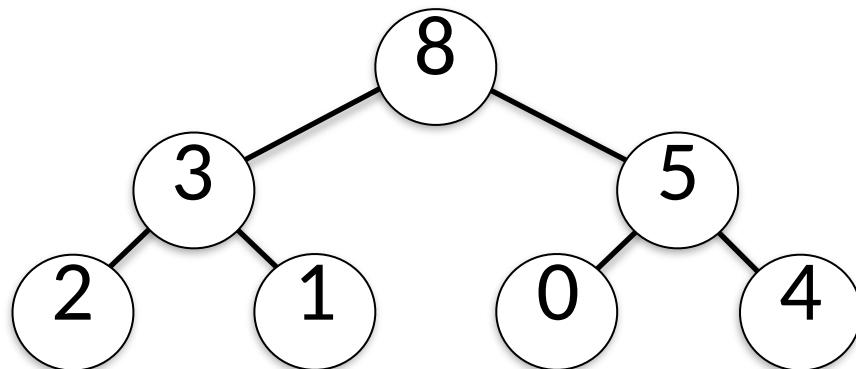
Fin tantque

Mettre val dans n

 // car la boucle s'arrête quand le parent n contient une
 valeur supérieure ou égale à val)

TAS - Suppression

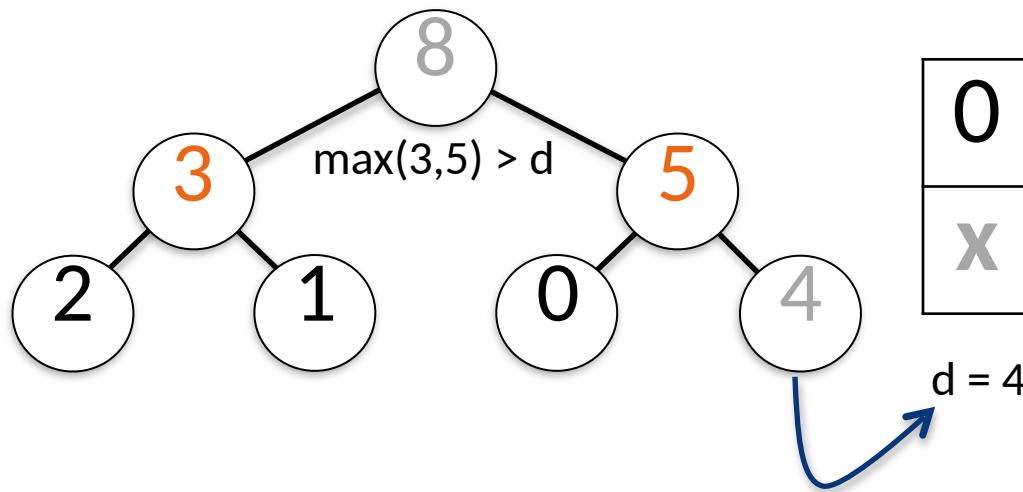
- On remplace l'élément à supprimer par la valeur du dernier élément du tableau et on la fait descendre.
- Suppression de 8 ? Suppression du dernier noeud (4) et placement de 4 en partant du haut : remplacement du noeud vide par le max de ses fils ou 4.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| x | 8 | 3 | 5 | 2 | 1 | 0 | 4 |

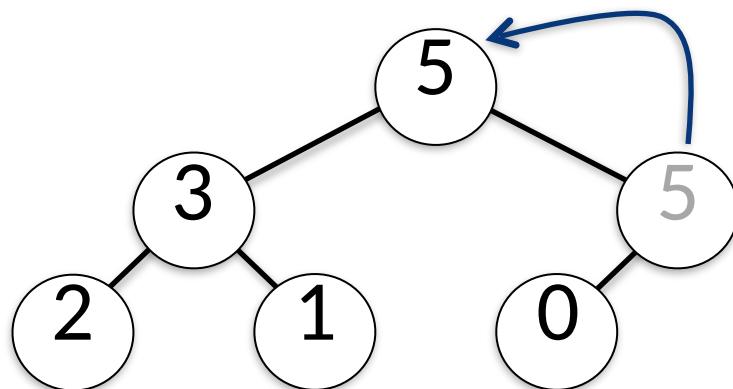
TAS - Suppression

- On remplace la racine par la valeur du dernier élément du tableau et on la fait descendre.
- Suppression de 8 ? Suppression du dernier noeud (4) et placement de 4 en partant du haut : remplacement du noeud vide par le max de ses fils ou 4.



TAS - Suppression

- On remplace la racine par la valeur du dernier élément du tableau et on la fait descendre.
- Suppression de 8 ? Suppression du dernier noeud (4) et placement de 4 en partant du haut : remplacement du noeud vide par le max de ses fils ou 4.

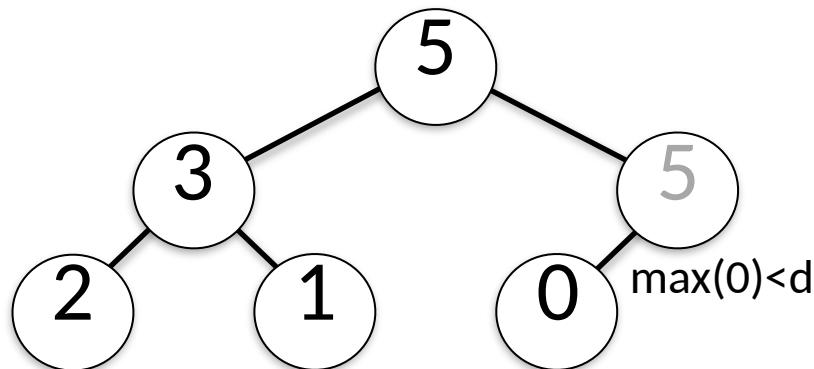


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| x | 5 | 3 | 5 | 2 | 1 | 0 | 4 |

$$d = 4$$

TAS - Suppression

- On remplace la racine par la valeur du dernier élément du tableau et on la fait descendre.
- Suppression de 8 ? Suppression du dernier noeud (4) et placement de 4 en partant du haut : remplacement du noeud vide par le max de ses fils ou 4.

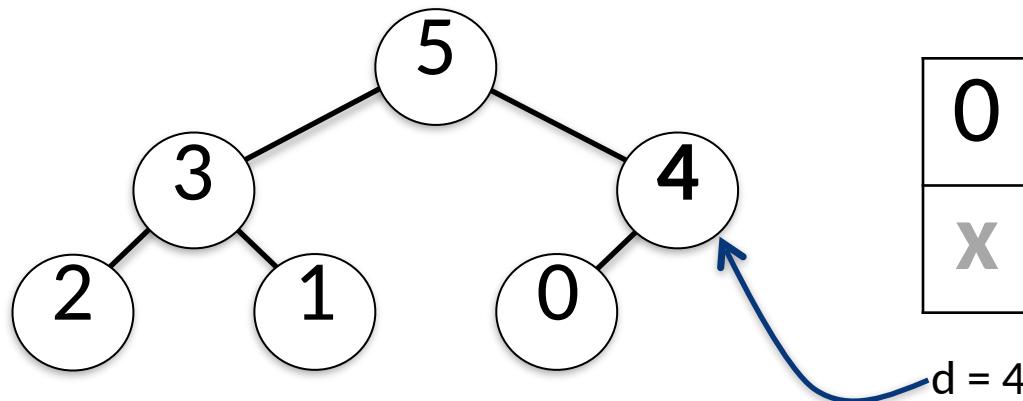


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| X | 8 | 3 | 5 | 2 | 1 | 0 | 4 |

$$d = 4$$

TAS - Suppression

- On remplace la racine par la valeur du dernier élément du tableau et on la fait descendre.
- Suppression de 8 ? Suppression du dernier noeud (4) et placement de 4 en partant du haut : remplacement du noeud vide par le max de ses fils ou 4.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| X | 8 | 3 | 5 | 2 | 1 | 0 | 4 |

TAS - Suppression

Mettre la valeur du sommet de côté pour le renvoyer à la fin

Utiliser un pointeur n partant de la racine

Tant que n a au moins un fils

Si le plus grand fils de n contient une valeur supérieur à celle du dernier noeud (dont on cherche la place dans le tas)

Remonter ce plus grand fils dans n

Faire pointer n sur ce plus grand fils

}

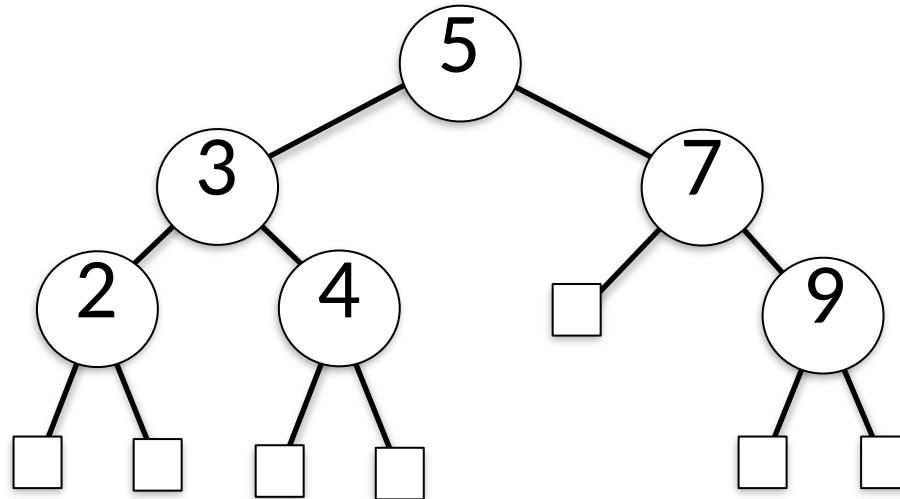
Décrémenter la taille du tas

Arbre Binaire de Recherche (ABR)

- Binary Search Tree en anglais (BST)

Arbre Binaire de Recherche (ABR)

- Arbre binaire : degré des nœuds ≤ 2
- Arbre ordonné : pour tout nœud de l'arbre
 - Valeur fils gauche < valeur
 - Valeur fils droit \geq valeur



ABR – Initialisation

```
abr *t = malloc(sizeof(*t));
```

```
t->z = malloc (sizeof (* (t->z)));
```

```
t->z->g = t->z;    t->z->d = t->z;
```

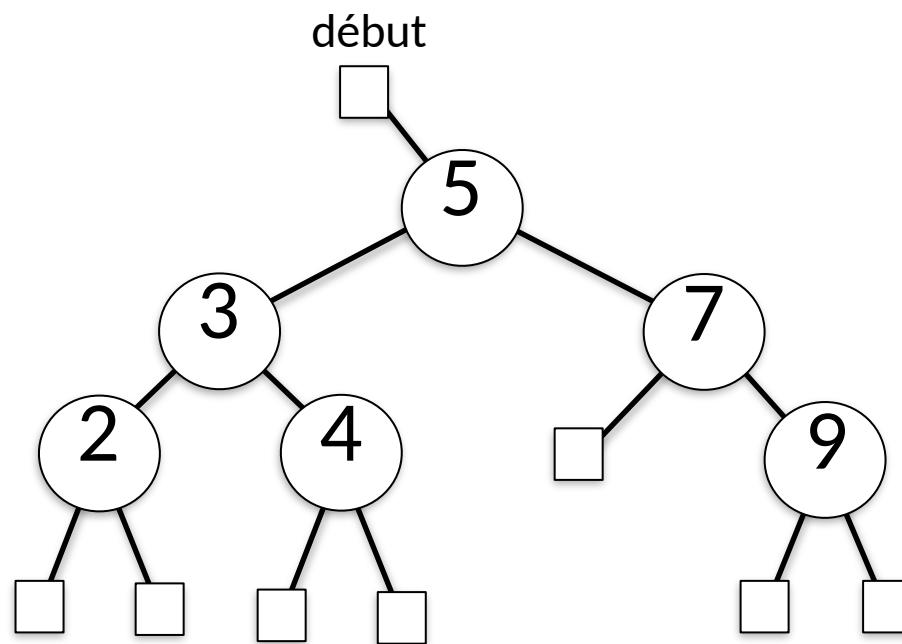
```
t->debut = malloc (sizeof(* (t->debut)));
```

```
t->debut->g = t->z;    t->debut->d = t->z;
```

```
t->debut->cle = MIN; // si l'on connaît MIN
```

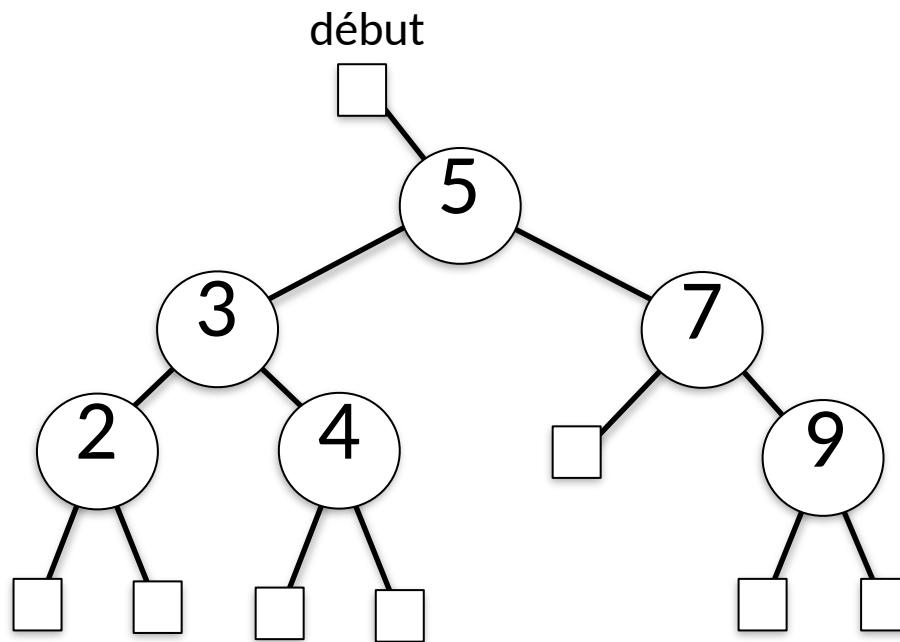
// t->debut est le plus petit élément : tous les autres seront situés dans son sous-arbre droit, donc plus grands que lui.

ABR - Tri



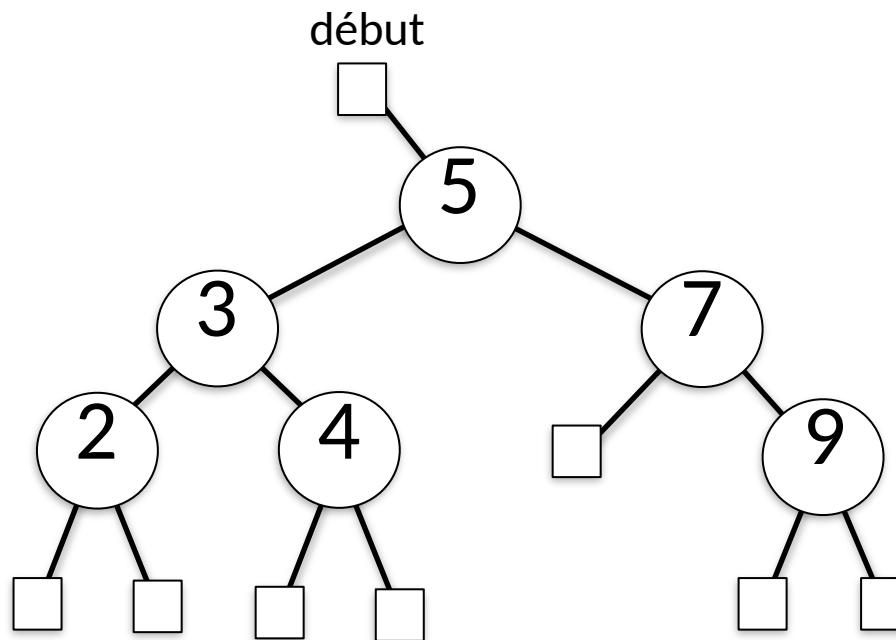
ABR - Tri

- Croissant -> 2,3,4,5,7,9 => *type de parcours ?*



ABR - Tri

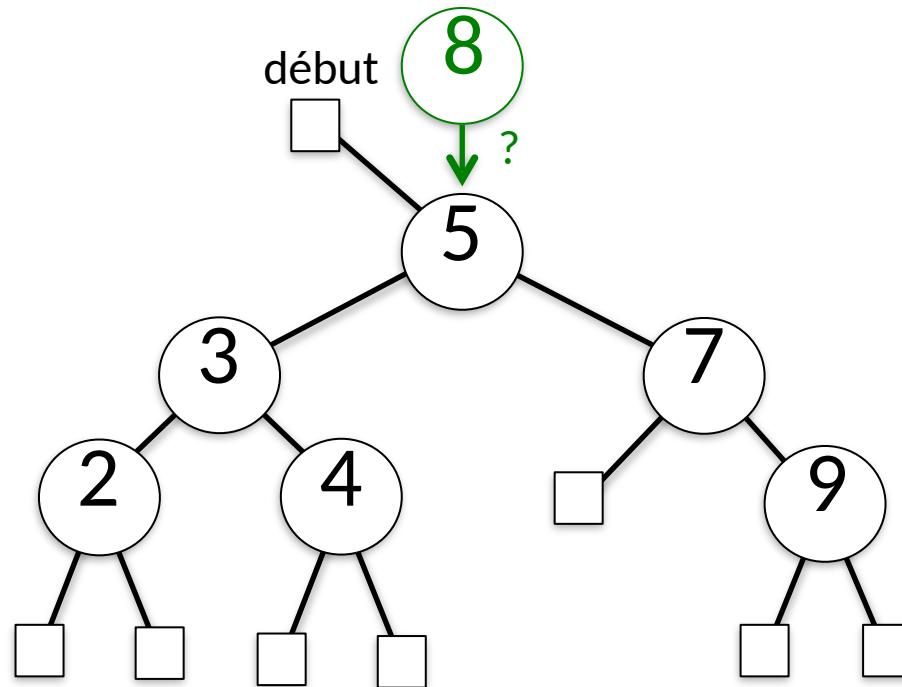
- Croissant -> 2,3,4,5,7,9 => **parcours infixé**



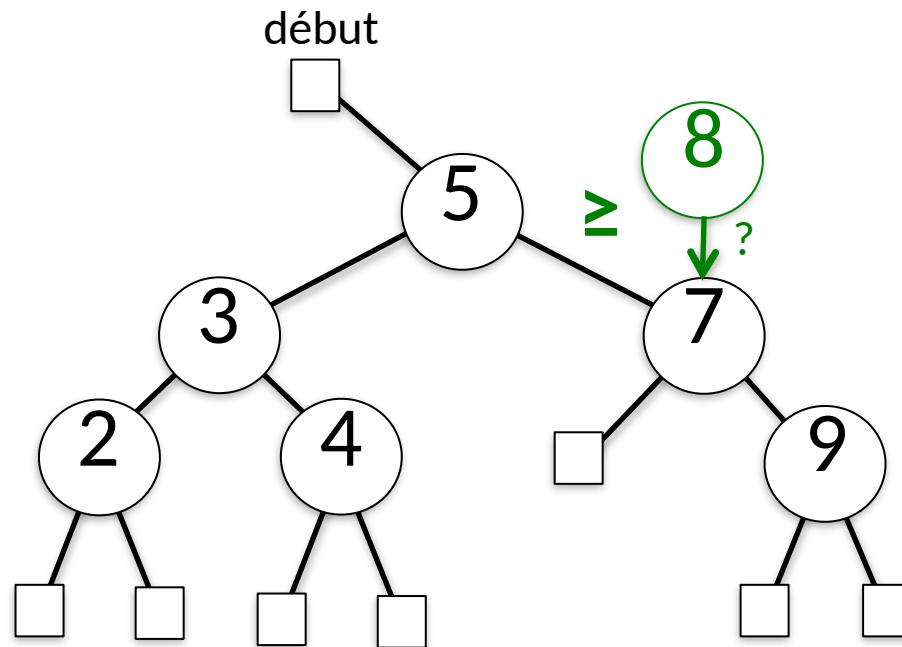
Insertion dans un ABR

Toujours par ajout d'une feuille

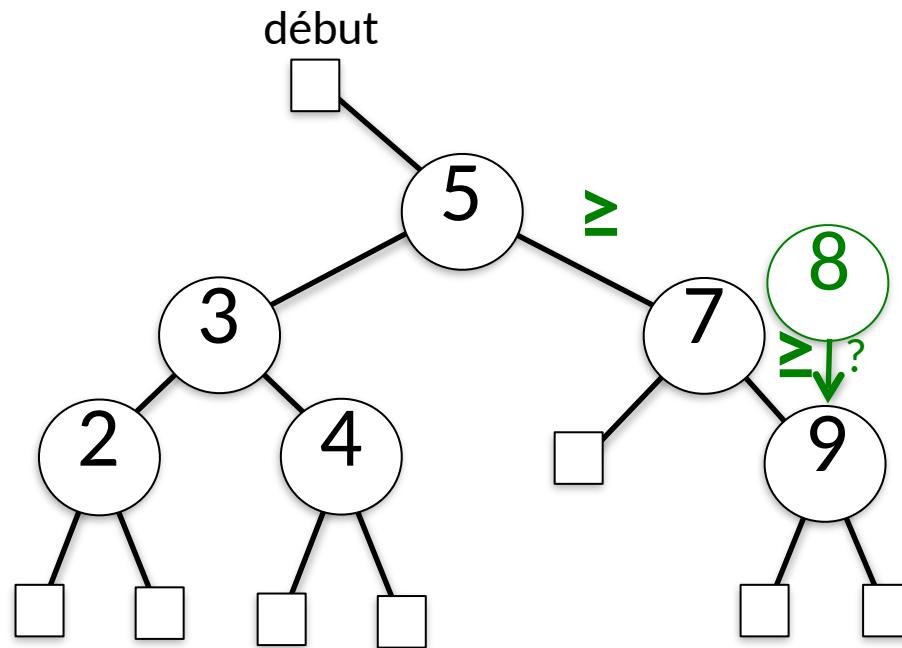
Insertion dans un ABR



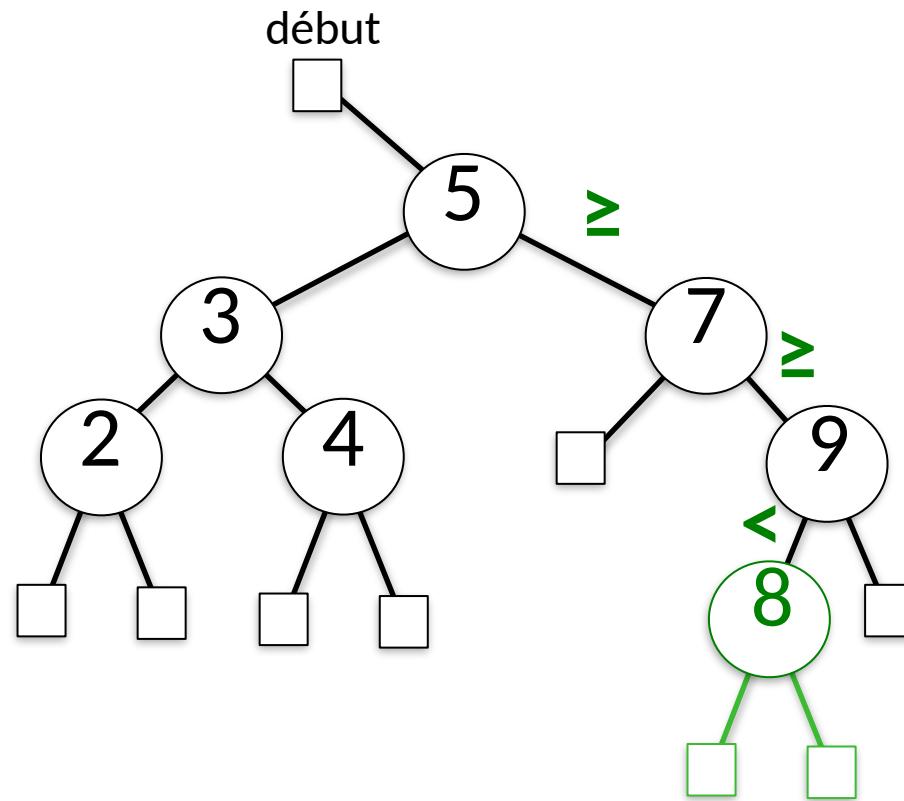
Insertion dans un ABR



Insertion dans un ABR

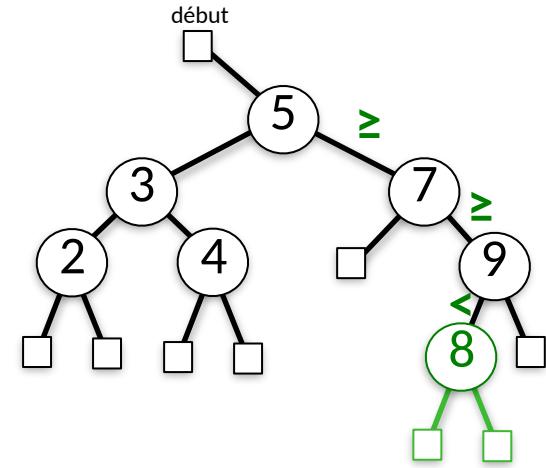


Insertion dans un ABR



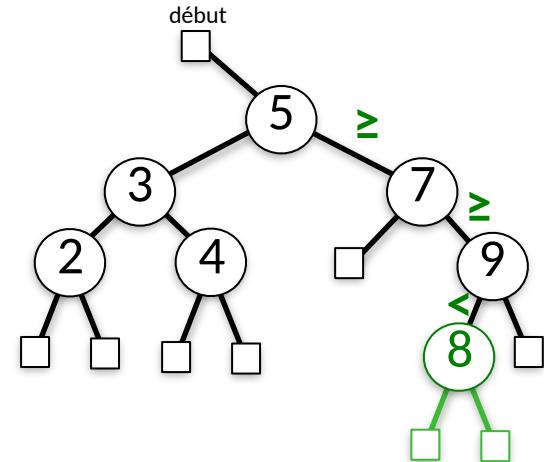
Insertion dans un ABR

```
// version itérative
void inserer(abr *a, char c) {
    noeud *p = a->debut; // parent de n
    noeud *n = p->d;
    while (n != a->z) {
        p = n;
        n = c < n->cle ? n->g : n->d;
    }
    // insertion d'un nouveau nœud sous p
    n = malloc(sizeof *n); n->cle = c;
    n->d = a->z; n->g = a->z;
    if (c < p->cle) p->g = n;
    else p->d = n;
}
```



Insertion dans un ABR

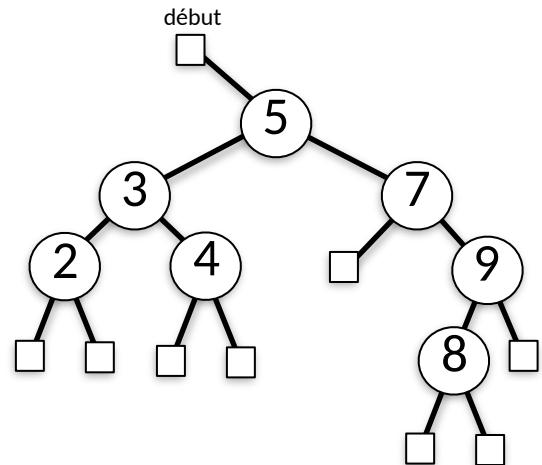
```
// Version récursive
void inserer(abr *a, char c) {
    insererRecursif(a->debut, c);
}
void insererRecursif(noeud *n, char c) {
    if (c < n->cle) { // insertion à gauche
        if (n->g->g == n->g) { // son propre fils => n->g == z
            noeud* f = malloc(sizeof *f); f->cle = c; f->d = n->g; f->g = n->g;
            n->g = f;
        } else insererRecursif(n->g, c);
    } else { // insertion à droite
        if (n->d->d == n->d) { // son propre fils => n->d == z
            noeud* f = malloc(sizeof *f); f->cle = c; f->d = n->d; f->g = n->d;
            n->d = f;
        } else insererRecursif(n->d, c);
    }
}
```



Recherche dans un ABR

// Similaire à la liste chaînée avec choix du suivant : fils droit ou fils gauche

```
noeud * rechercher(abr *a, char c) {  
    noeud *n = a->d;  
    a->z->cle = c;  
    while (c != n->cle) {  
        n = c < n->cle ? n->g : n->d;  
    }  
    return n;  
    // ici n = a->z si non trouvé !  
}
```



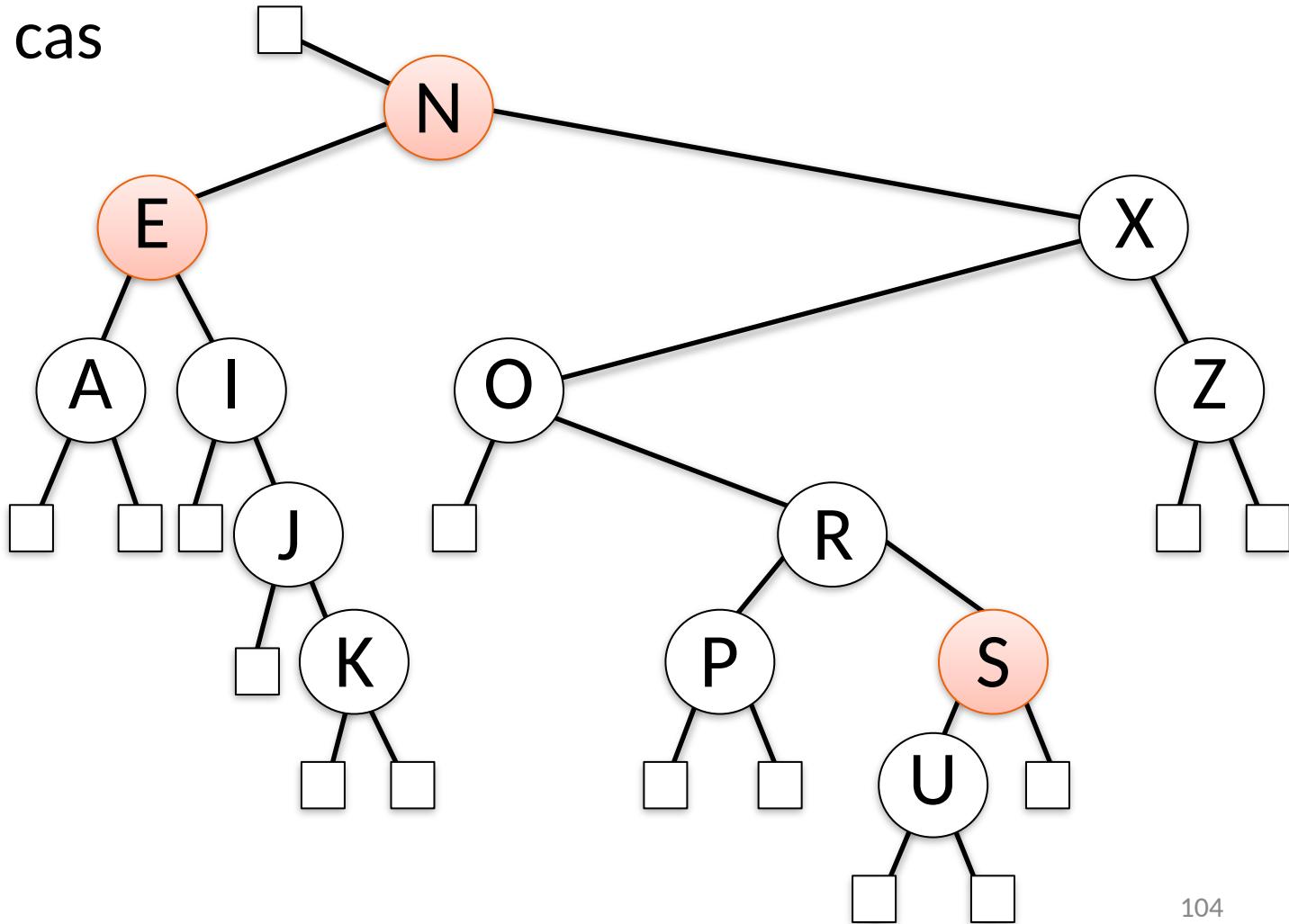
Suppression dans un ABR

- Différents cas

S ?

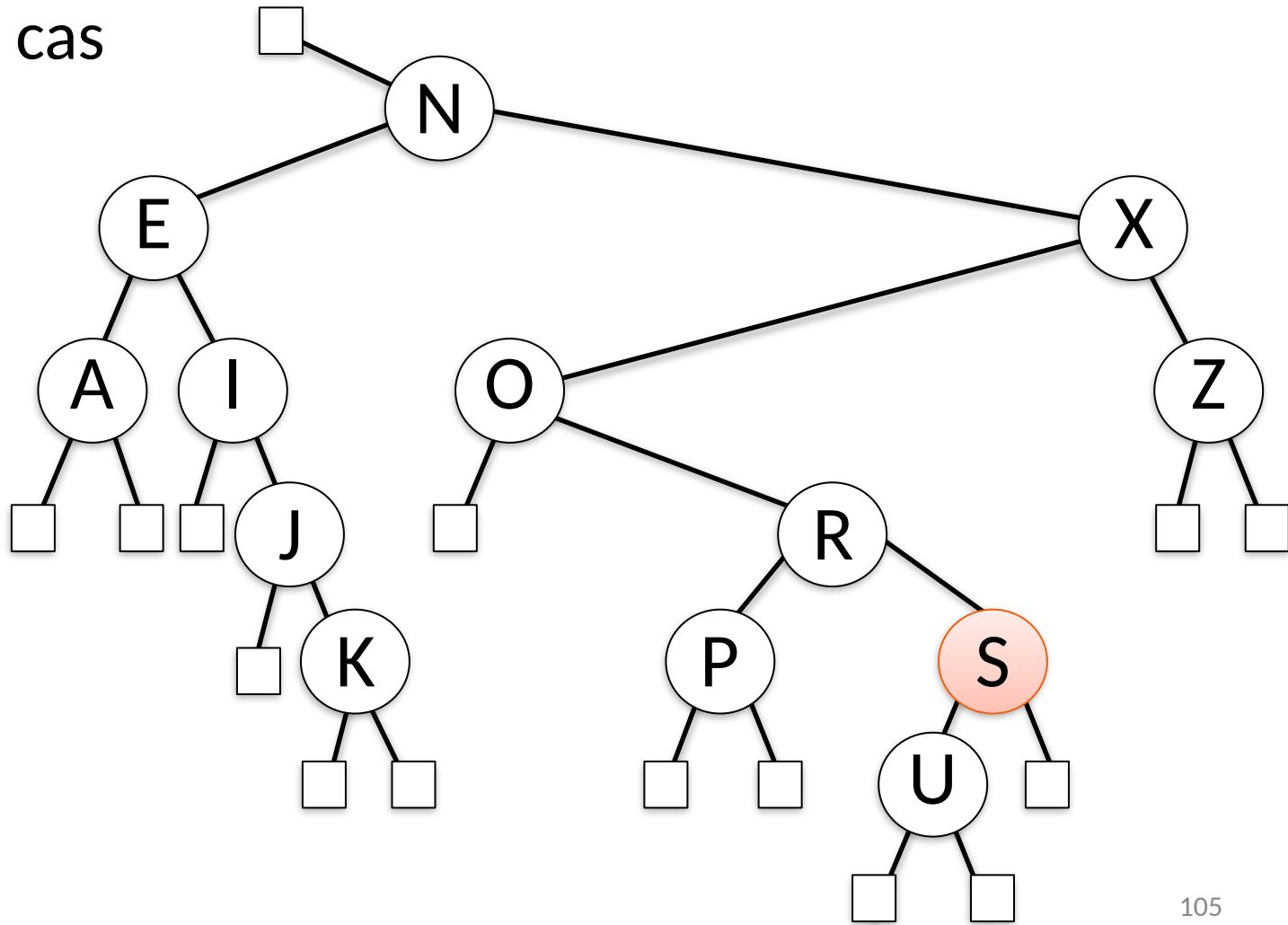
E ?

N ?



Suppression dans un ABR

- Différents cas

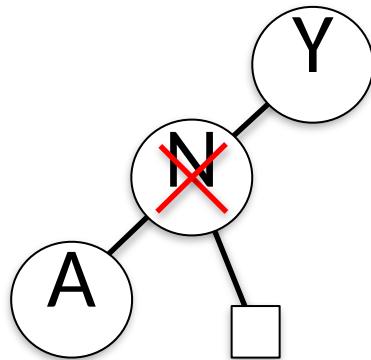


Suppression dans un ABR

- On étudie ici le côté droit du noeud à supprimer, on pourra de manière symétrique en déduire ce qui se passerait à gauche.

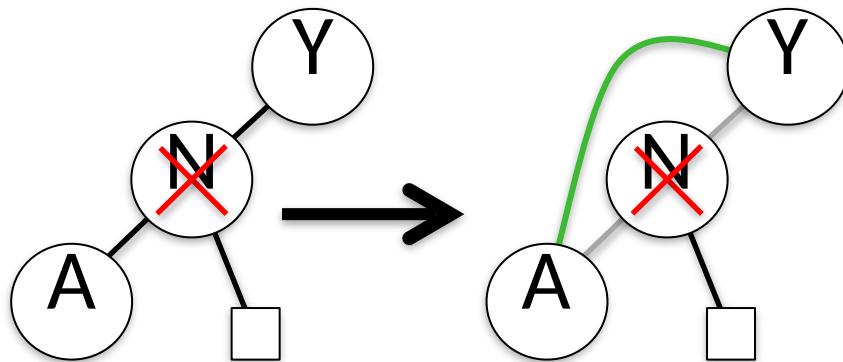
Suppression dans un ABR

- 1^{er} cas : N sans fils droit



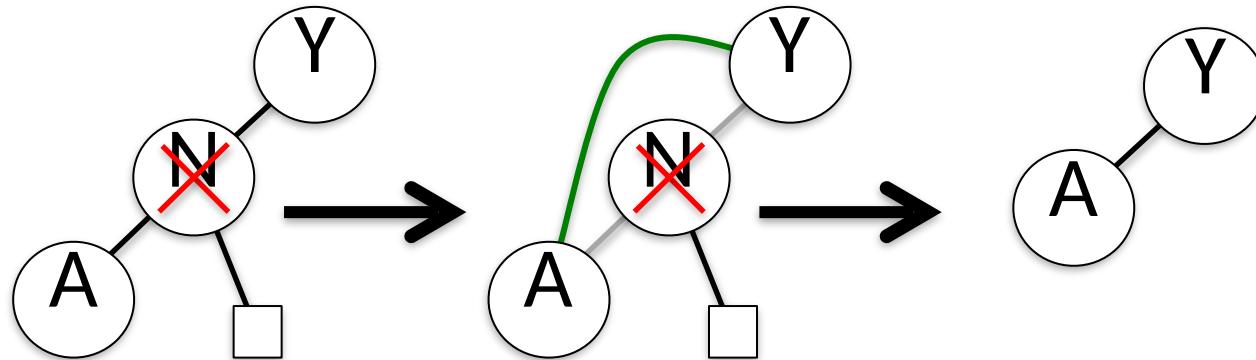
Suppression dans un ABR

- 1^{er} cas : N sans fils droit



Suppression dans un ABR

- 1^{er} cas : N sans fils droit



`y->g = n->g;`

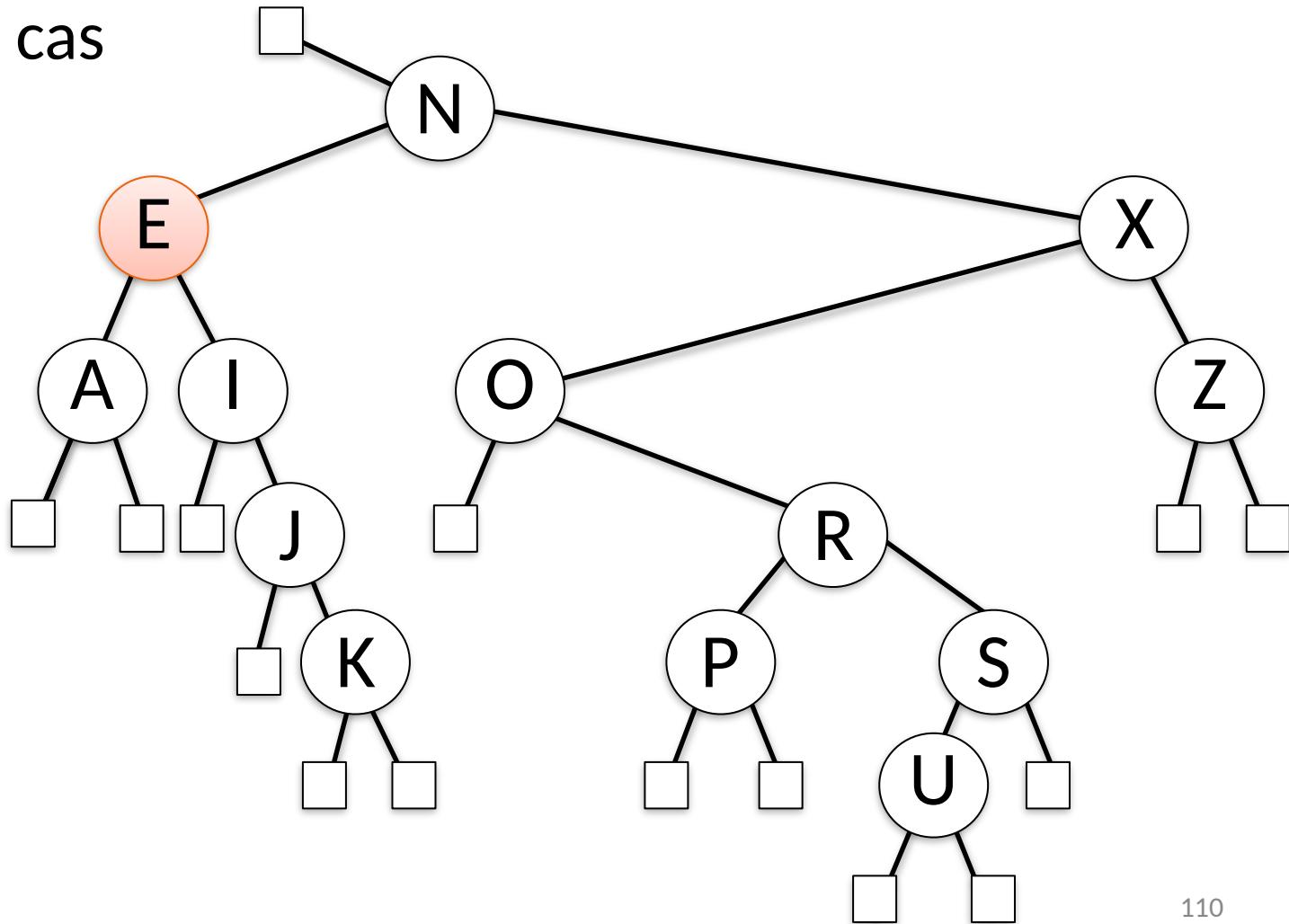
// ou si n était le fils droit de Y :

// `y->d = n->g`

`free(n);`

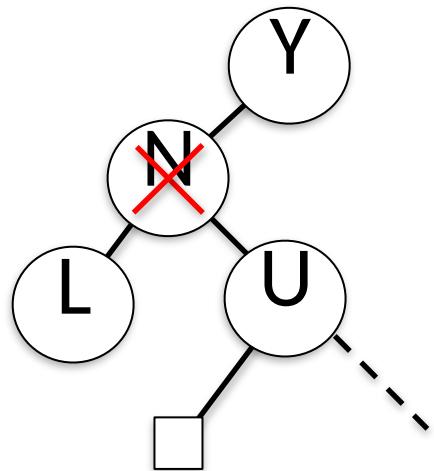
Suppression dans un ABR

- Différents cas



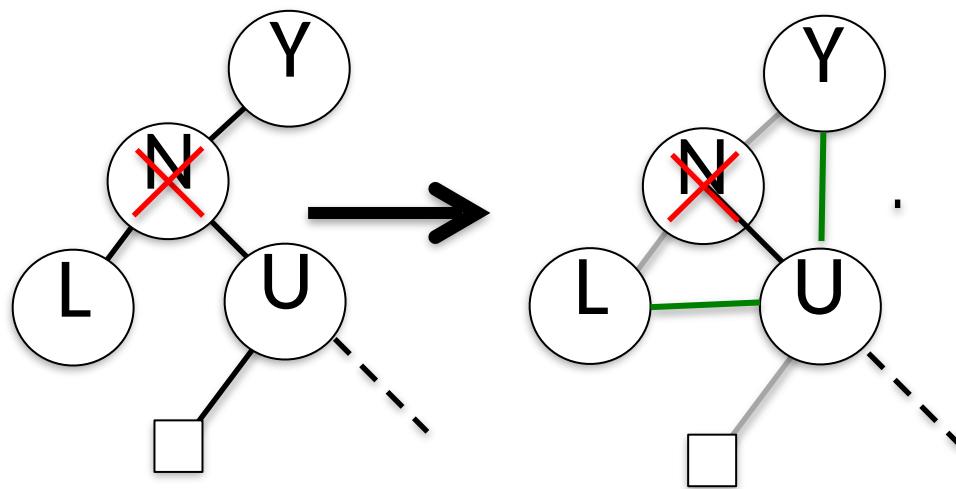
Suppression dans un ABR

- 2^{ème} cas : N avec un fils droit n'ayant pas de fils gauche



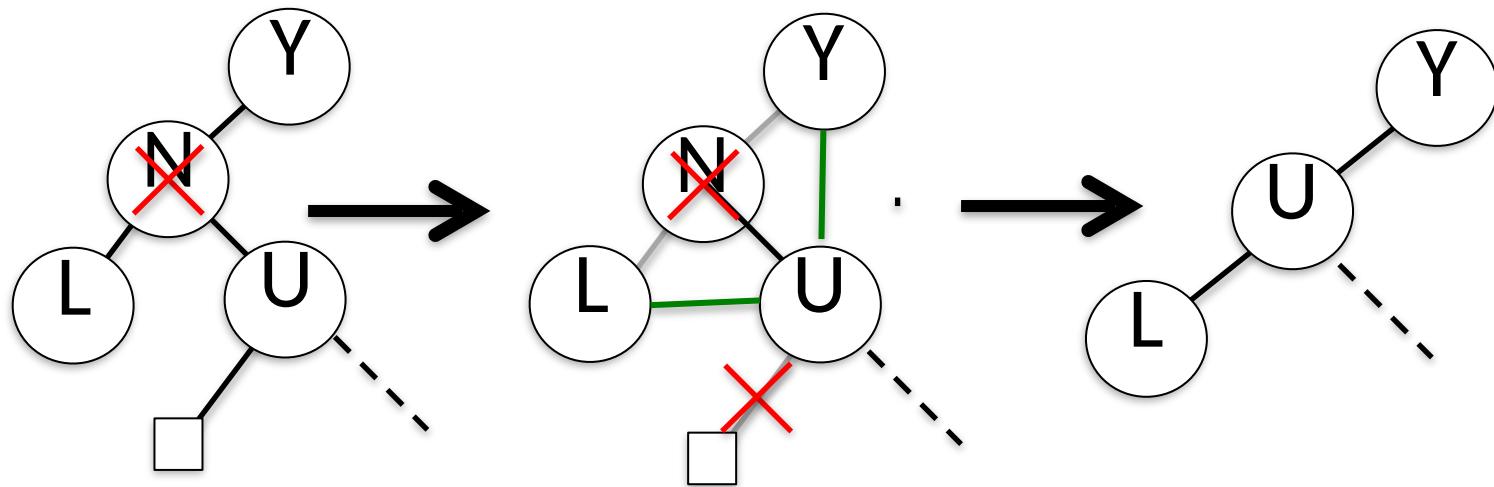
Suppression dans un ABR

- 2^{ème} cas : N avec un fils droit n'ayant pas de fils gauche



Suppression dans un ABR

- 2^{ème} cas : N avec un fils droit n'ayant pas de fils gauche

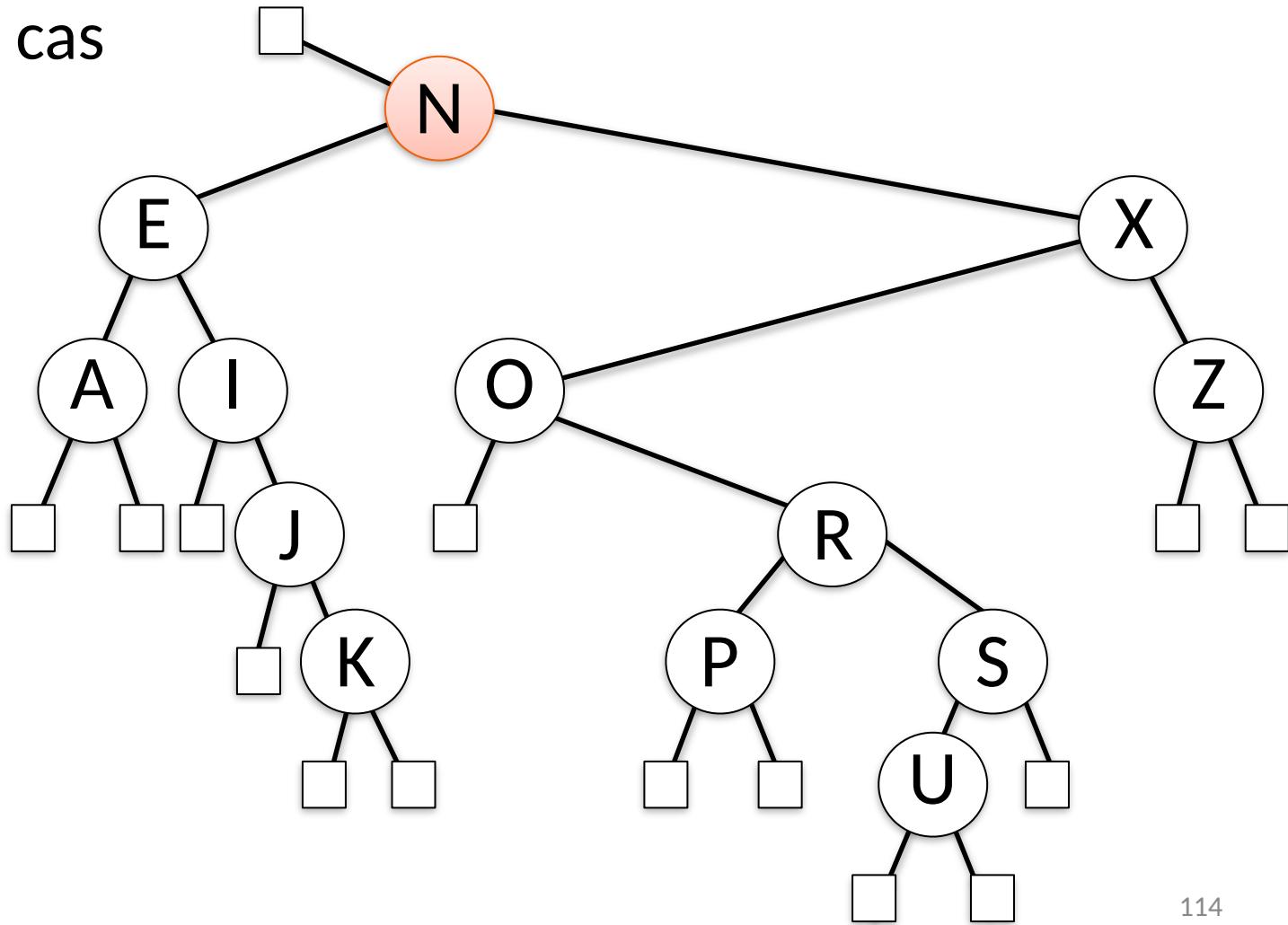


$u \rightarrow g = n \rightarrow g;$

$y \rightarrow g = n \rightarrow d$ // ou $y \rightarrow d = n \rightarrow d$ si $n = y \rightarrow d$

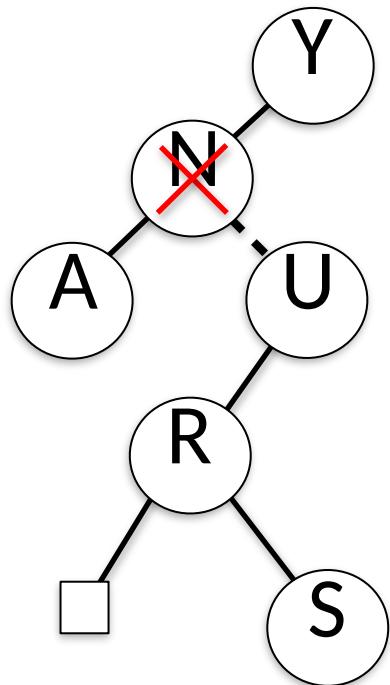
Suppression dans un ABR

- Différents cas



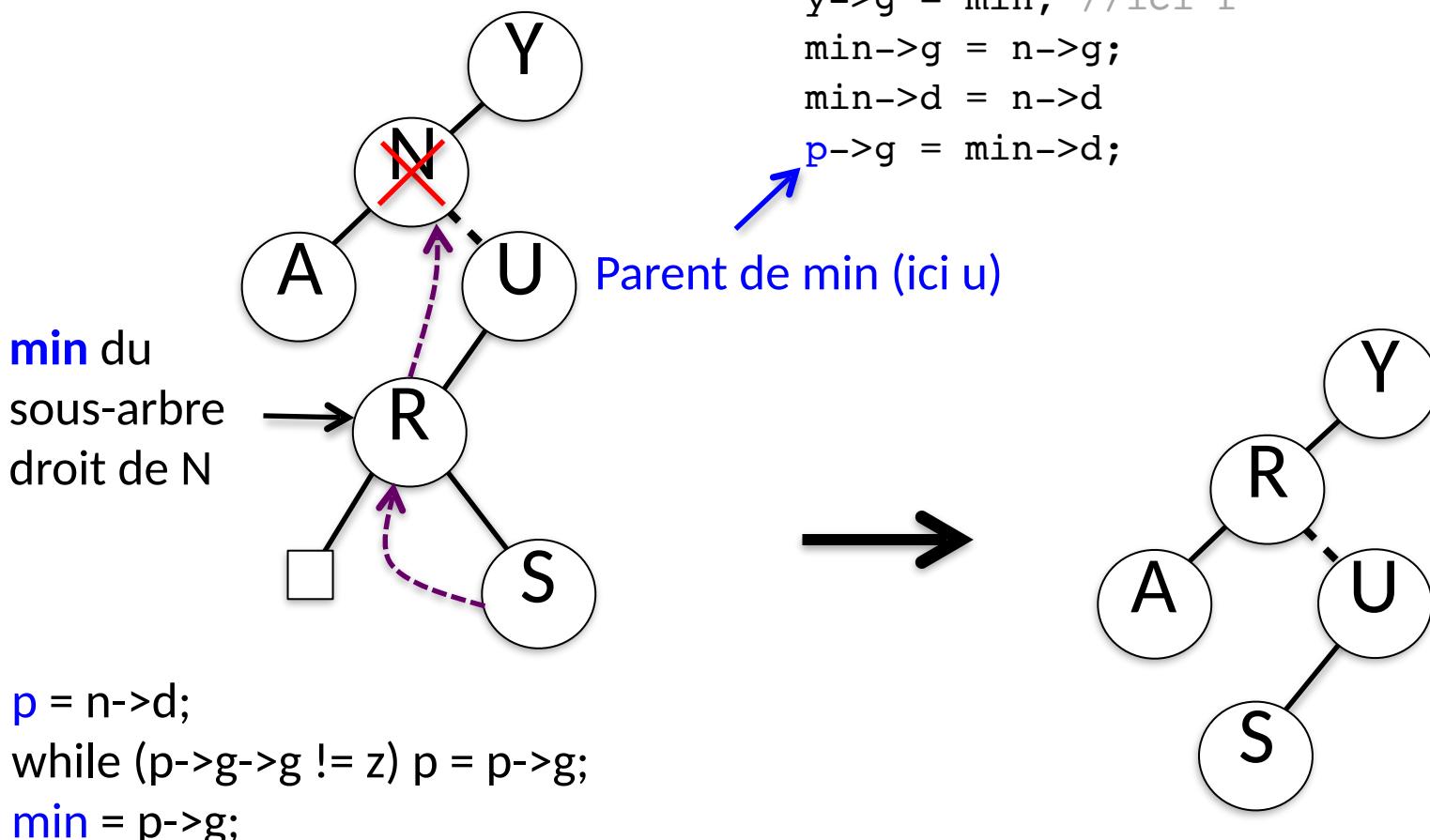
Suppression dans un ABR

- 3ème cas : tous les autres



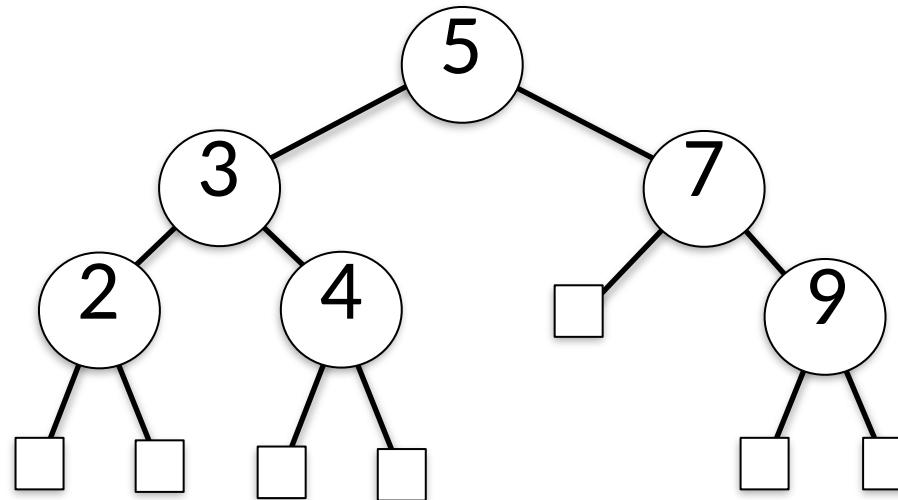
Suppression dans un ABR

- 3ème cas : tous les autres



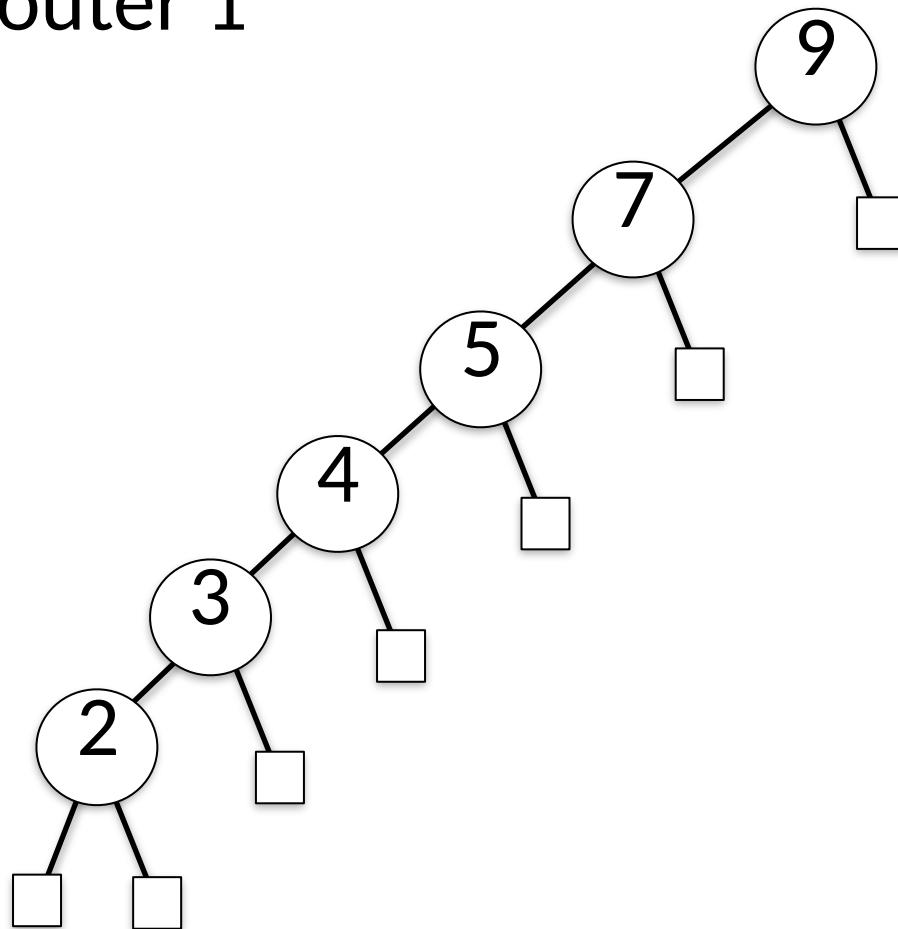
Complexité

- Insertion ? Ajouter 1



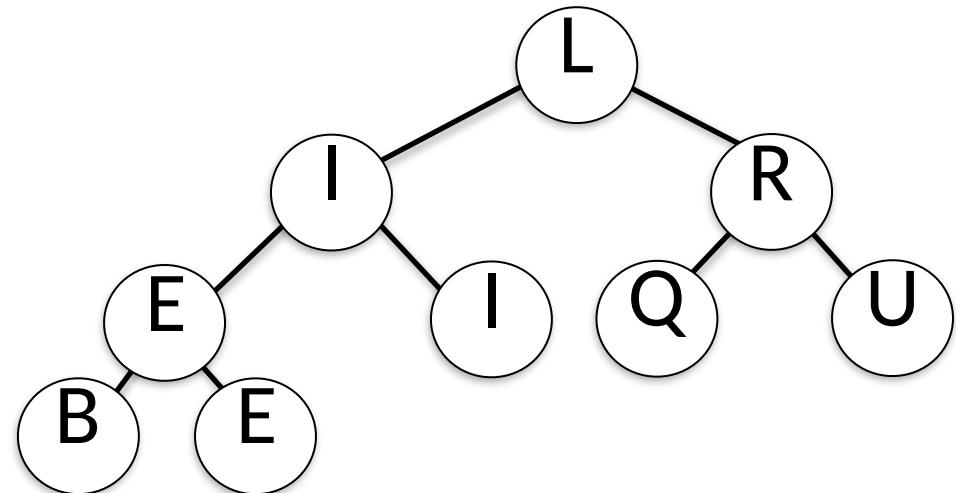
Complexité

- Insertion ? Ajouter 1

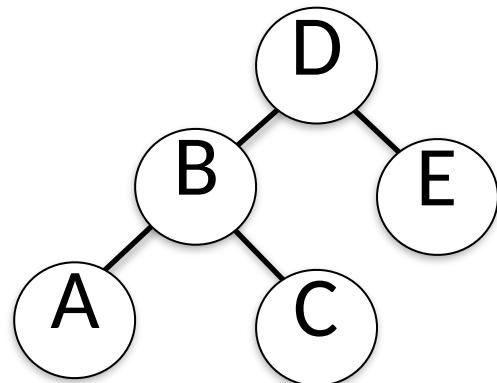


Arbre équilibré - Complexité

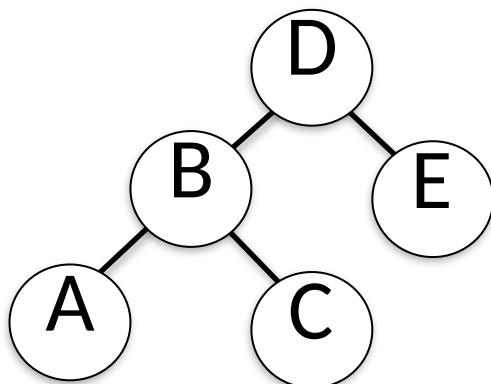
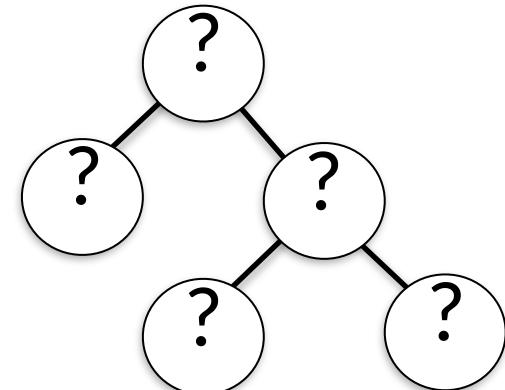
- Hauteur d'un arbre équilibré en fonction de son nombre de nœuds N de l'ordre de $\log_2 N$:
- Insertion ? Ajouter A : 4 comparaisons maxi
(9 éléments, $\log_2 9 = 3.17$) => $O(\log(n))$



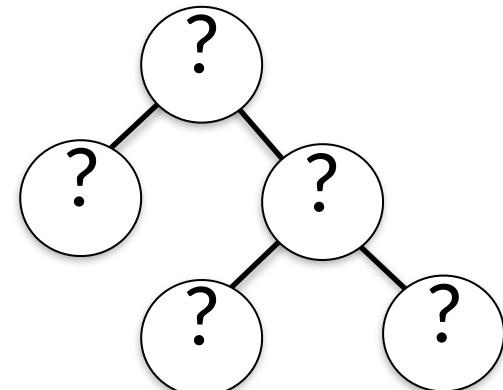
Arbre équilibré - Rotation



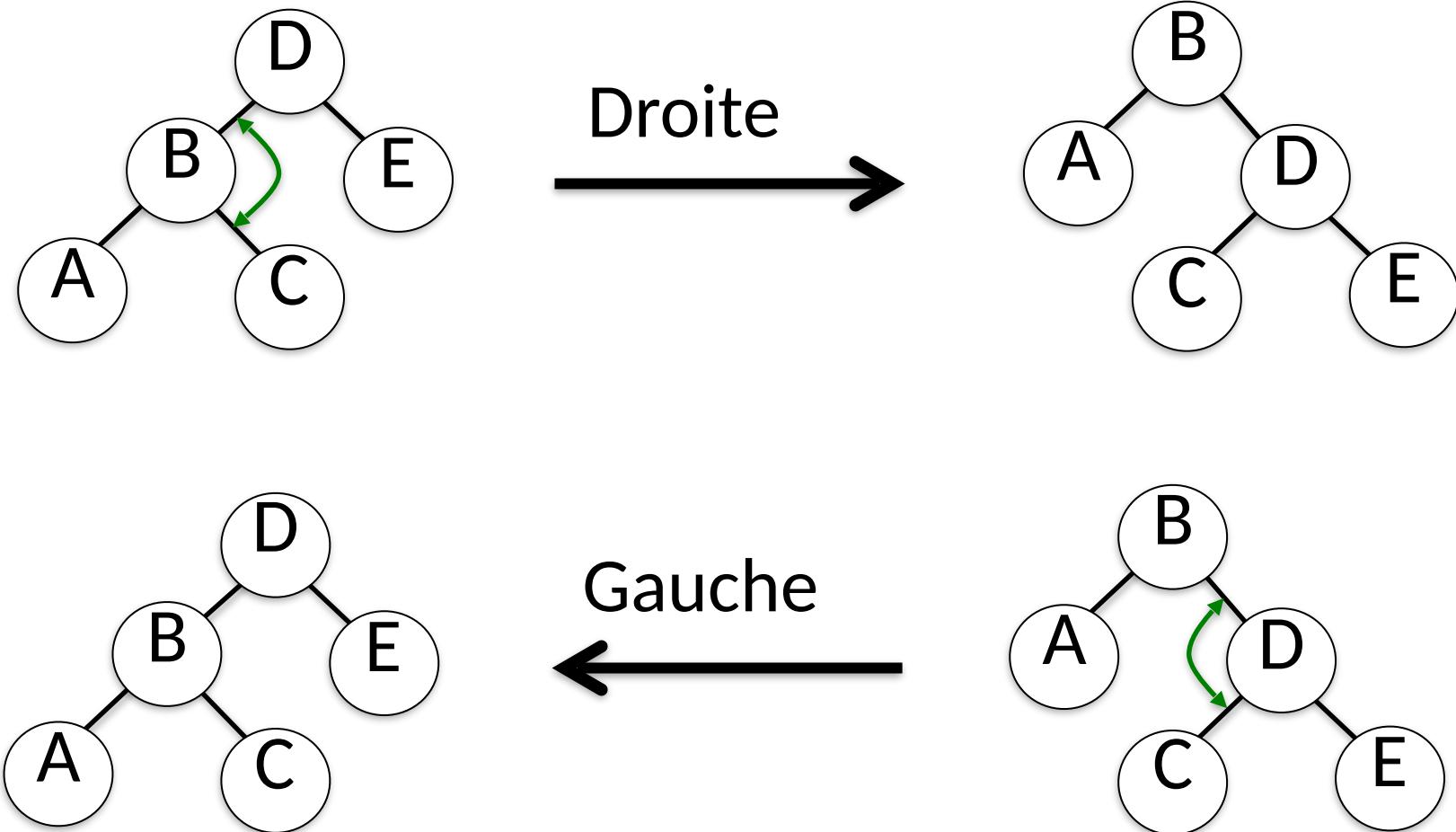
Droite →



← Gauche

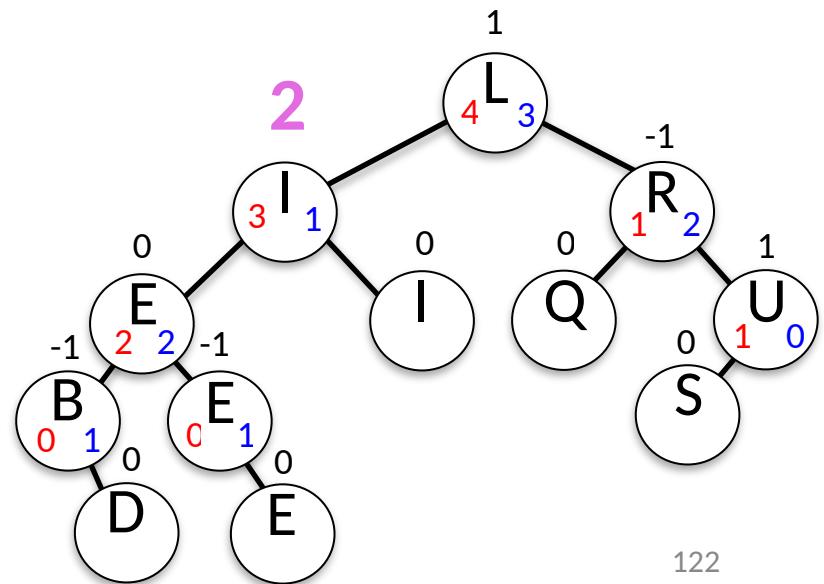
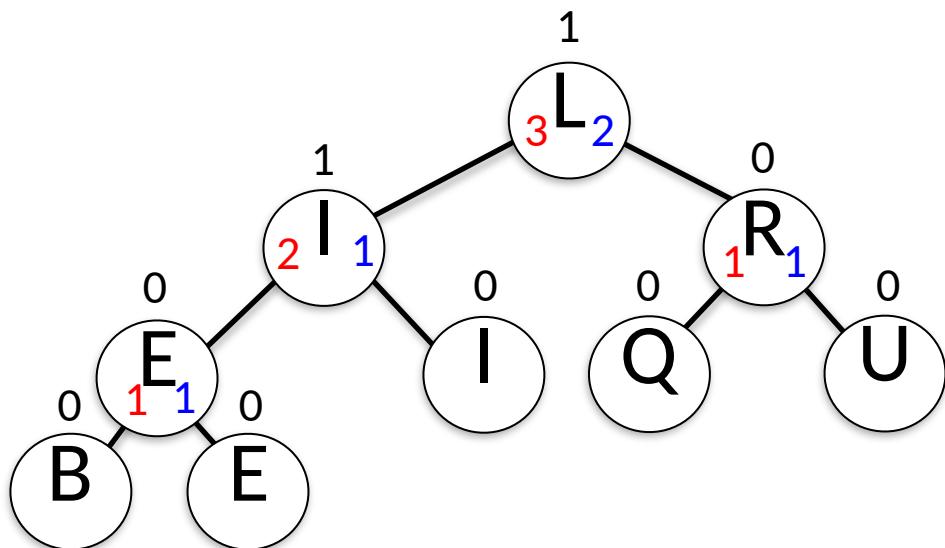


Arbre équilibré - Rotation



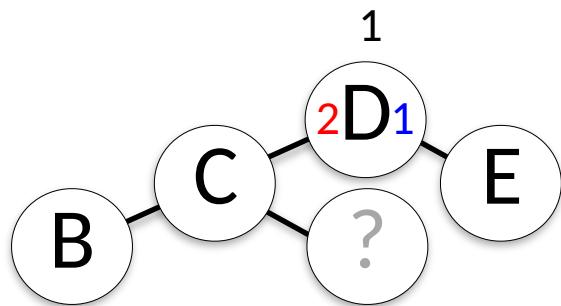
Arbre AVL (Adelson-Velsky et Landis)

- L'équilibre d'un nœud est la différence de hauteur de ses sous-arbres.
- Un arbre AVL est équilibré lorsque l'équilibre de chacun de ses nœuds est inférieur ou égal à 1 en valeur absolue.



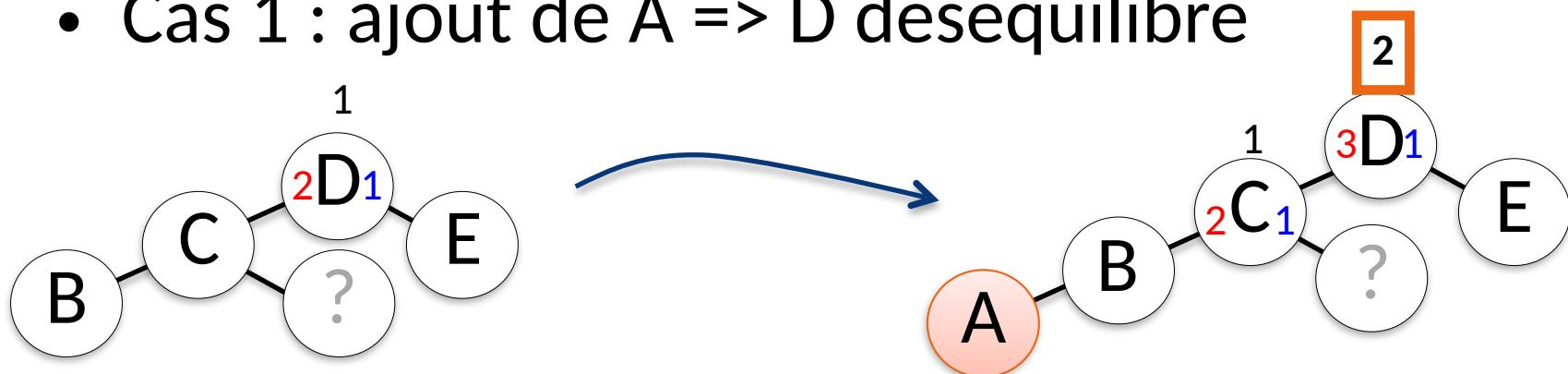
Arbre AVL - Equilibrage

- Cas 1



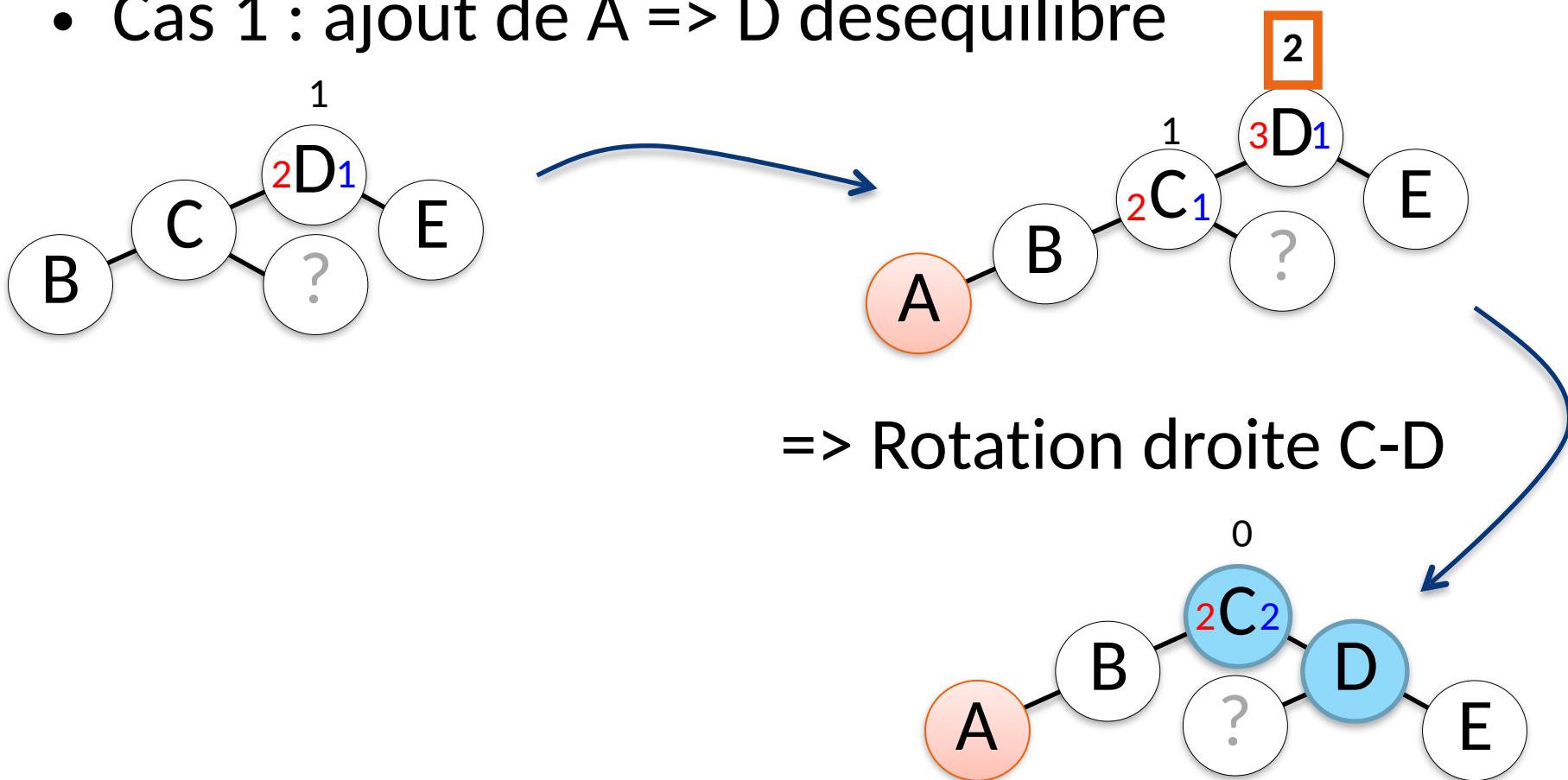
Arbre AVL - Equilibrage

- Cas 1 : ajout de A => D déséquilibré



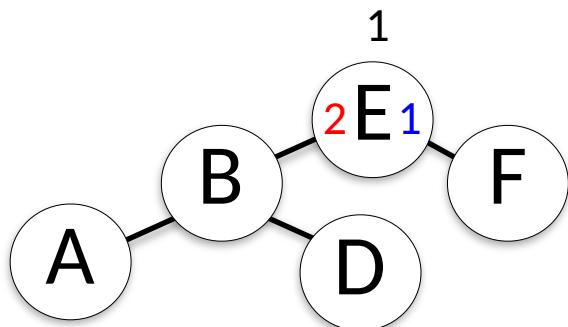
Arbre AVL - Equilibrage

- Cas 1 : ajout de A => D déséquilibré



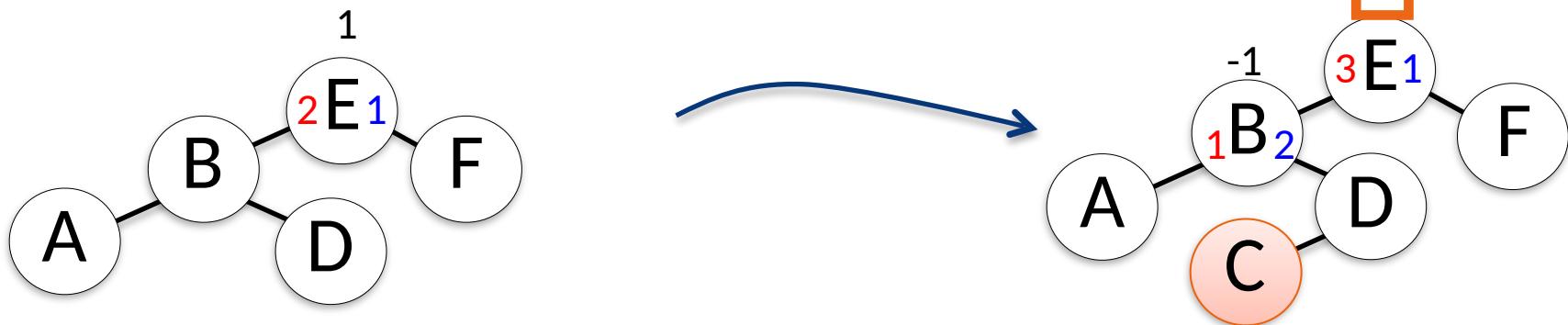
Arbre AVL - Equilibrage

- Cas 2



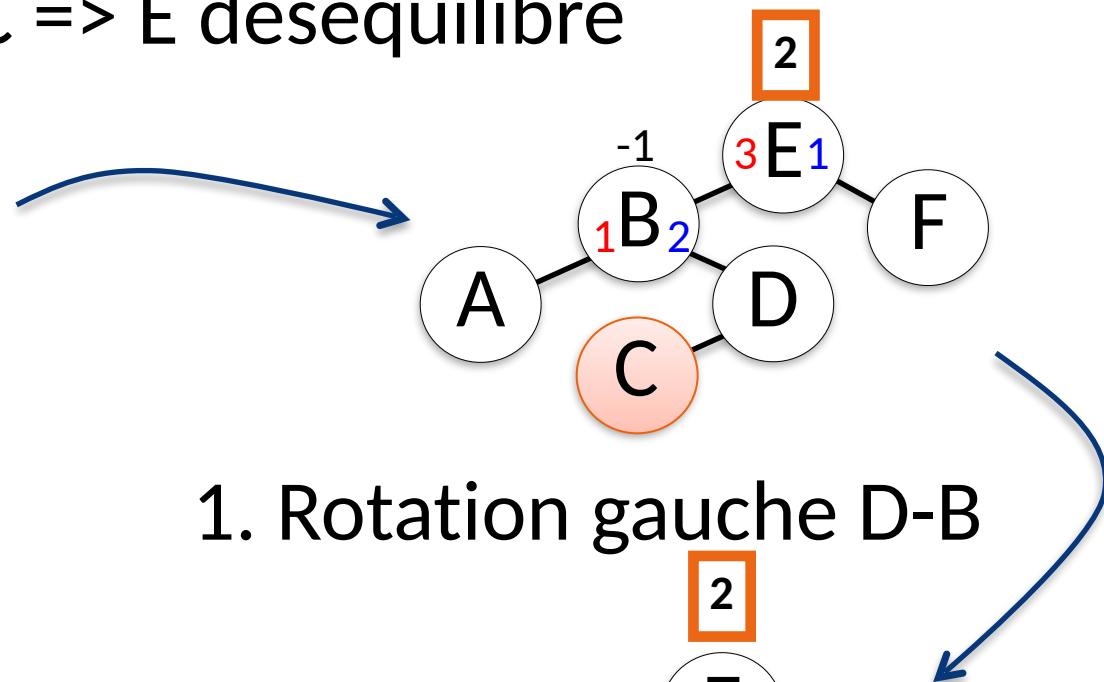
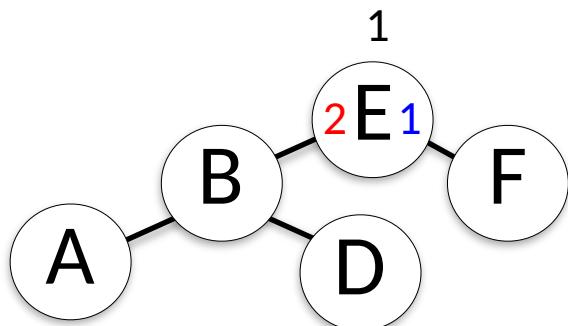
Arbre AVL - Equilibrage

- Cas 2 : ajout de C => E déséquilibré

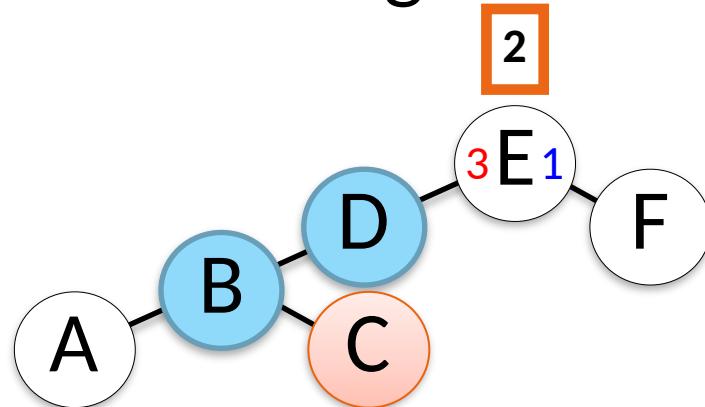


Arbre AVL - Equilibrage

- Cas 2 : ajout de C => E déséquilibré

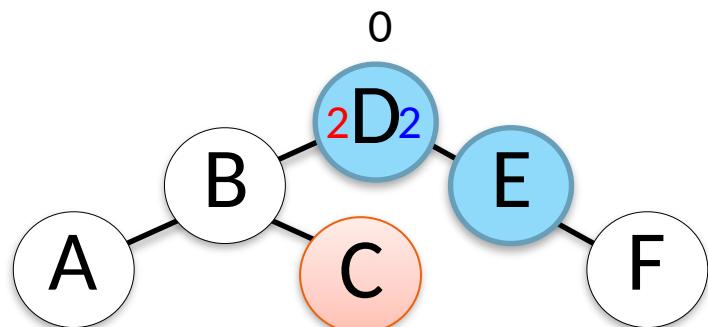
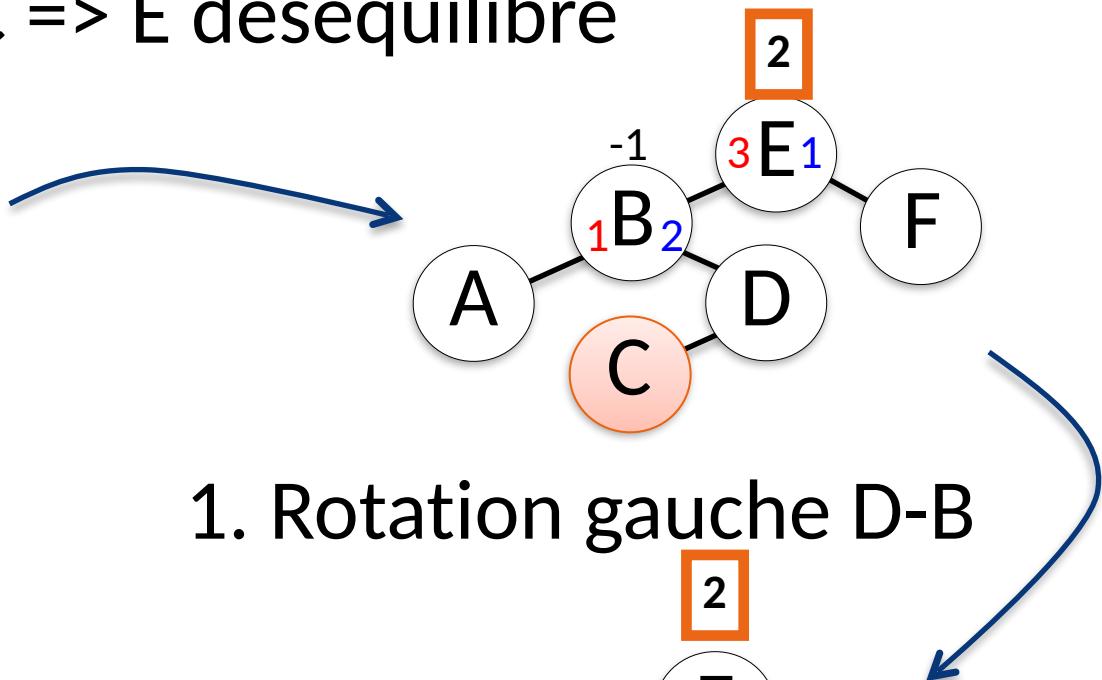
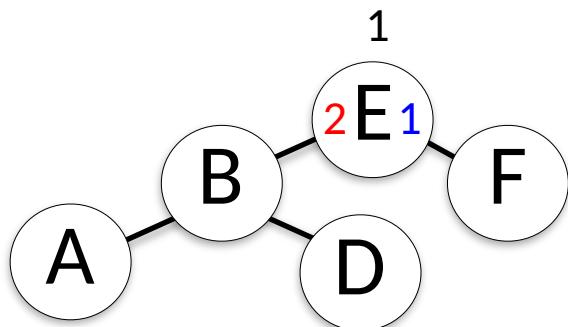


1. Rotation gauche D-B



Arbre AVL - Equilibrage

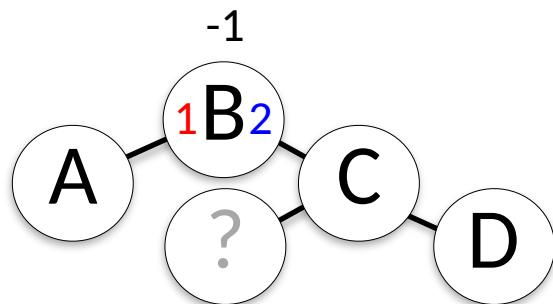
- Cas 2 : ajout de C => E déséquilibré



2. Rotation droite D-E

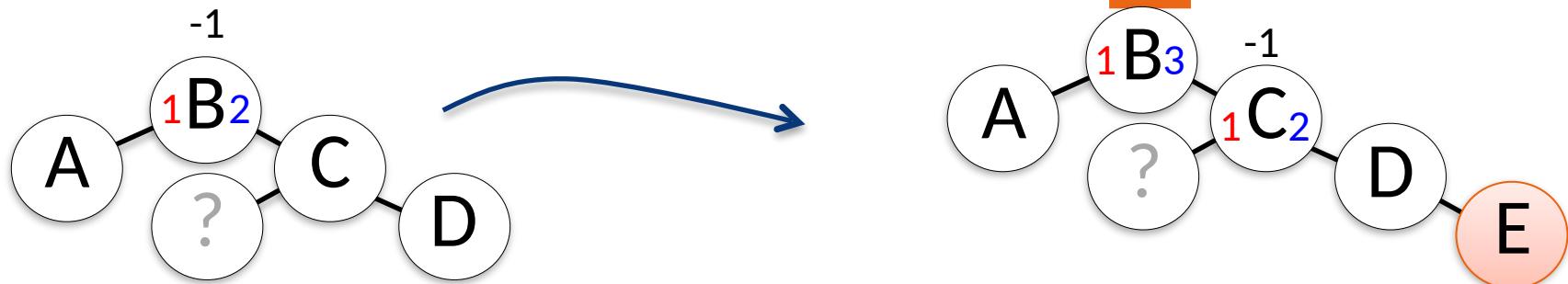
Arbre AVL - Equilibrage

- Cas 3



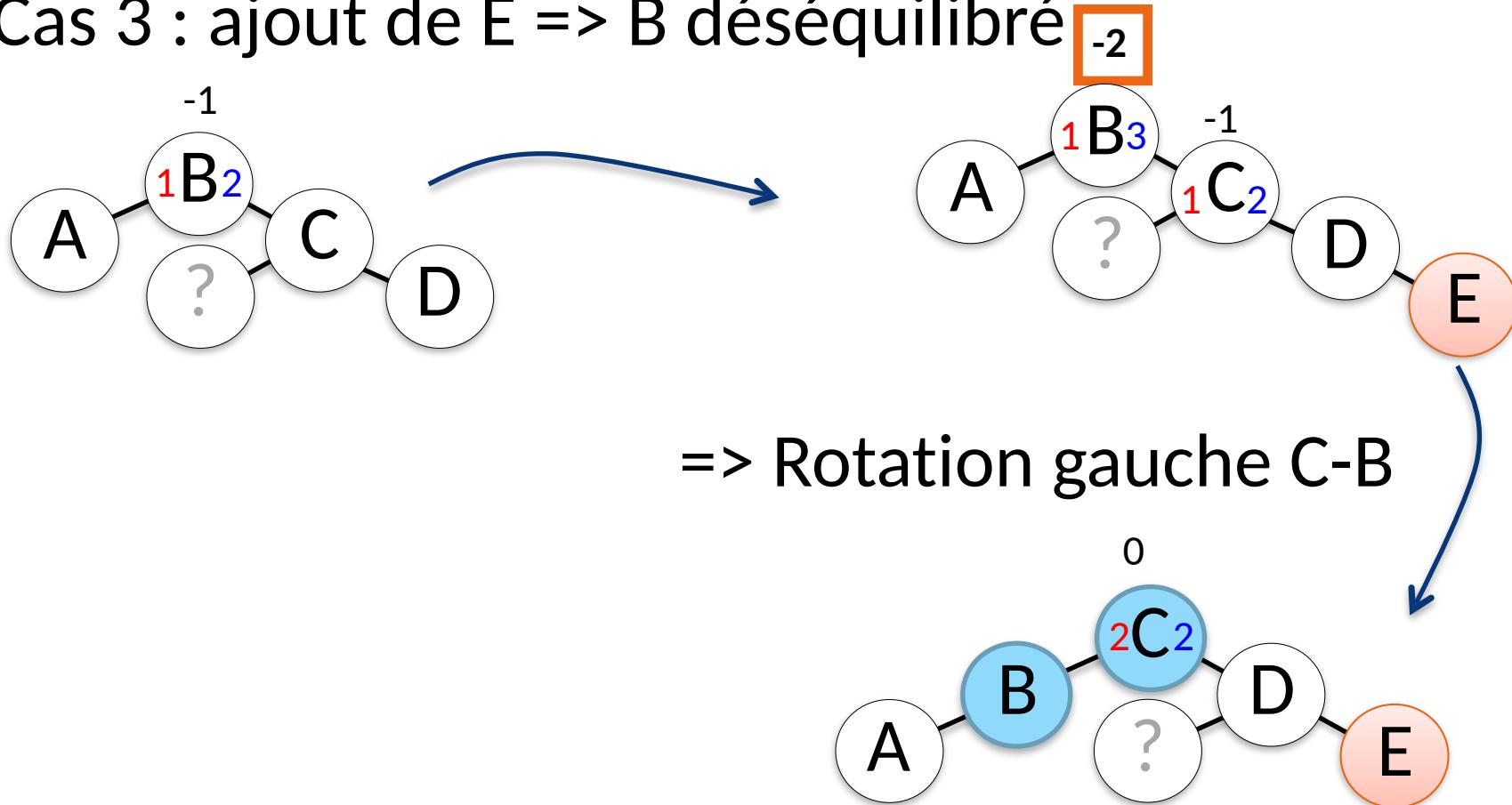
Arbre AVL - Equilibrage

- Cas 3 : ajout de E => B déséquilibré -2



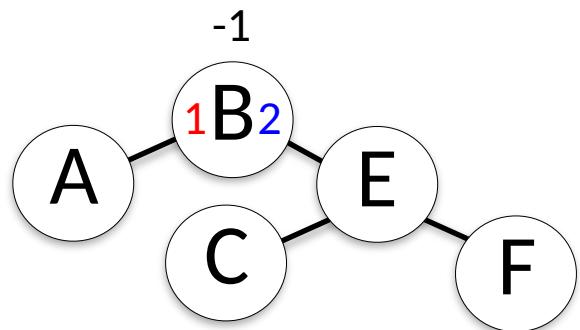
Arbre AVL - Equilibrage

- Cas 3 : ajout de E => B déséquilibré



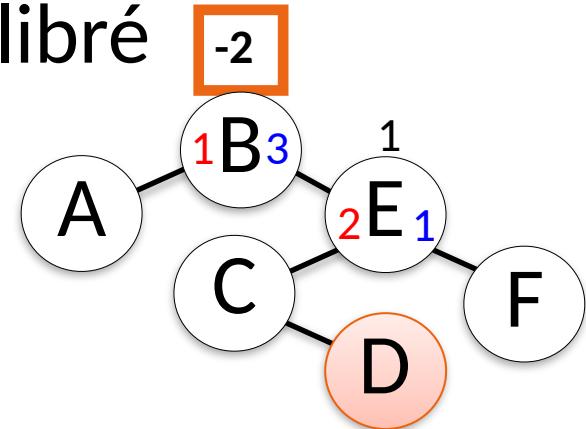
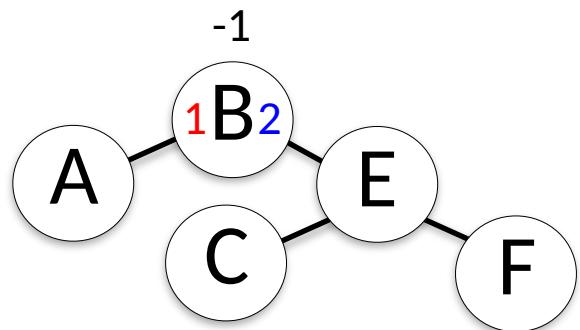
Arbre AVL - Equilibrage

- Cas 4



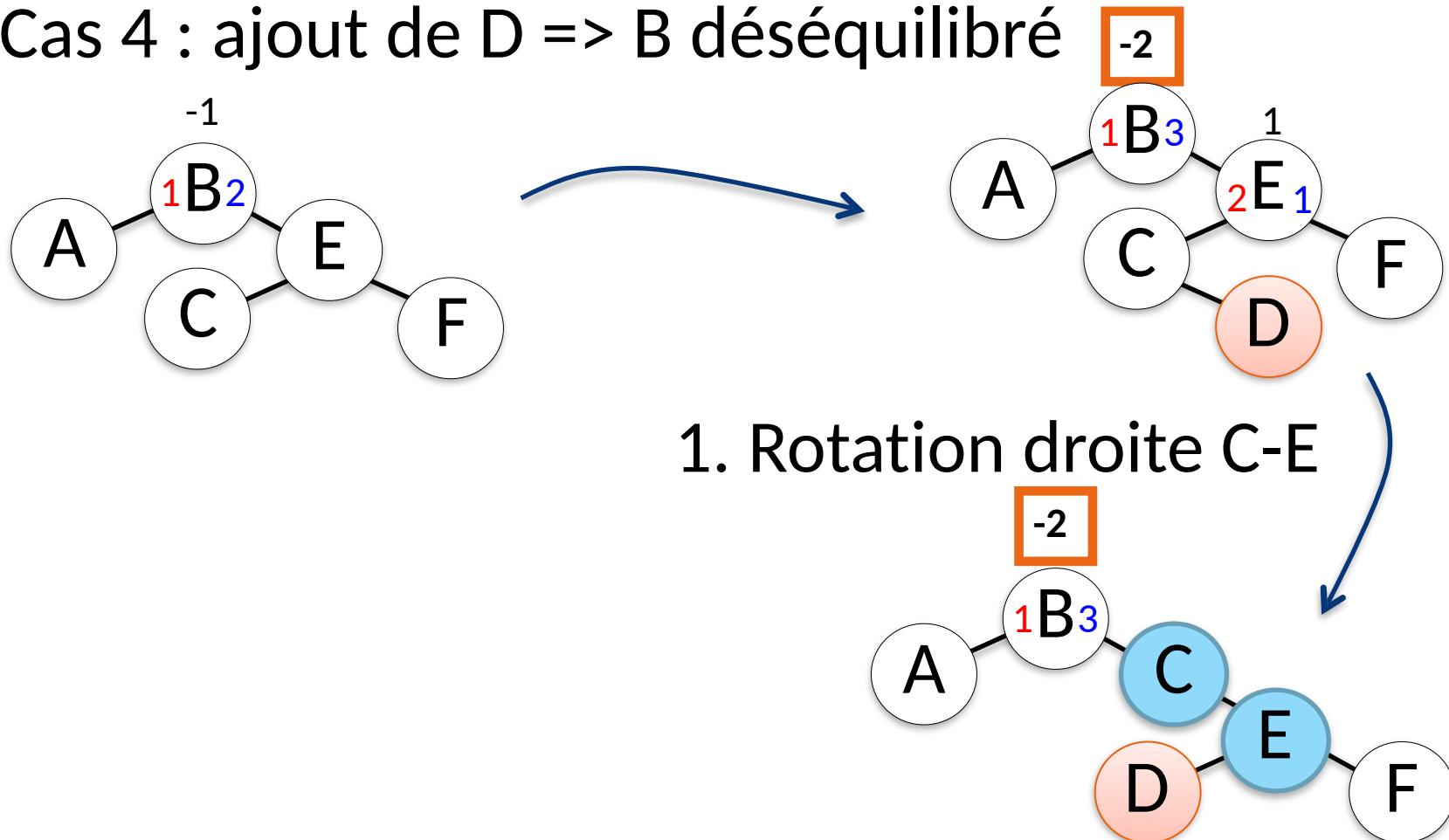
Arbre AVL - Equilibrage

- Cas 4 : ajout de D => B déséquilibré



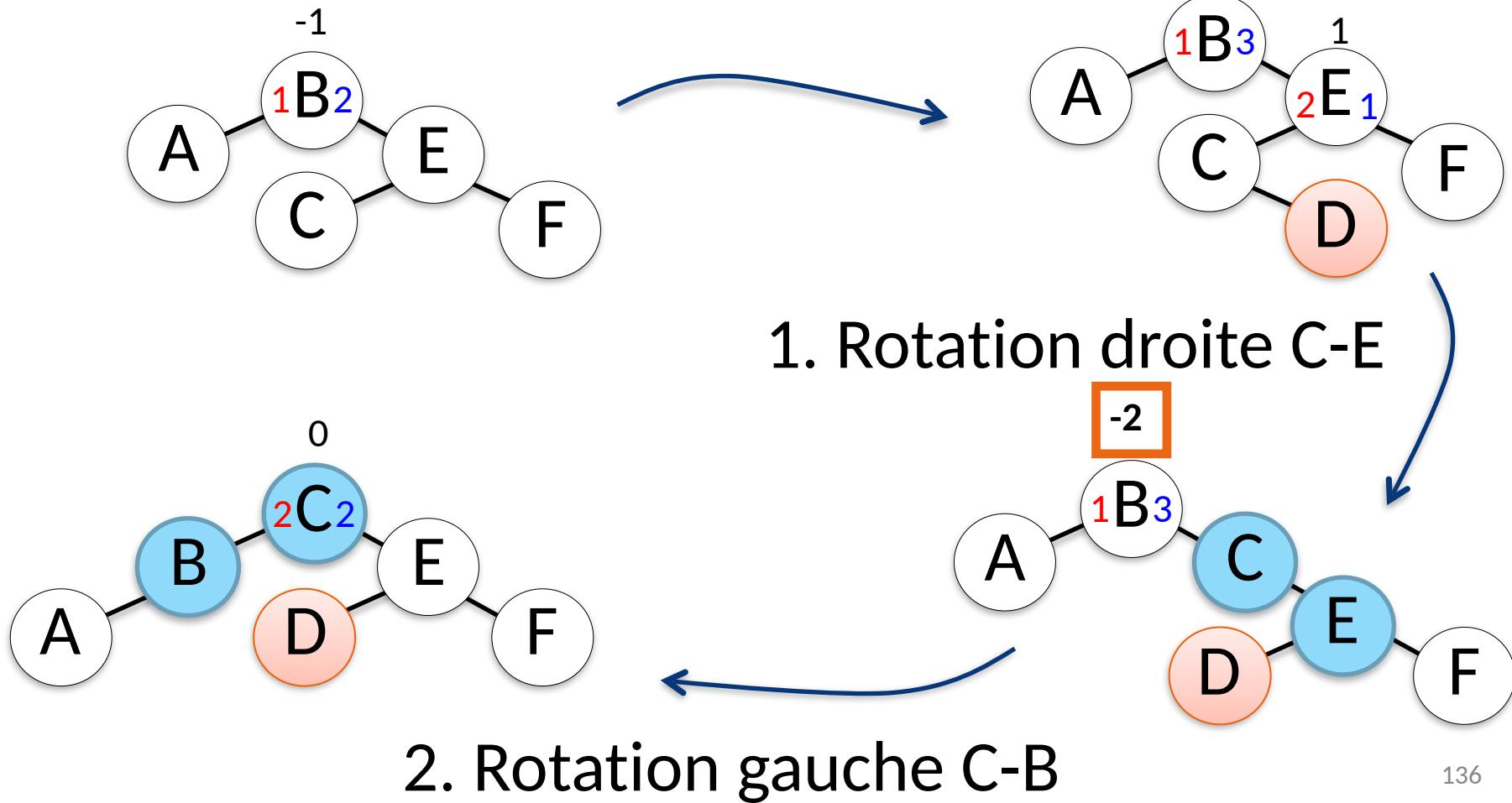
Arbre AVL - Equilibrage

- Cas 4 : ajout de D => B déséquilibré



Arbre AVL - Equilibrage

- Cas 4 : ajout de D => B déséquilibré



Arbre AVL - Equilibrage

- Deux rotations maximales sont nécessaires pour rééquilibrer un arbre après une insertion ou une suppression
- A partir du noeud modifié le plus bas, remonter dans l'arbre et vérifier l'équilibre des noeuds rencontrés.
- Possibilité de conserver un lien vers le père de chaque nœud ou d'alimenter une pile lors de la descente dans l'arbre.

Arbre AVL - Equilibrage

Équilibrage de n

si équilibre de n = 2 alors

 si équilibre du fils gauche de n < 0 alors

 rotation gauche du fils gauche de n

 rotation droite de n

sinon si équilibre de n = -2

 si équilibre du fils droit de n < 0 alors

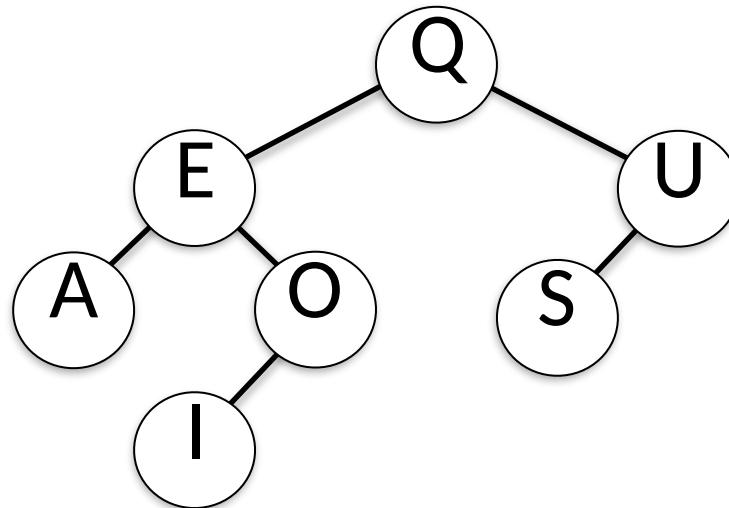
 rotation droite du fils droit de n

 rotation gauche de n

fin

ABR - Implantation par tableau

- **Liens-pères**, deux tableaux :
 - Tableau des clés
 - Tableau des indices du père de chaque nœud



| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| S | E | Q | U | O | I | A |
| 3 | 2 | | 2 | 1 | 4 | 1 |

Exercices

- Mise en œuvre d'un arbre binaire par pointeurs, valeurs de type `unsigned char` ou `void*`, avec noeuds factices
- Insertion dans l'arbre considéré comme un ABR
- Parcours : en largeur, préfixé, infixé, postfixé (versions récursive et itérative)
- Recherche dans l'ABR
- Suppression de l'ABR
- Mise en œuvre d'un tas d'entiers par tableau : ajout / suppression
- Création d'un AST à partir d'une file en NPI puis évaluation récursive.
- Suppression dans un ABR
- Equilibrage d'un ABR (\Rightarrow AVL)