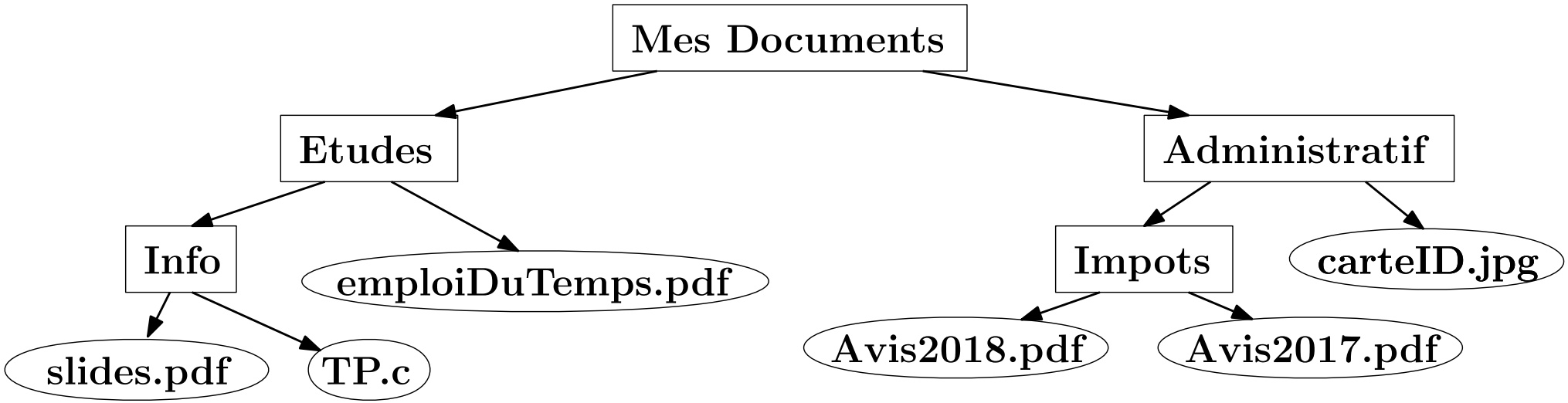
Chapitre 8 :
d'autres Types de Données Abstraits:
Arbres

1) Definition et vocabulaire
2) Arbres binaires
3) Parcours d'arbres en largeur et en profondeur
4) Arbres binaires de Recherche (ABR)
5) Parcours d'arbres récursifs

1) Definition et vocabulaire

- On va voir maintenant un autre Type de Données Abstrait: **l'arbre** (nom complet: **arbre enraciné**).

- Il fonctionne comme un système de dossier/fichier sur un ordinateur, et on le dessine généralement à l'envers en informatique: la racine en haut, les feuilles en bas.

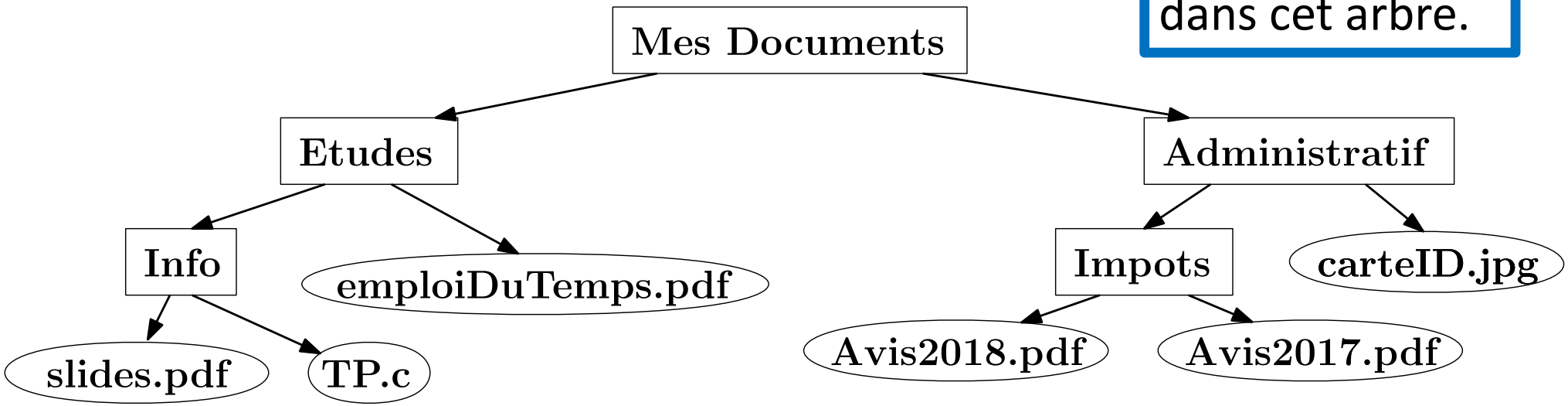


1) Definition et vocabulaire

Vocabulaire:

- **un noeud: une position dans l'arbre**
- les fils d'un nœud: les nœuds juste en dessous
- le père d'un nœud: le nœud juste en-dessus

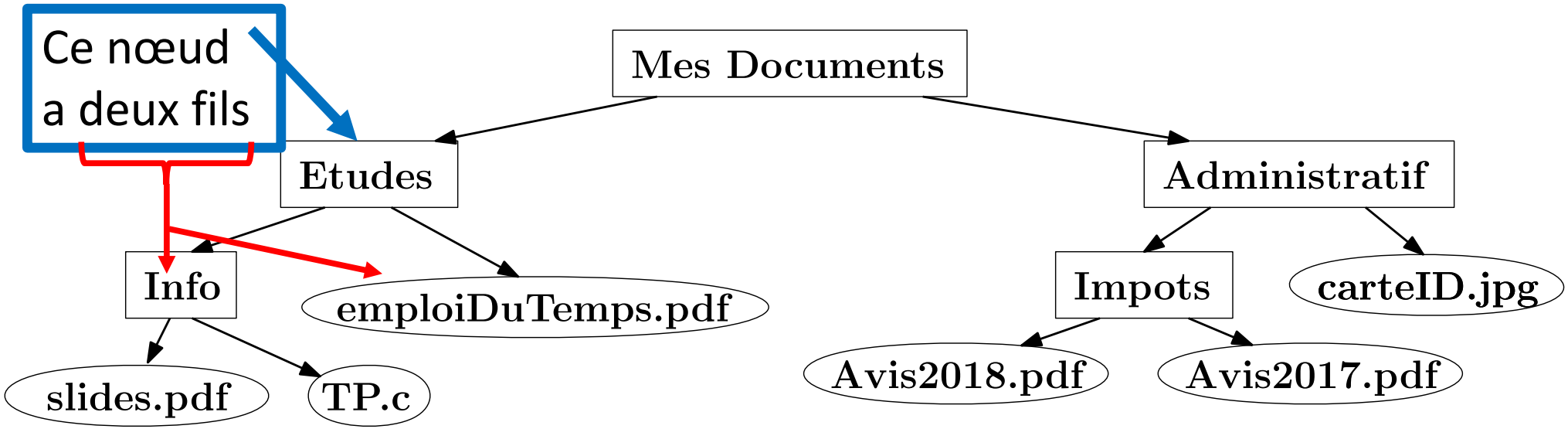
Il y a **11 nœuds** dans cet arbre.



1) Definition et vocabulaire

Vocabulaire:

- un noeud: une position dans l'arbre
- **les fils d'un nœud: les nœuds juste en dessous**
- le père d'un nœud: le nœud juste en-dessus

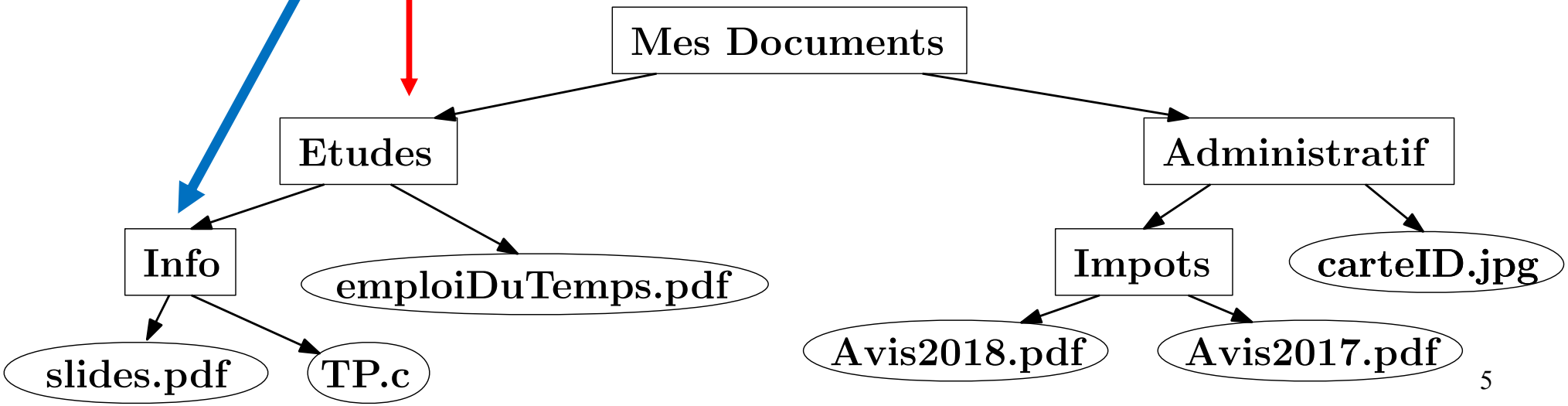


1) Definition et vocabulaire

Vocabulaire:

- un noeud: une position dans l'arbre
- les fils d'un nœud: les nœuds juste en dessous
- **le père d'un nœud: le nœud juste en-dessus**

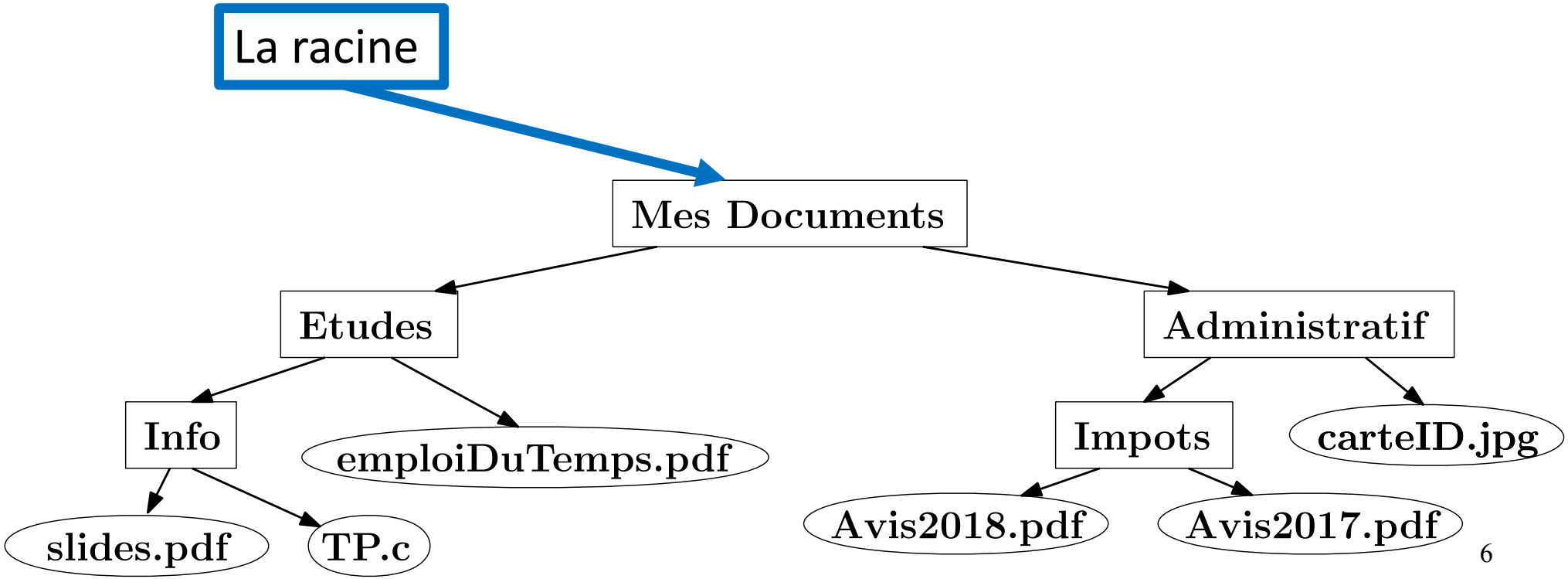
Le père du nœud dont la donnée est "Info" est le nœud dont la donnée est "Etudes"



1) Definition et vocabulaire

Vocabulaire:

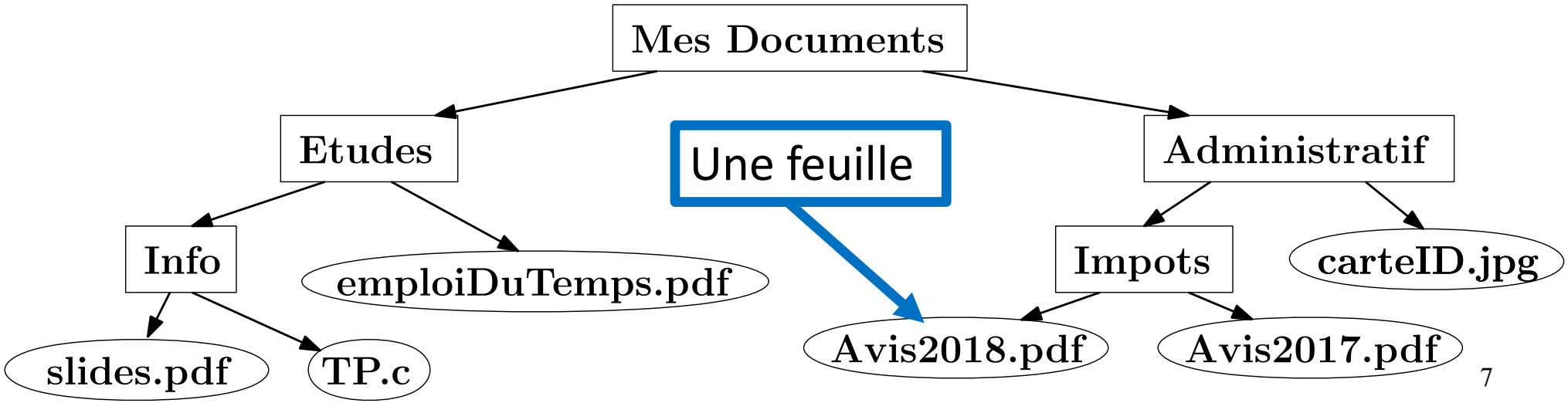
- **la racine est l'unique nœud sans père (tout en haut)**
- une feuille est un nœud qui n'a pas de fils
- un nœud interne est un nœud qui a au moins un fils



1) Definition et vocabulaire

Vocabulaire:

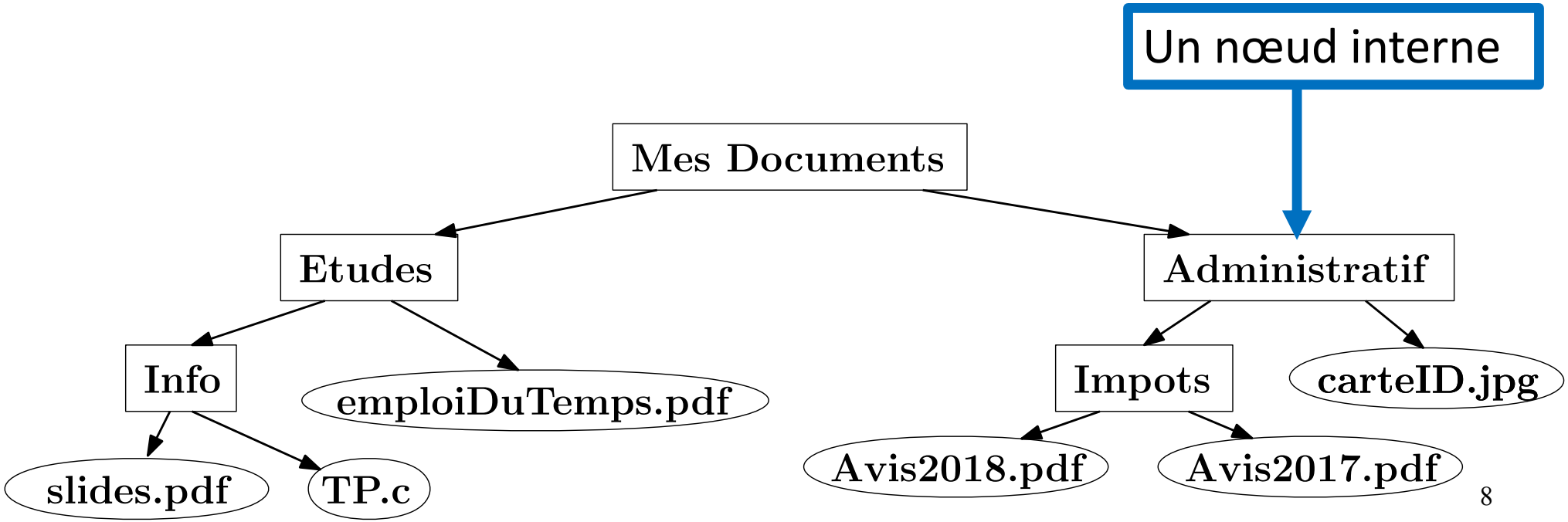
- la racine est l'unique nœud sans père (tout en haut)
- **une feuille est un nœud qui n'a pas de fils**
- un nœud interne est un nœud qui a au moins un fils



1) Definition et vocabulaire

Vocabulaire:

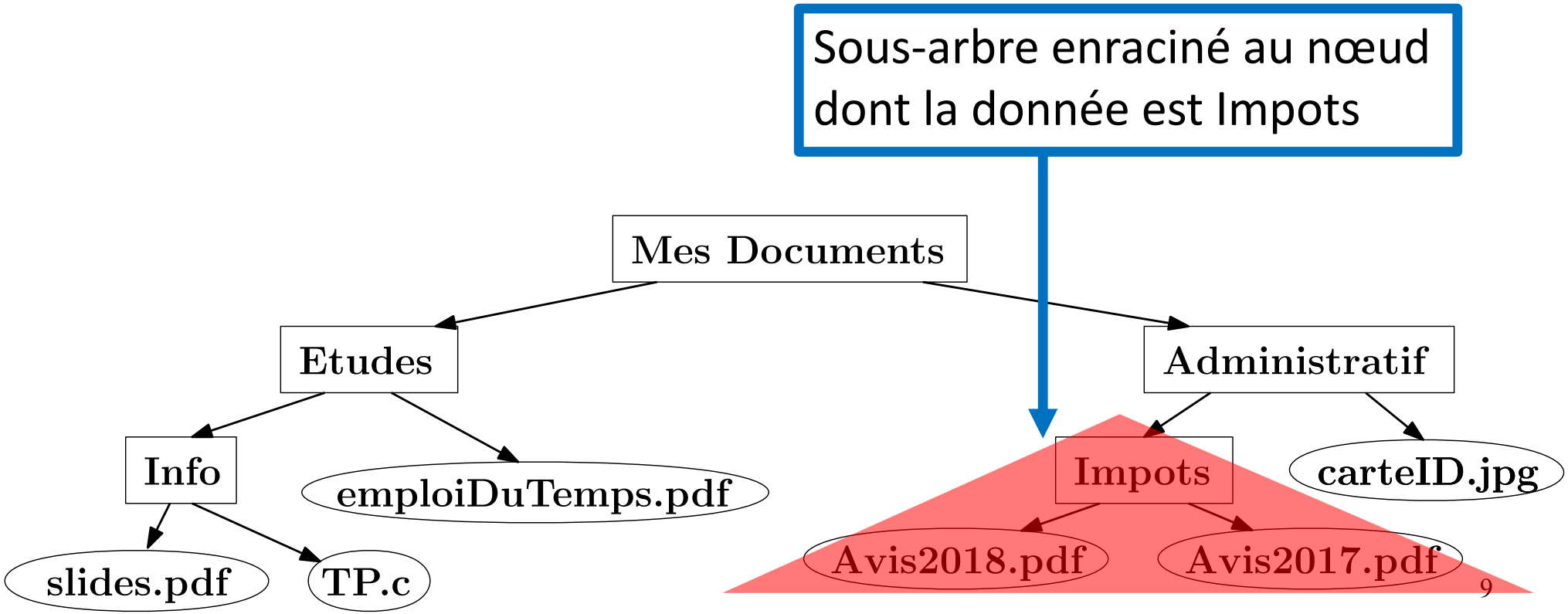
- la racine est l'unique nœud sans père (tout en haut)
- une feuille est un nœud qui n'a pas de fils
- **un nœud interne est un nœud qui a au moins un fils**



1) Definition et vocabulaire

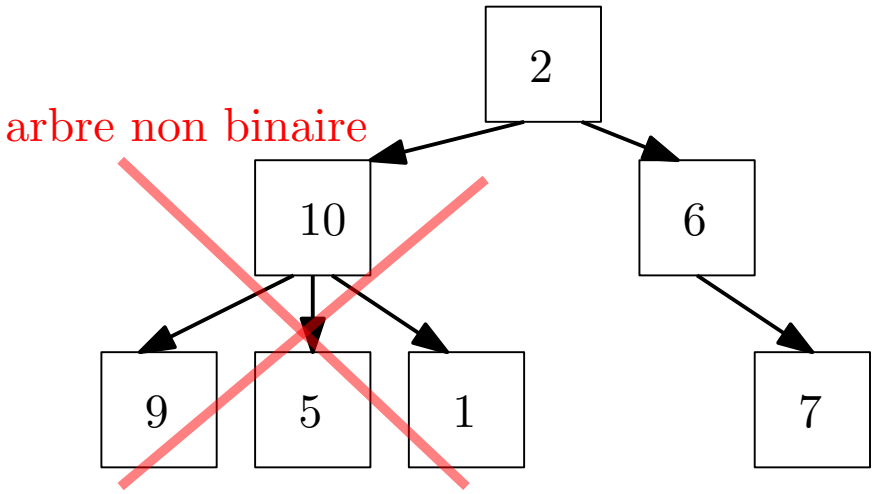
Vocabulaire:

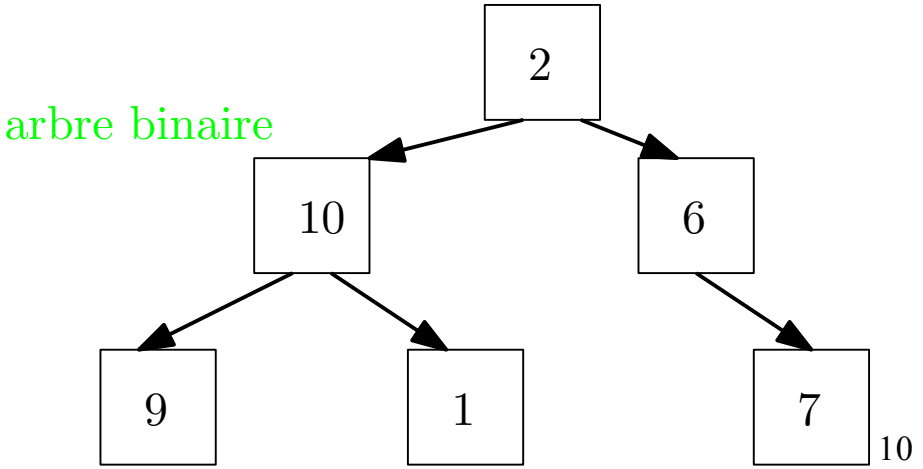
- si l'on coupe l'arbre au-dessus d'un certain nœud, on obtient le **sous-arbre enraciné** en ce nœud.



2) Arbres binaires

- Pour simplifier, on va se restreindre à un type d'arbre particulier: les **arbres binaires.**
- Cela signifie qu'**un nœud peut avoir 0, 1 ou 2 fils (mais pas plus)**
- Comme un noeud a au plus deux fils, on pourra parler du fils gauche et du fils droit.





2) Arbres binaires

- Pour simplifier la représentation, on supposera que chaque nœud a **exactement 2 fils** (un fils gauche et un fils droit), mais **ces nœuds auront le droit d'être vides**.
- On représentera un **nœud vide par la valeur spéciale -1.**
- Un nœud aura donc trois champs: data, filsG, filsD.

- *Note: il est assez courant de trouver des définitions d'arbres avec un quatrième champ:* pere

2) Arbres binaires

Spécifications du Type de Données Abstrait nœud:

- **Créer un noeud**

noeud N:= NewNoeud()

Les champs filsG et filsD seront ainsi automatiquement initialisés au nœud fictif -1 (ou NULL selon vos goûts).

- **Lire/écrire** dans le champ **data** d'un noeud (là où est stockée la véritable donnée): N.data où N doit être une variable de type noeud.

- **Lire/écrire** dans le champ **filsG ou filsD** d'un noeud: N.filsG:=N1

2) Arbres binaires

Spécifications du Type de Données Abstrait nœud:

- **Créer un noeud**

noeud N:= NewNoeud()

Complexité: 0 op.

Les champs filsG et filsD seront ainsi automatiquement initialisés au nœud fictif -1 (ou NULL selon vos goûts).

- **Lire/écrire** dans le champ **data** d'un noeud (là où est stockée la véritable donnée): N.data où N doit être une variable de type noeud.

Complexité: 1 op.

- **Lire/écrire** dans le champ **filsG** ou **filsD** d'un noeud:

N.filsG:=N1

Complexité: 1 op.

2) Arbres binaires

Exemple:

nœud N:=NewNoeud()

N.data:=2

noeud M:=NewNoeud()

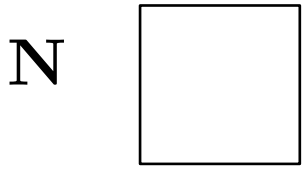
M.data:=10

N.filsG:=M

noeud P:=NewNoeud()

P.data:=6

N.filsD:=P



2) Arbres binaires

Exemple:

nœud N:=NewNoeud()

N.data:=2

noeud M:=NewNoeud()

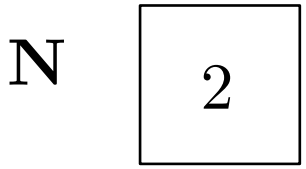
M.data:=10

N.filsG:=M

noeud P:=NewNoeud()

P.data:=6

N.filsD:=P



2) Arbres binaires

Exemple:

nœud N:=NewNoeud()

N.data:=2

noeud M:=NewNoeud()

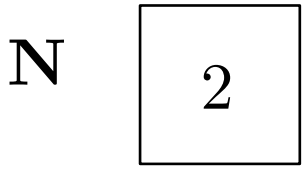
M.data:=10

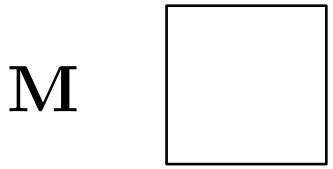
N.filsG:=M

noeud P:=NewNoeud()

P.data:=6

N.filsD:=P





2) Arbres binaires

Exemple:

nœud N:=NewNoeud()

N.data:=2

noeud M:=NewNoeud()

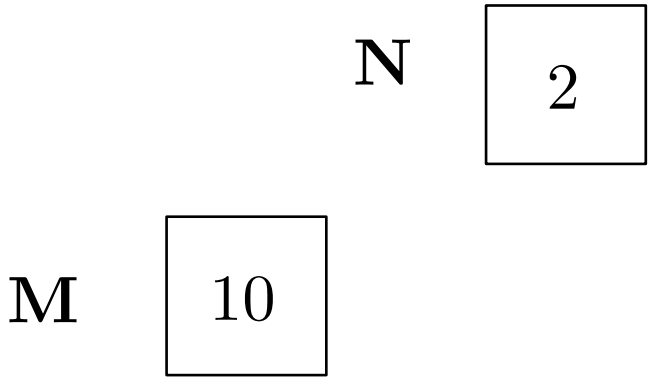
M.data:=10

N.filsG:=M

noeud P:=NewNoeud()

P.data:=6

N.filsD:=P



2) Arbres binaires

Exemple:

nœud N:=NewNoeud()

N.data:=2

noeud M:=NewNoeud()

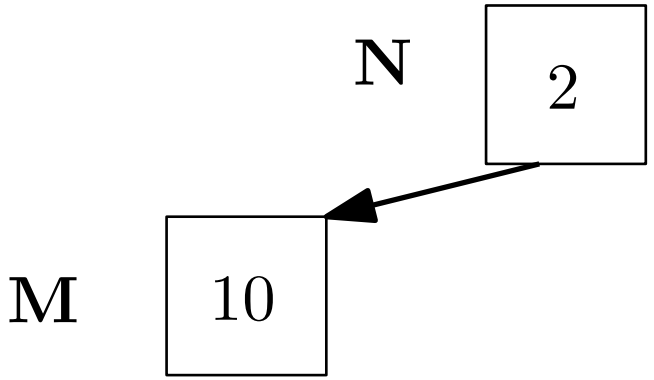
M.data:=10

N.filsG:=M

noeud P:=NewNoeud()

P.data:=6

N.filsD:=P



2) Arbres binaires

Exemple:

nœud N:=NewNoeud()

N.data:=2

noeud M:=NewNoeud()

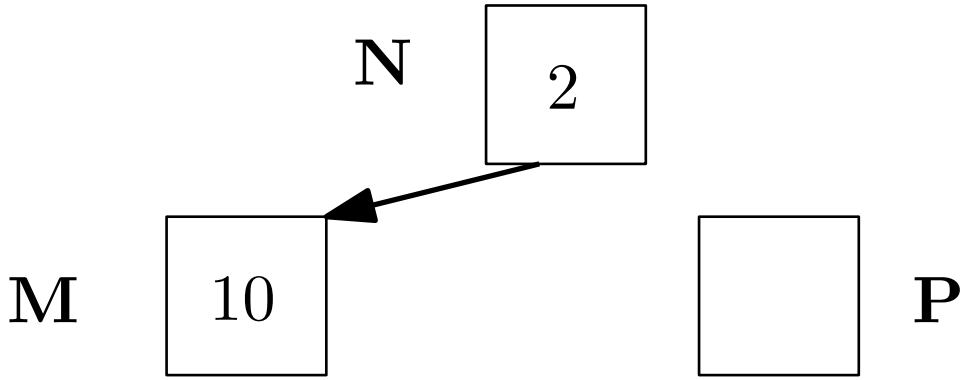
M.data:=10

N.filsG:=M

noeud P:=NewNoeud()

P.data:=6

N.filsD:=P



2) Arbres binaires

Exemple:

nœud N:=NewNoeud()

N.data:=2

noeud M:=NewNoeud()

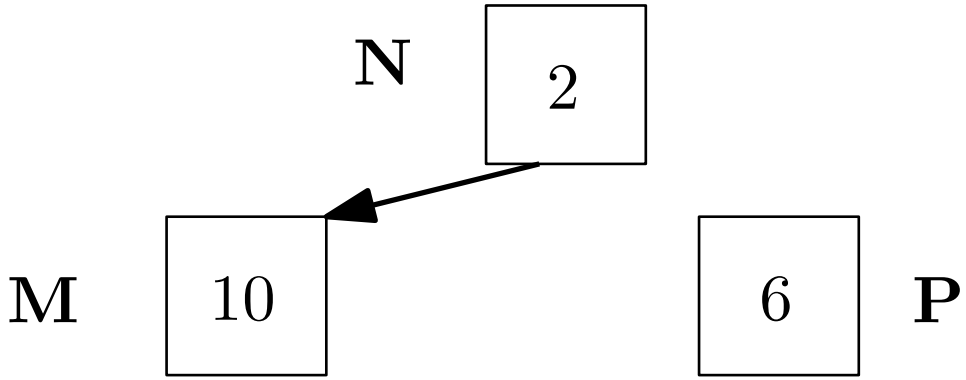
M.data:=10

N.filsG:=M

noeud P:=NewNoeud()

P.data:=6

N.filsD:=P



2) Arbres binaires

Exemple:

nœud N:=NewNoeud()

N.data:=2

noeud M:=NewNoeud()

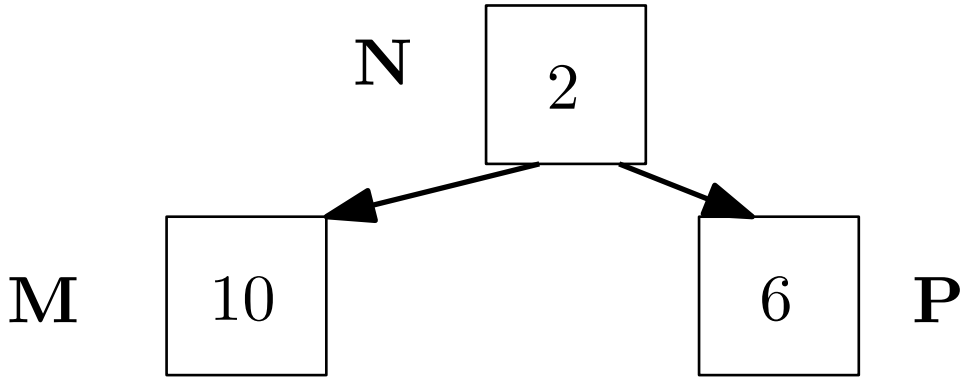
M.data:=10

N.filsG:=M

noeud P:=NewNoeud()

P.data:=6

N.filsD:=P



2) Arbres binaires

Fonctions que l'on va écrire:

- ajouter un fils gauche
- ajouter un fils droit
- est-ce-qu'un nœud est une feuille?
- combien de fils un nœud passé en argument a-t-il ?

2) Arbres binaires

```
Proc ajouteFilsG(nœud N, nombre valeur):
    Si N.filsG== -1: /* si N n'a pas de fils gauche pour l'instant*/
        nœud M:=NewNoeud()
        M.data:=valeur
        N.filsG:=M
Proc ajouteFilsD(nœud N, nombre valeur):
    Si N.filsD == -1: /* si N n'a pas de fils droit pour l'instant*/
        nœud M:=NewNoeud()
        M.data:=valeur
        N.filsD:=M
```

2) Arbres binaires

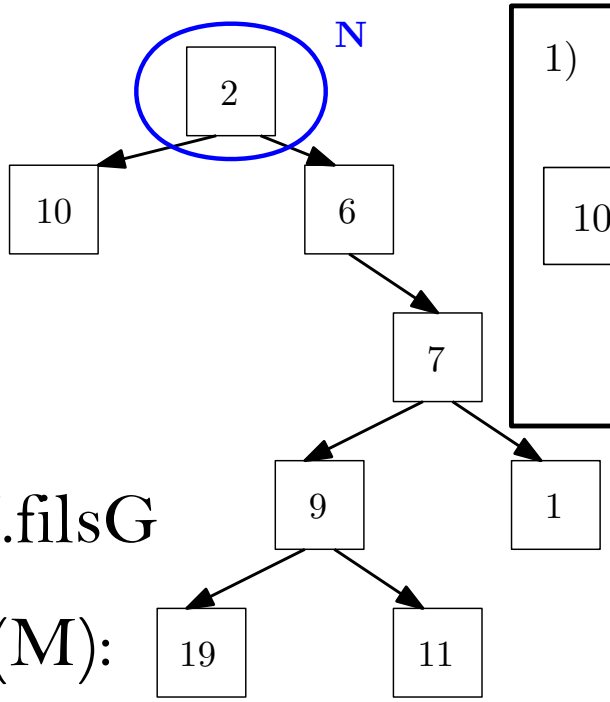
```
Fun bool estFeuille(nœud N):
    Si N.filsG== -1 et N.filsD==-1:
        Renvoyer Vrai
    Sinon:
        Renvoyer Faux
     /* ou tout simplement:
        Renvoyer (N.filsG==-1) et (N.filsD==-1) */
```

2) Arbres binaires

```
Fun int combienFils(nœud N):
    int compteur:=0
    Si N.filsG !=-1: // Si N a un fils gauche
        compteur++
    Si N.filsD!=-1: // Si N a un fils droit
        compteur++
    Retourner compteur
```

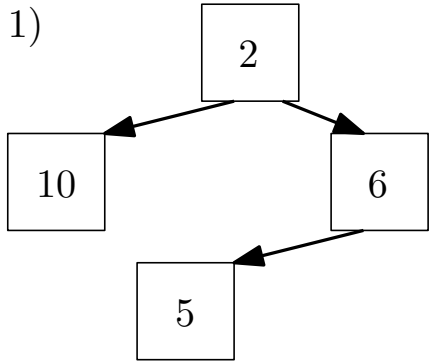
Arbres binaires : QCM

On suppose que le début de l'algorithme (la partie en pointillés) construit l'arbre suivant, dont le nœud N est la racine. Quel est l'état de la mémoire à la fin de l'exécution?




```
Début
    .....
    nœud M:=N.filsG
    Si estFeuille(M):
        ajouteFilsD(M, 5)
    int n:=combienFils(N.filsD.filsD)
    bool b:=estFeuille(M.filsD)
Fin
```

1)



n	2
b	Faux

2) Arbre inchangé

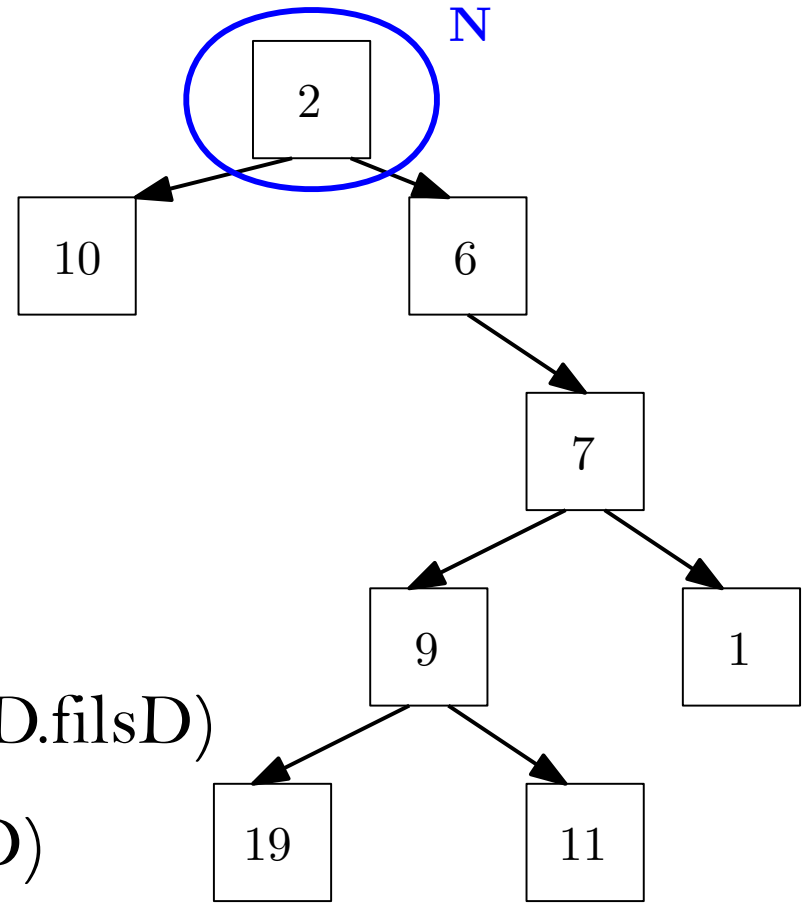
n	2
b	Vrai

3) Aucune des deux propositions

Arbres binaires : QCM

On suppose que le début de l'algorithme (la partie en pointillés) construit l'arbre suivant, dont le nœud N est la racine. Quel est l'état de la mémoire à la fin de l'exécution?

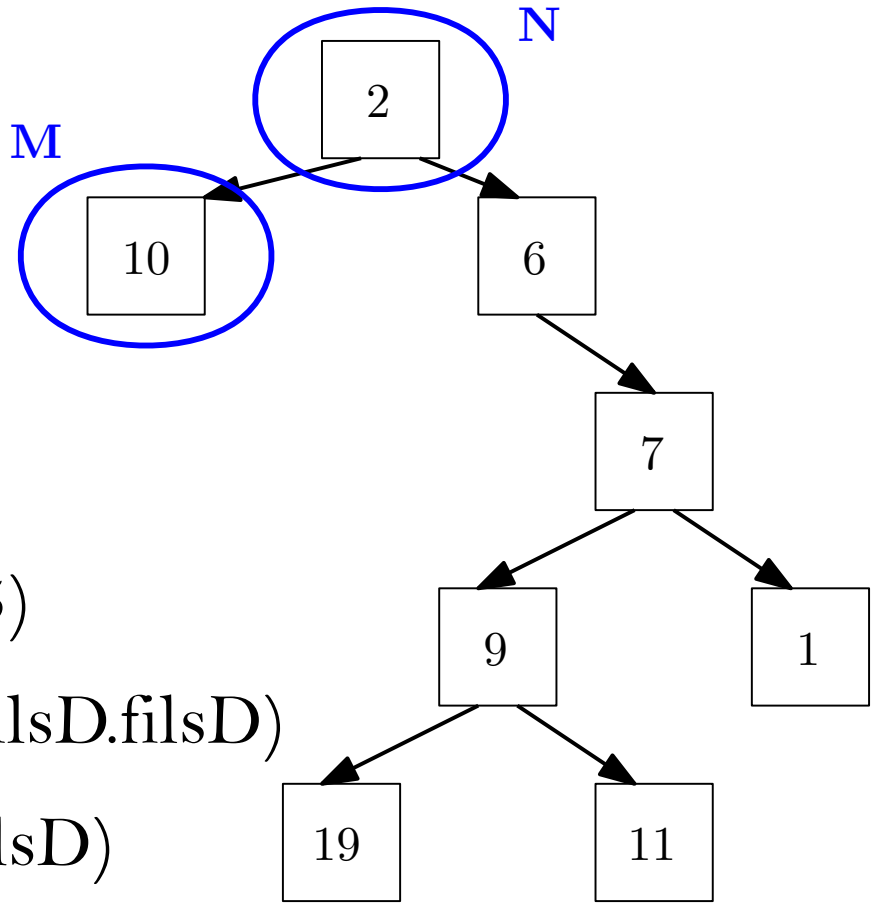
```
Début
    .....
    nœud M:=N.filsG
    Si estFeuille(M):
        ajouteFilsD(M, 5)
    int n:=combienFils(N.filsD.filsD)
    bool b:=estFeuille(M.filsD)
Fin
```



Arbres binaires : QCM

On suppose que le début de l'algorithme (la partie en pointillés) construit l'arbre suivant, dont le nœud N est la racine. Quel est l'état de la mémoire à la fin de l'exécution?

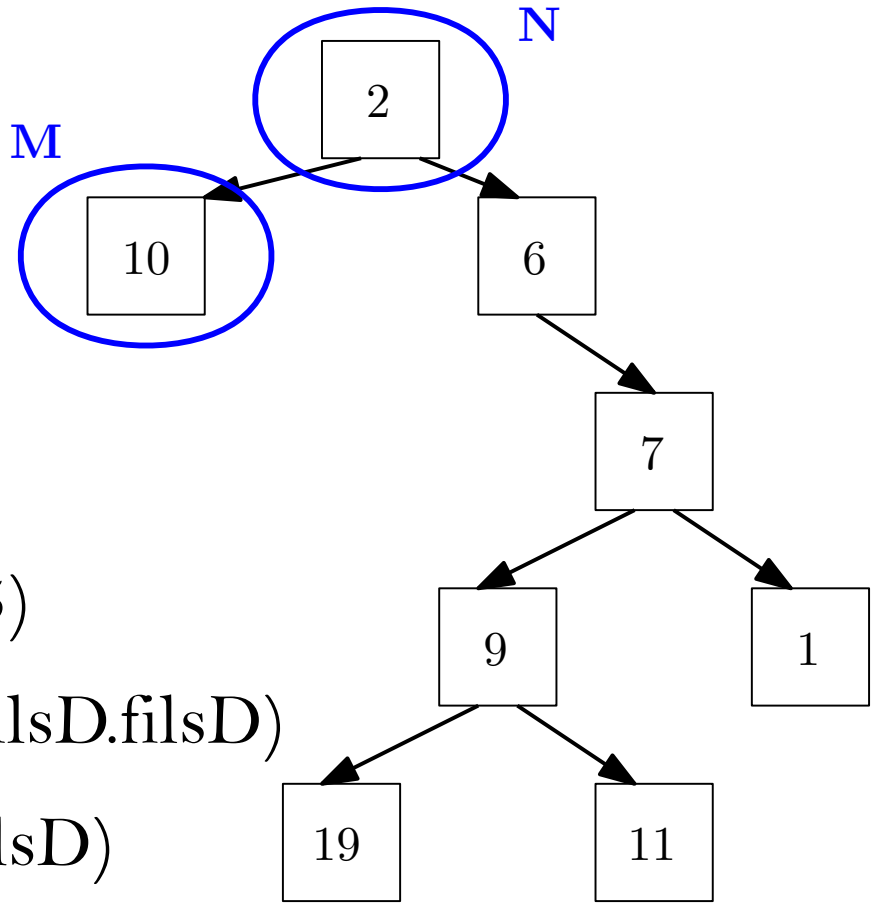
```
Début
    .....
    nœud M:=N.filsG
    Si estFeuille(M):
        ajouteFilsD(M, 5)
    int n:=combienFils(N.filsD.filsD)
    bool b:=estFeuille(M.filsD)
Fin
```



Arbres binaires : QCM

On suppose que le début de l'algorithme (la partie en pointillés) construit l'arbre suivant, dont le nœud N est la racine. Quel est l'état de la mémoire à la fin de l'exécution?

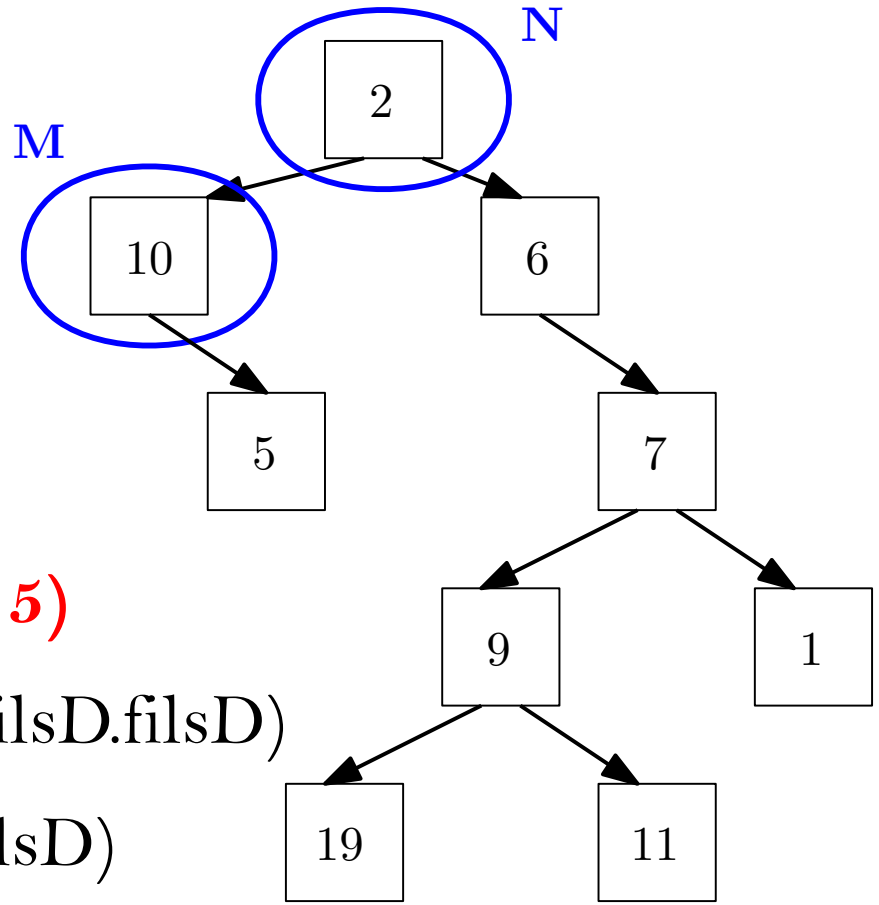
```
Début
    .....
    nœud M:=N.filsG
    Si estFeuille(M):
        ajouteFilsD(M, 5)
    int n:=combienFils(N.filsD.filsD)
    bool b:=estFeuille(M.filsD)
Fin
```



Arbres binaires : QCM

On suppose que le début de l'algorithme (la partie en pointillés) construit l'arbre suivant, dont le nœud N est la racine. Quel est l'état de la mémoire à la fin de l'exécution?

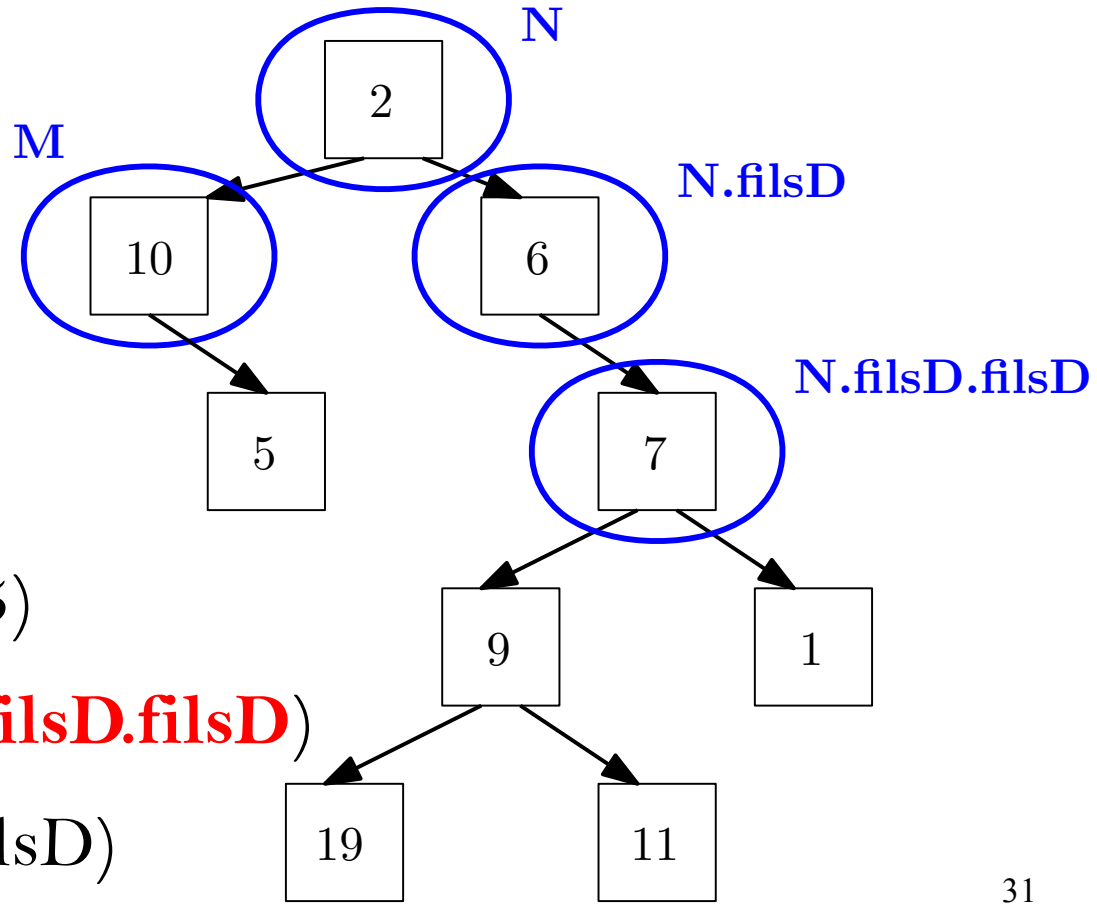
```
Début
    .....
    nœud M:=N.filsG
    Si estFeuille(M):
        ajouteFilsD(M, 5)
    int n:=combienFils(N.filsD.filsD)
    bool b:=estFeuille(M.filsD)
Fin
```



Arbres binaires : QCM

On suppose que le début de l'algorithme (la partie en pointillés) construit l'arbre suivant, dont le nœud N est la racine. Quel est l'état de la mémoire à la fin de l'exécution?

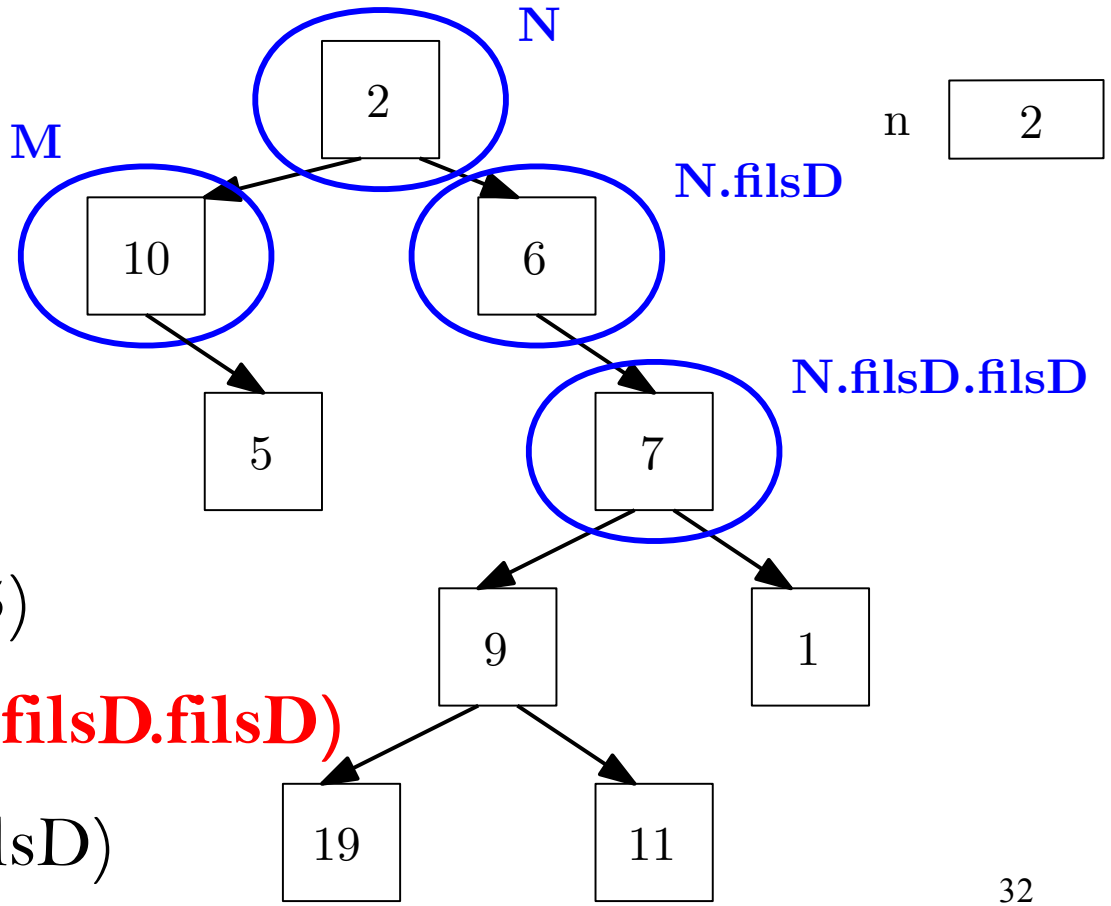
```
Début
    .....
    nœud M:=N.filsG
    Si estFeuille(M):
        ajouteFilsD(M, 5)
    int n:=combienFils(N.filsD.filsD)
    bool b:=estFeuille(M.filsD)
Fin
```



Arbres binaires : QCM

On suppose que le début de l'algorithme (la partie en pointillés) construit l'arbre suivant, dont le nœud N est la racine. Quel est l'état de la mémoire à la fin de l'exécution?

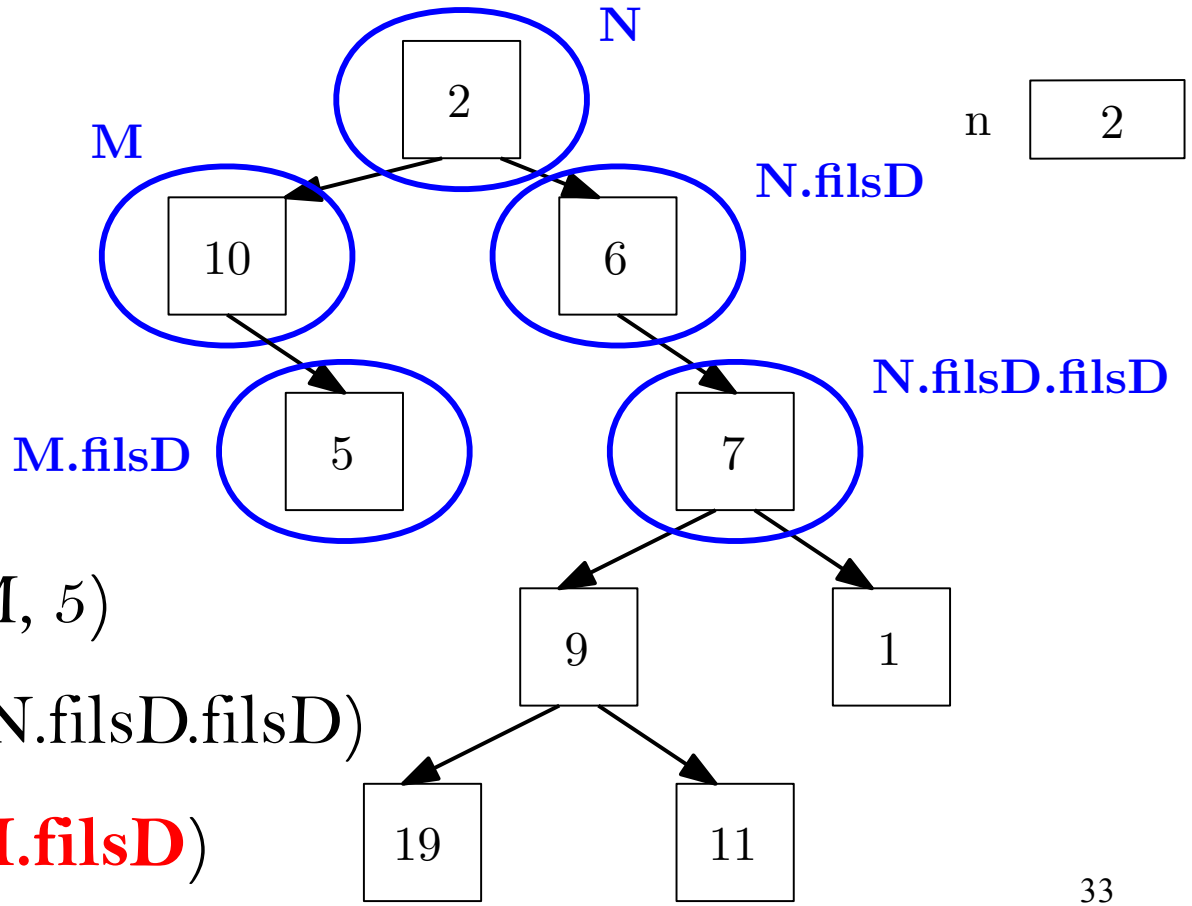
```
Début
    .....
    nœud M:=N.filsG
    Si estFeuille(M):
        ajouteFilsD(M, 5)
    int n:=combienFils(N.filsD.filsD)
    bool b:=estFeuille(M.filsD)
Fin
```



Arbres binaires : QCM

On suppose que le début de l'algorithme (la partie en pointillés) construit l'arbre suivant, dont le nœud N est la racine. Quel est l'état de la mémoire à la fin de l'exécution?

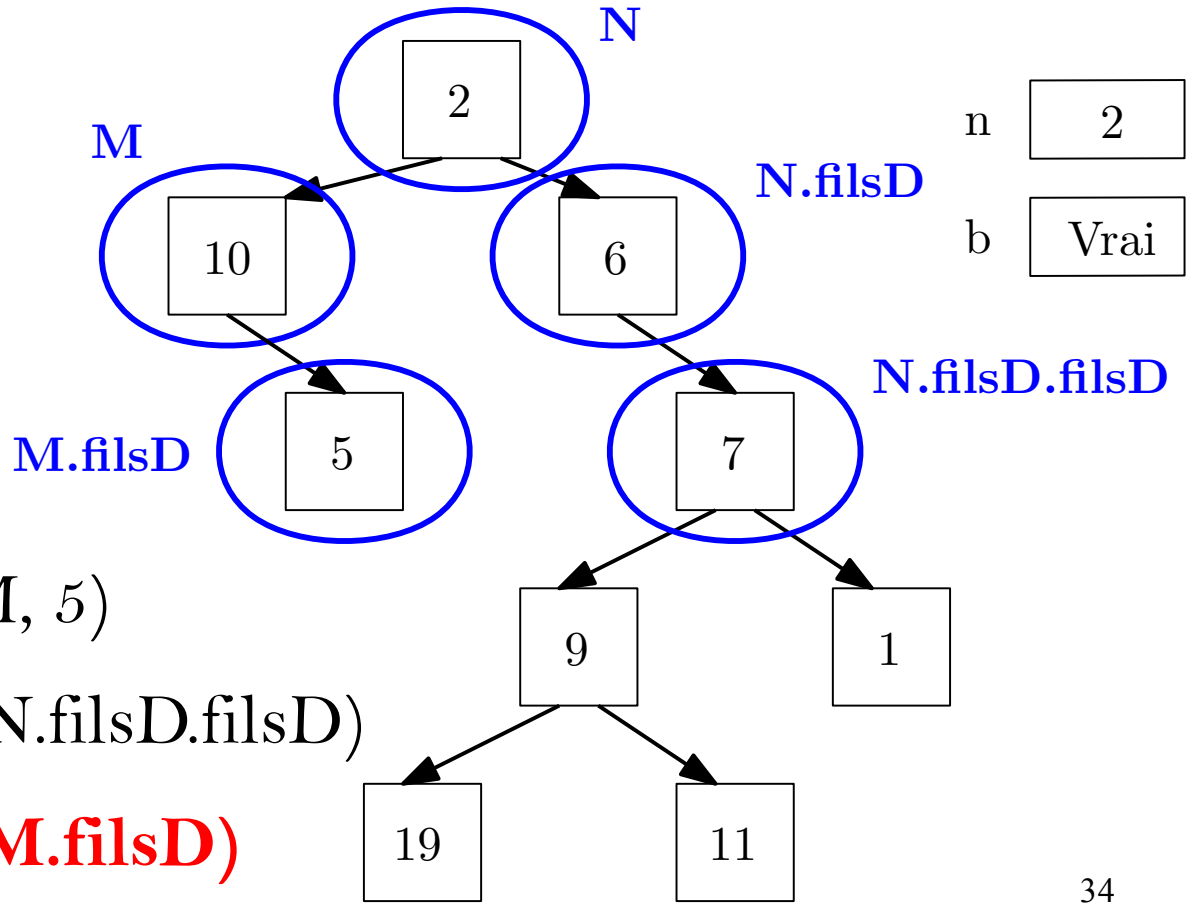
```
Début
    .....
    nœud M:=N.filsG
    Si estFeuille(M):
        ajouteFilsD(M, 5)
    int n:=combienFils(N.filsD.filsD)
    bool b:=estFeuille(M.filsD)
Fin
```



Arbres binaires : QCM

On suppose que le début de l'algorithme (la partie en pointillés) construit l'arbre suivant, dont le nœud N est la racine. Quel est l'état de la mémoire à la fin de l'exécution?

```
Début
    .....
    nœud M:=N.filsG
    Si estFeuille(M):
        ajouteFilsD(M, 5)
    int n:=combienFils(N.filsD.filsD)
    bool b:=estFeuille(M.filsD)
Fin
```



3) Parcours d'arbres

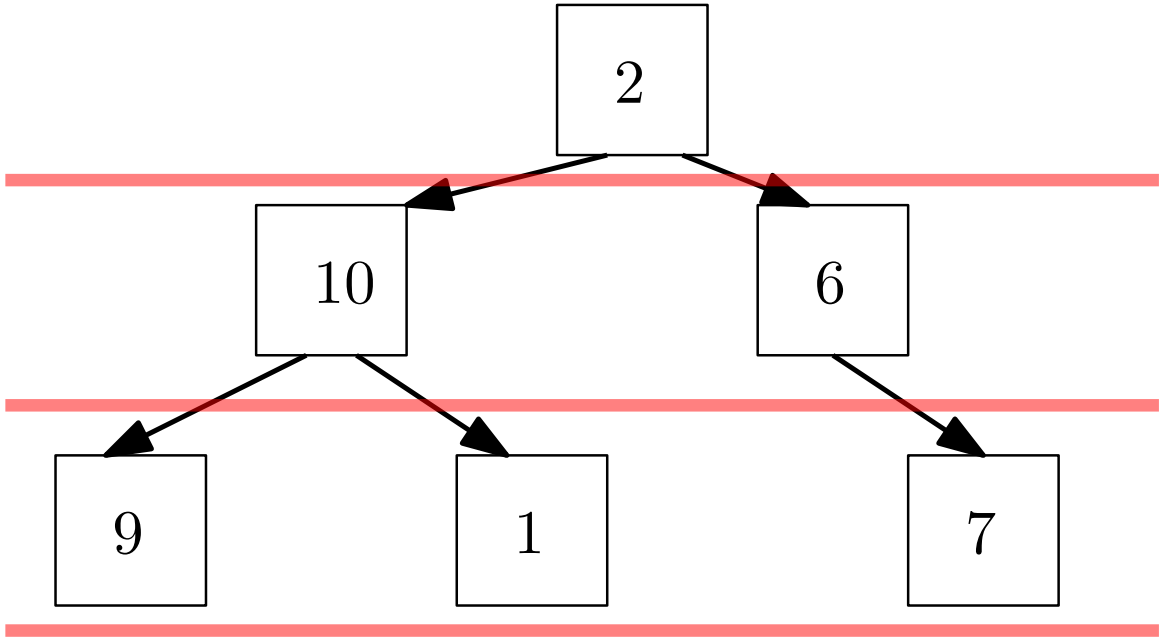
Il y a plusieurs façons de parcourir tous les nœuds d'un arbre.

Deux sont très importantes:

- **Parcours en largeur: on parcourt "étage par étage"**
- **Parcours en profondeur: on explore toute la branche jusqu'en bas à chaque fois.**

3) Parcours d'arbres

- **Parcours en largeur: on parcourt "étage par étage", de** gauche à droite à l'intérieur d'un étage.
- *Exemple:*

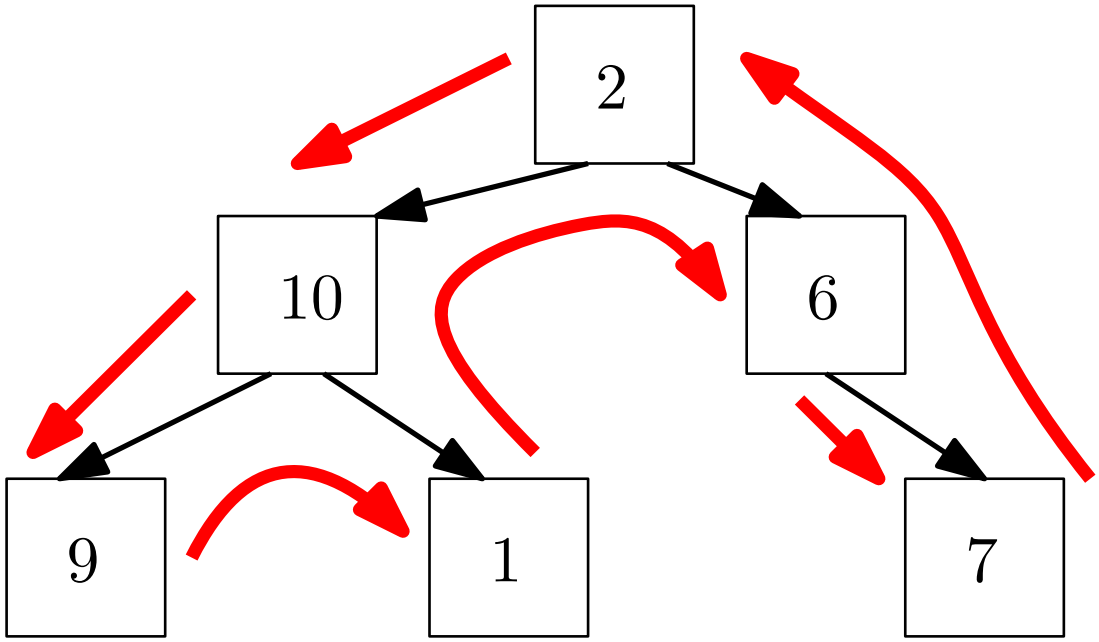


- Le **parcours en largeur** visite les nœuds dans l'ordre suivant: **2, 10, 6, 9, 1, 7.**

3) Parcours d'arbres

- **Parcours en profondeur:** on explore toute la branche **jusqu'en bas à chaque fois**, et quand on est bloqués on **remonte seulement d'un cran**, tant qu'il reste des nœuds à visiter dans cette branche.

- *Exemple:*



- Le **parcours en profondeur** visite les nœuds dans l'ordre suivant: **2, 10, 9, 1, 6, 7.**

3) Parcours d'arbres

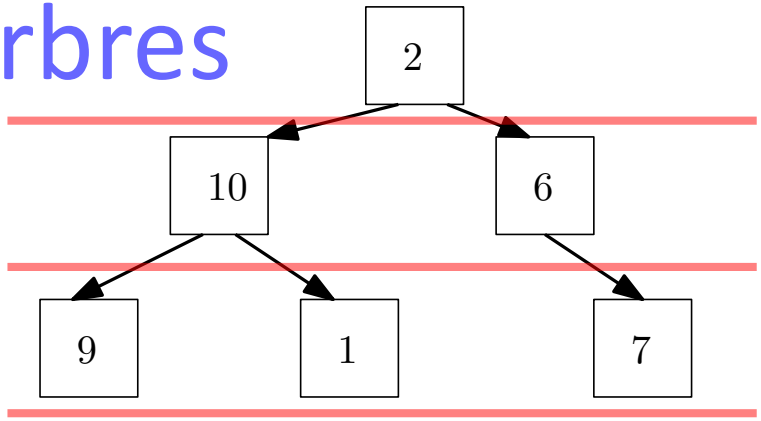
- On va écrire les algorithmes correspondant à ces parcours.
- **Le parcours en largeur s'écrit avec une file**

("First in, First out")

- **Le parcours en profondeur s'écrit avec une pile**

("Last in, first out")

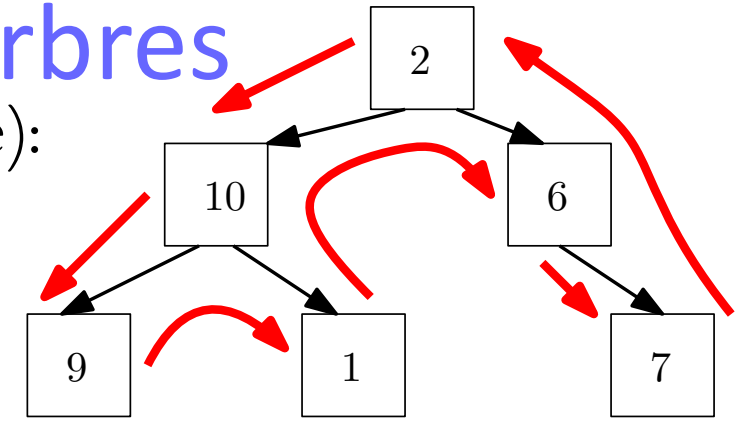
3) Parcours d'arbres



Parcours voulu: **2, 10, 6, 9, 1, 7.**

```
Proc parcoursEnLargeur(nœud racine):
    file F:=NewFile()
    enfiler(F, racine)
    nœud N
    Tant que non(estVide(F)):
        N:=valeurDebut(F)
        print(N.data)
        defiler(F)
        Si N.filsG!= -1: /* si N a un fils gauche */
            enfiler(F, N.filsG)
        Si N.filsD!=-1:
            enfiler(F, N.filsD)
```

3) Parcours d'arbres

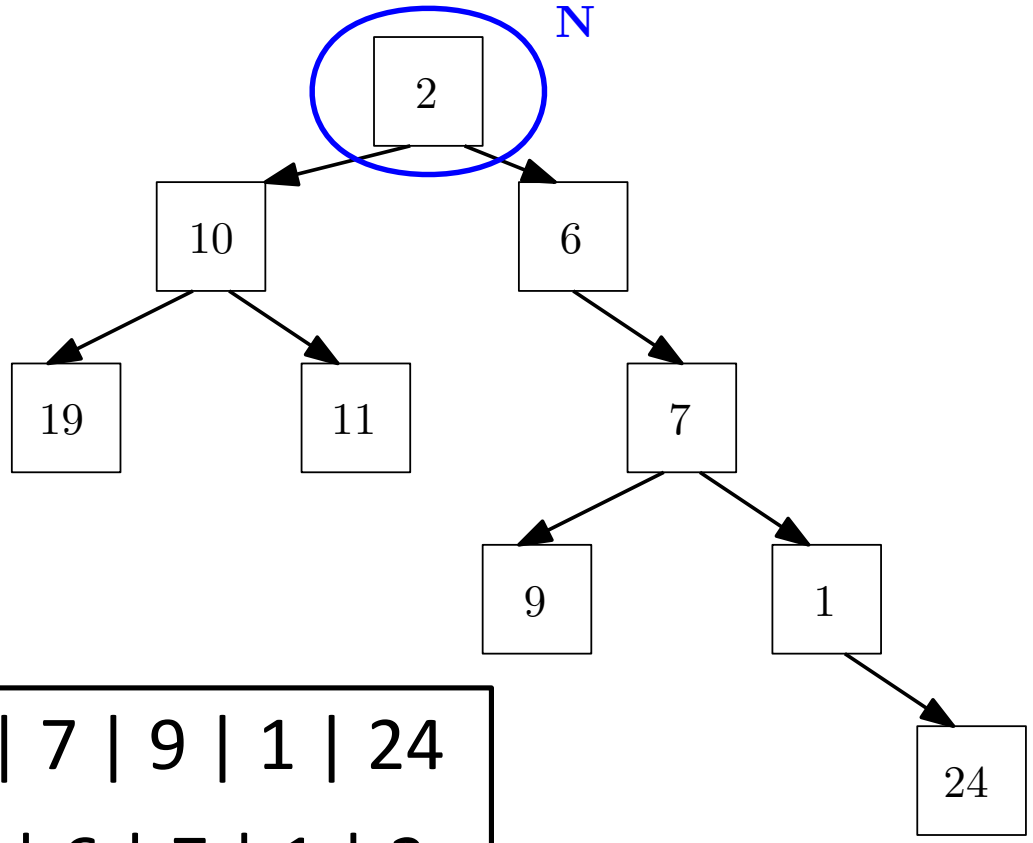


Parcours voulu: **2, 10, 9, 1, 6, 7.**

```
Proc parcoursEnProfondeur(nœud racine):
    pile P:=NewPile()
    empiler(P, racine)
    nœud N
    Tant que non(estVide(P)):
        N:=valeurTop(P)
        print(N.data)
        depiler(P)
        Si N.filsD!= -1: /* si N a un fils droit */
            empiler(P, N.filsD)
        Si N.filsG!=-1:
            empiler(P, N.filsG)
```

3) Parcours d'arbres: QCM 1/2

Quel affichage est provoqué par l'appel **parcoursEnLargeur(N)**?



1) 2 |10 | 19 | 11 | 6 | 7 | 9 | 1 | 24

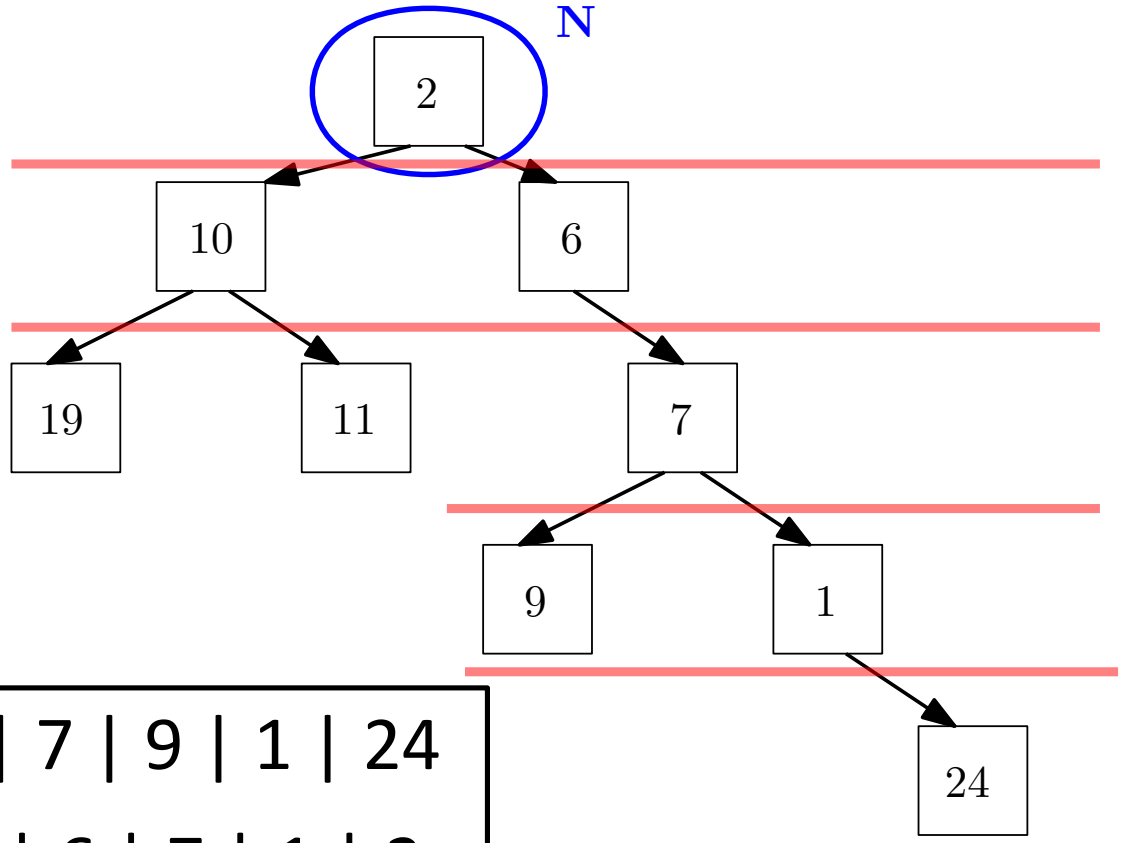
2) 19 |11 | 9 | 24 | 10 | 6 | 7 | 1 | 2

3) 2 | 10 |6 | 19 | 11 | 7 | 9 | 1 | 24

4) 24 | 1 | 9 | 7 | 11 | 19 | 6 | 10 | 2

3) Parcours d'arbres: QCM 1/2

Quel affichage est provoqué par l'appel **parcoursEnLargeur(N)**?



1) 2 |10 | 19 | 11 | 6 | 7 | 9 | 1 | 24

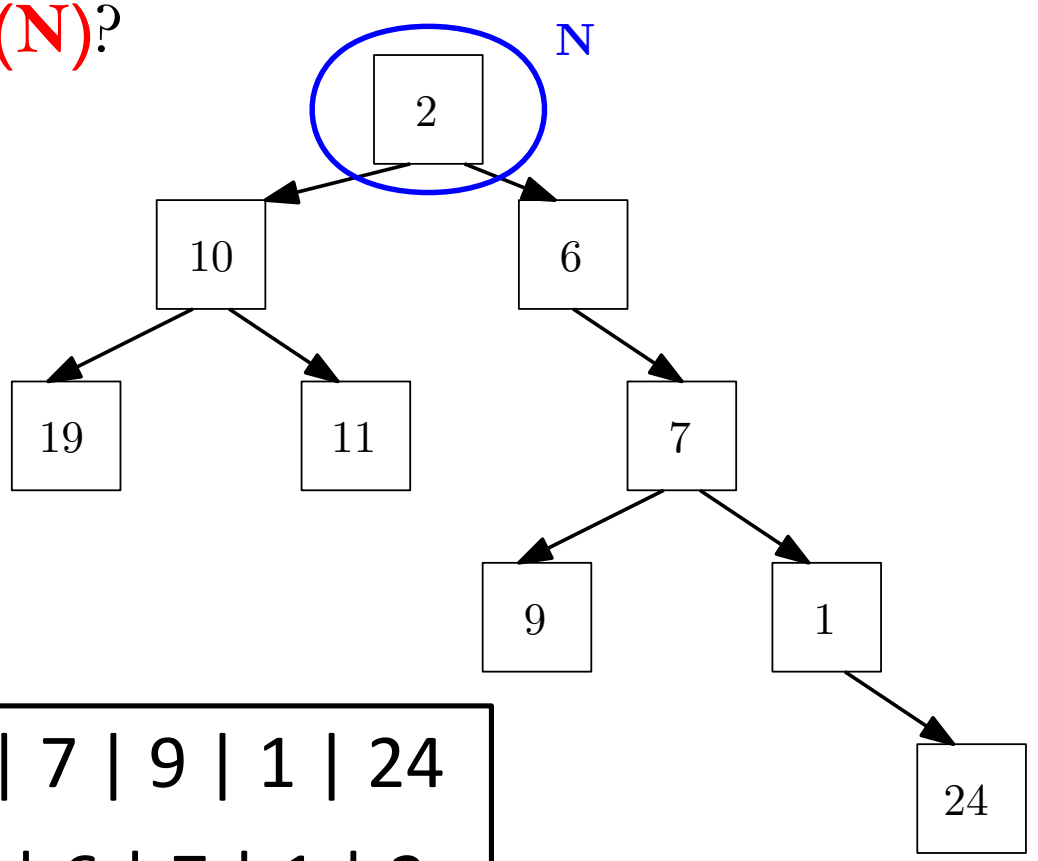
2) 19 |11 | 9 | 24 | 10 | 6 | 7 | 1 | 2

3) 2 | 10 |6 | 19 | 11 | 7 | 9 | 1 | 24

4) 24 | 1 | 9 | 7 | 11 | 19 | 6 | 10 | 2

3) Parcours d'arbres: QCM 2/2

Quel affichage est provoqué par l'appel **parcoursEnProfondeur(N)**?



1) 2 |10 | 19 | 11 | 6 | 7 | 9 | 1 | 24

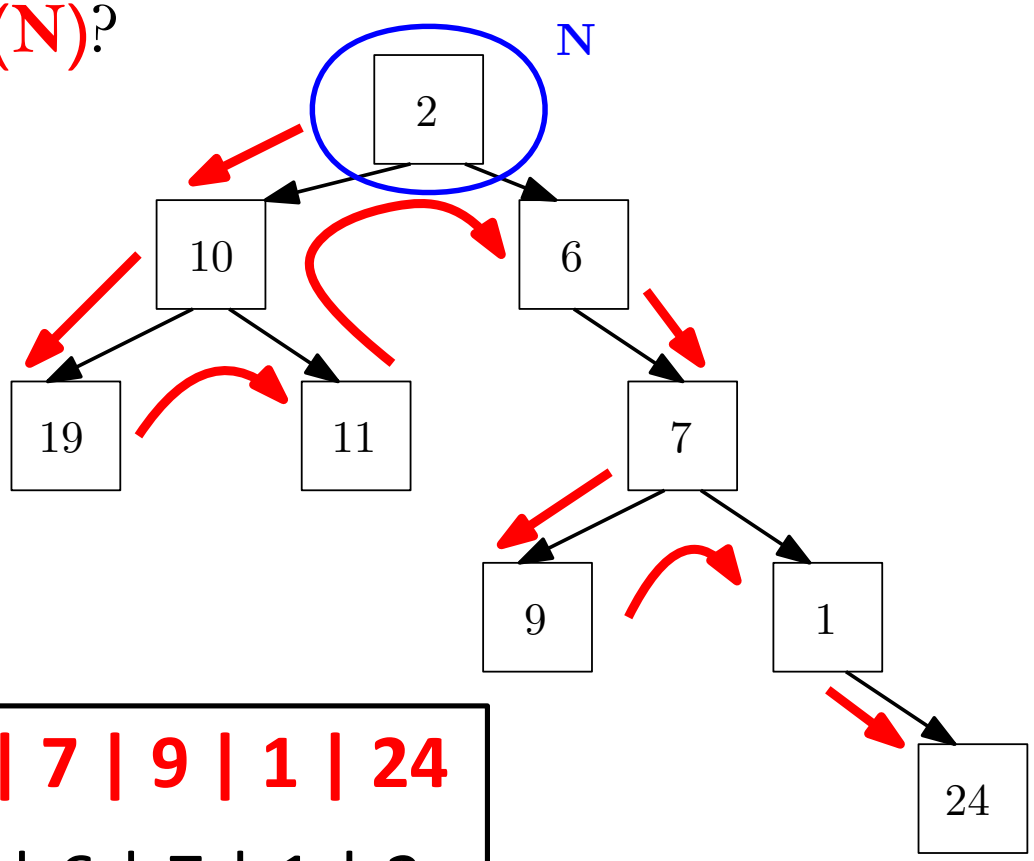
2) 19 |11 | 9 | 24 | 10 | 6 | 7 | 1 | 2

3) 2 | 10 |6 | 19 | 11 | 7 | 9 | 1 | 24

4) 24 | 1 | 9 | 7 | 11 | 19 | 6 | 10 | 2

3) Parcours d'arbres: QCM 2/2

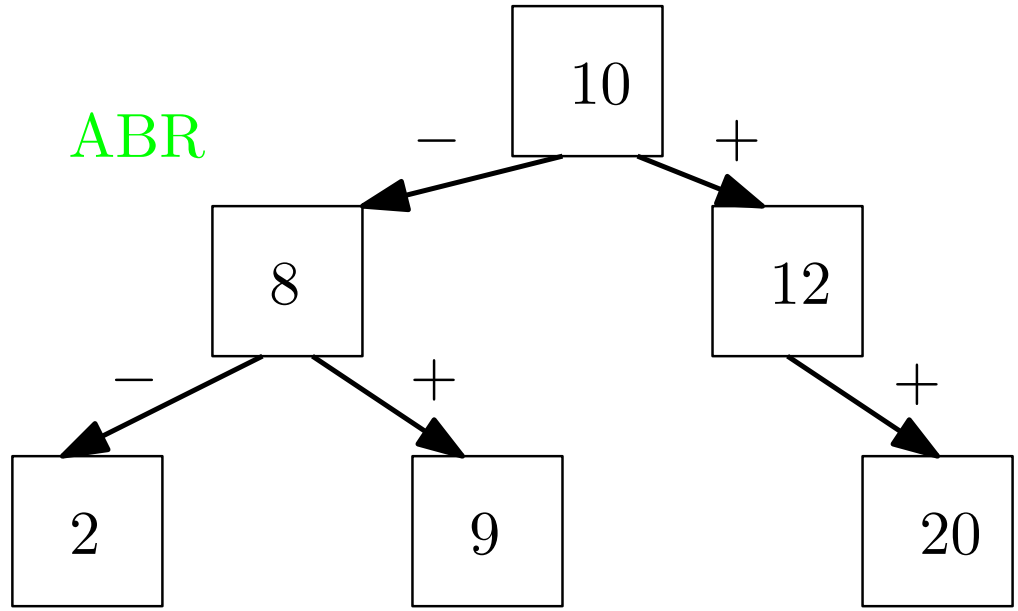
Quel affichage est provoqué par l'appel **parcoursEnProfondeur(N)**?



1) **2 |10 | 19 | 11 | 6 | 7 | 9 | 1 | 24**
2) 19 |11 | 9 | 24 | 10 | 6 | 7 | 1 | 2
3) 2 | 10 |6 | 19 | 11 | 7 | 9 | 1 | 24
4) 24 | 1 | 9 | 7 | 11 | 19 | 6 | 10 | 2

4) Arbres binaires de recherche (ABR)

Un **arbre binaire de recherche** est un arbre binaire avec une propriété particulière, qui permet de stocker des nombres de façon à pouvoir insérer ou rechercher une valeur de manière efficace.



4) Arbres binaires de recherche (ABR)

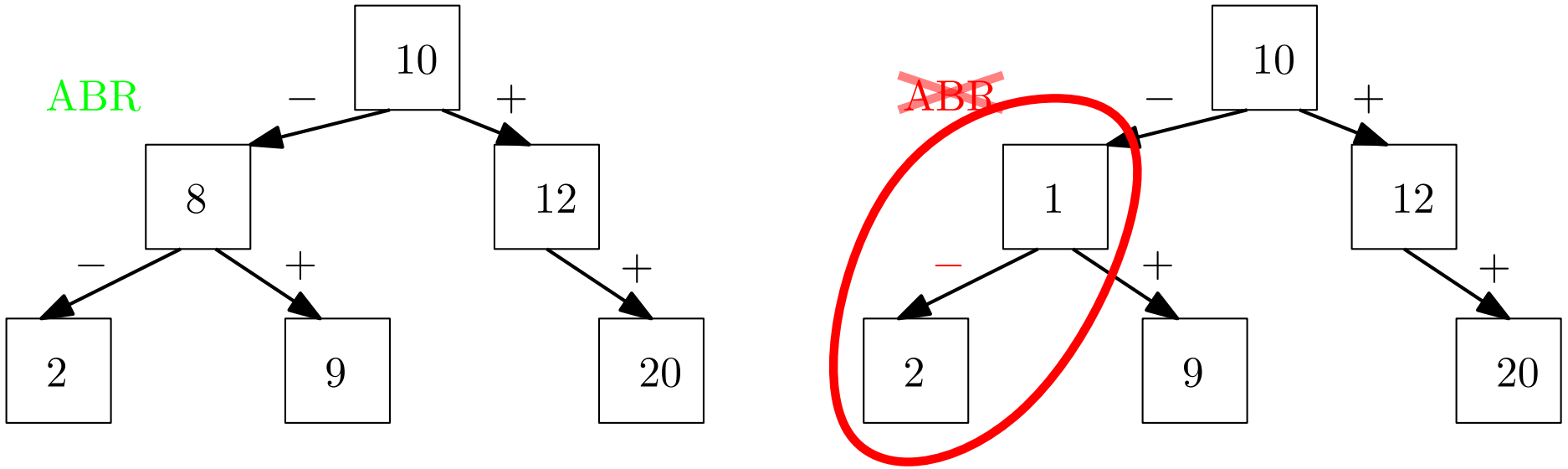
Definition: Un arbre binaire est un **arbre binaire de recherche (ABR)** si tous les nœuds ont une partie data remplie avec un nombre[1], et si pour chacun de ses nœuds N, les deux propositions suivantes sont vraies:

- toutes les données dans le **sous-arbre enraciné dans le fils gauche de N sont inférieures ou égales à $\mathrm{N.data}$**
- toutes les données dans le **sous-arbre enraciné dans le fils droit de N sont supérieures ou égales à $\mathrm{N.data}$**

[1]Plus exactement: avec un élément d'un ensemble totalement ordonné. On pourrait donc avoir des caractères ordonnés grâce à l'ordre ASCII.

4) Arbres binaires de recherche (ABR)

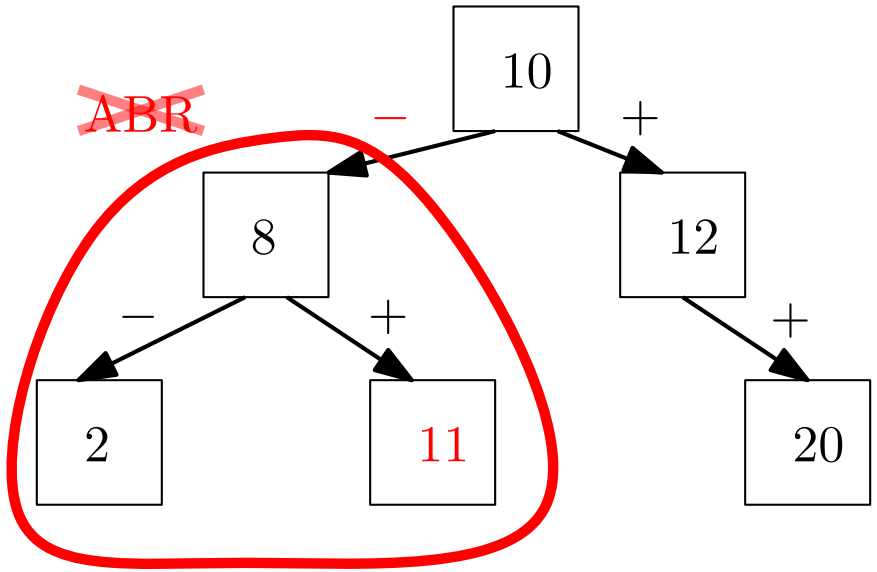
Exemples:



4) Arbres binaires de recherche (ABR)

Exemples:

Attention! Pour vérifier si un arbre binaire est un ABR, **il ne suffit pas de regarder les fils directs de chaque nœud**: il faut considérer **tout le sous-arbre gauche** et **tout le sous-arbre droit!**



Le sous-arbre gauche de 10 ne doit contenir que des données inférieures ou égales à 10, mais il contient 11!

→ Ce n'est pas un ABR.

4) Arbres binaires de recherche (ABR)

Opérations les plus importantes sur les ABR:

- **rechercher si un nombre est présent** comme donnée dans l'arbre:

  Fun bool contient(nœud racine, nombre cible)

- **insérer un nombre** en conservant les propriétés d'ABR:

  Proc ajoute(nœud racine, nombre valeur)

4) Arbres binaires de recherche (ABR)

Rechercher si un nombre cible est présent comme donnée dans l'arbre: c'est simple!

- **Soit la cible est égale à la donnée de la racine** → *on a trouvé*
- **Soit elle est plus petite** → *on doit chercher dans le sous-arbre gauche, s'il n'est pas vide*
- **Soit elle est plus grande** → *on doit chercher dans le sous-arbre droit, s'il n'est pas vide.*

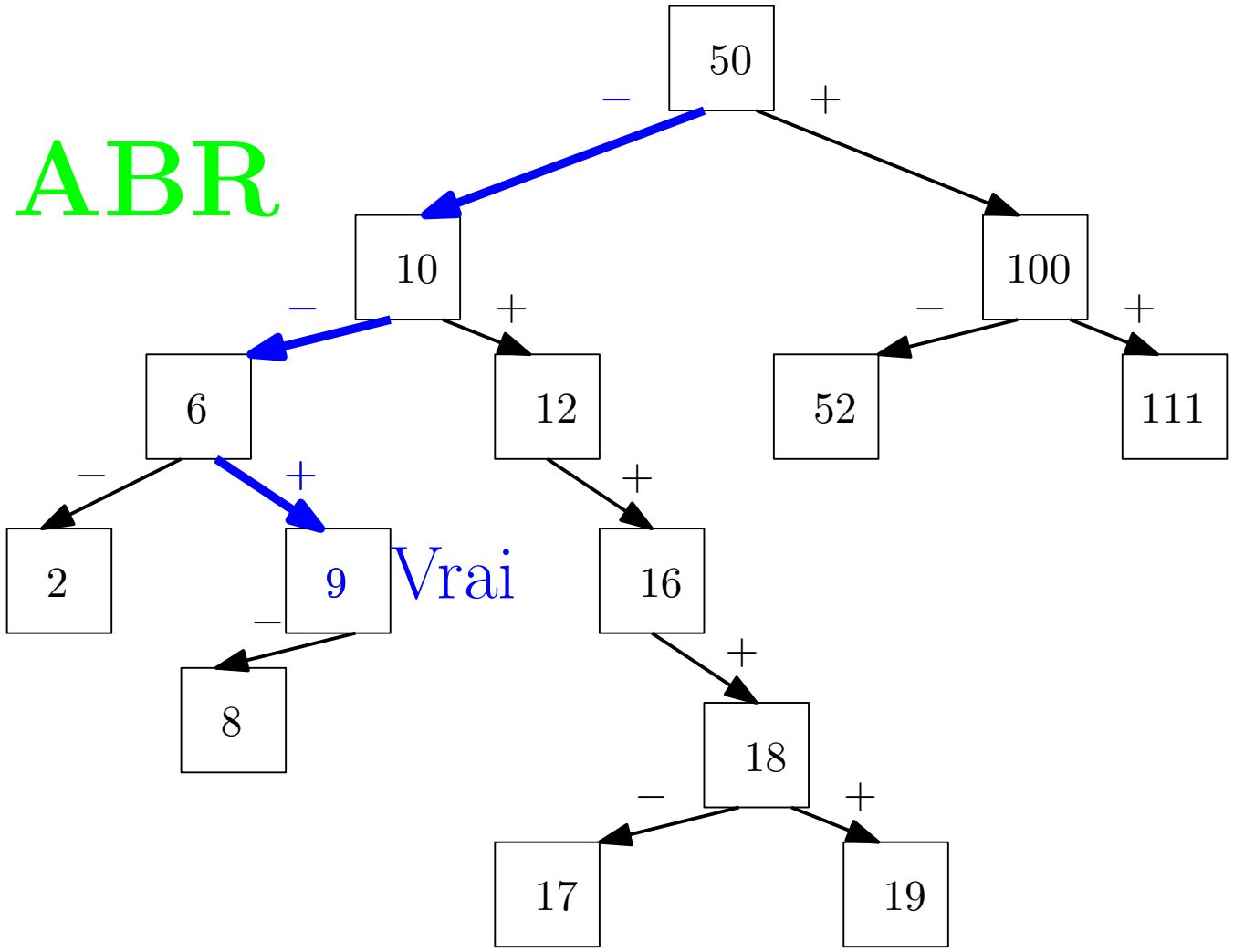
4) Arbres binaires de recherche (ABR)

On suppose que N est la racine d'un ABR.

```
Fun bool contient(nœud N, nombre cible):
    Tant que N != -1:
        Si N.data==cible:
            Retourner Vrai
        SinonSi N.data>cible:
            N:=N.filsG
        Sinon:
            N:=N.filsD
    /* Si on arrive ici, c'est qu'on finit par tomber sur un arbre vide
    sans avoir trouvé la valeur cible */
    Retourner Faux
```

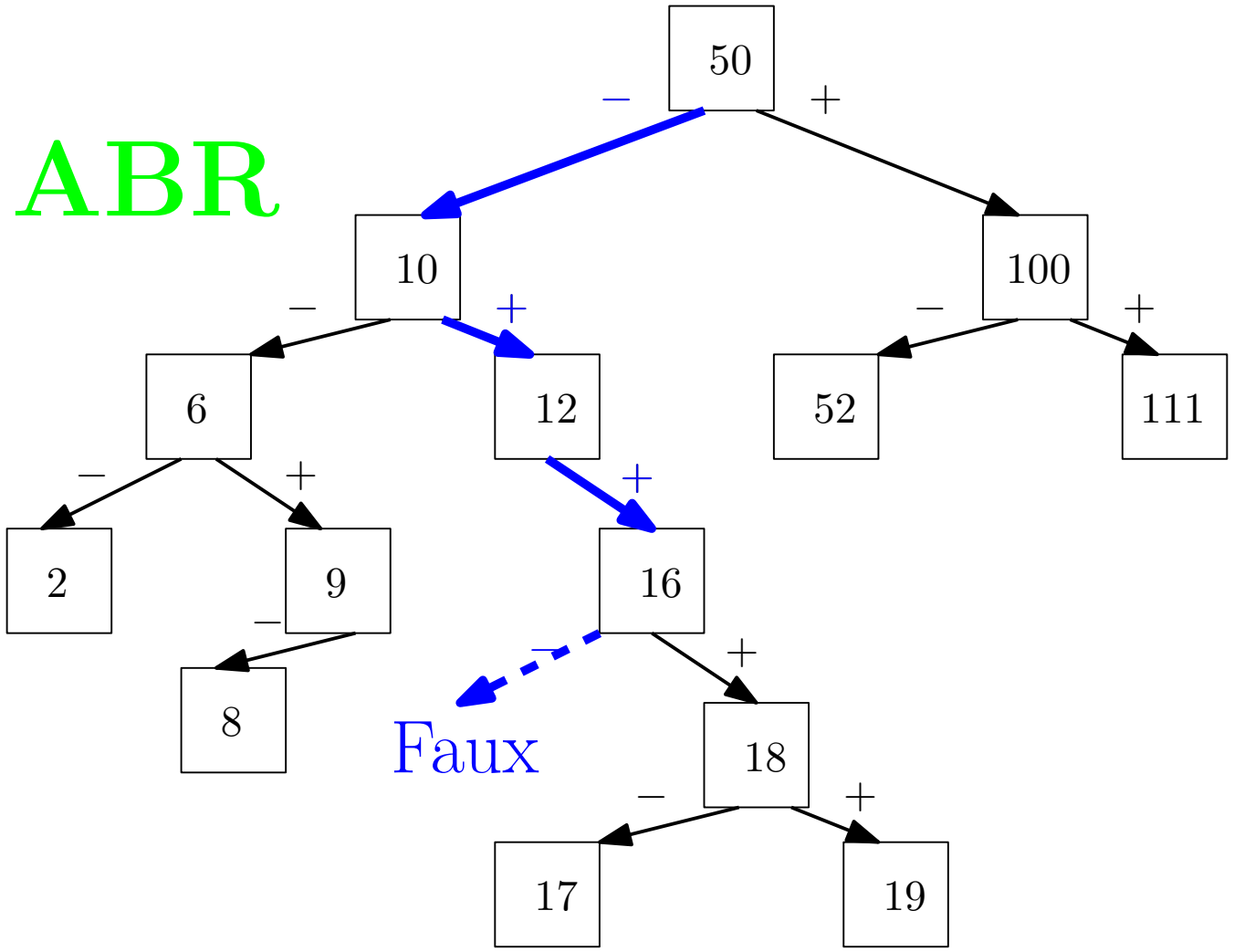
4) Arbres binaires de recherche (ABR)

Exemple: nœuds parcourus par l'appel **contient(N, 9)**, en supposant que N est le nœud racine de l'arbre ci-dessous:



4) Arbres binaires de recherche (ABR)

Exemple: nœuds parcourus par l'appel **contient(N, 15)**, en supposant que N est le nœud racine de l'arbre ci-dessous:



4) Arbres binaires de recherche (ABR)

Pour **insérer une nouvelle valeur**: on va créer un nœud avec cette valeur comme donnée, puis **on va l'accrocher comme feuille** dans l'arbre. Pour **savoir où est le bon endroit** pour conserver les prop. d'ABR, on suit la **même logique** que précédemment:

- Soit la valeur à ajouter est **plus petite** que la donnée du nœud exploré→ *on se déplace vers le fils gauche s'il n'est pas vide, et on recommence*
- Soit la valeur à ajouter est **plus grande** que la donnée du nœud exploré → *on se déplace vers le fils droit s'il n'est pas vide, et on recommence*
- On finit par tomber sur un fils vide, *où l'on accroche la nouvelle feuille.*

4) Arbres binaires de recherche (ABR)

On suppose que N est la racine d'un ABR.

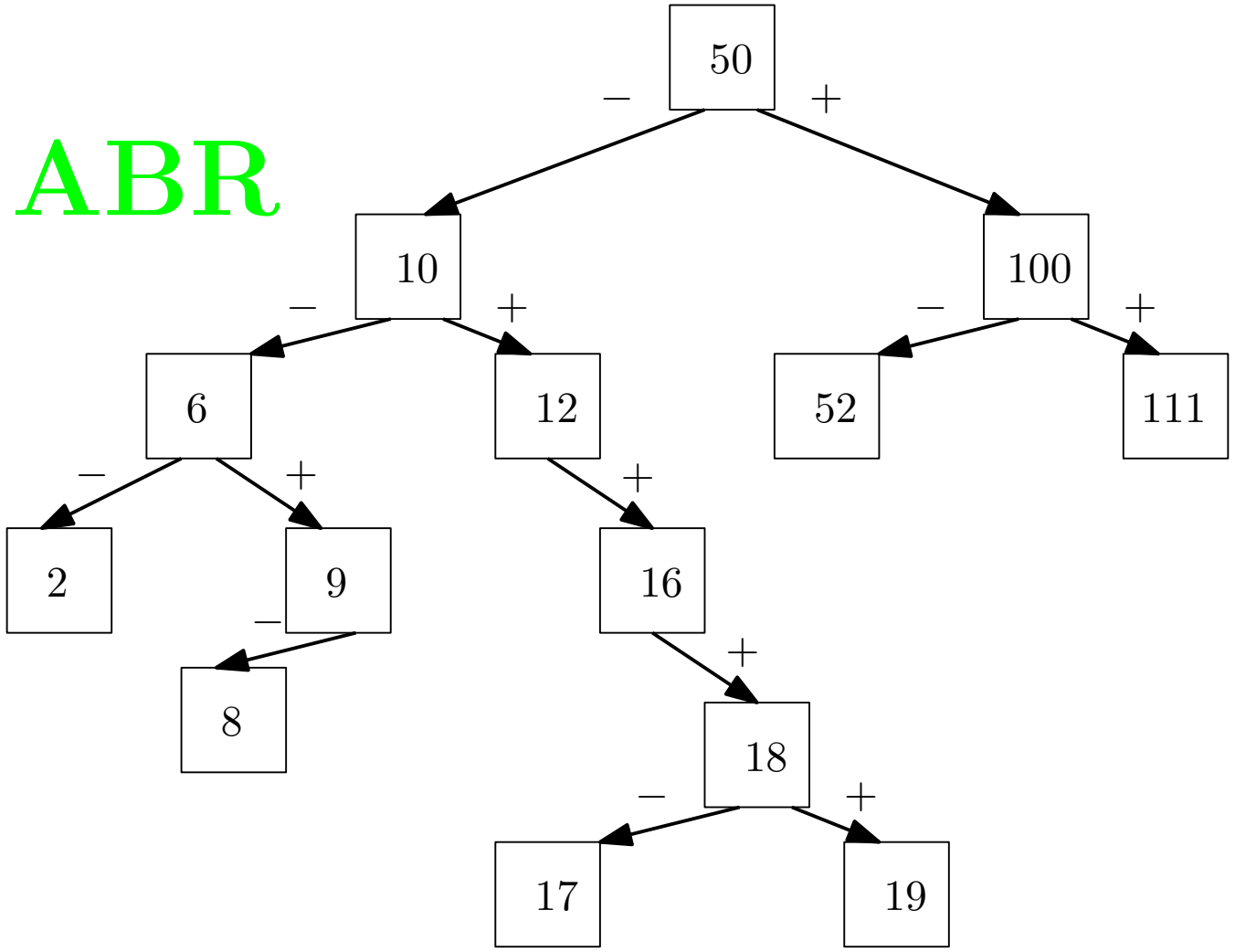
```
Proc ajoute(nœud N, nombre valeur):
    nœud M:=NewNoeud()
    M.data:=valeur
    bool stop:=Faux // Servira à détecter quand on doit s'arrêter
    Tant que non(stop):
        Si valeur < N.data:
            Si N.filsG==-1: // Si N n'a pas de fils gauche
                N.filsG:=M // on ajoute la feuille ici
                stop:=Vrai
            Sinon:
                N:=N.filsG // on continue vers la gauche
```

4) Arbres binaires de recherche (ABR)

```
Proc ajoute(nœud N, nombre valeur):
    Tant que non(stop):
        Sinon // alors la valeur est ≥ à la donnée du nœud N
            Si N.filsD==-1: // si N n'a pas de filsD
                N.filsD:=M // on ajoute la feuille ici
                stop:=Vrai
            Sinon:
                N:=N.filsD // on continue vers la droite
```

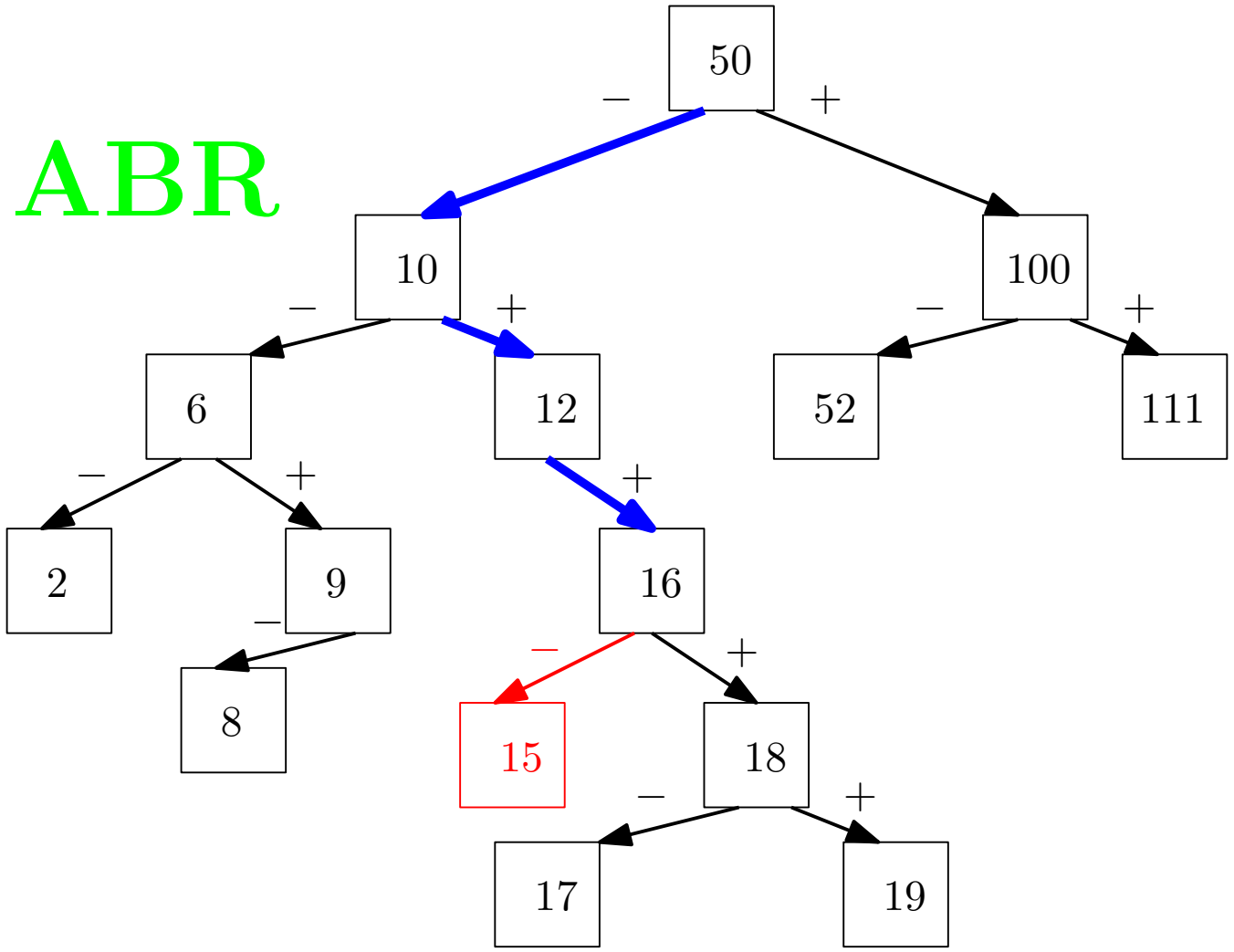
4) Arbres binaires de recherche (ABR)

Exemple: nœuds parcourus par l'appel **ajoute(N, 15)**, en supposant que N est le nœud racine de l'arbre ci-dessous:



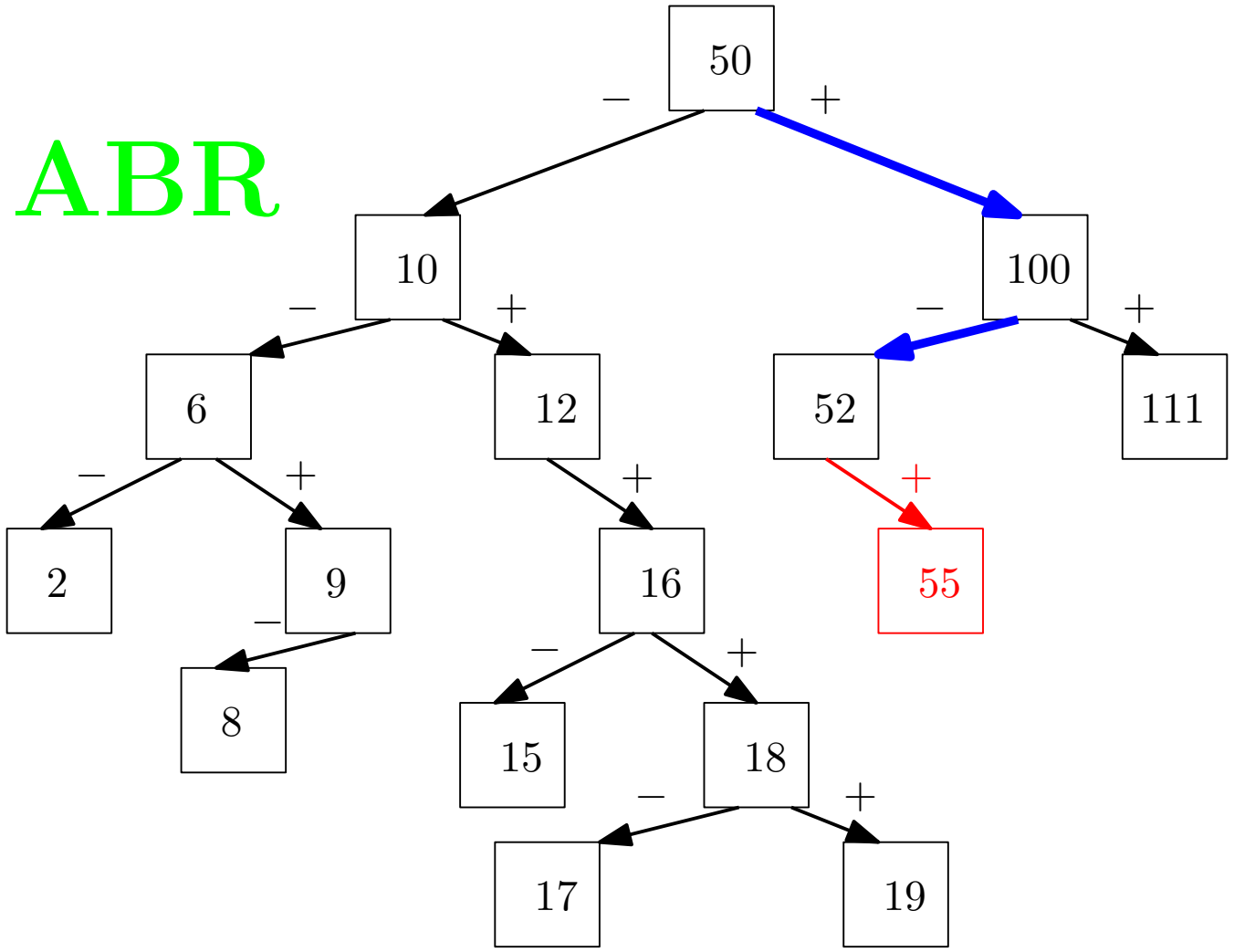
4) Arbres binaires de recherche (ABR)

Exemple: nœuds parcourus par l'appel **ajoute(N, 15)**, en supposant que N est le nœud racine de l'arbre ci-dessous:



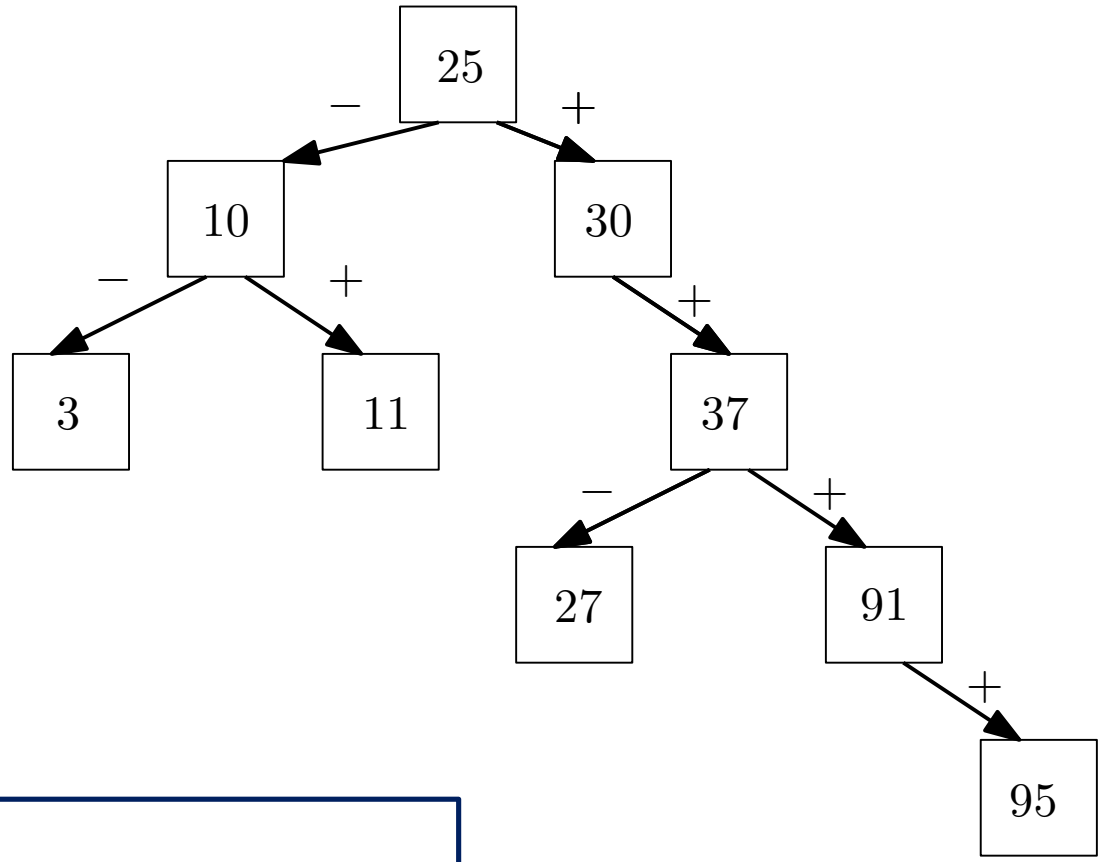
4) Arbres binaires de recherche (ABR)

Exemple: nœuds parcourus par l'appel **ajoute(N, 55)**, en supposant que N est le nœud racine de l'arbre ci-dessous:



4) Arbres binaires de recherche (ABR)

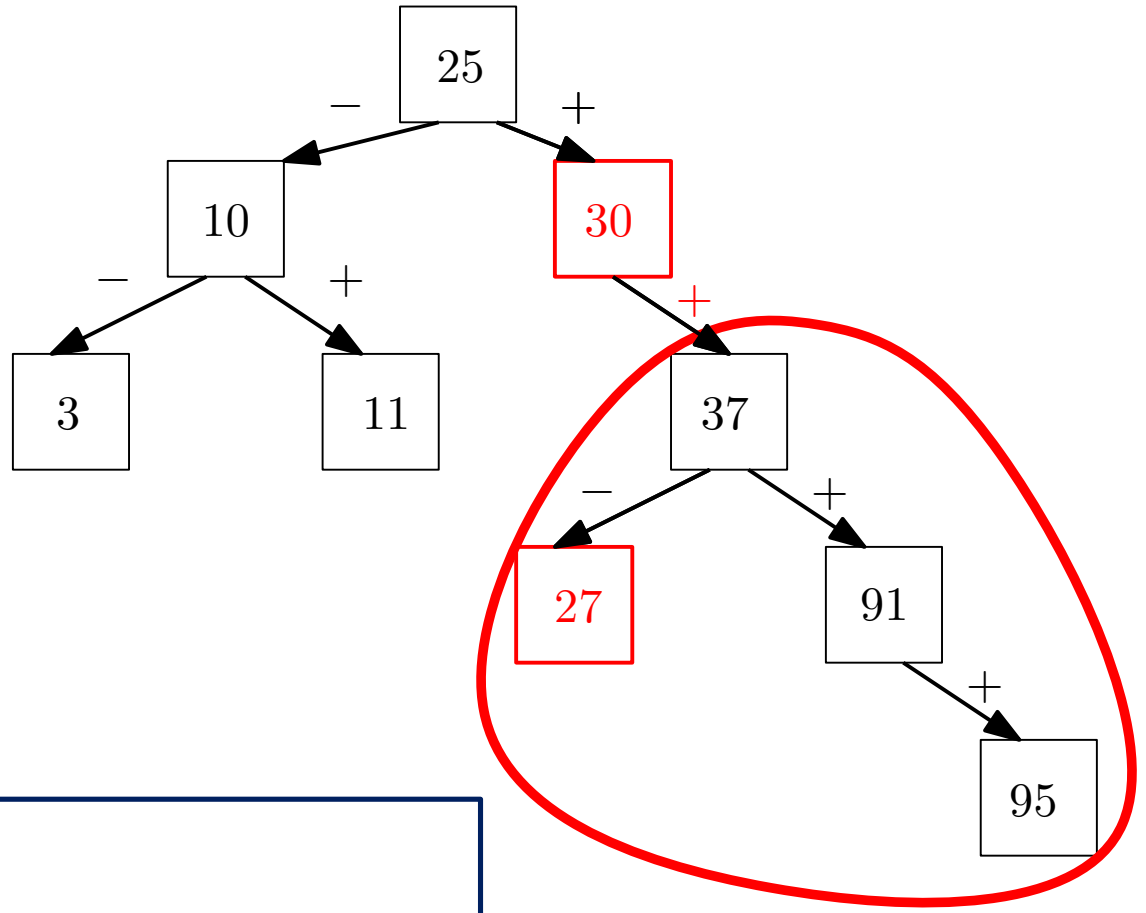
QCM: l'arbre ci-dessous est-il un ABR?



1) Oui
2) Non
3) On ne peut pas savoir

4) Arbres binaires de recherche (ABR)

QCM: l'arbre ci-dessous est-il un ABR?



1) Oui
2) Non
3) On ne peut pas savoir

5) Parcours d'arbres récursifs

Il existe trois autres parcours d'arbres couramment utilisés, qui permettent d'obtenir trois affichages différents des données de l'arbre:

- **affichage prefixe**
- **affichage infixe**
- **affichage post-fixe**

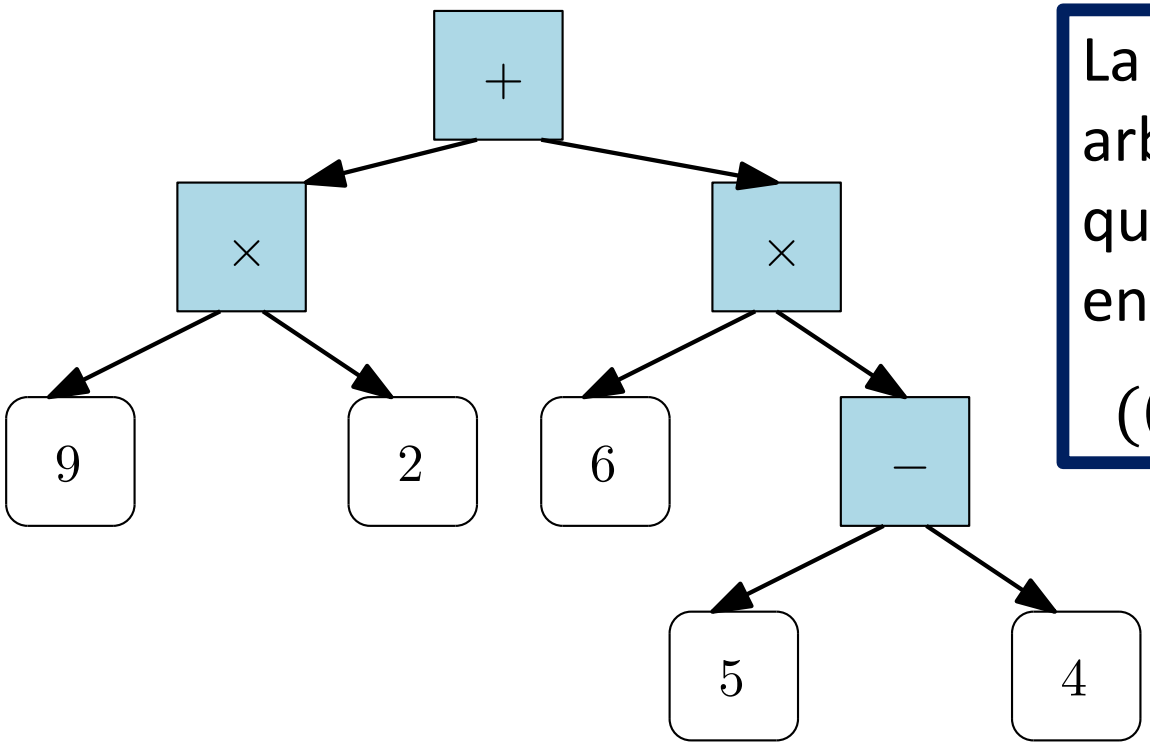
Ces trois affichages s'obtiennent grâce à des fonctions **récursives** : c'est-à-dire que **la fonction se rappelle elle-même** sur un sous-arbre plus petit!

```
Proc affichagePrefixe(nœud N):
    ...
    affichagePrefixe(...)
```

5) Parcours d'arbres récursifs

Pour bien comprendre ces trois affichages, il faut imaginer des arbres binaires qui représentent des formules arithmétiques:

- les nœuds internes contiennent chacun un symbole mathématique parmi +, -, x, /, et ont tous deux fils
- les feuilles contiennent un nombre



La formule associée à cet arbre, écrite selon la forme qu'on utilise habituellement en maths, est:

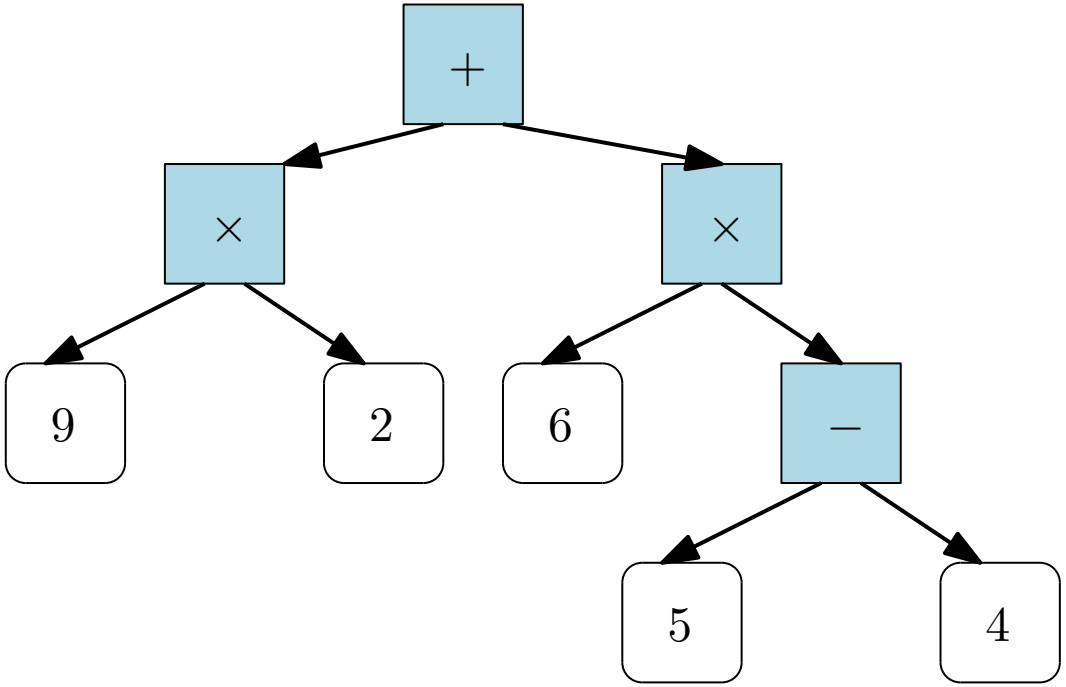
$$((9 \times 2) + (6 \times (5 - 4)))$$

5) Parcours d'arbres récursifs

L'**écriture infixe** est celle qu'on utilise habituellement en maths:

- **d'abord**, on écrit tout ce qui concerne le sous-arbre **gauche**
- puis, on écrit la donnée du nœud **racine**
- puis, on écrit tout ce qui concerne le sous-arbre **droit**

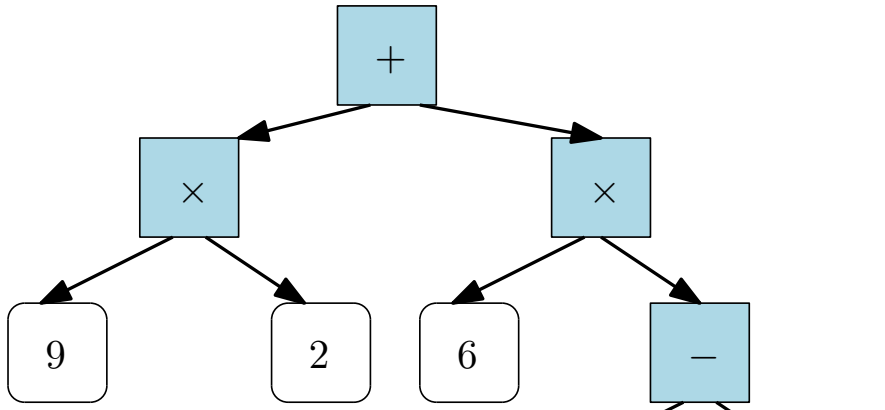
On ajoute des parenthèses pour améliorer la lecture.



Formule en **écriture infixe**:

$$((9 \times 2) + (6 \times (5 - 4)))$$

5) Parcours d'arbres récursifs



Formule en **écriture infixe**:

$$((9 \times 2) + (6 \times (5 - 4)))$$

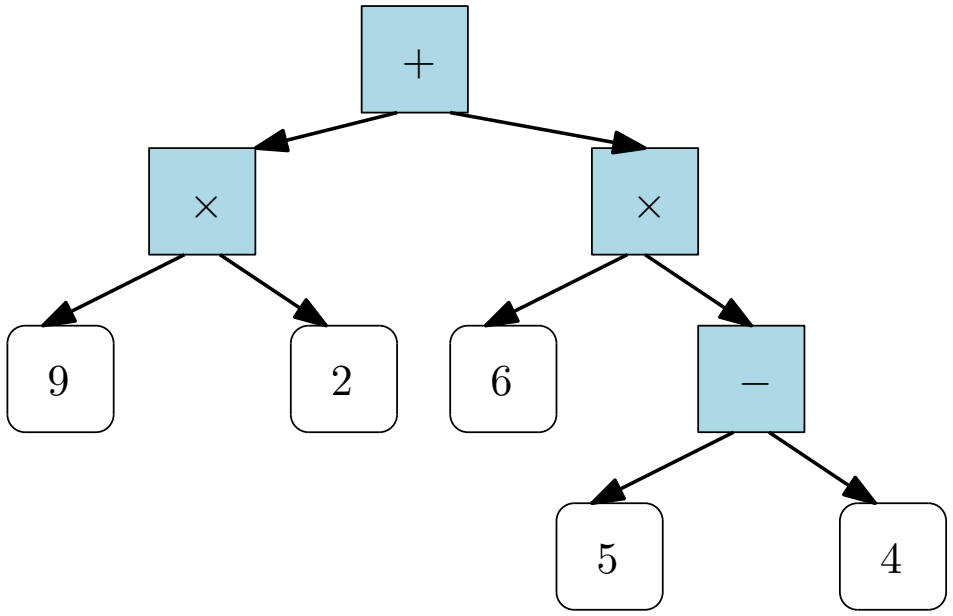
```
Proc affichageInfixe(nœud N):
    Si estFeuille(N):
        print(N.data)
    Sinon:
        print( "(" ) /* Parenthèse pour l'esthétisme */
        affichageInfixe(N.filsG) //on traite le sous-arbre gauche
        print(N.data) // puis la racine
        affichageInfixe(N.filsD) // puis le sous-arbre droit
        print( ")" )
```

5) Parcours d'arbres récursifs

L'**écriture préfixe** consiste à écrire le signe de l'opération avant les opérandes :

- **d'abord**, on écrit la donnée du nœud **racine**
- puis, on écrit tout ce qui concerne le sous-arbre **gauche**
- puis, on écrit tout ce qui concerne le sous-arbre **droit**

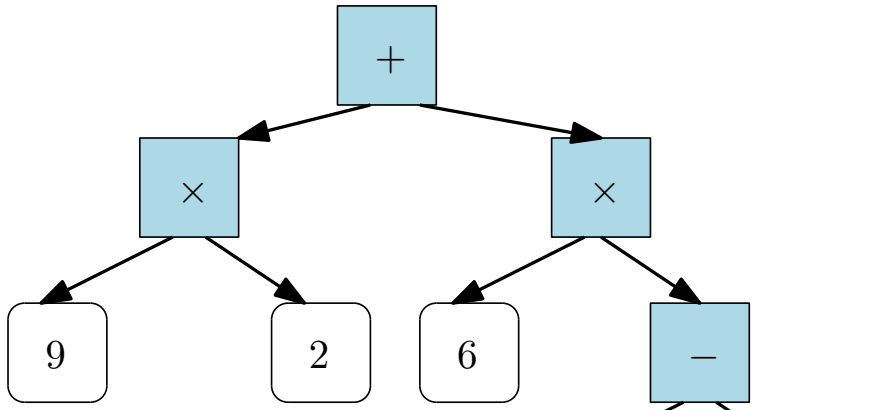
On ajoute des parenthèses pour améliorer la lecture.



Formule en **écriture préfixe**:

$$(+(\times\ 9\ 2)\ (\times\ 6\ (-\ 5\ 4)))$$

5) Parcours d'arbres récursifs



Formule en **écriture préfixe**:

(+(× 9 2) (× 6 (− 5 4)))

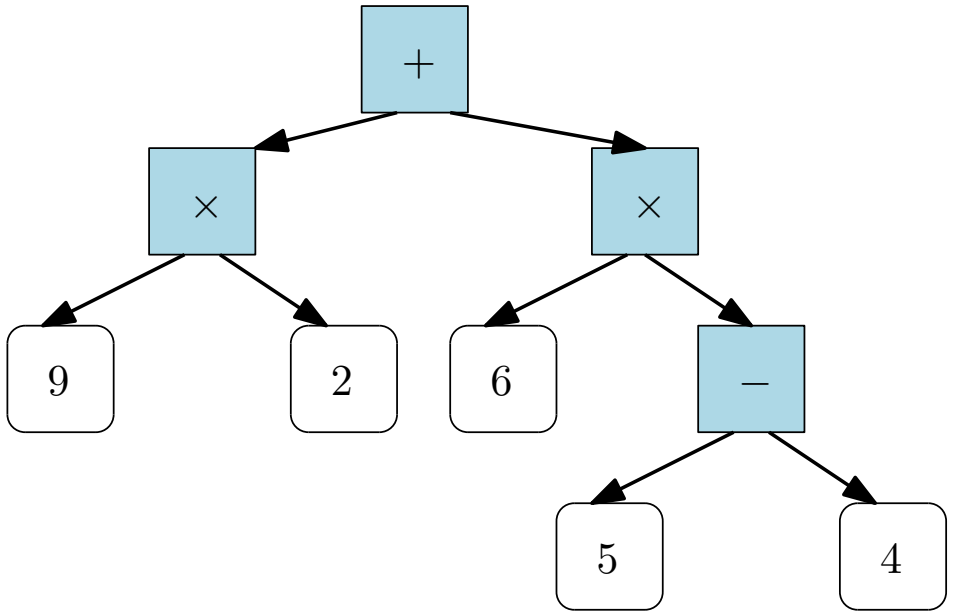
```
Proc affichagePrefixe(nœud N):
    Si estFeuille(N):
        print(N.data)
    Sinon:
        print( "(" ) /* Parenthèse pour l'esthétisme */
        print(N.data) // on traite la racine
        affichagePrefixe(N.filsG) //puis le sous-arbre gauche
        affichagePrefixe(N.filsD) // puis le sous-arbre droit
        print( ")" )
```

5) Parcours d'arbres récursifs

L'**écriture postfixe** consiste à écrire le signe de l'opération après les opérandes :

- **d'abord**, on écrit tout ce qui concerne le sous-arbre **gauche**
- puis, on écrit tout ce qui concerne le sous-arbre **droit**
- puis, on écrit la donnée du nœud **racine**

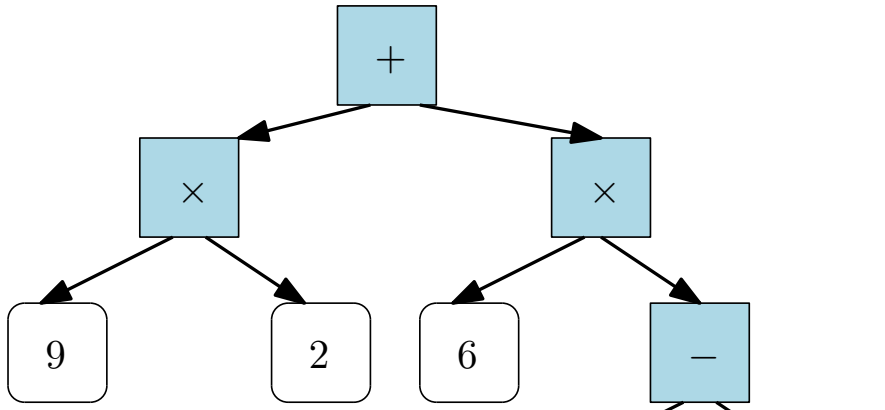
On ajoute des parenthèses pour améliorer la lecture.



Formule en **écriture postfixe**:

$$((9\ 2\ \times)\ (6\ (5\ 4\ -)\ \times)+)$$

5) Parcours d'arbres récursifs



Formule en **écriture postfixe**:

((9 2 ×) (6 (5 4 −) ×)+)

```
Proc affichagePostfixe(nœud N):
    Si estFeuille(N):
        print(N.data)
    Sinon:
        print( "(" ) /* Parenthèse pour l'esthétisme */
        affichagePostfixe(N.filsG) //on traite le sous-arbre gauche
        affichagePostfixe(N.filsD) // puis le sous-arbre droit
        print(N.data) // puis la racine
        print( ")" )
```