

5 Arbres

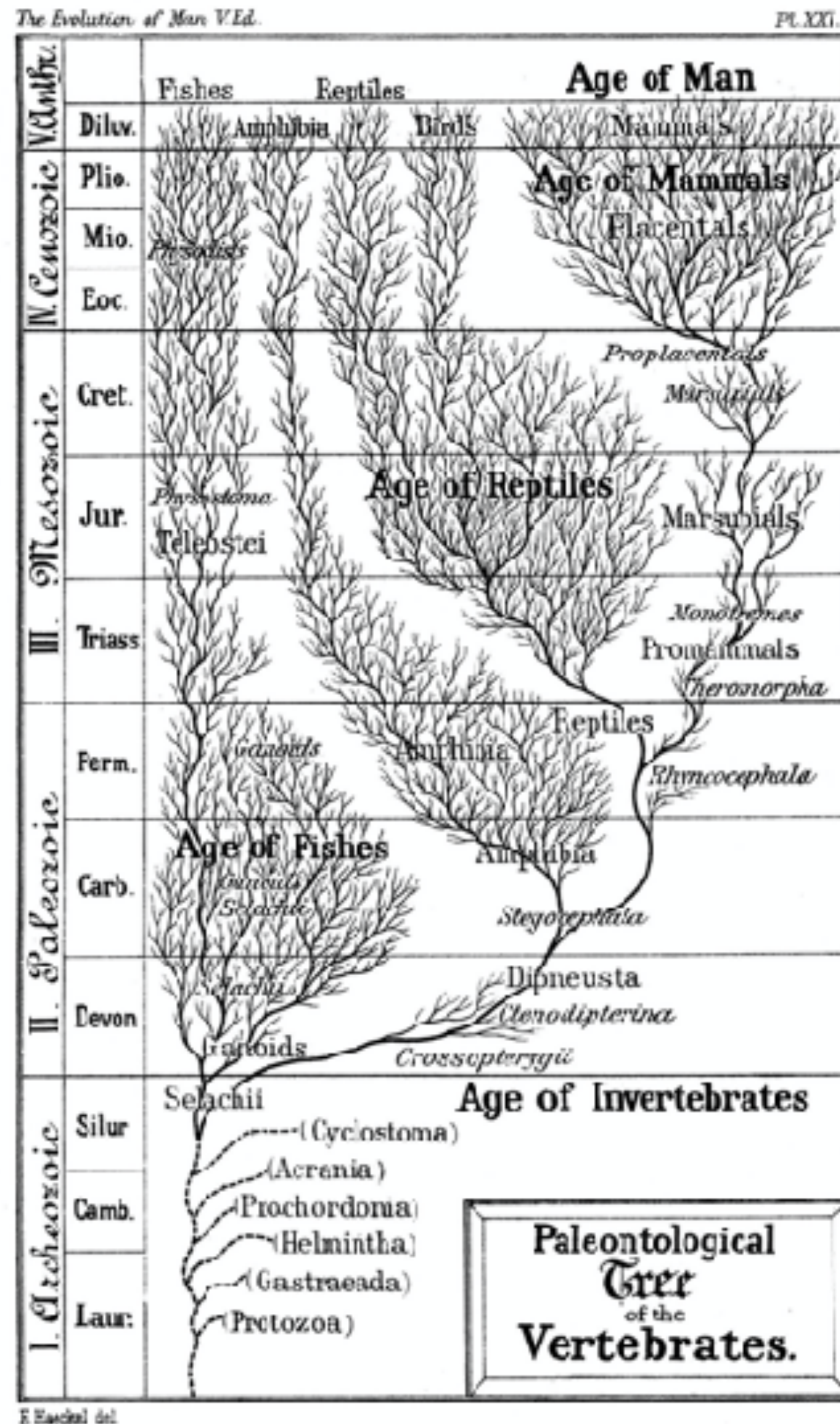
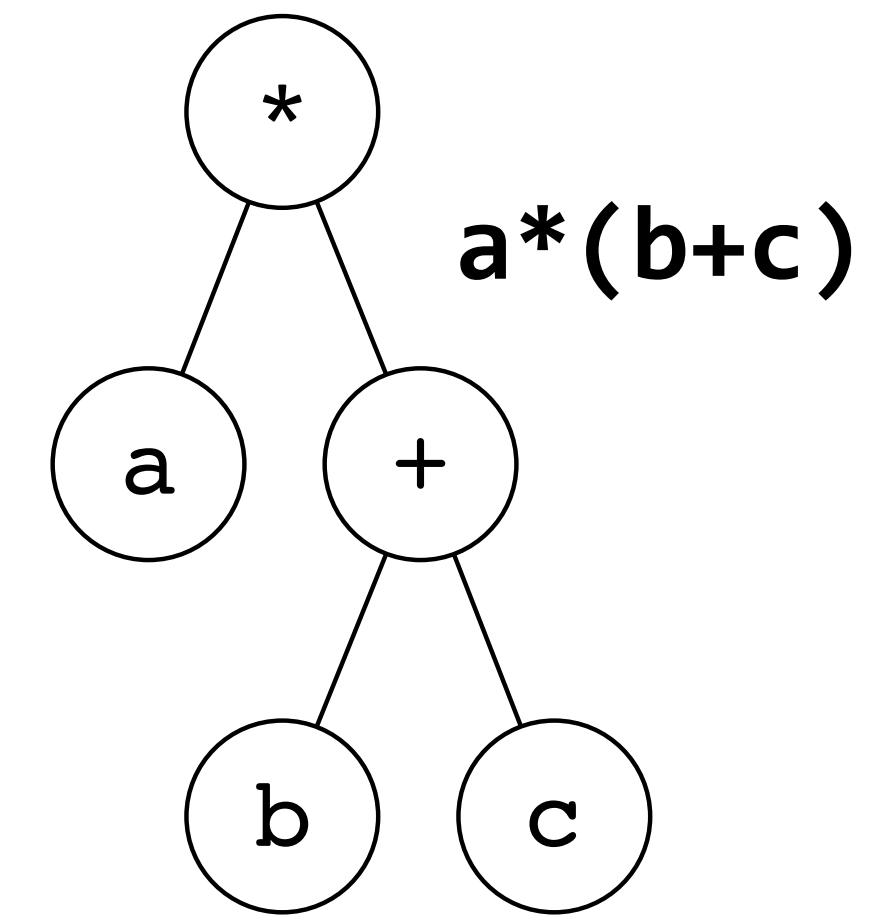


1 . Introduction

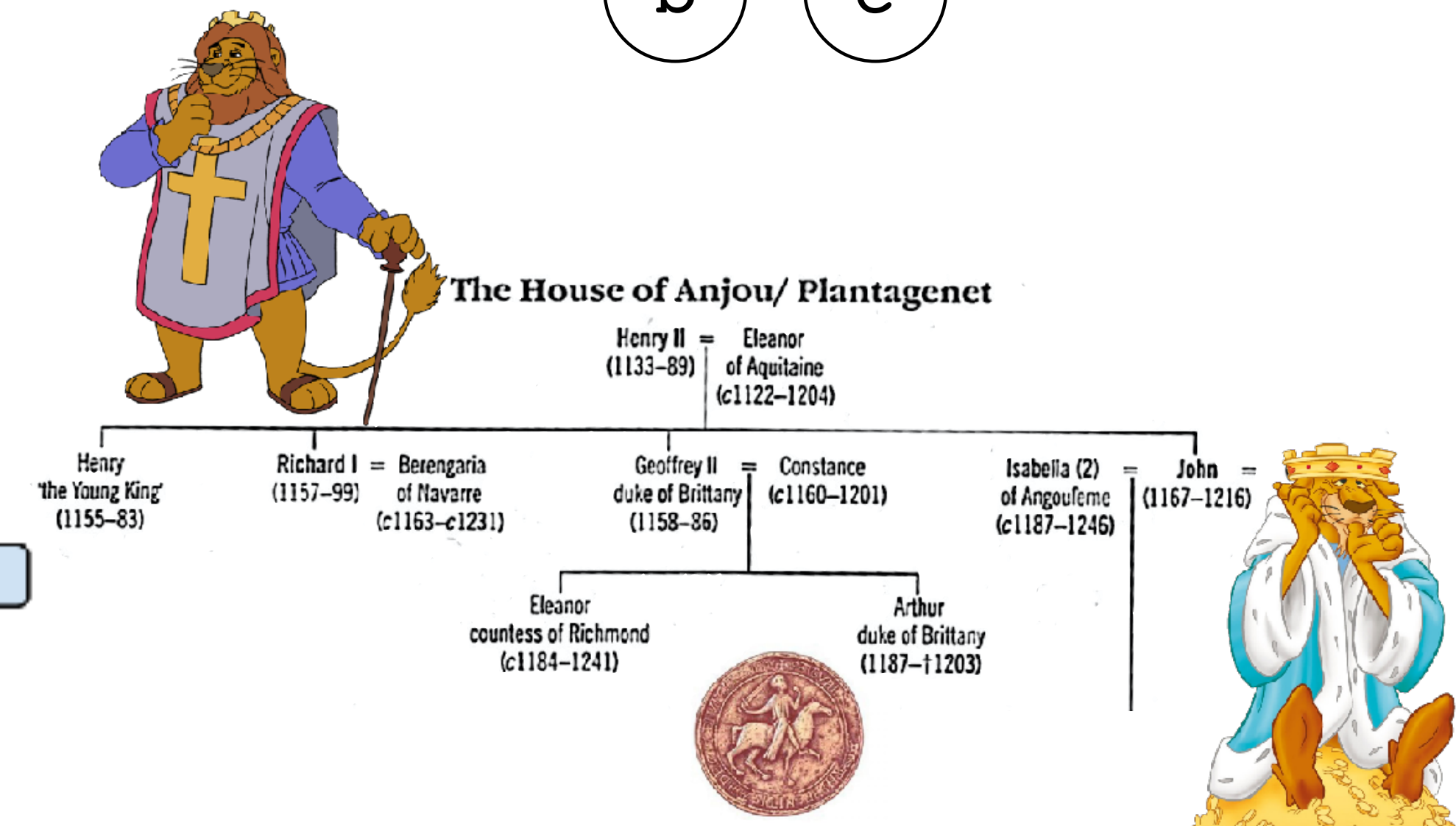
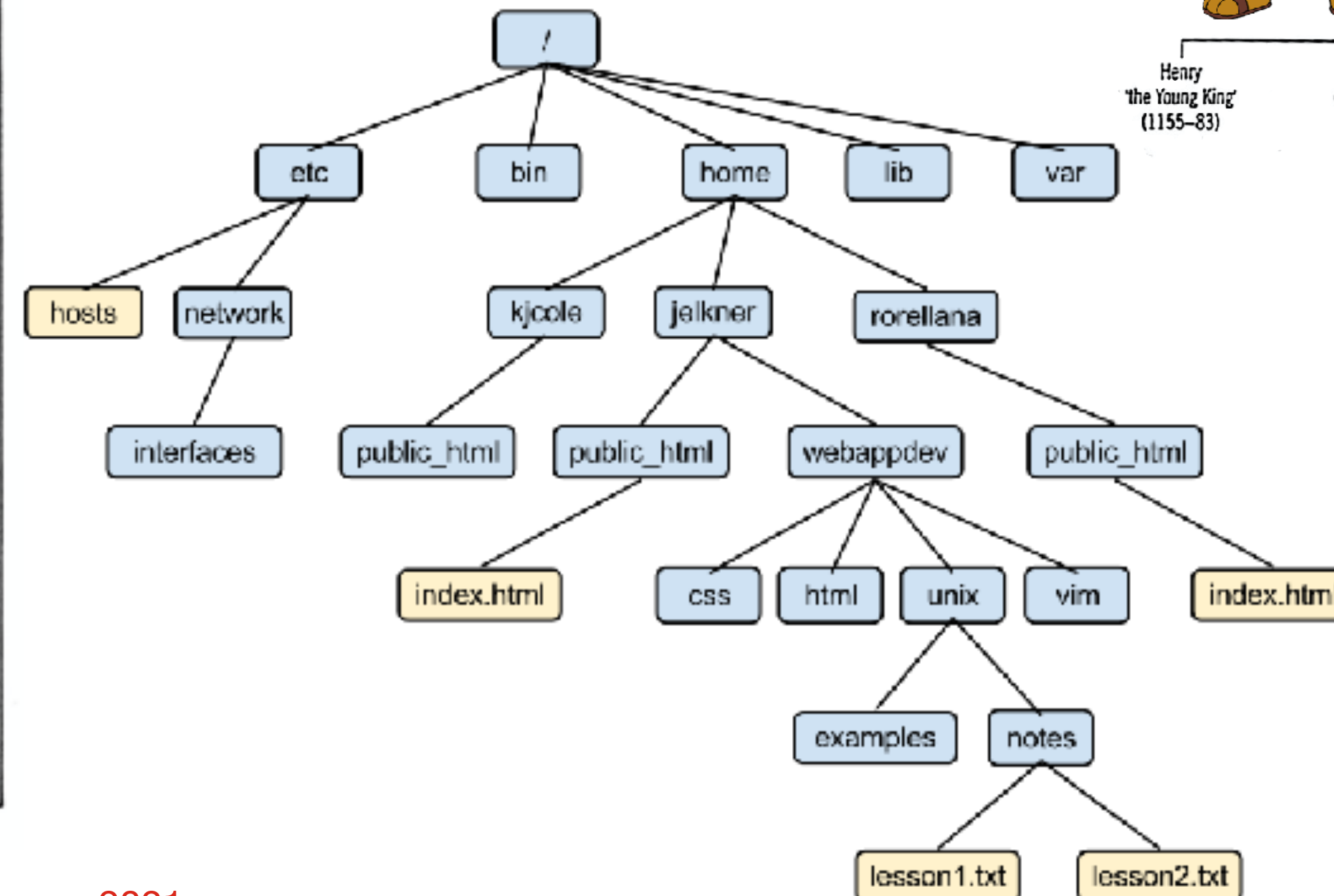


Exemples





set	Set (class template)
multiset	Multiple-key set (class template)
map	Map (class template)
multimap	Multiple-key map (class template)



Qu'est-ce qu'un arbre?



Extension du concept de liste chaînée,
avec plusieurs éléments suivants.

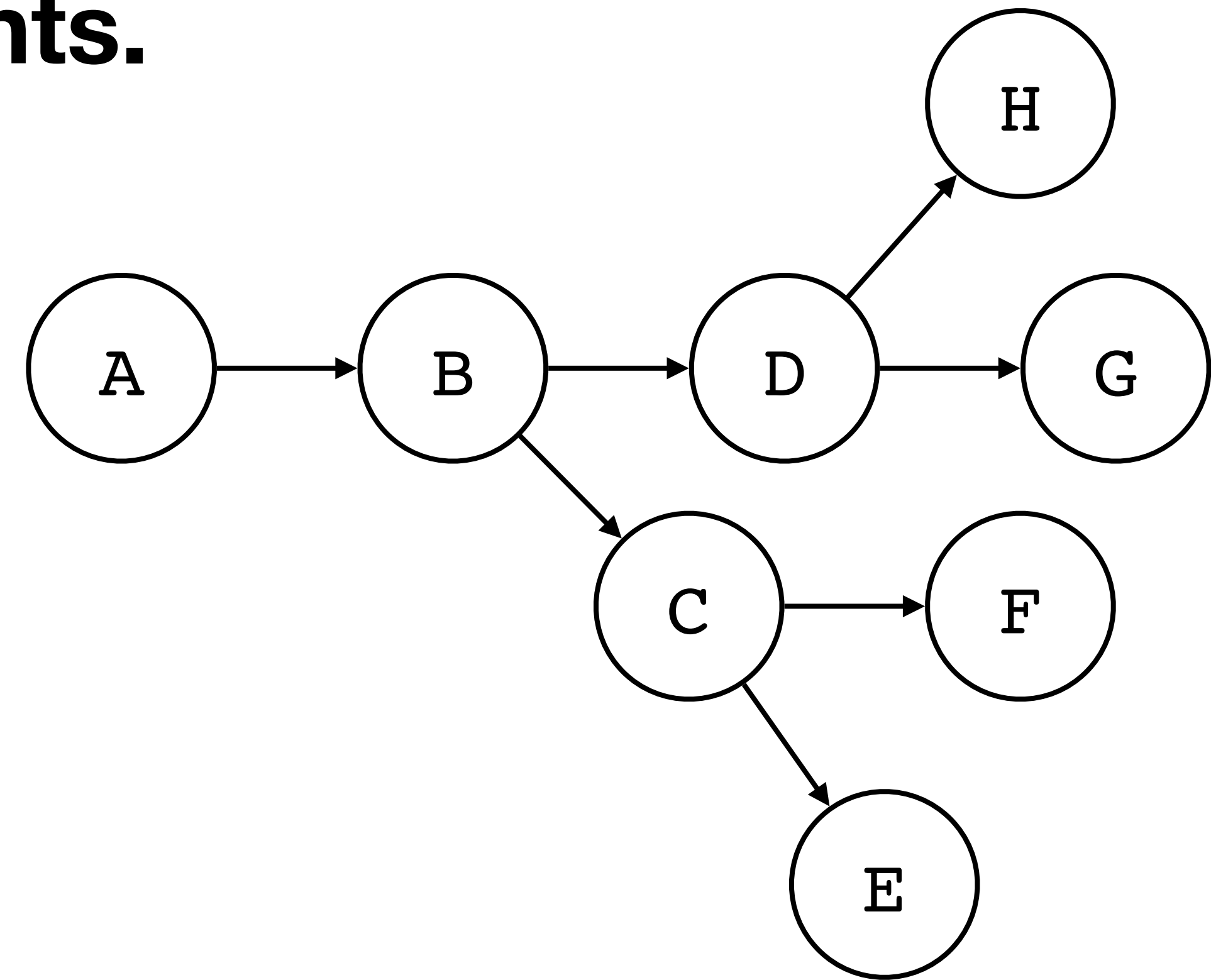
Maillon $\Leftrightarrow$ **Noeud**

Précédent $\Leftrightarrow$ **Parent**

Suivant $\Leftrightarrow$ **Enfants**

Début $\Leftrightarrow$ **Racine**

Valeur $\Leftrightarrow$ **Etiquettes**

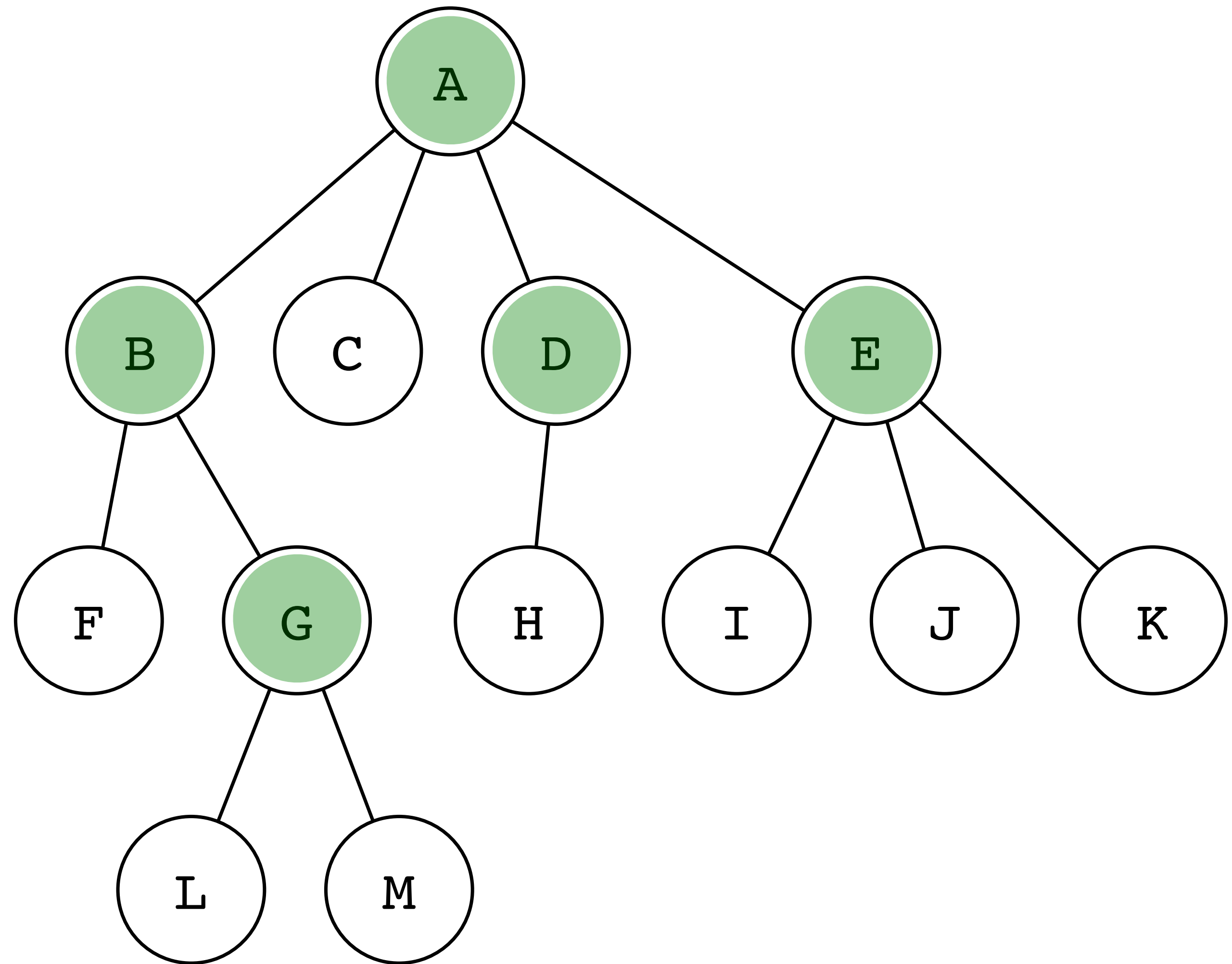


Nomenclature des noeuds



Noeuds internes

ont au moins un
enfant

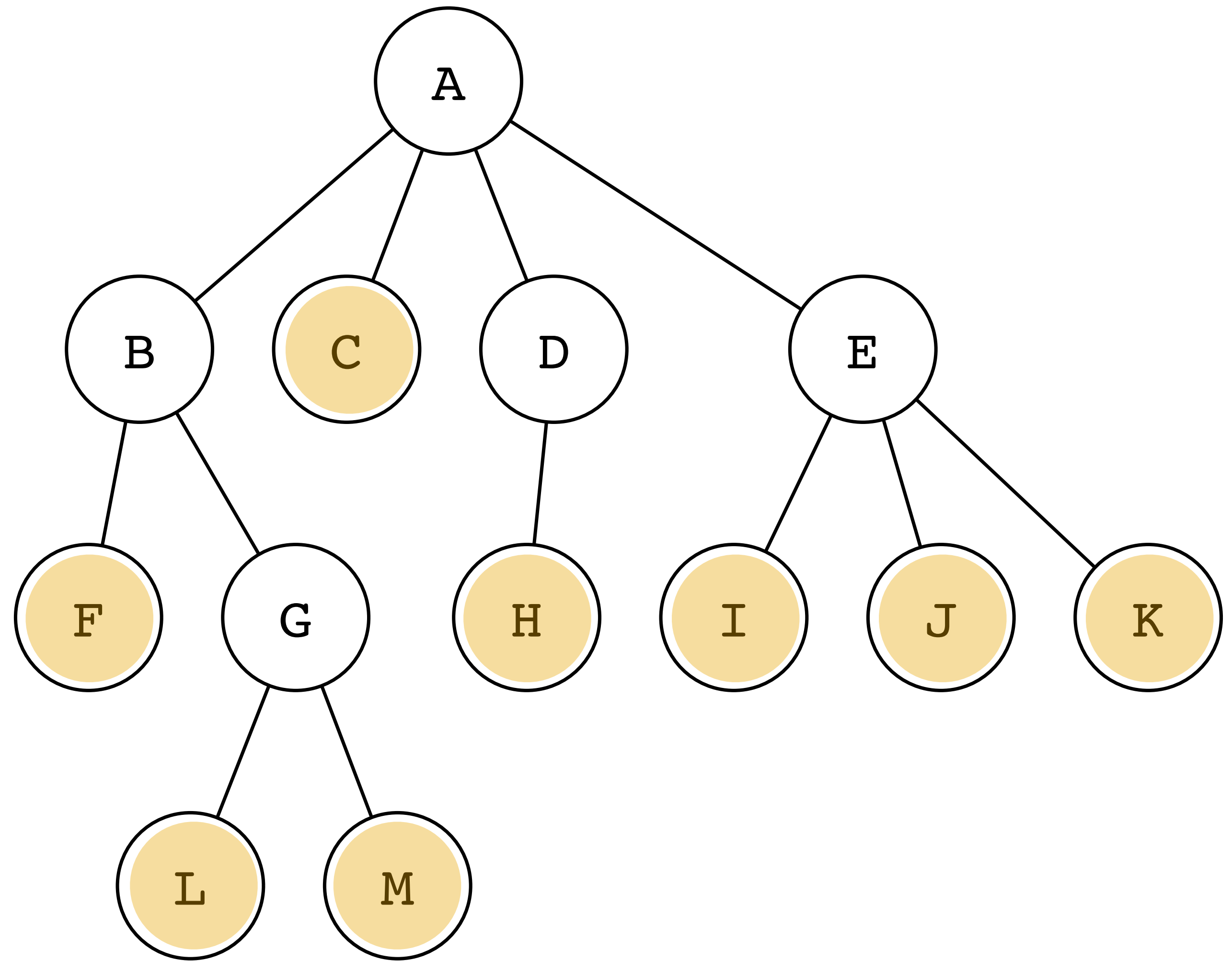


Nomenclature des noeuds



Noeuds internes
ont au moins un
enfant

Feuilles
n'ont pas d'enfant



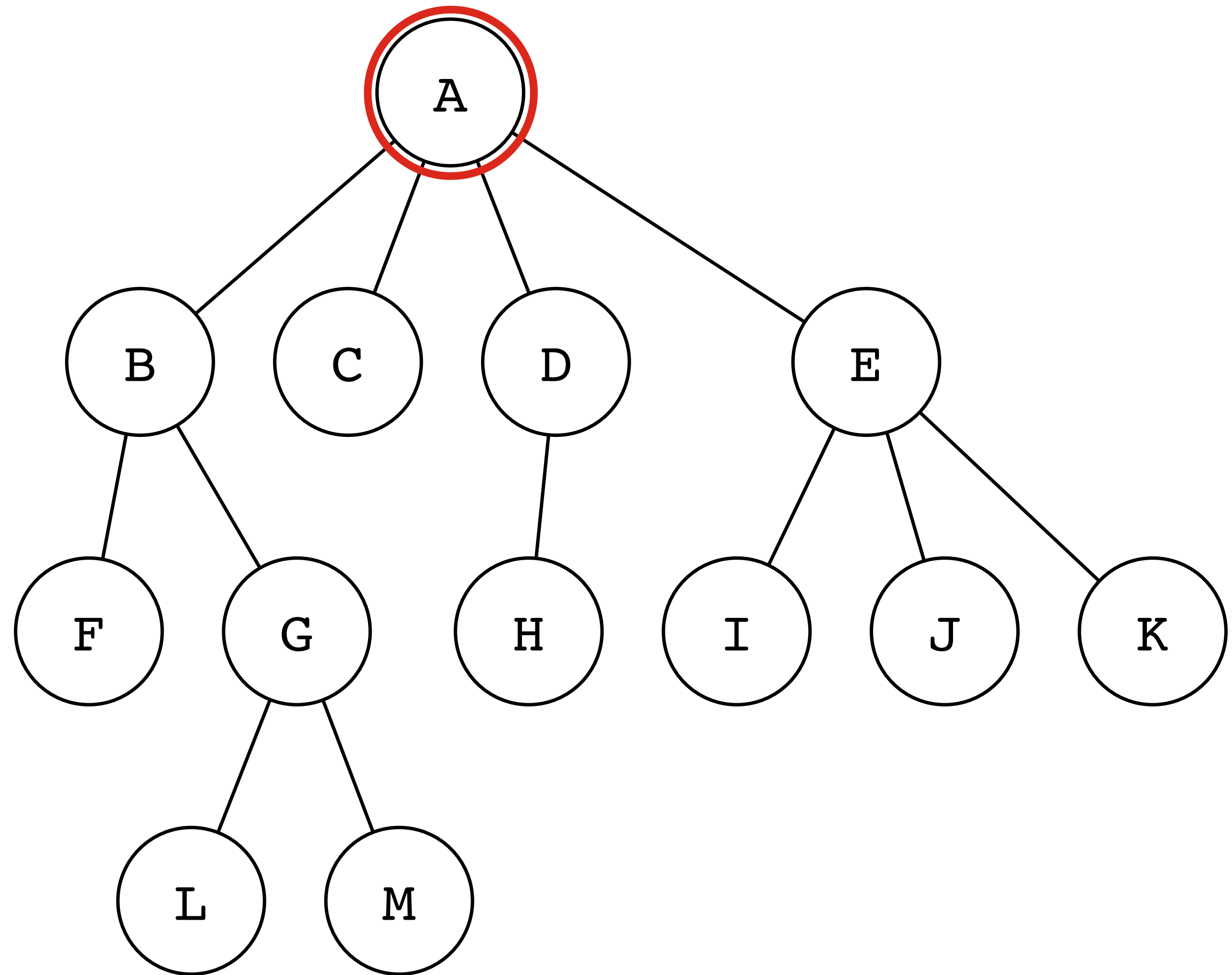


Noeuds internes
ont au moins un
enfant

Feuilles
n'ont pas d'enfant

Racine

n'a pas de parent



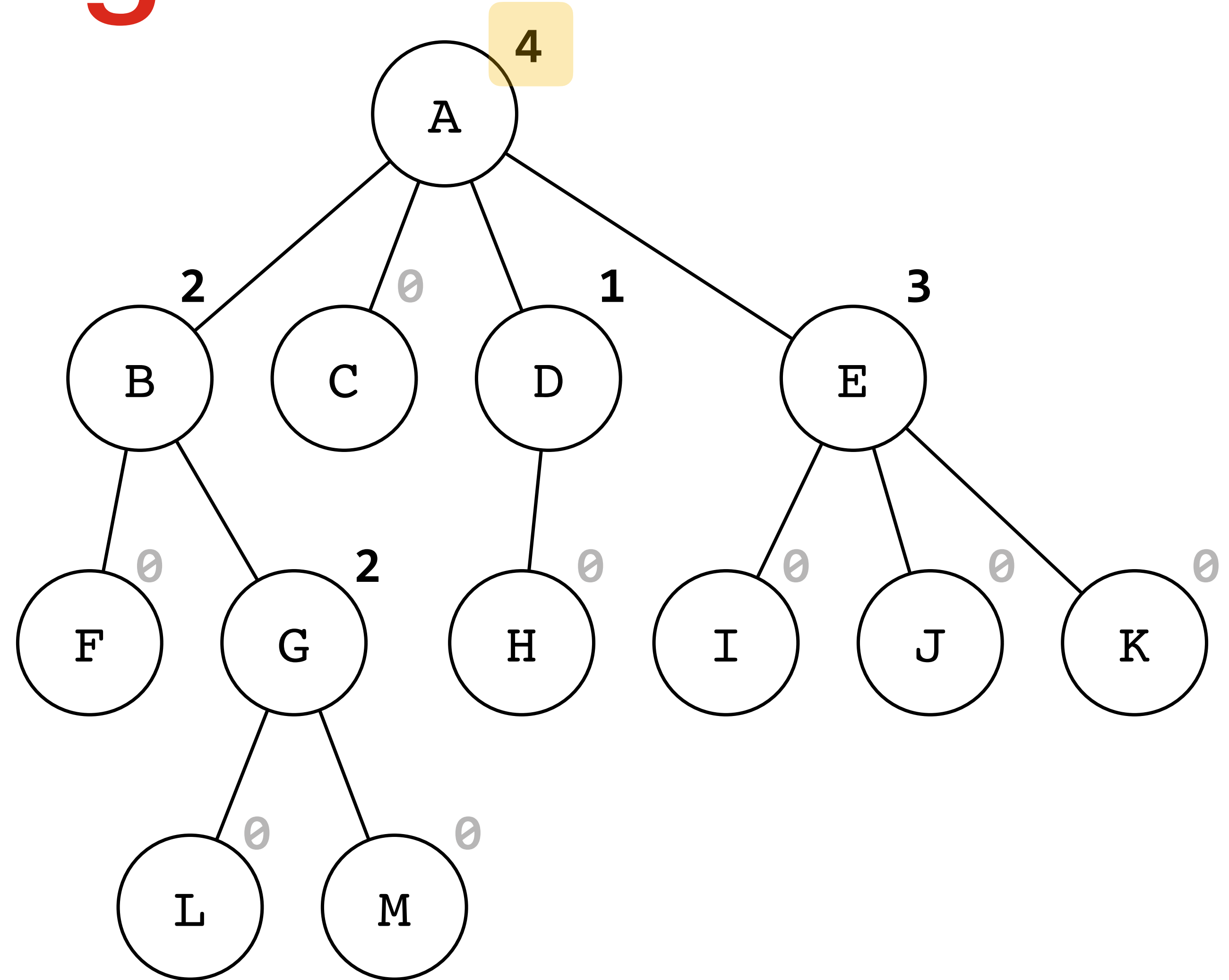


Degrés

Degré d'un noeud
Nombre d'enfants

Degré d'un arbre

Degré max de ses noeuds

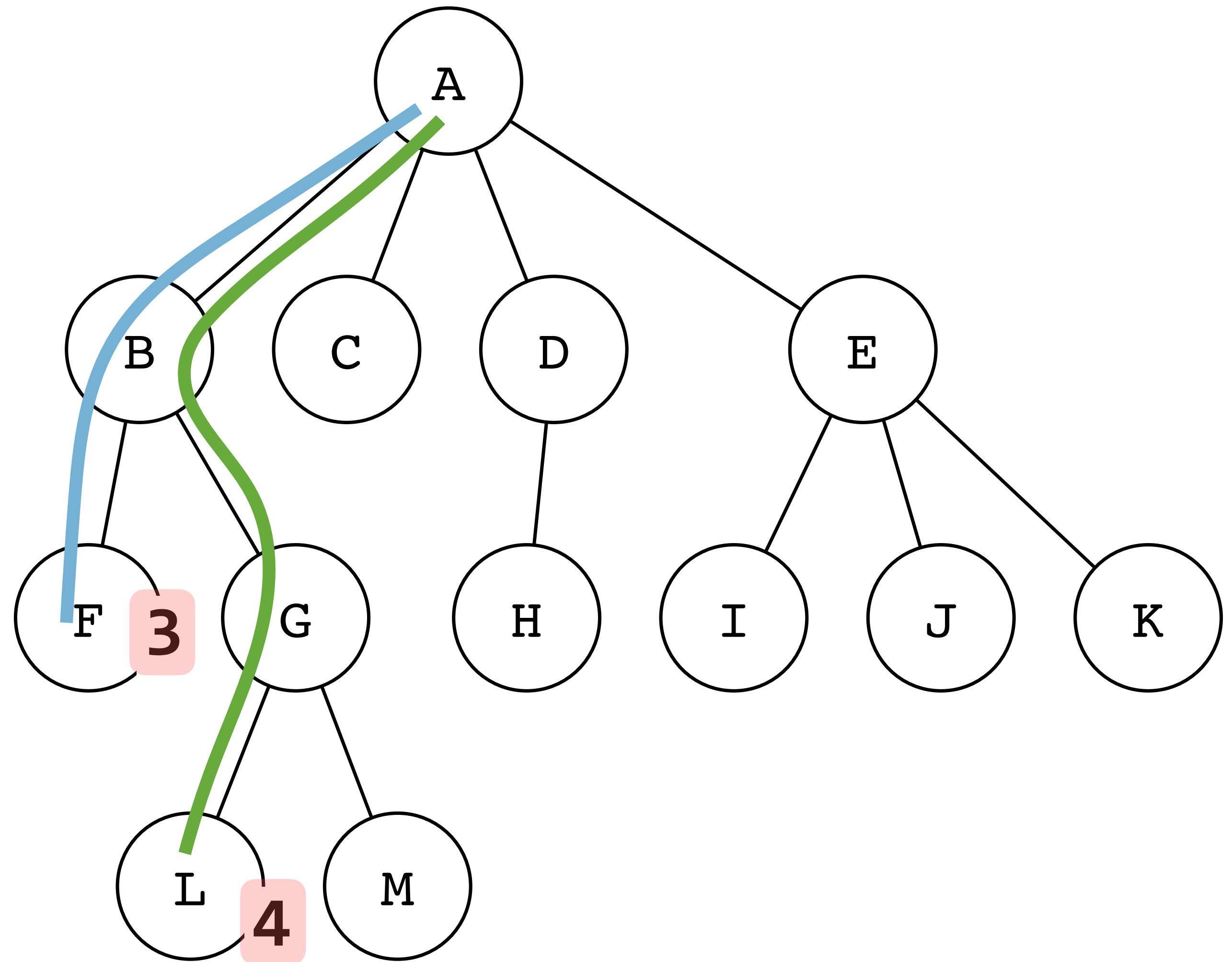




Chemins

Chemin d'un noeud
Suite des noeuds le
reliant à la racine

Niveau d'un noeud
Longueur de son
chemin



Chemin de F : **A-B-F**

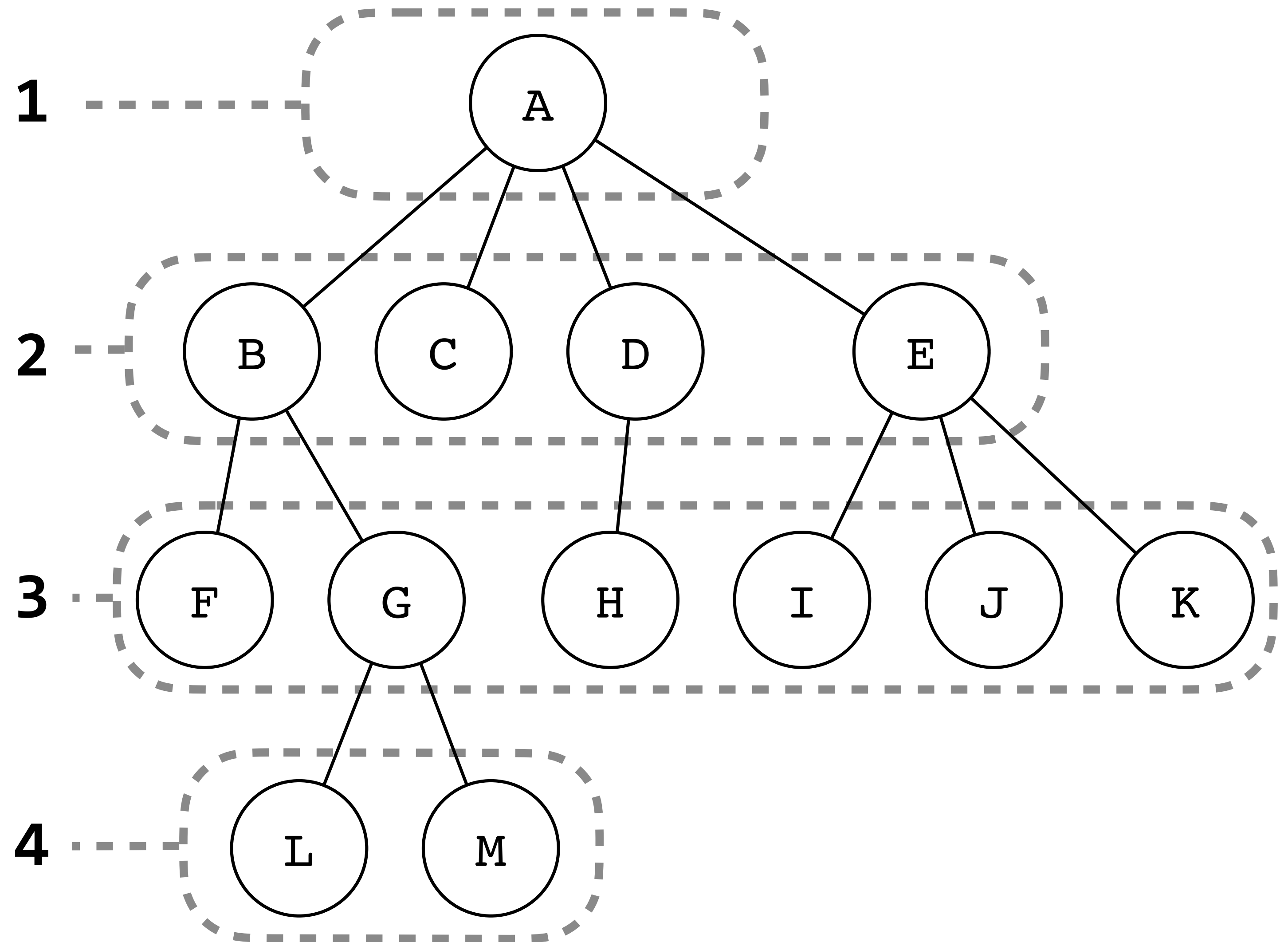
Chemin de L : **A-B-G-L**



Chemins

Chemin d'un noeud
Suite des noeuds le
reliant à la racine

Niveau d'un noeud
Longueur de son
chemin

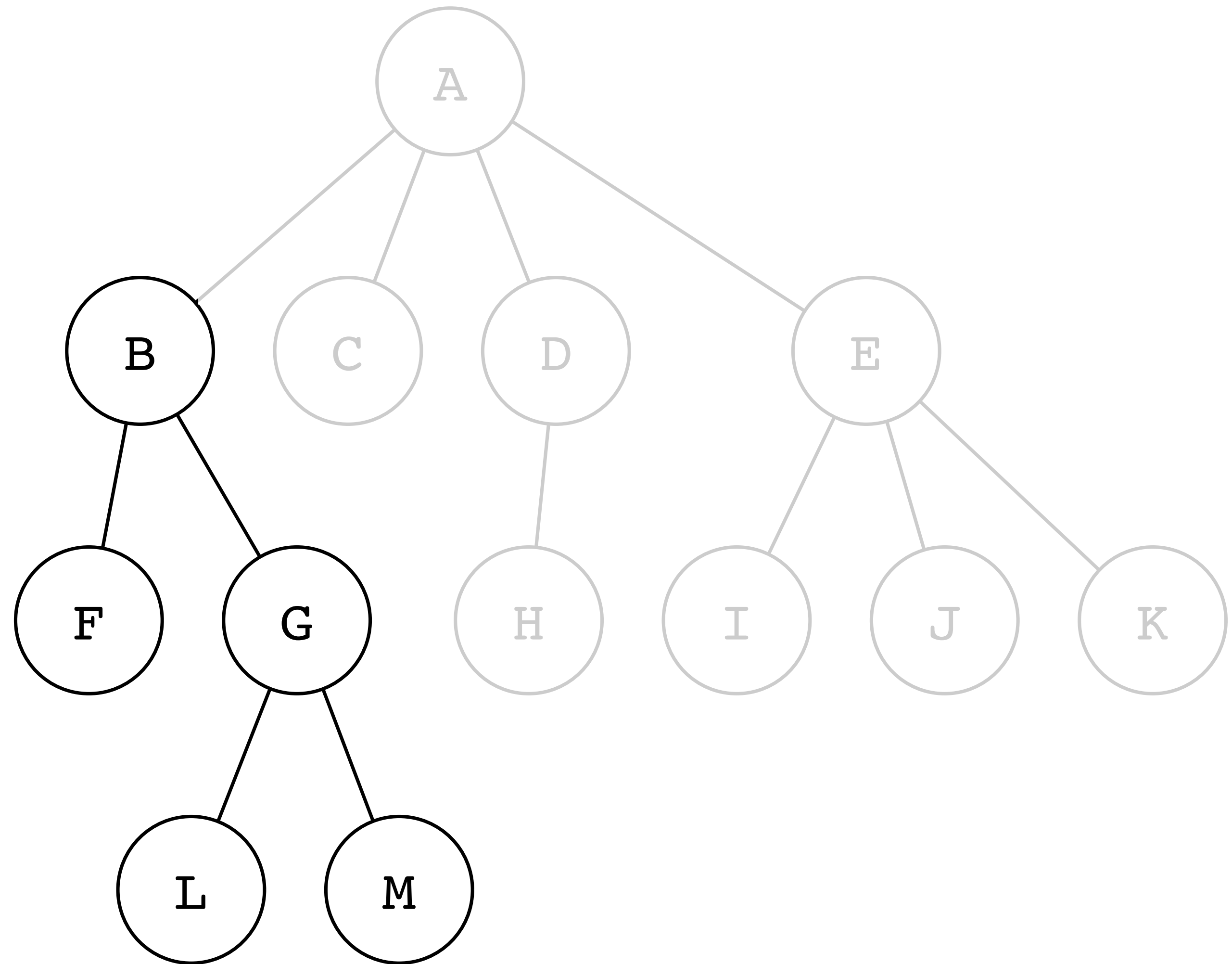


Hauteur et sous-arbres



Sous-arbre

Arbre dont la racine est enfant dans un autre arbre





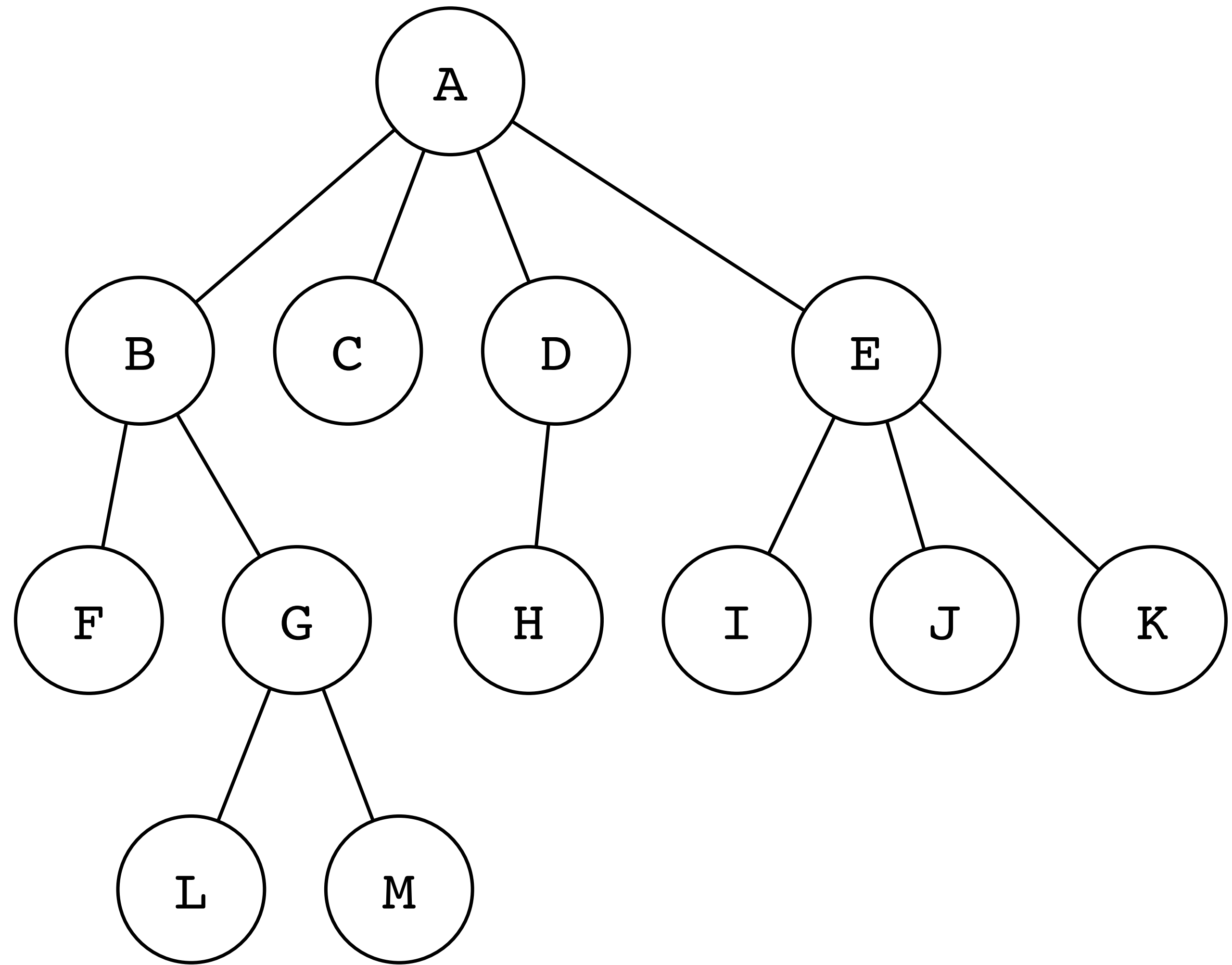
Sous-arbre

Arbre dont la racine est enfant dans un autre arbre

Hauteur d'un arbre

Niveau maximum
parmi tous ses noeuds

$$H(v) = 1 + \max_{e \in \text{enfants}(v)} H(e)$$





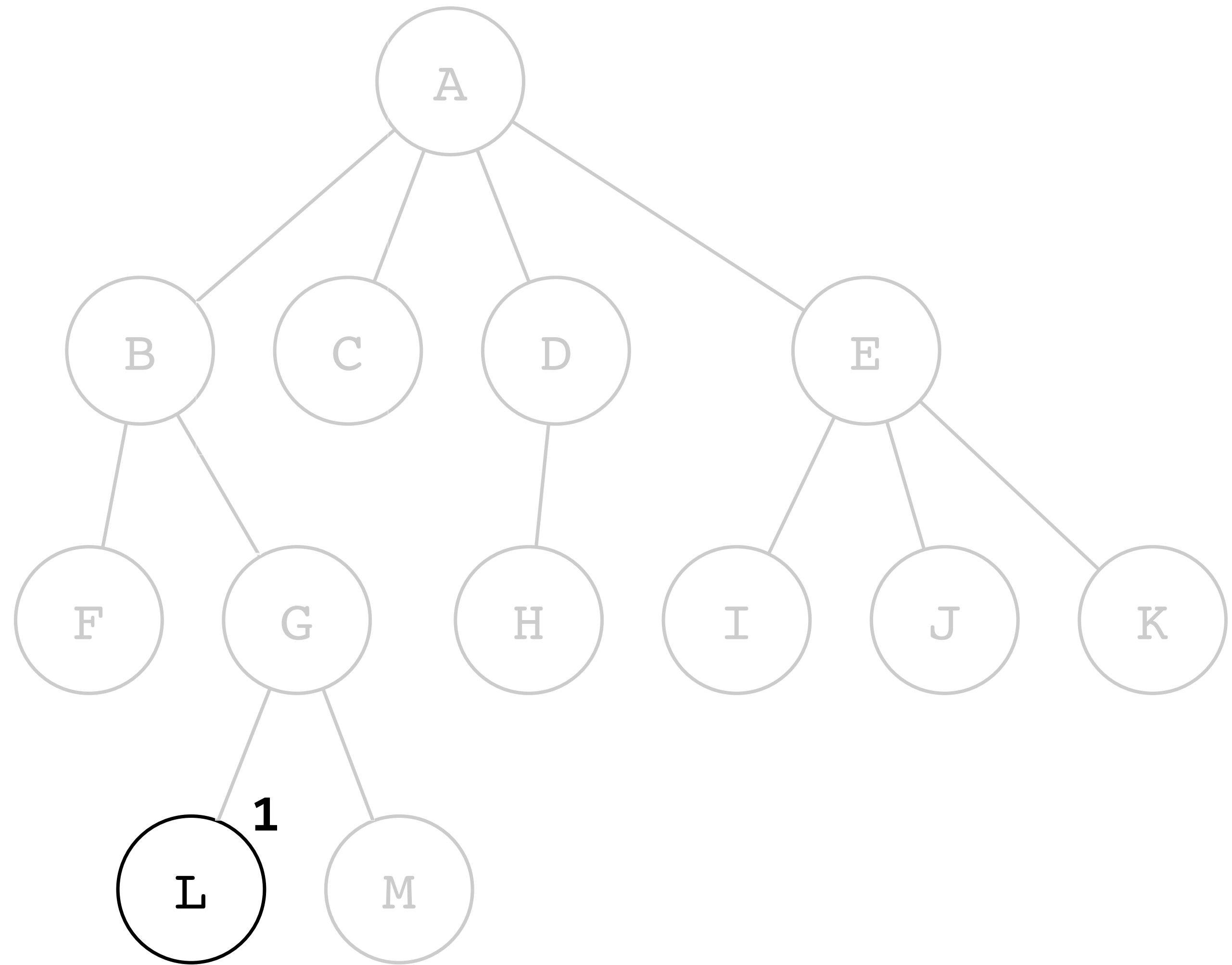
Sous-arbre

Arbre dont la racine est enfant dans un autre arbre

Hauteur d'un arbre

Niveau maximum
parmi tous ses noeuds

$$H(v) = 1 + \max_{e \in \text{enfants}(v)} H(e)$$





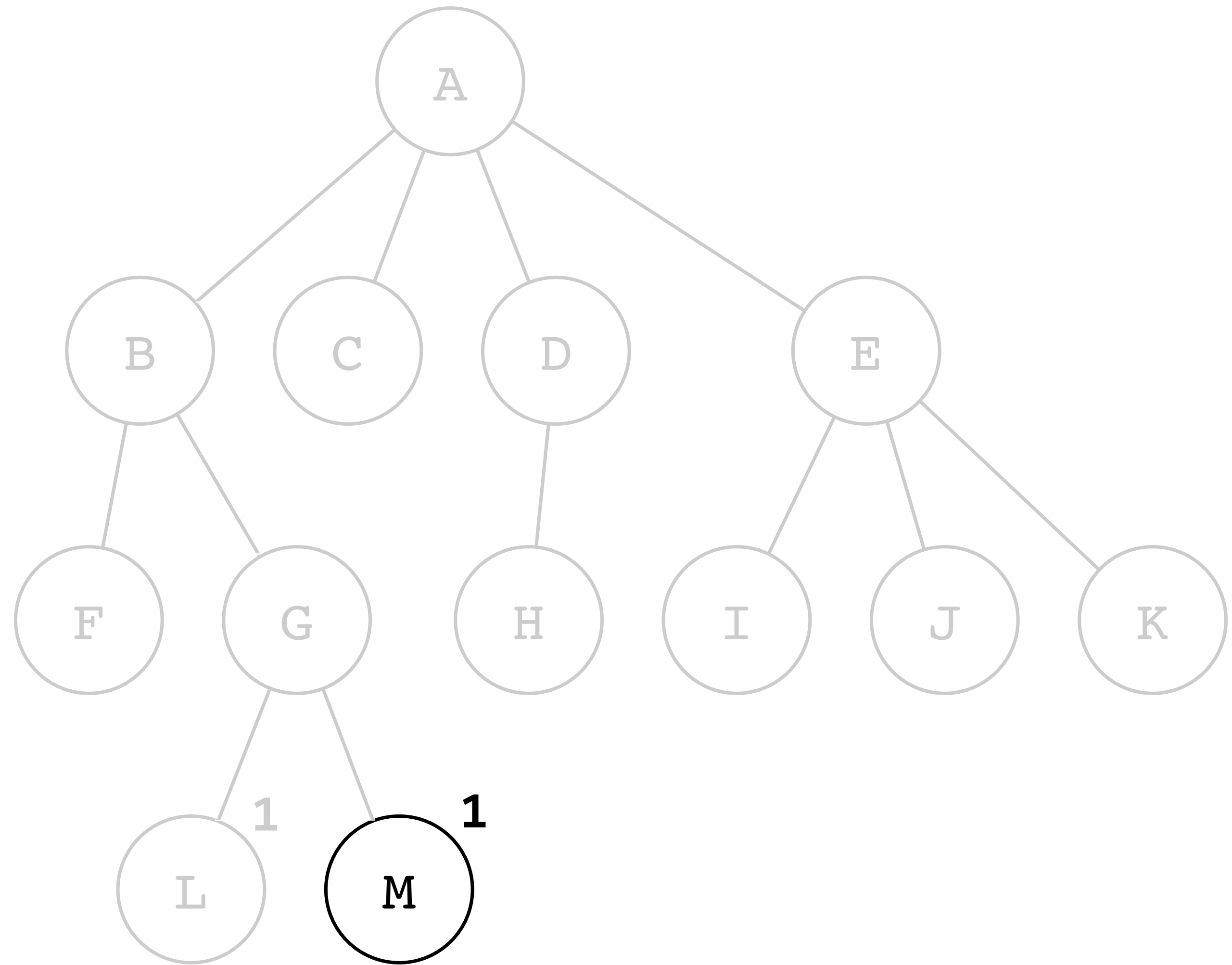
Sous-arbre

Arbre dont la racine est enfant dans un autre arbre

Hauteur d'un arbre

Niveau maximum
parmi tous ses noeuds

$$H(v) = 1 + \max_{e \in \text{enfants}(v)} H(e)$$





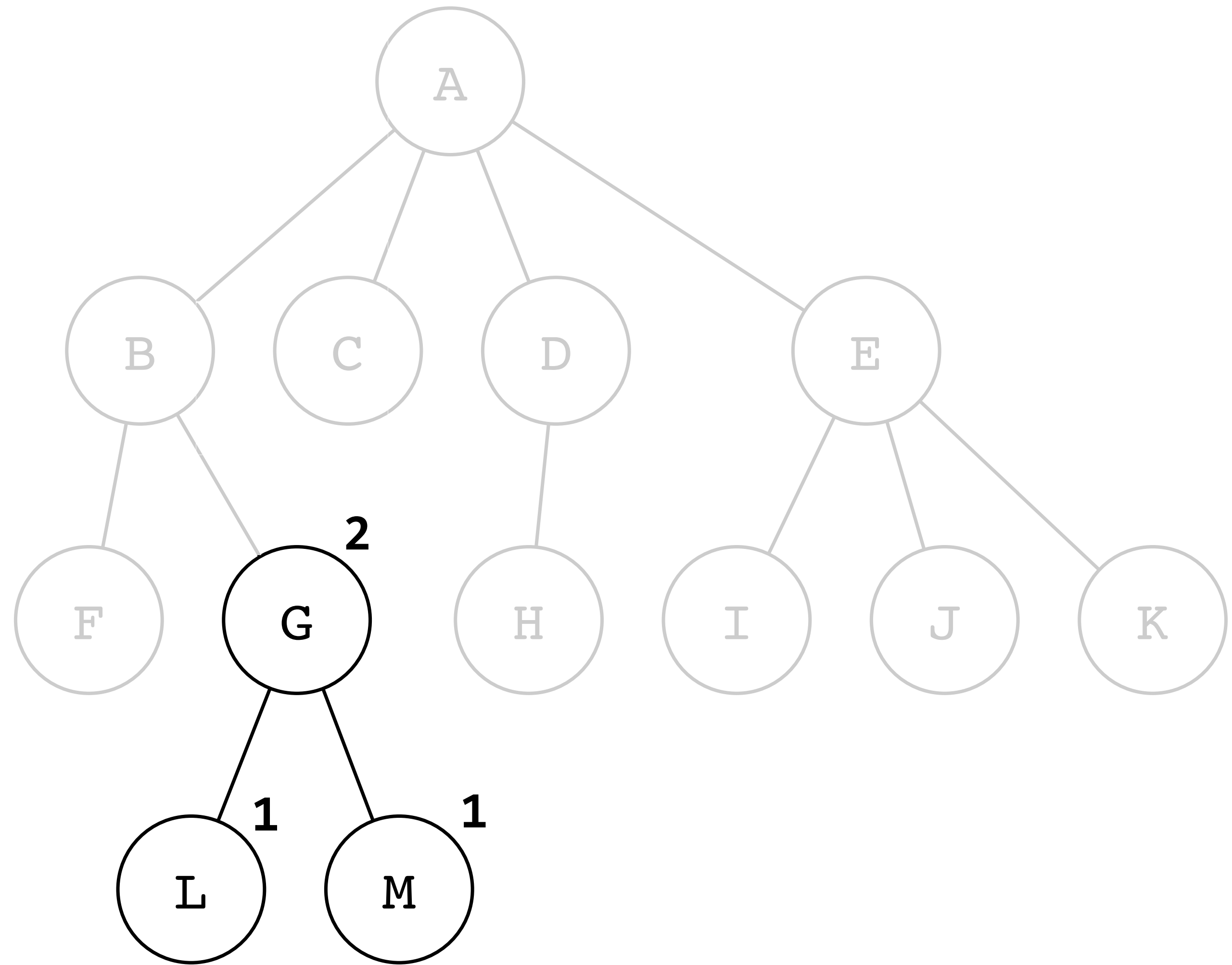
Sous-arbre

Arbre dont la racine est enfant dans un autre arbre

Hauteur d'un arbre

Niveau maximum parmi tous ses noeuds

$$H(v) = 1 + \max_{e \in \text{enfants}(v)} H(e)$$



Hauteur et sous-arbres



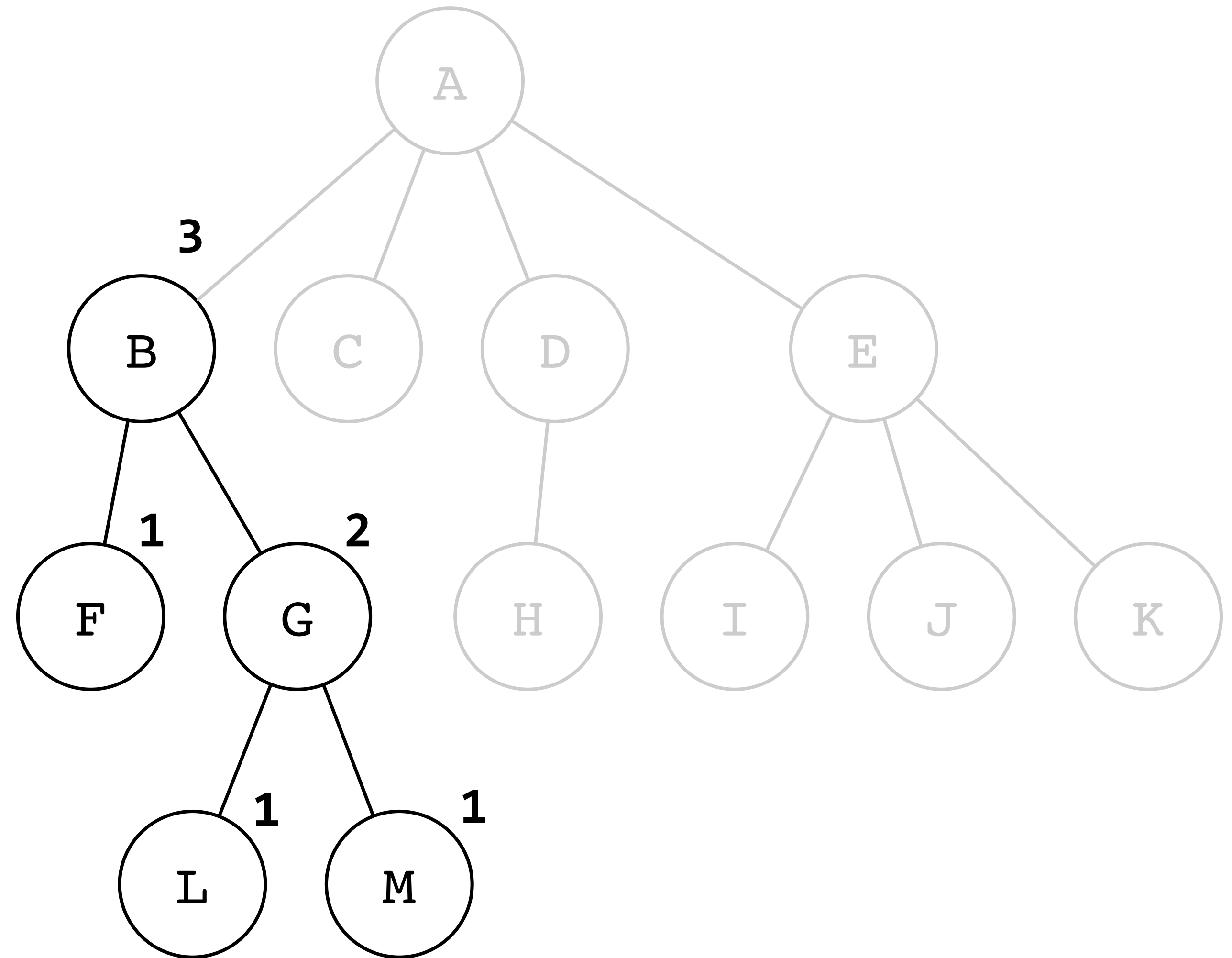
Sous-arbre

Arbre dont la racine est enfant dans un autre arbre

Hauteur d'un arbre

Niveau maximum
parmi tous ses noeuds

$$H(v) = 1 + \max_{e \in \text{enfants}(v)} H(e)$$



Hauteur et sous-arbres



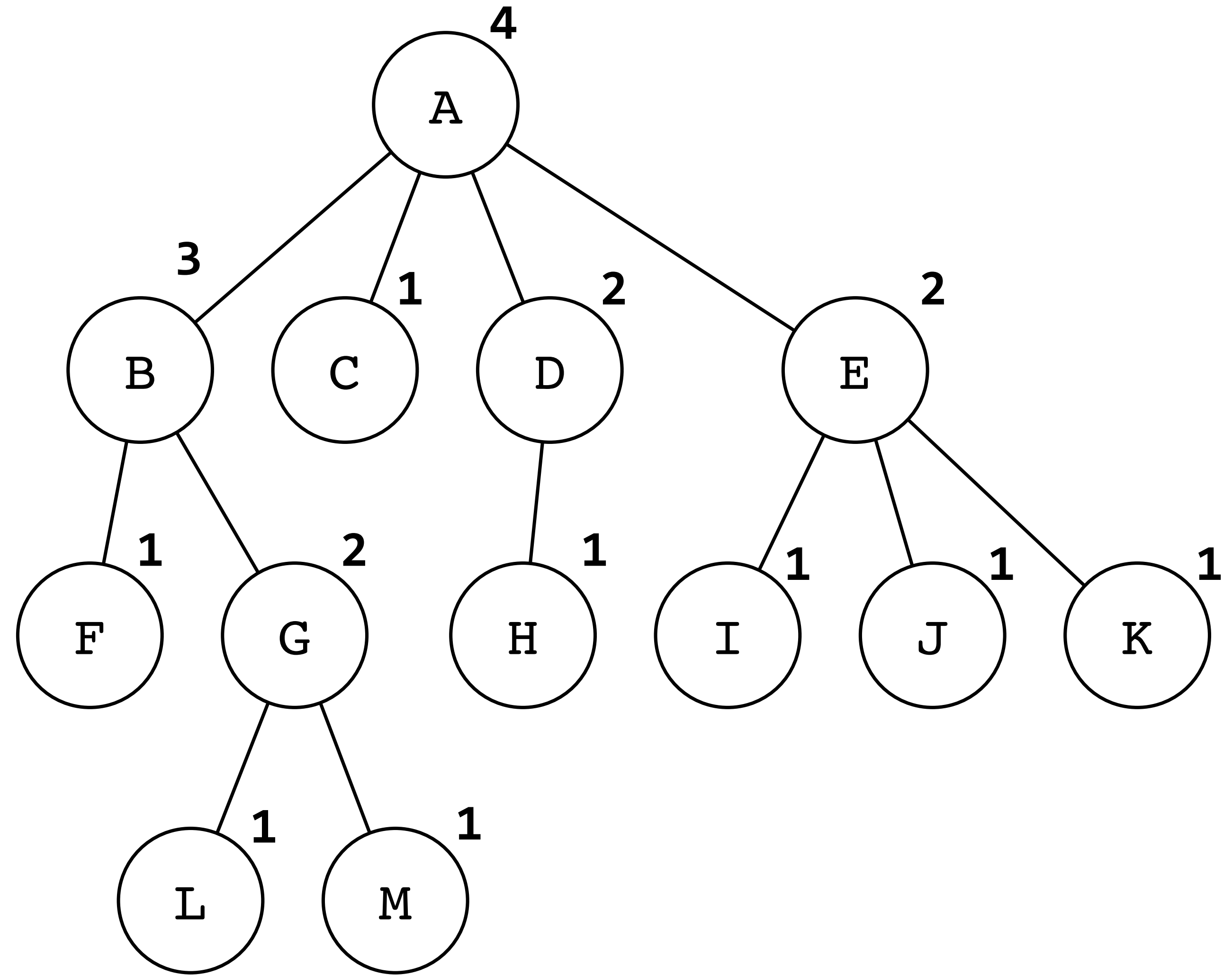
Sous-arbre

Arbre dont la racine est enfant dans un autre arbre

Hauteur d'un arbre

Niveau maximum
parmi tous ses noeuds

$$H(v) = 1 + \max_{e \in \text{enfants}(v)} H(e)$$



(Hauteur vs. Niveau)

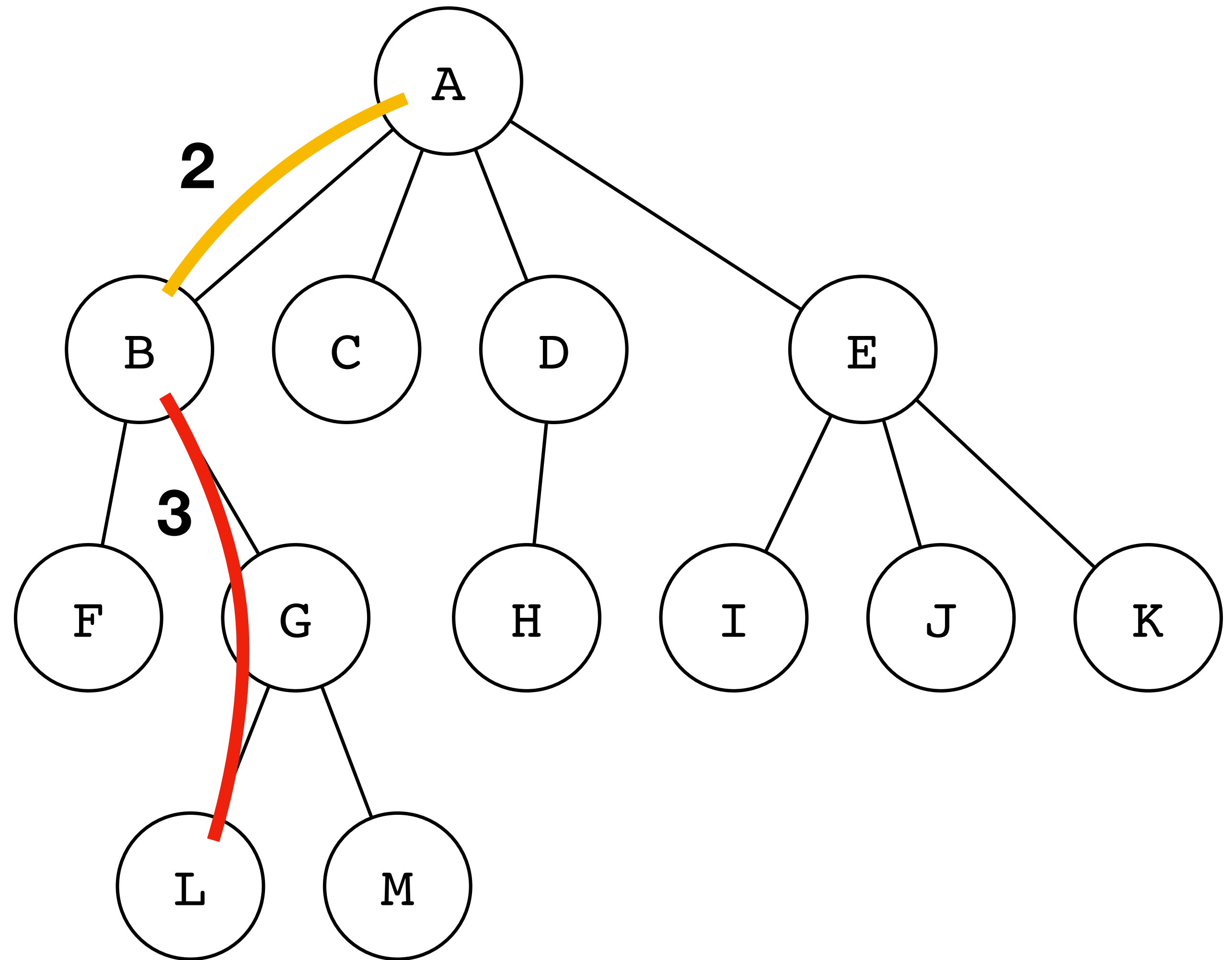


Niveau de n

Distance entre n et la racine de l'arbre.

Hauteur de n (*a.k.a.* $H(n)$)

Distance entre n et le plus éloigné de ses enfants.



(où “distance” entre A et B est le nombre de noeuds dans le chemin entre A et B)

Arbre vide



- Arbre sans aucun noeud
- A une hauteur 0

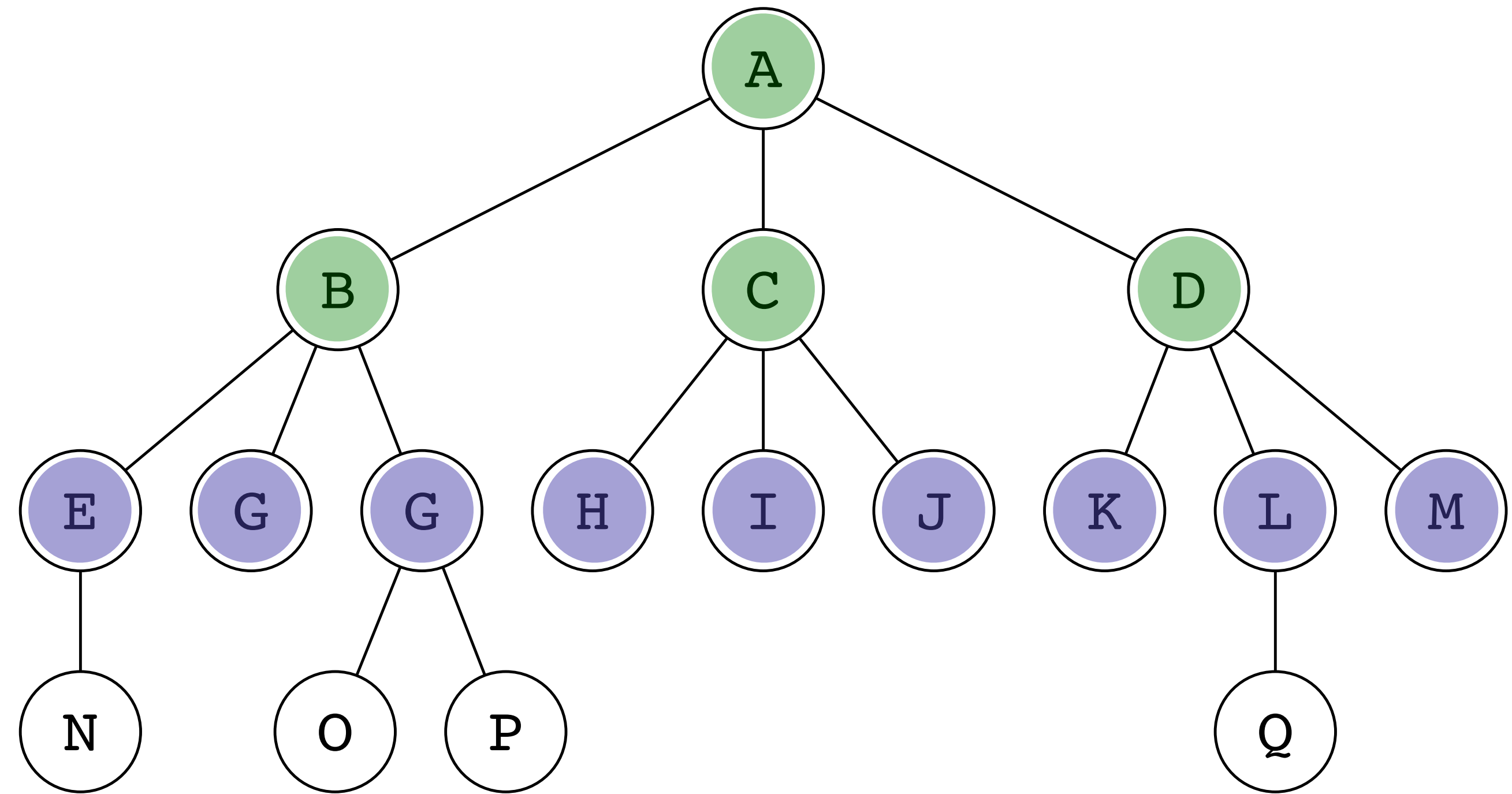
Arbre plein de degré d



Défini par un degré d (ici 3)

Si h est sa hauteur (ici 4) :

- Tout noeud de niveau $< h - 1$ a un degré d .
- Tout noeud de niveau $h - 1$ a un degré $\leq d$.
- (Tout noeud de niveau h est une feuille, forcément)



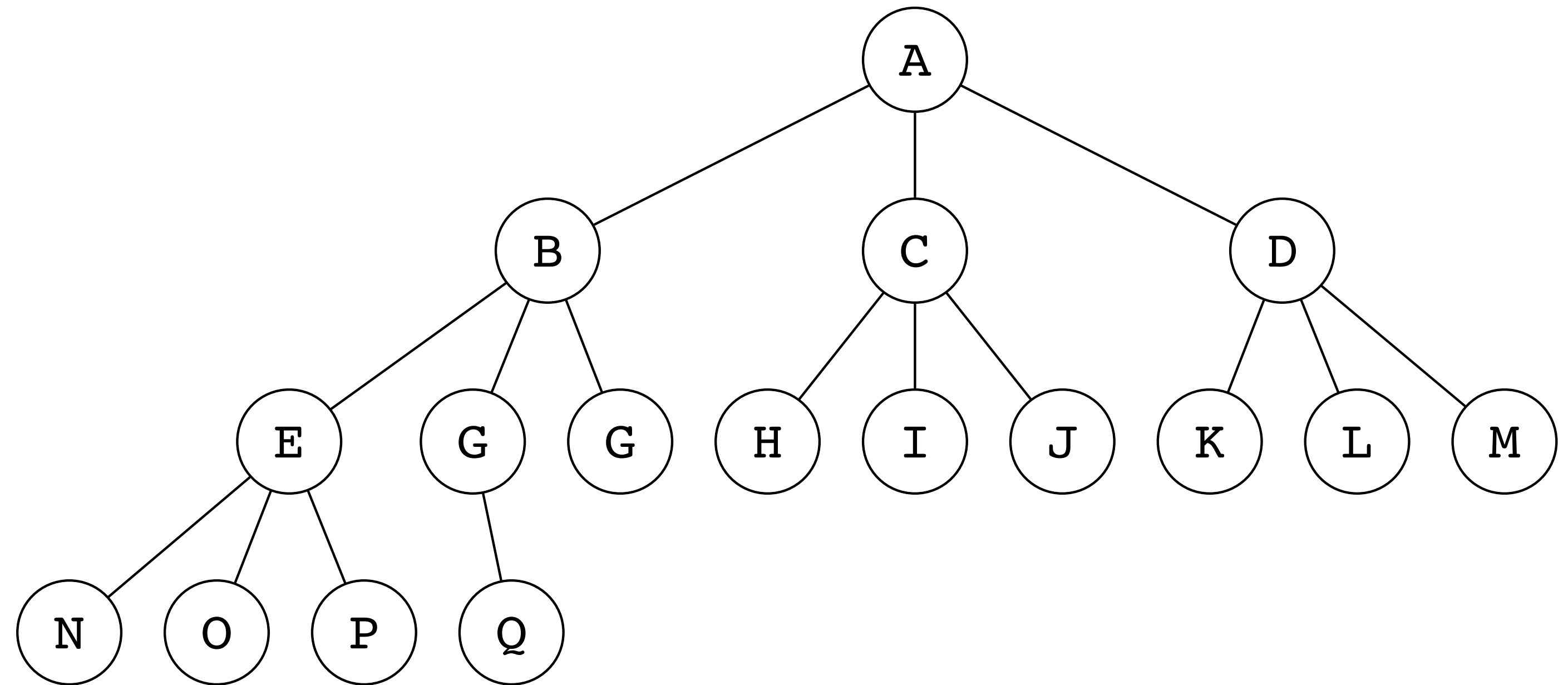
Arbre complet



Un arbre complet est:

- Plein, et
- Tel que le dernier niveau est rempli par la gauche.

(Au chapitre 4, un tas était un arbre binaire complet)



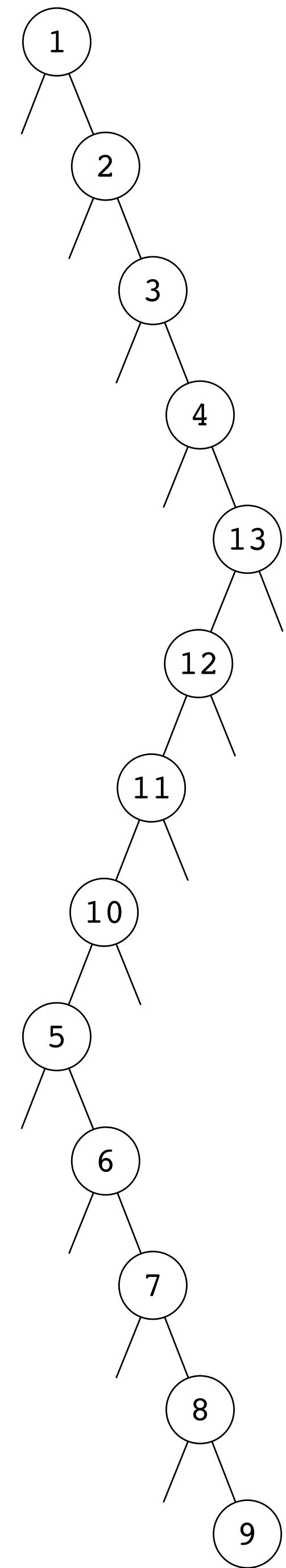
Arbre dégénéré

(a.k.a. de degré 1)



Un arbre de degré 1 est dit **dégénéré**.

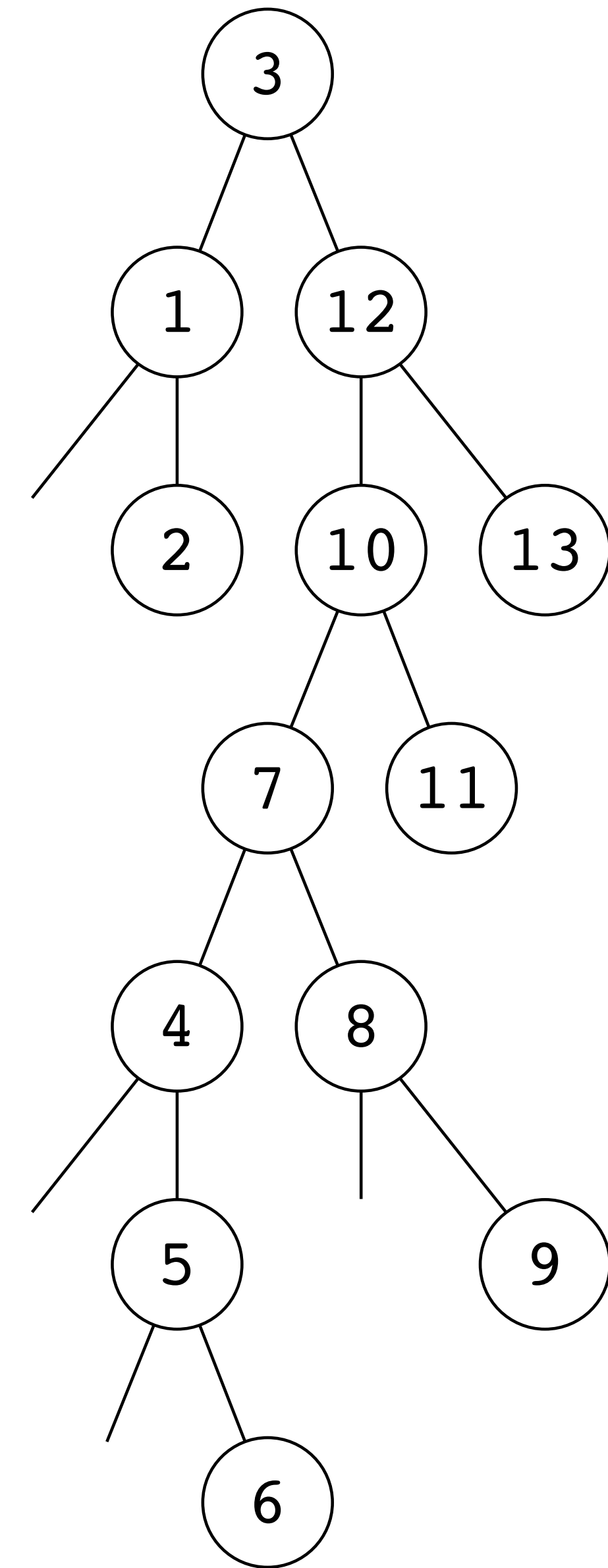
- Topologiquement équivalent à une liste chaînée
- Pire cas des algorithmes dépendant de la hauteur de l'arbre
- Parfois souhaitable comme structure intermédiaire (équilibrage par linéarisation / arborisation)



Arbre binaire



Un arbre de degré ≤ 2 est dit **binnaire**.
(Pourquoi “ $\leq$ ” ?)



Arbre binaire

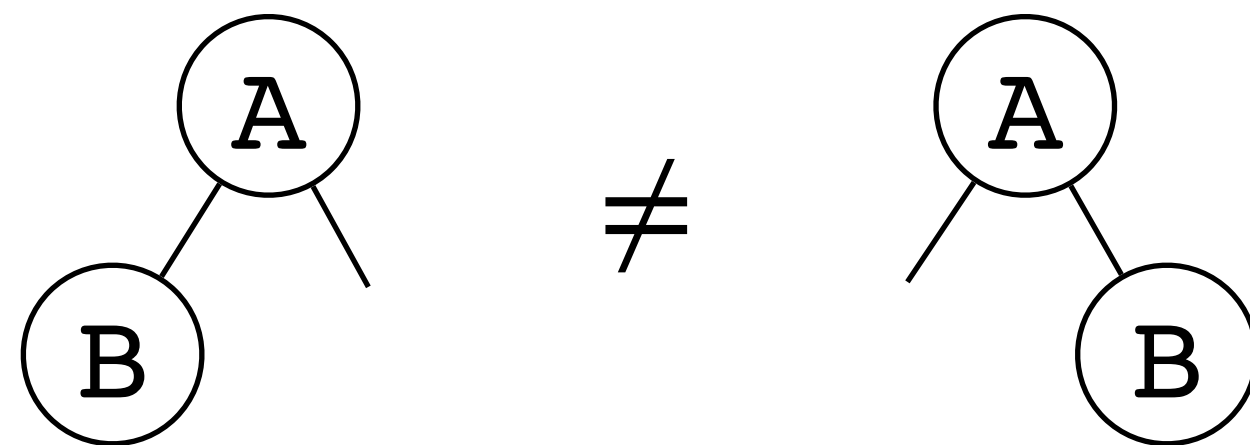


Un arbre de degré ≤ 2 est dit **binaire**.

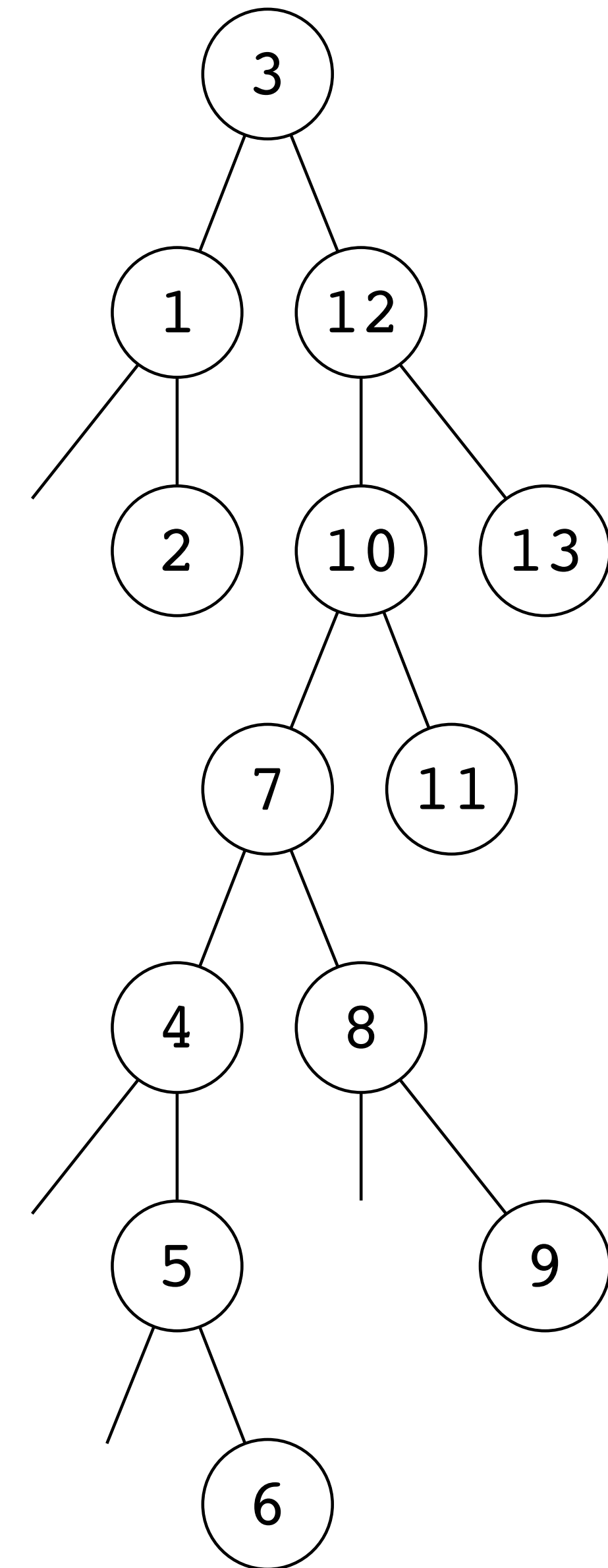
(Pourquoi “ $\leq$ ” ?)

*(Un arbre dégénéré est
un arbre binaire !)*

Distinction entre noeud gauche et droit,
même s’il n’y a qu’un noeud.



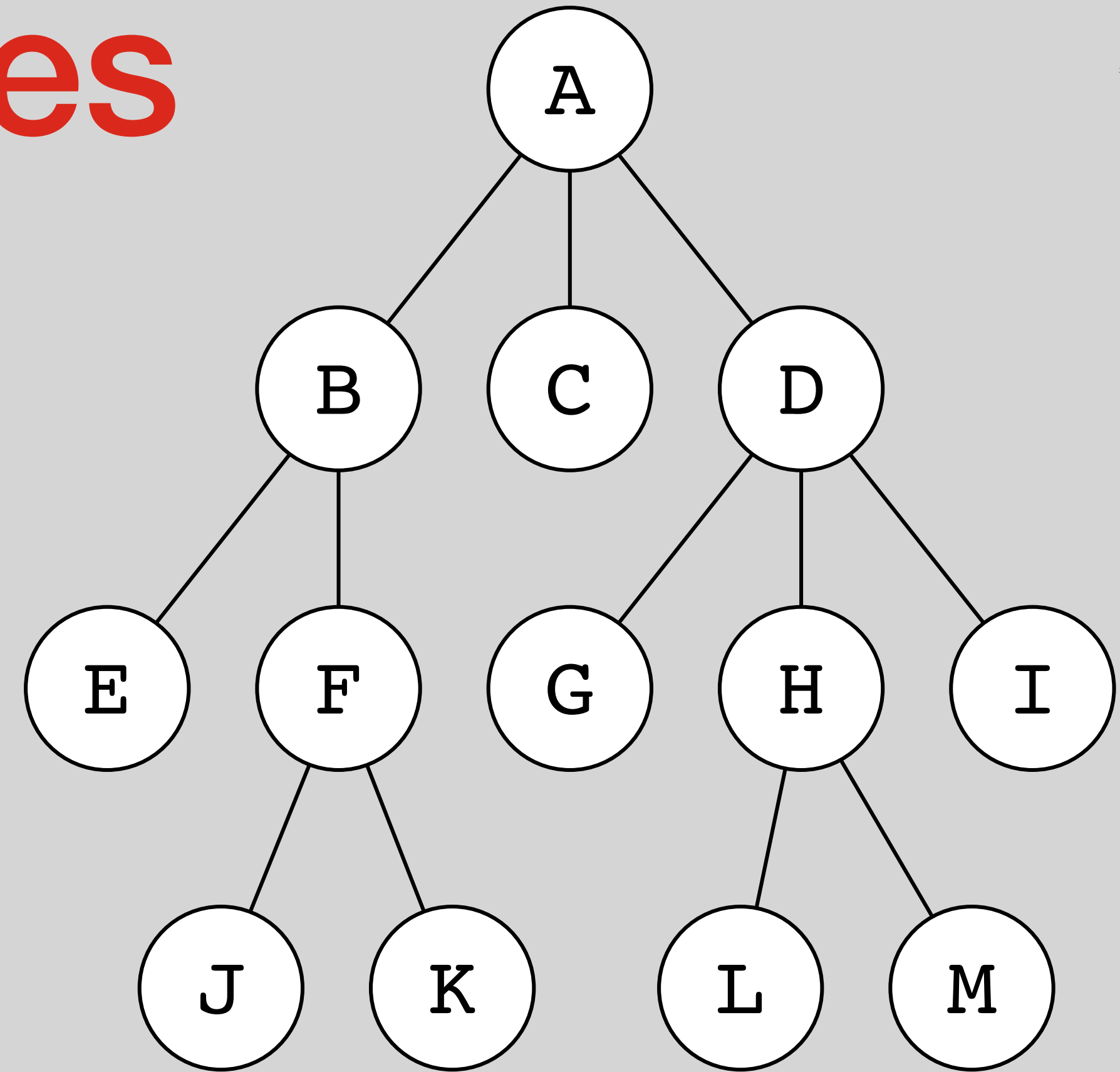
(Étudié en détail dans la suite du chapitre)



Exercices



- Quel est le degré de cet arbre ?
- Quel est le chemin du noeud F ?
- Quel est le niveau du noeud M ?
- Quelle est la racine ?
- Quels sont les noeuds internes ?
- Quelles sont les feuilles ?
- Quelle est la racine du sous-arbre dans lequel F a un niveau de 2 ?
- Quels sous-arbres sont binaires ? Binaires pleins ? Binaires complets ?
- Que doit-on changer pour que cet arbre soit binaire ?
 - Serait-ce alors un arbre plein ? Complet ?



Solution



- Quel est le degré de cet arbre ? 3, à cause de A et D
- Quel est le chemin du noeud F ? A B F
- Quel est le niveau du noeud M ? 4
- Quelle est la racine ? A
- Quels sont les noeuds internes ? A B D F H
- Quelles sont les feuilles ? C E G I J K L M
- Quelle est la racine du sous-arbre dans lequel F a un niveau de 2 ? B
- Quels sous-arbres sont binaires ? Binaires pleins ? Binaires complets ?

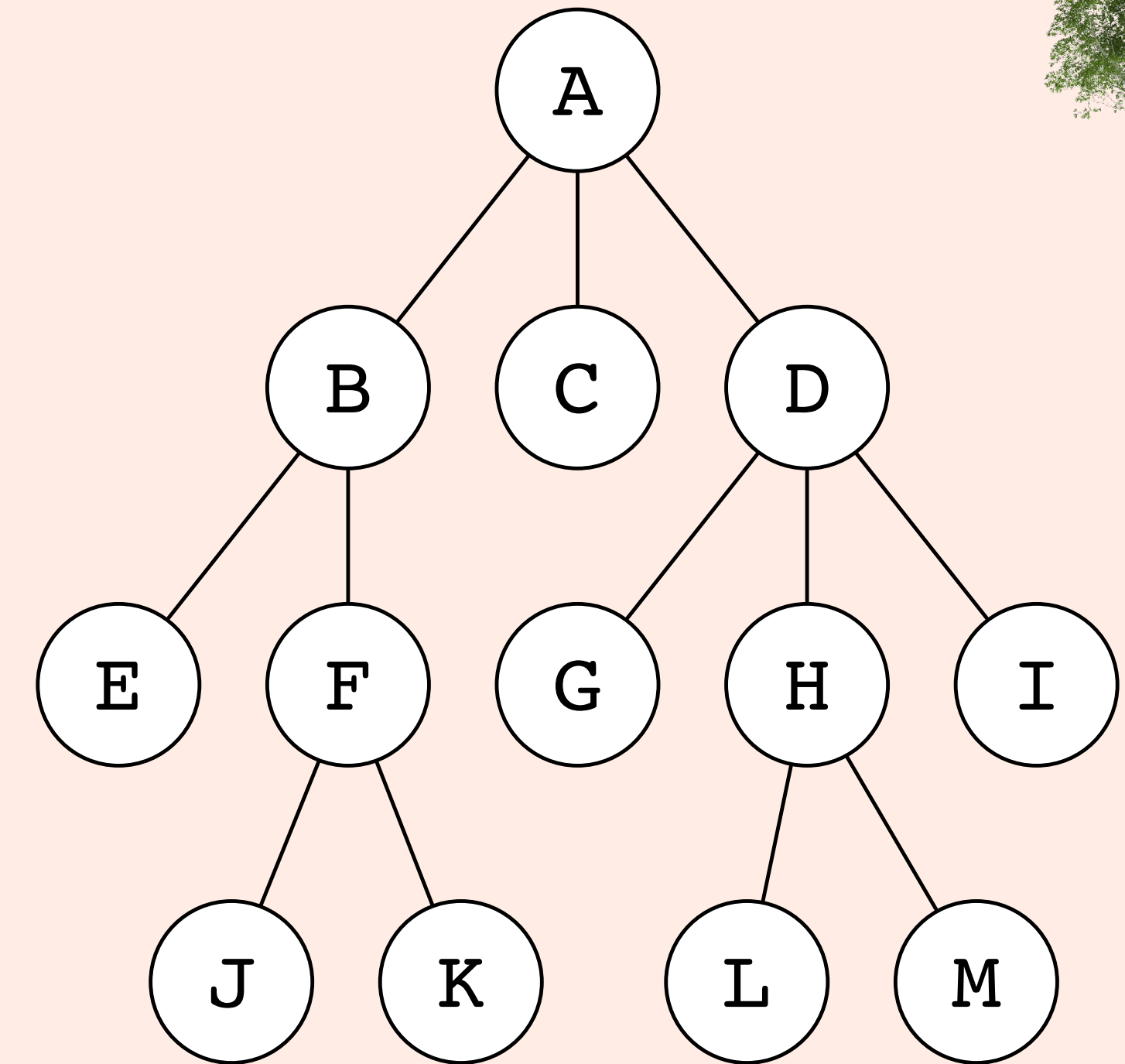
Binaires complets : Les feuilles, FJK, HLM.

Binaires pleins mais pas complets : BEFJK.

Binaires mais pas pleins : Aucun

- Que doit-on changer pour que cet arbre soit binaire ? Retirer C et G (ou I)
 - Serait-ce alors un arbre plein ? Complet ?

Il serait plein mais pas complet : les enfants ne seraient pas tout à gauche (à cause de E)



2. Mise en oeuvre





Un noeud par élément, qui stocke

- son étiquette
- les pointeurs vers ses enfants (*dans structure au choix*)
- (*optionnel : un pointeur vers le noeud parent*)

```
structure Noeud<T>  
    T étiquette  
    Tableau<Noeud*> enfants
```

```
structure Noeud<T>  
    T étiquette  
    Liste<Noeud*> enfants
```

```
structure Noeud<T>  
    T étiquette  
    Noeud* parent  
    Tableau<Noeud*> enfants
```

```
structure Noeud<T>  
    T étiquette  
    Noeud* parent  
    Liste<Noeud*> enfants
```

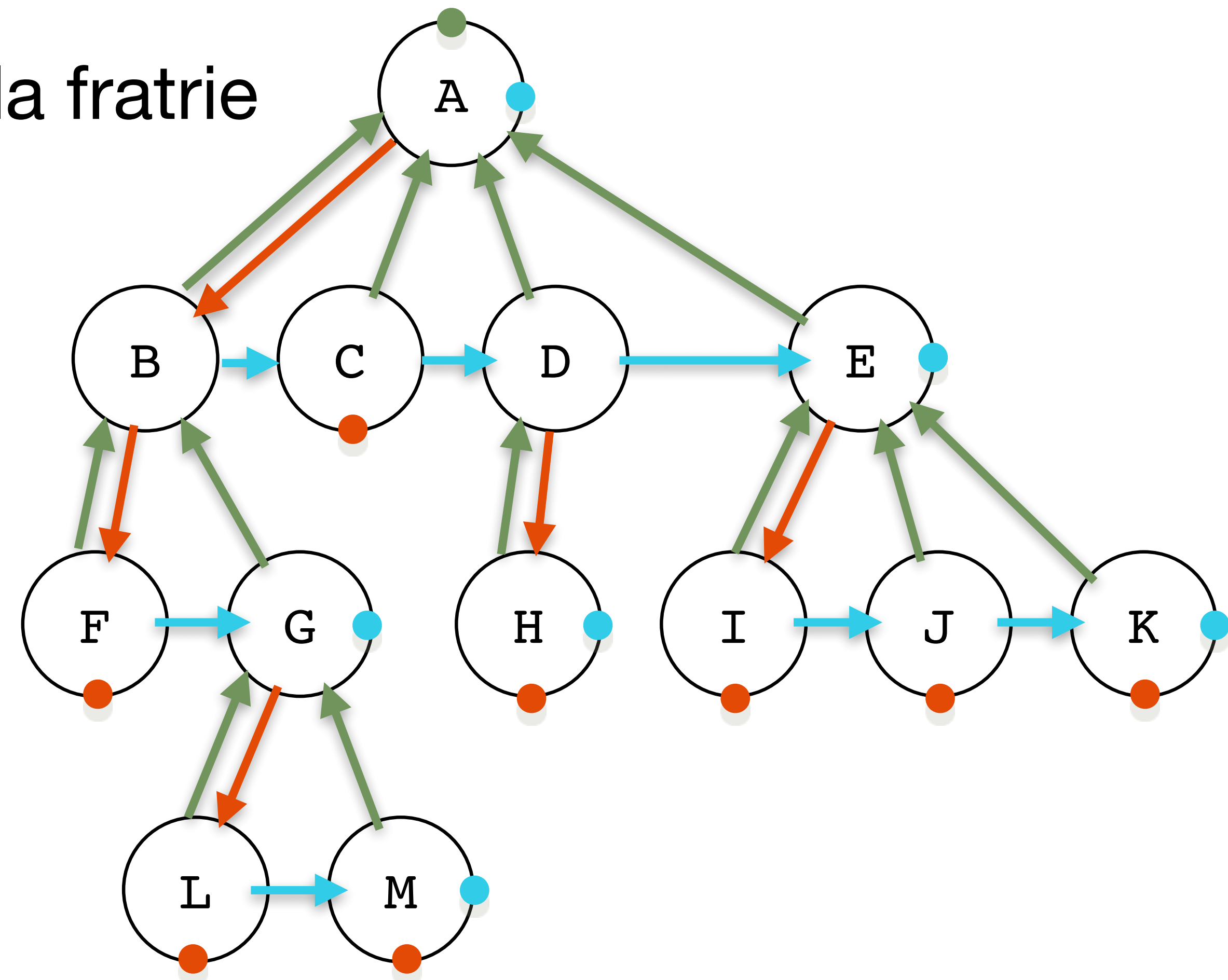
Sans utiliser de structures linéaires



- Un lien **descendant** vers l'ainé des enfants
- Un lien **horizontal** vers le puiné dans la fratrie
- Un lien **montant** vers le parent

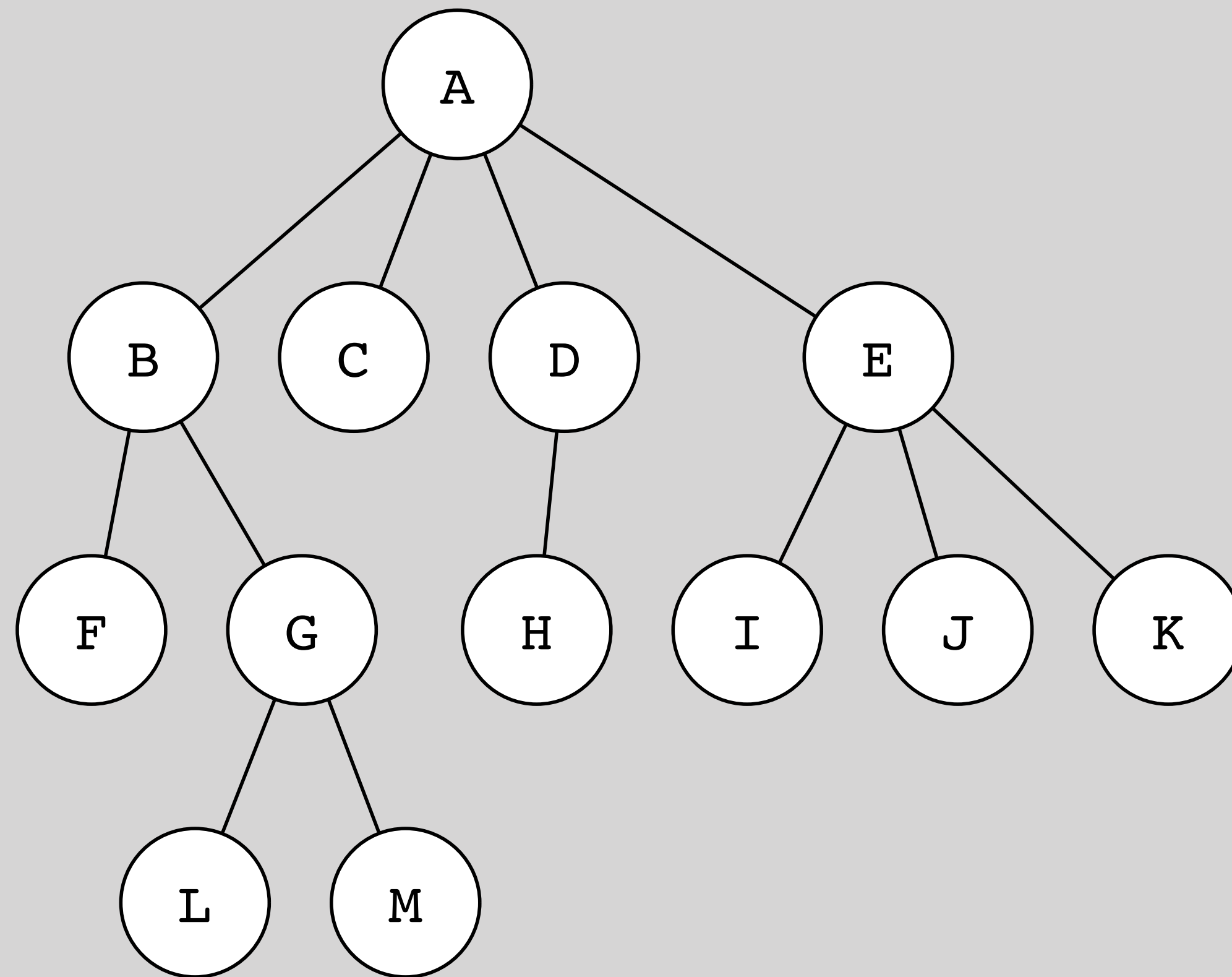
```

structure Noeud<T>
  T étiquette
  Noeud* ainé
  Noeud* puiné
  Noeud* parent
  
```



Exercice

Quel est le nombre minimum de pointeurs et/ou `size_t` utilisés par les structures ci-contre pour stocker l'arbre ci-dessous?



```
template <typename T> struct Node1 {  
    T etiquette;  
    Node1* parent;  
    array<Node1*,DEGRE> enfants;  
};
```

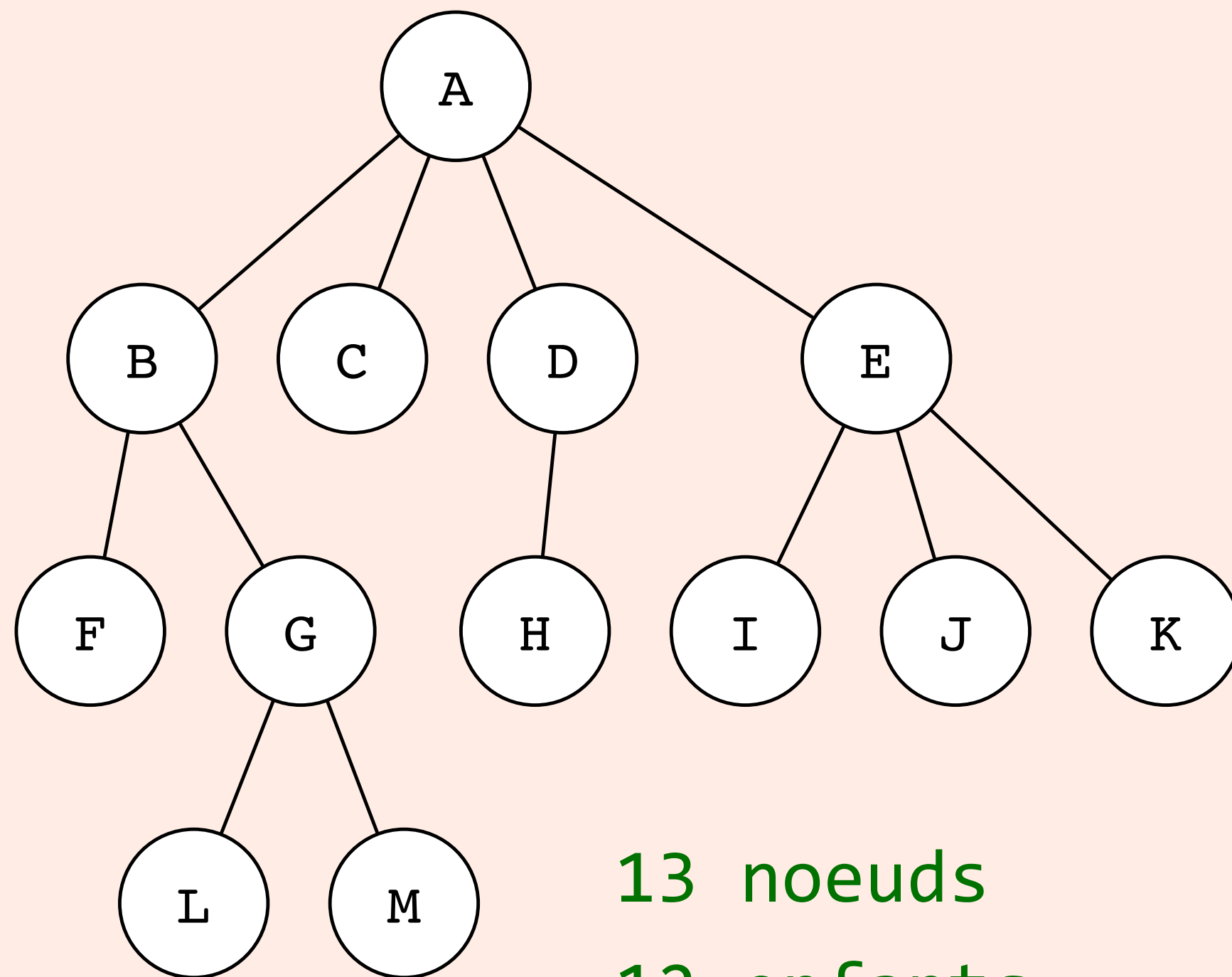
```
template <typename T> struct Node2 {  
    T etiquette;  
    Node2* parent;  
    vector<Node2*> enfants;  
};
```

```
template <typename T> struct Node3 {  
    T etiquette;  
    Node3* parent;  
    list<Node3*> enfants;  
};
```

```
template <typename T> struct Node4 {  
    T etiquette;  
    Node4* parent;  
    forward_list<Node4*> enfants;  
};
```

```
template <typename T> struct Node5 {  
    T etiquette;  
    Node5* parent;  
    Node5* aîné;  
    Node5* puîné;  
};
```

Quel est le nombre minimum de pointeurs et/ou `size_t` utilisés par les structures ci-contre pour stocker l'arbre ci-dessous?



13 noeuds

12 enfants

Exercice

```
template <typename T> struct Node1 {
    T etiquette;
    Node1* parent;
    array<Node1*,DEGRE> enfants;
};
```

```
template <typename T> struct Node2 {
    T etiquette;
    Node2* parent;
    vector<Node2*> enfants;
};
```

```
template <typename T> struct Node3 {
    T etiquette;
    Node3* parent;
    list<Node3*> enfants;
};
```

```
template <typename T> struct Node4 {
    T etiquette;
    Node4* parent;
    forward_list<Node4*> enfants;
};
```

```
template <typename T> struct Node5 {
    T etiquette;
    Node5* parent;
    Node5* aîné;
    Node5* puîné;
};
```



Degré 4

$$13 \times (4+1) = 65$$

Pour chaque
noeud

1 parent
4 pour l'array

Chaque vector de n coute 3+n

$$13 \times (3+1) + 12 = 64$$

Pour chaque
noeud

1 parent
3 pour le vector

Le pointeur de
chaque enfant
dans son parent

Chaque liste de n coute 3+3n

$$13 \times (3+1) + 12 \times 3 = 88$$

Pour chaque
noeud

1 parent
3 pour la liste

Le pointeur de
chaque enfant
dans son parent

Le cout de stockage
de ce pointeur dans
la liste

Chaque liste de n coute 1+2n

$$13 \times (1+1) + 12 \times 2 = 50$$

Pour chaque
noeud

1 parent
1 pour la liste

Le pointeur de
chaque enfant
dans son parent

Le cout de stockage
de ce pointeur dans
la liste

3 ptr par élément

$$13 \times 3 = 39$$

Pour chaque
noeud

3 pointeurs

3. Parcours





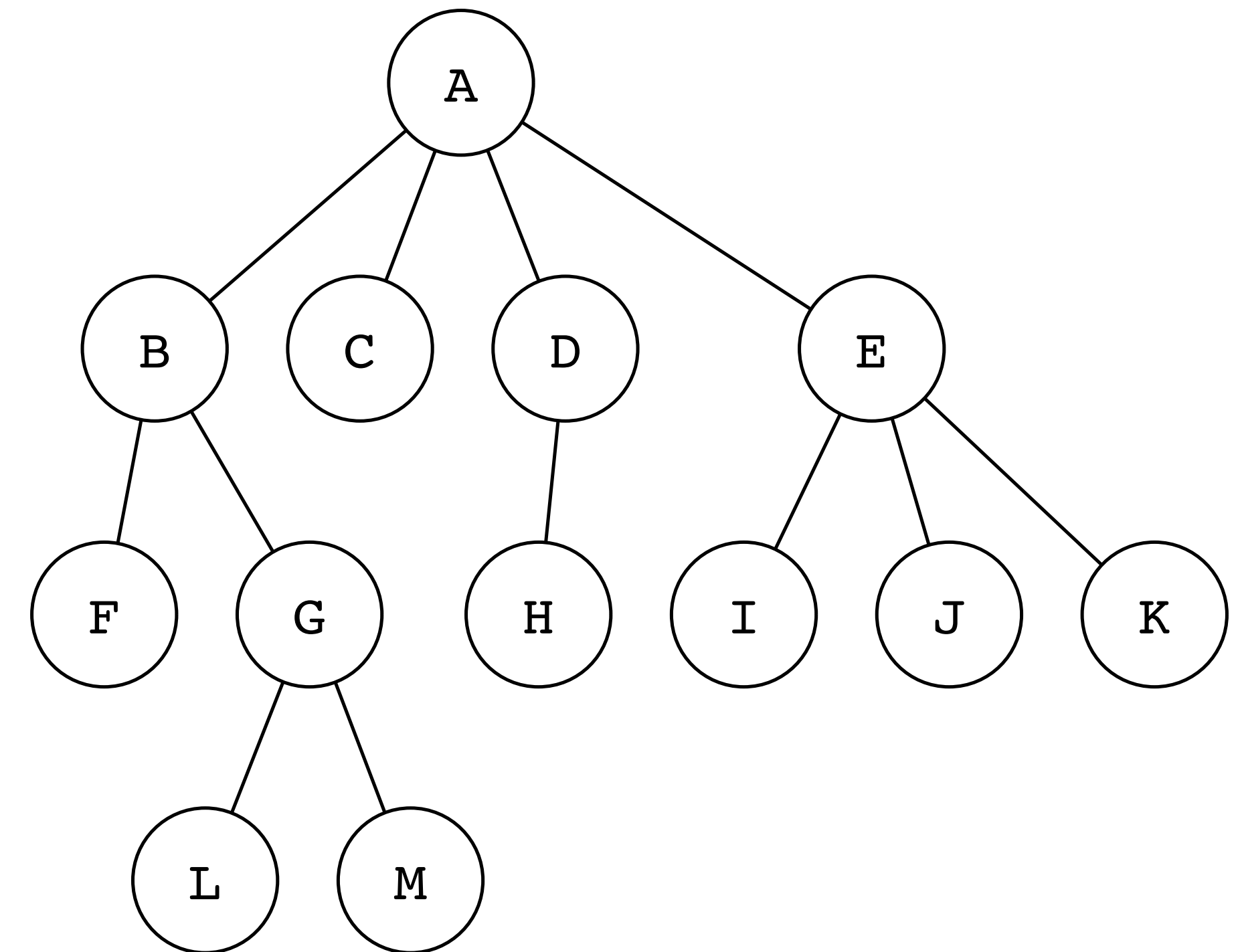
Comment **visiter** tous les noeuds d'un arbre ?

Récurivement :

- Visiter la racine, puis
- Pour chaque enfant, parcourir le sous-arbres dont il est la racine.

Cas trivial :

Rien à faire pour un arbre vide.



```
fonction profondeur (r, fn)
  si r != ∅
    fn(r) // visite de r
    pour tout enfant e de r
      profondeur(e, fn)
```

(Primogéniture)



👑 King Albert II (born 1934)

👑 King Philippe (b. 1960)

(1) Princess Elisabeth, Duchess of Brabant (b. 2001)^[1]

(2) Prince Gabriel (b. 2003)^[1]

(3) Prince Emmanuel (b. 2005)^[1]

(4) Princess Eléonore (b. 2008)^[1]

(5) Princess Astrid, Archduchess of Austria-Este (b. 1962)^[1]

(6) Prince Amedeo, Archduke of Austria-Este (b. 1986)^[1]

(7) Archduchess Anna Astrid of Austria-Este (b. 2016)

(8) Archduke Maximilian of Austria-Este (b. 2019)

(9) Princess Maria Laura, Archduchess of Austria-Este (b. 1988)^[1]

(10) Prince Joachim, Archduke of Austria-Este (b. 1991)^[1]

(11) Princess Luisa Maria, Archduchess of Austria-Este (b. 1995)^[1]

(12) Princess Laetitia Maria, Archduchess of Austria-Este (b. 2003)^[1]

(13) Prince Laurent (b. 1963)^[1]

(14) Princess Louise (b. 2004)^[1]

(15) Prince Nicolas (b. 2005)^[1]

(16) Prince Aymeric (b. 2005)^[1]

Qui succèdera à Philippe, roi des Belges ?

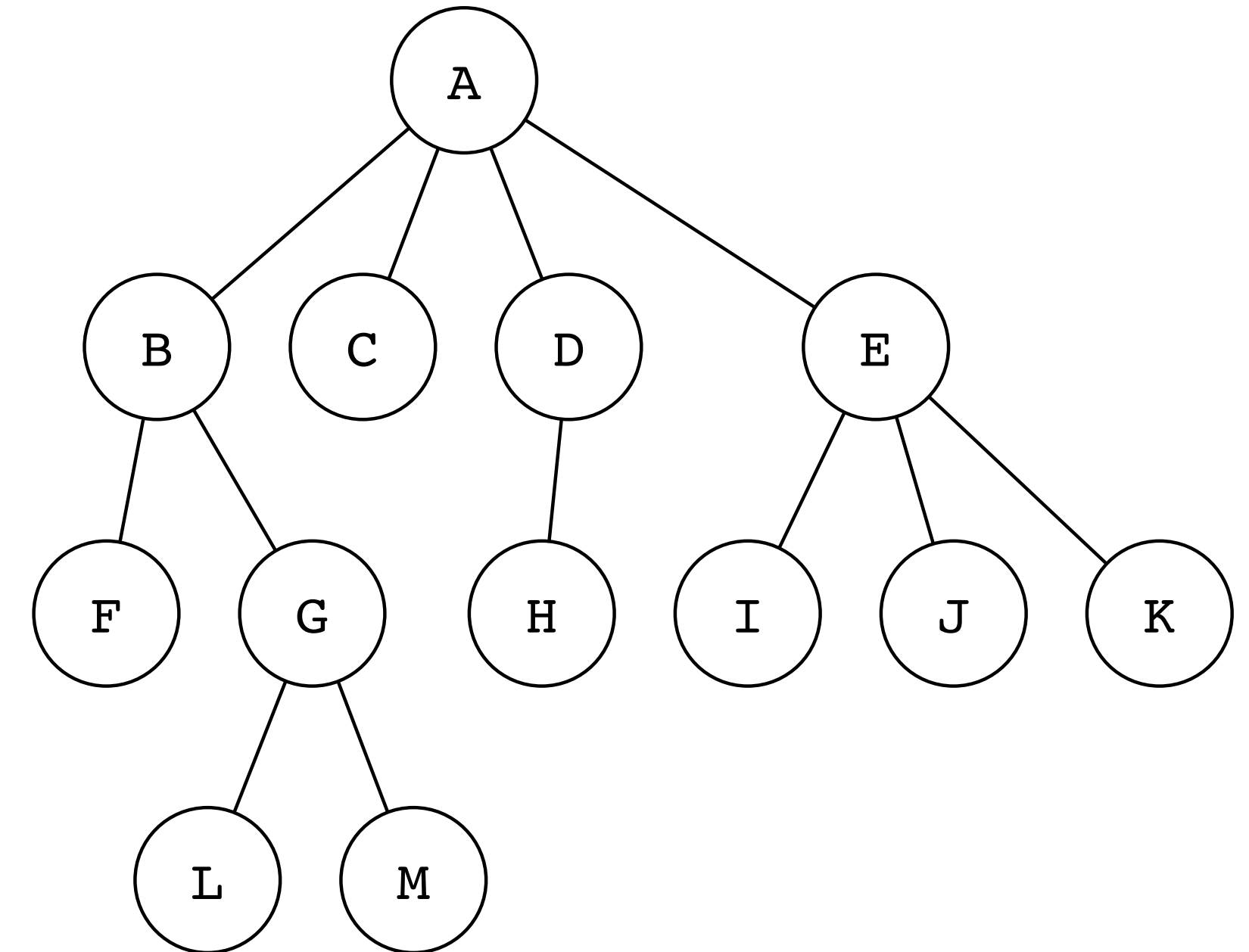


Parcours post-ordonné



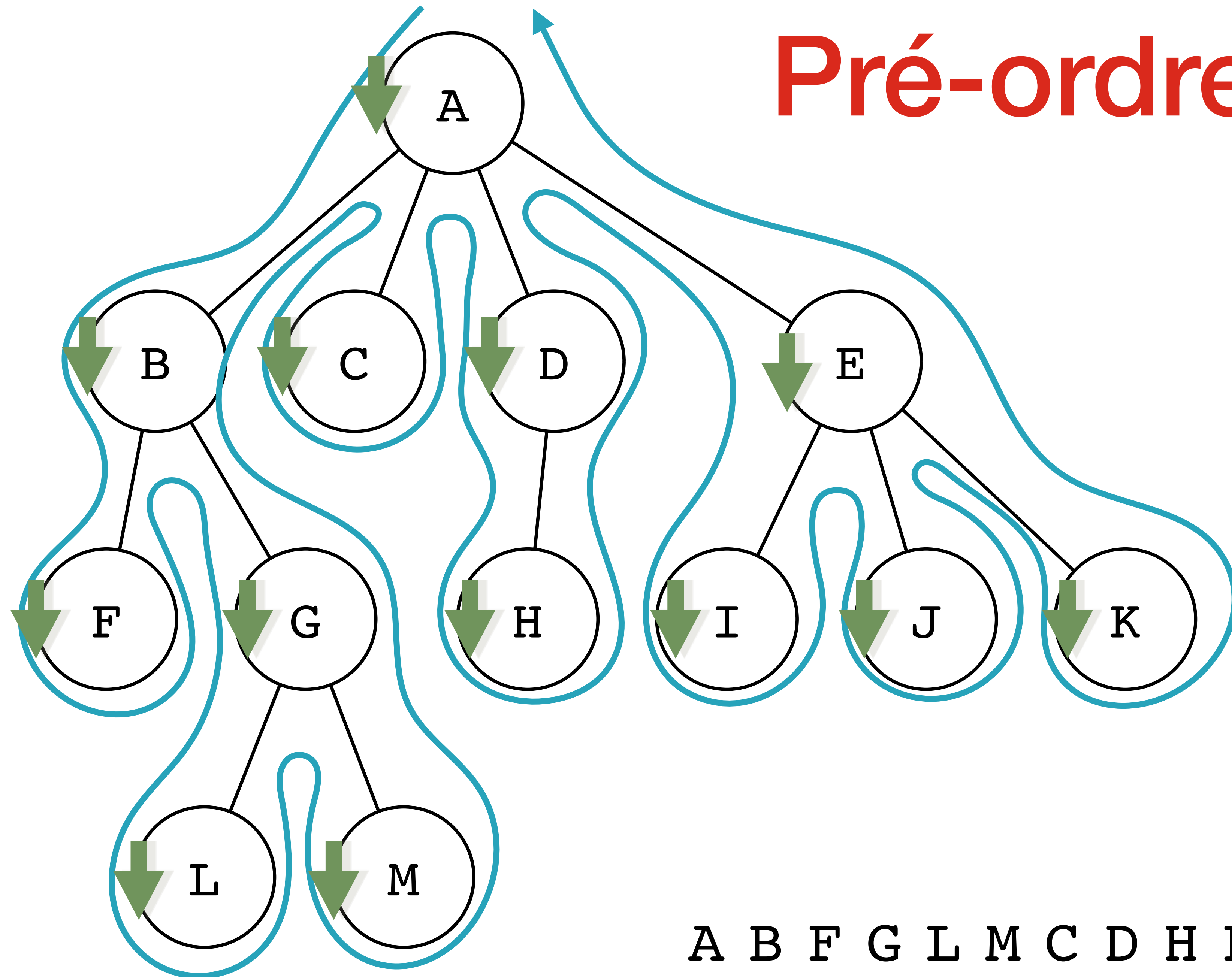
Rien n'impose *quand* on visite la racine :
peut être **avant** ou **après** les enfants.

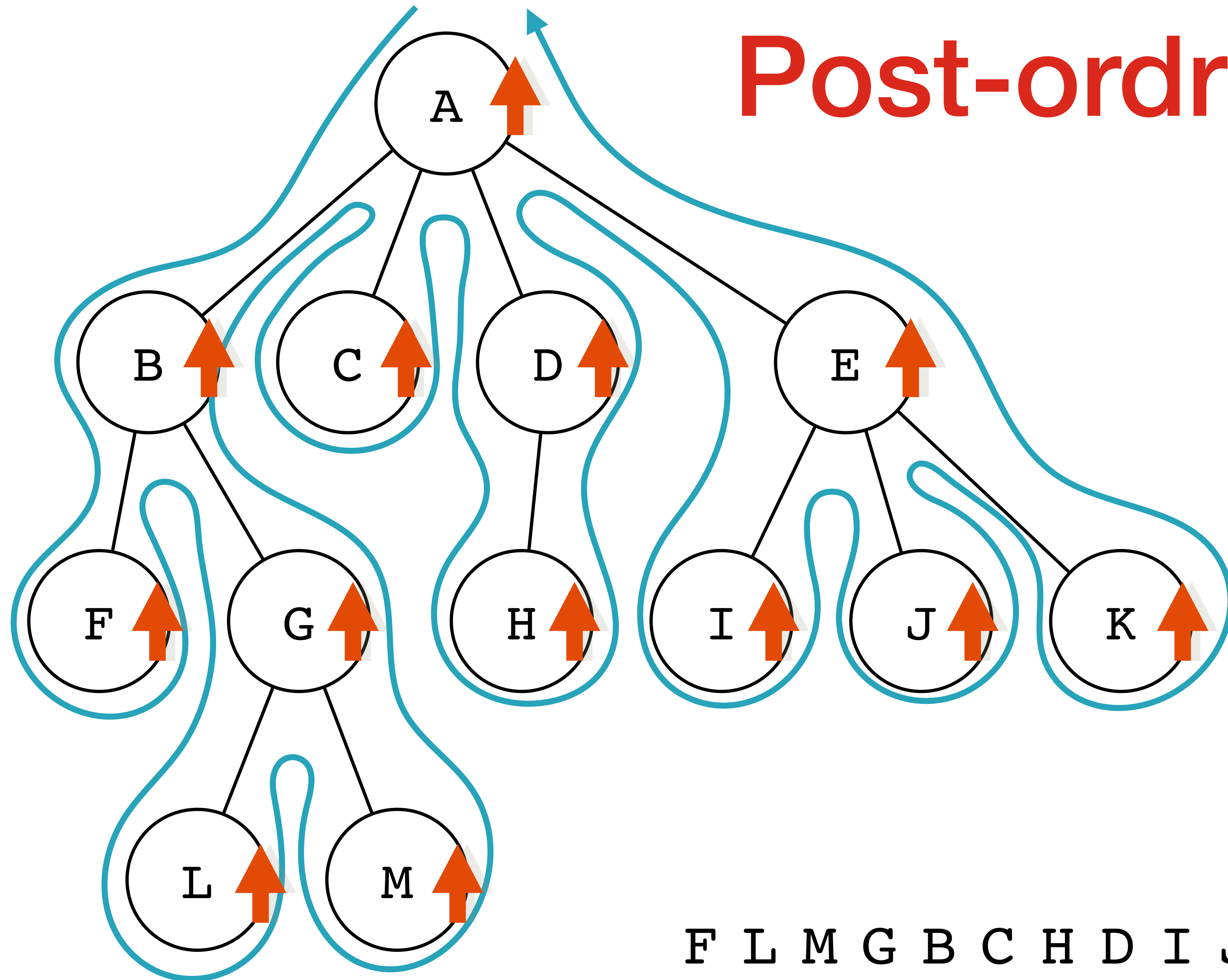
*Parfois utile, par exemple
pour détruire la structure.*



```
fonction préordre (r, fn)
  si r != ∅
    fn(r) // visite de r
    pour tout enfant e de r
      profondeur(e, fn)
```

```
fonction postordre (r, fn)
  si r != ∅
    pour tout enfant e de r
      postordre(e, fn)
    fn(r) // visite de r
```







- Copier en pré-ordre

```
fonction copier (r)
  si r != ∅
    r2 ← nouveau noeud d'étiquette r.étiquette
    pour tout enfant e de r
      e2 ← copier(e)
      Ajouter e2 aux enfants de r2
  retourner r2
```

- Détruire en post-ordre

```
fonction détruire (ref r)
  si r != ∅
    pour tout enfant e de r
      détruire(e)
  effacer le noeud r
```

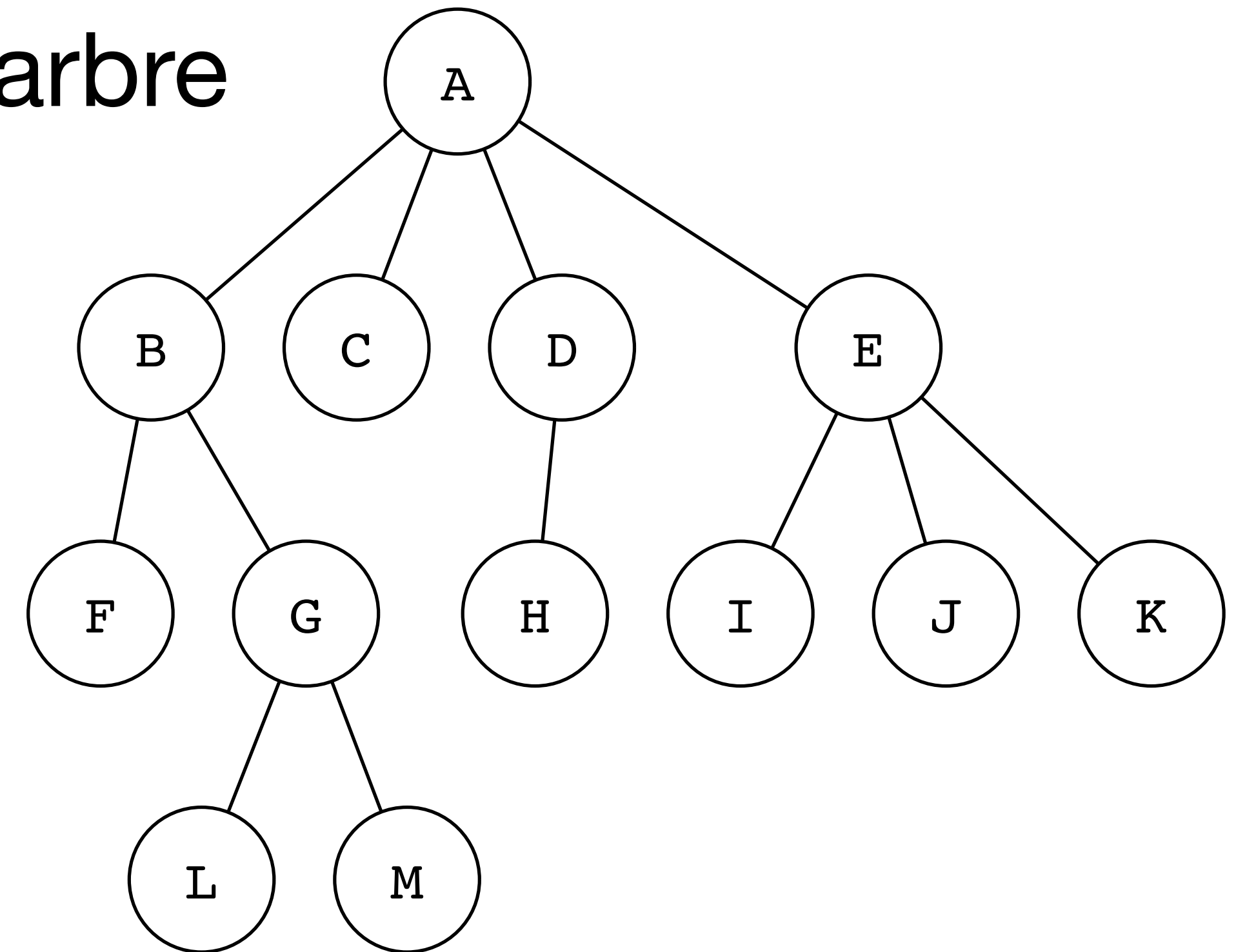


Comment visiter tous les noeuds d'un arbre
niveau par niveau ?

Comment savoir que

- F vient après E
- L vient après K

Complicé récursivement...



À chaque noeud visité, on veut noter ses enfants “pour plus tard”.

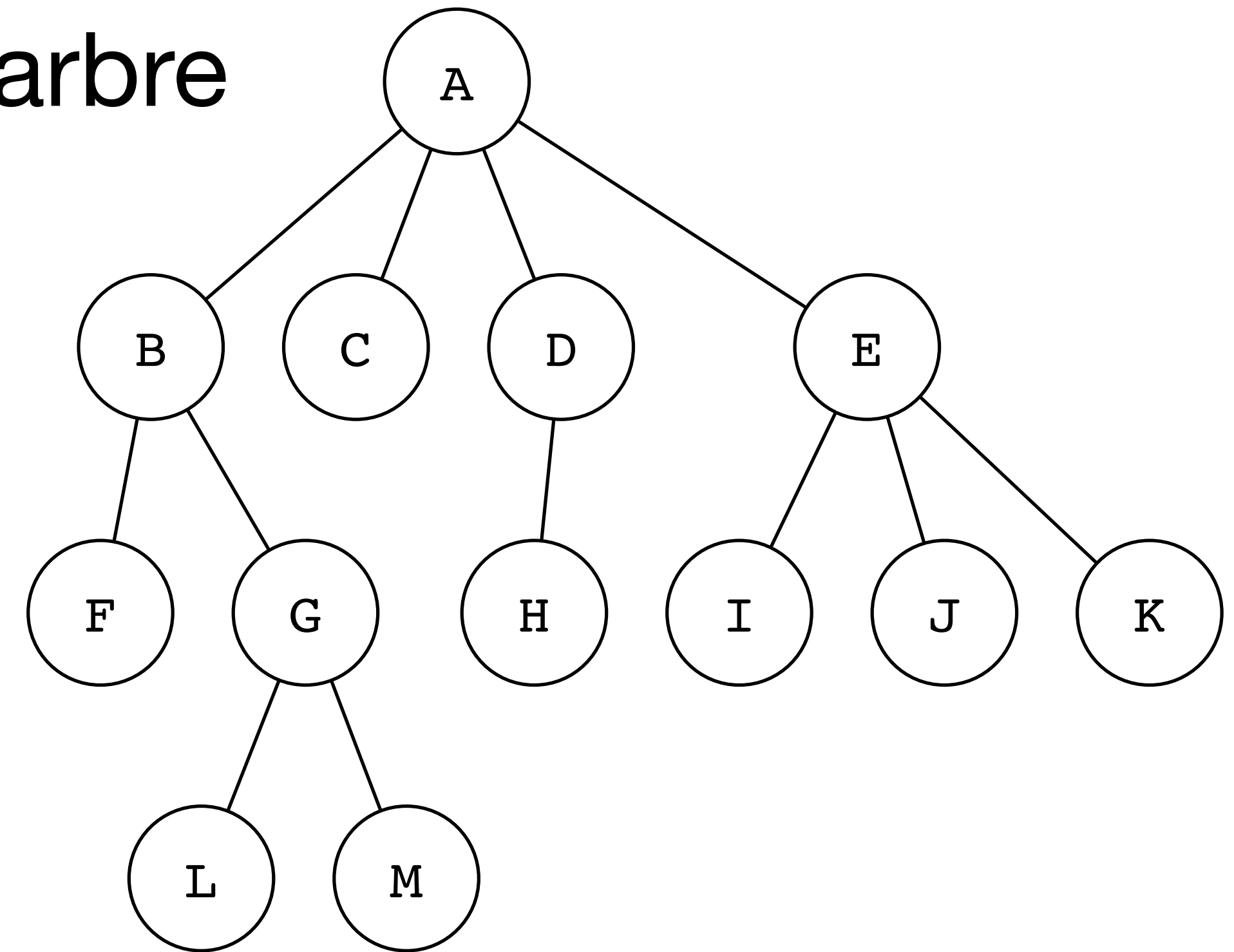


Comment visiter tous les noeuds d'un arbre
niveau par niveau ?

Comment savoir que

- F vient après E
- L vient après K

Complicé récursivement...



À chaque noeud visité, on veut noter ses enfants “pour plus tard”.
mettre ses enfants en file d’attente.
utiliser une queue.



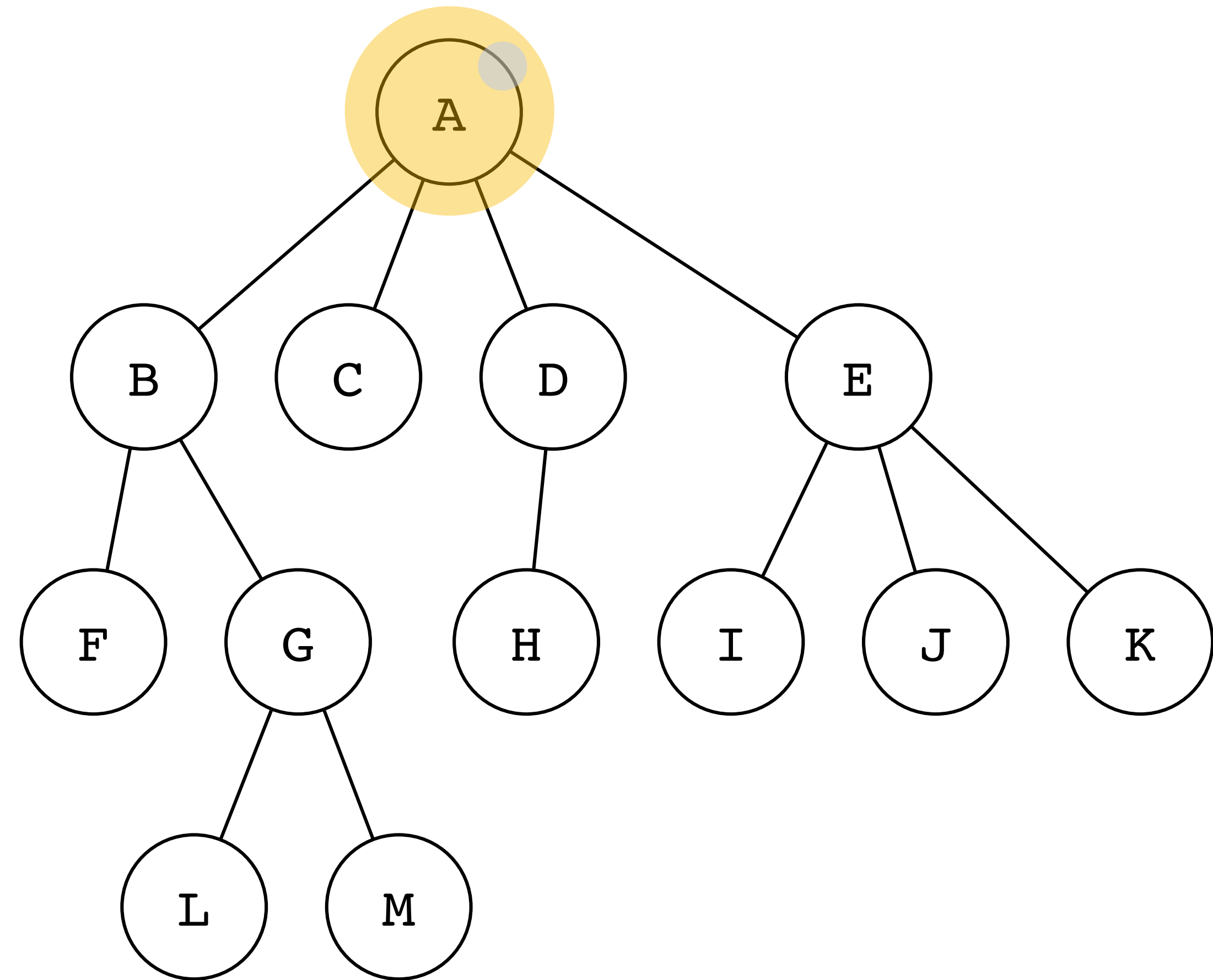
À chaque noeud visité :

- Chaque enfant est mis en file d'attente.

File d'attente ●

Ordre de visite ●

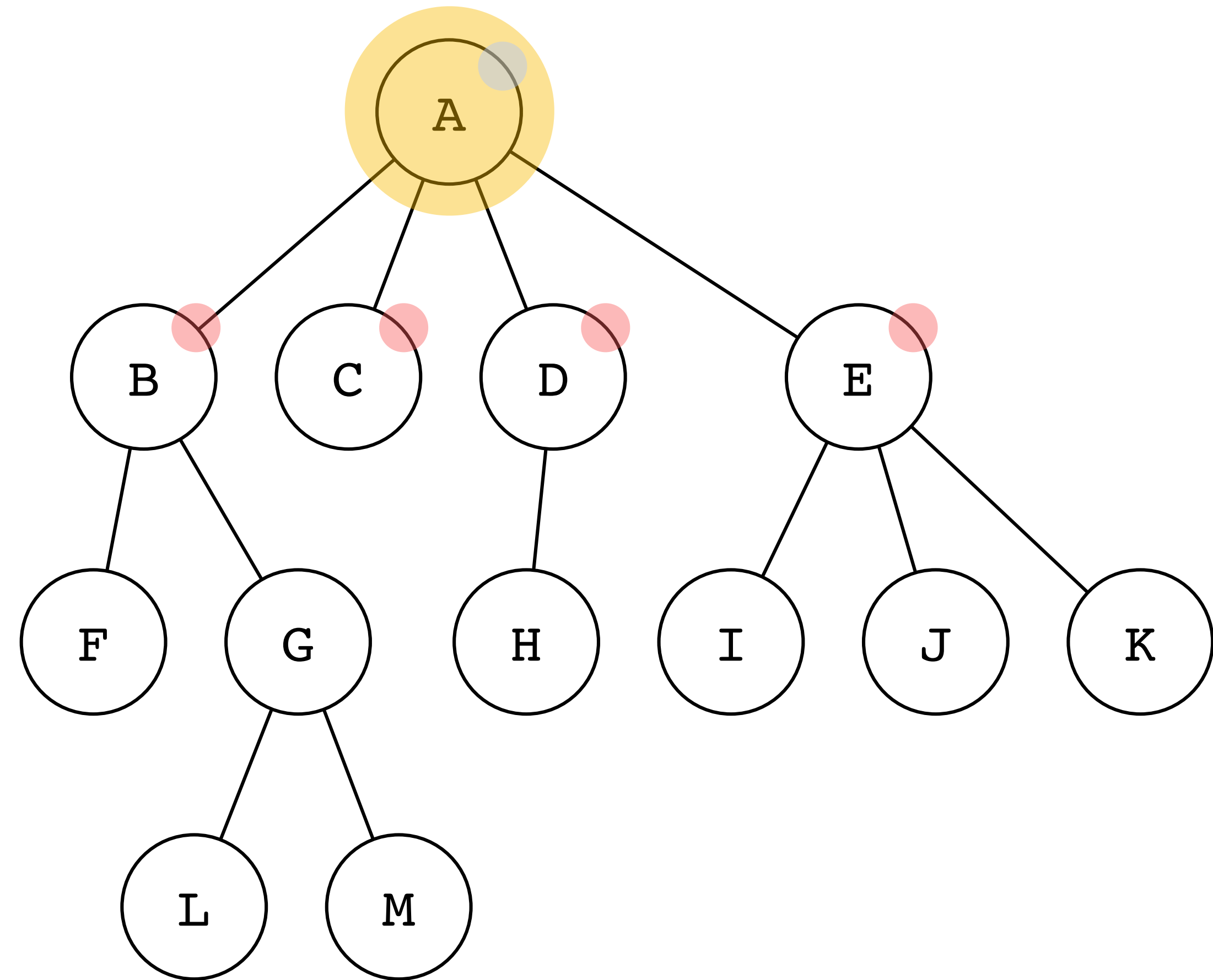
A





À chaque noeud visité :

- Chaque enfant est mis en file d'attente.



File d'attente ●

B C D E

Ordre de visite ●

A



À chaque noeud visité :

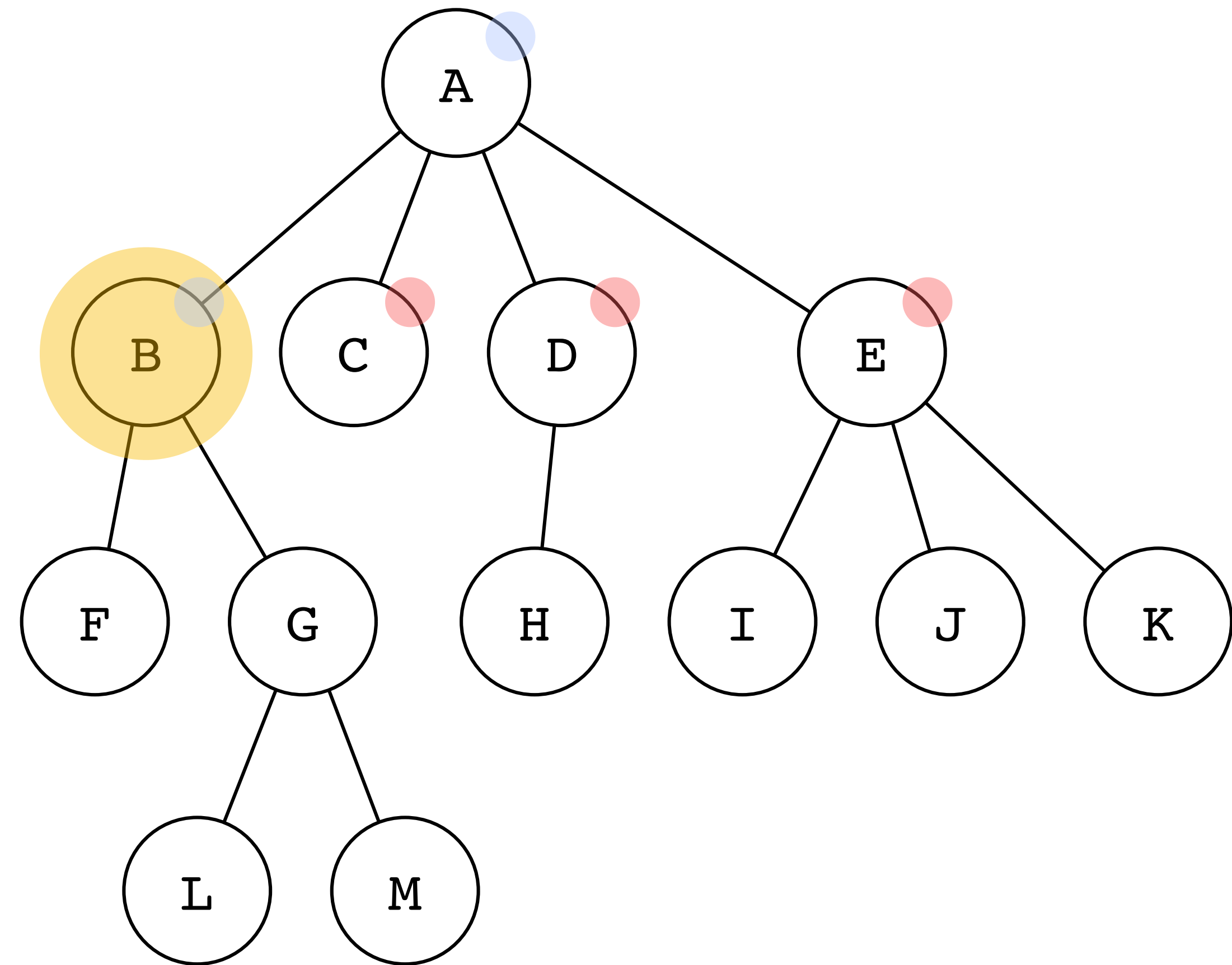
- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

C D E

Ordre de visite ●

A B





À chaque noeud visité :

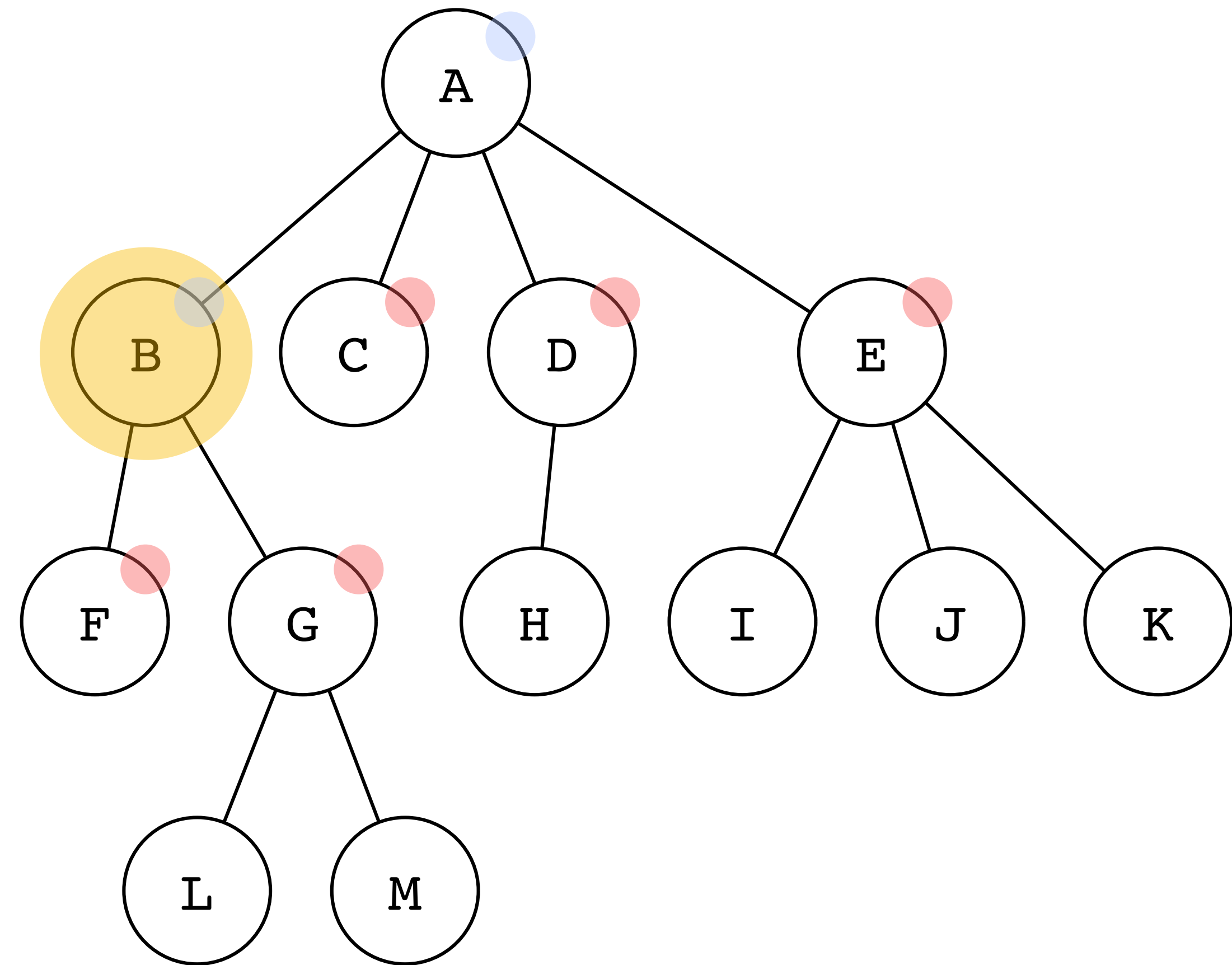
- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

C D E F G

Ordre de visite ●

A B





À chaque noeud visité :

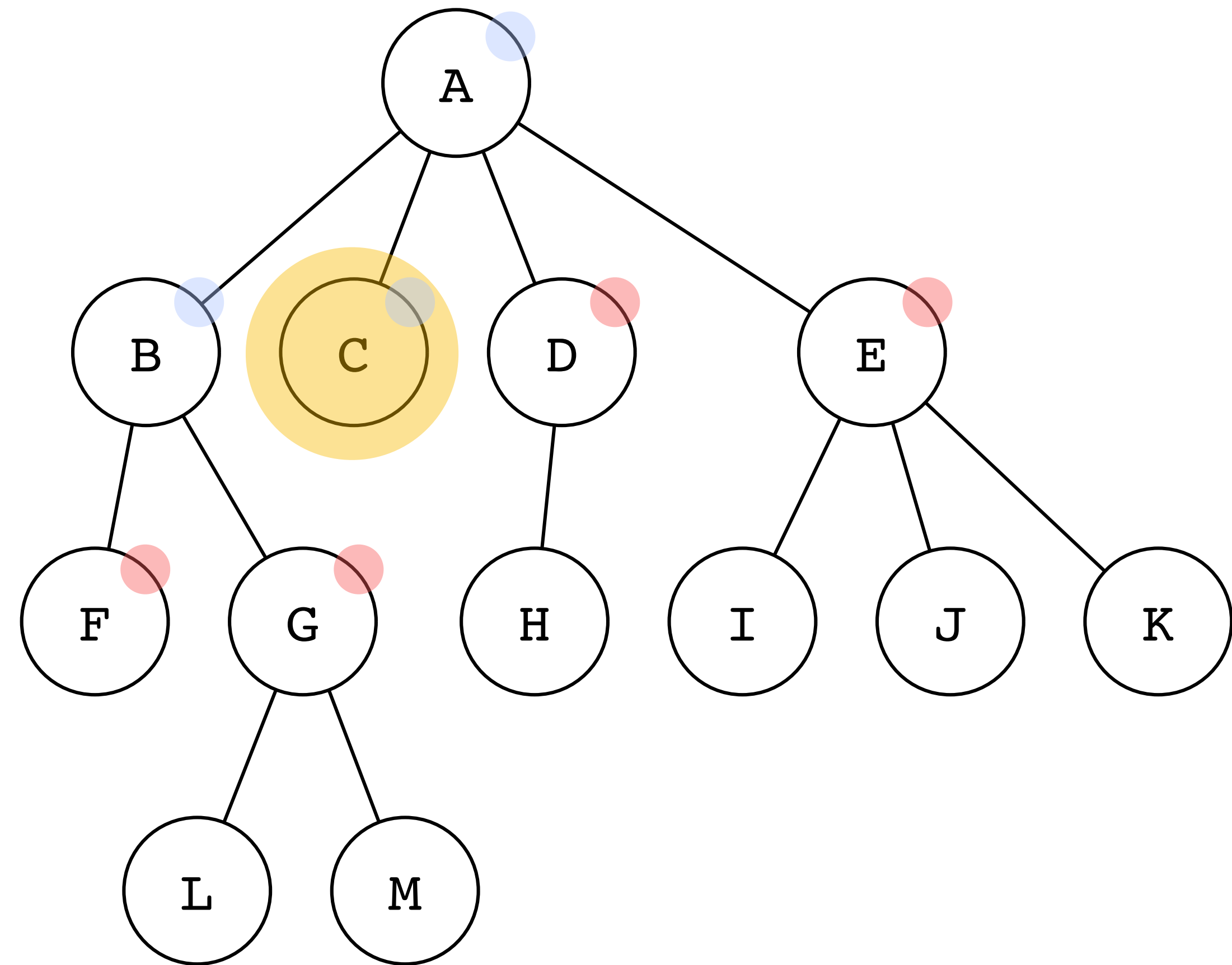
- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

D E F G

Ordre de visite ●

A B C





À chaque noeud visité :

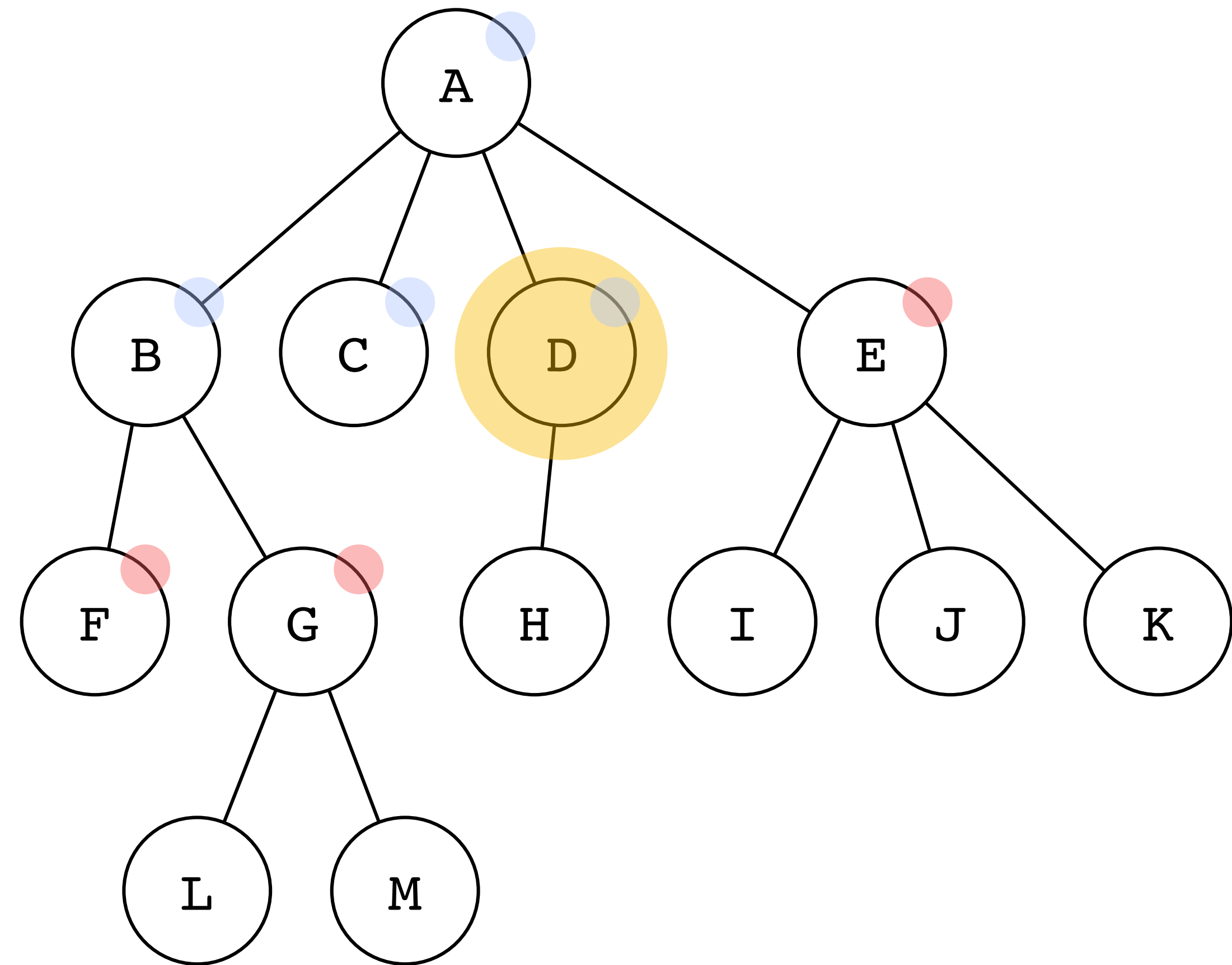
- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

E F G

Ordre de visite ●

A B C D





À chaque noeud visité :

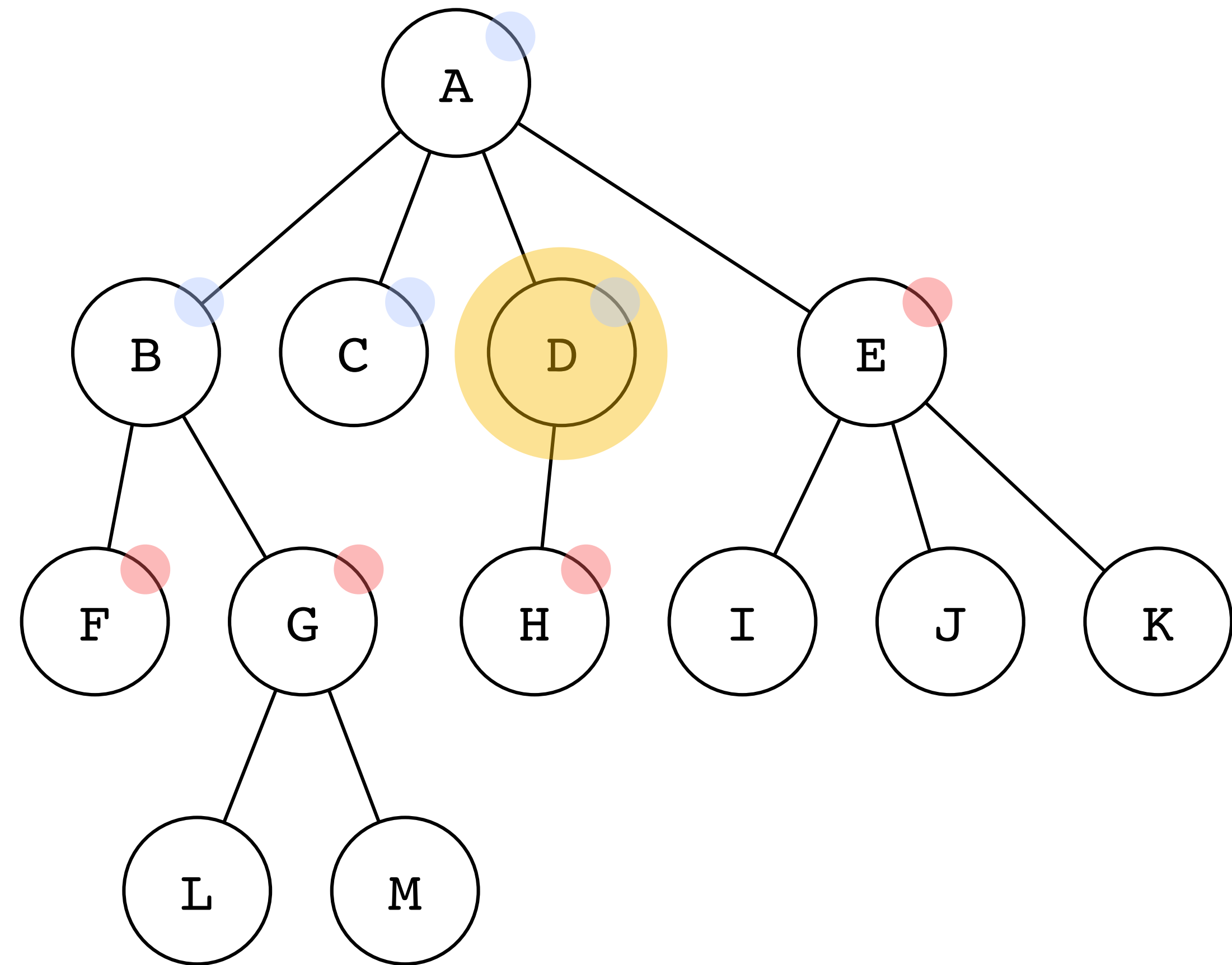
- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

E F G H

Ordre de visite ●

A B C D





À chaque noeud visité :

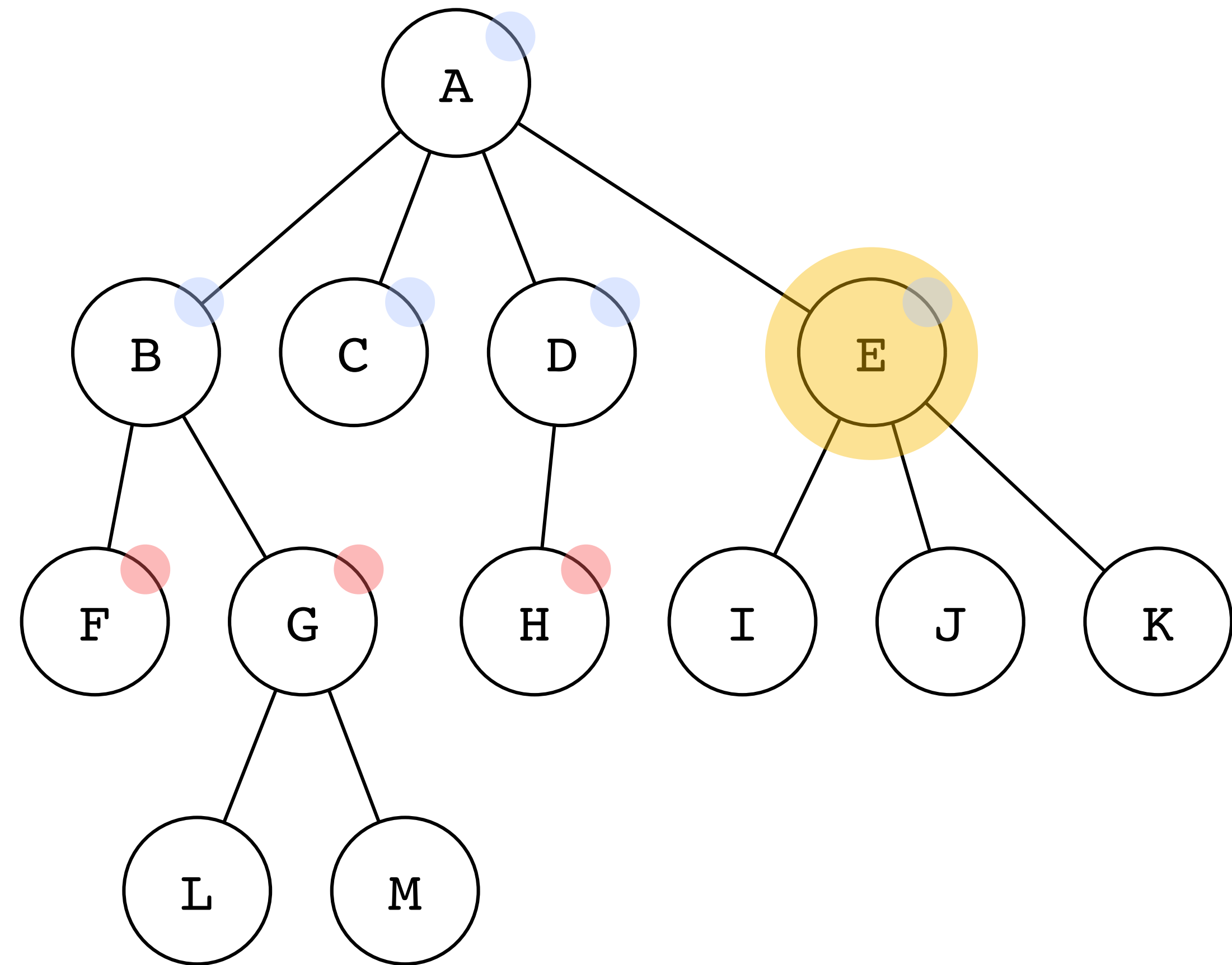
- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

F G H

Ordre de visite ●

A B C D E





À chaque noeud visité :

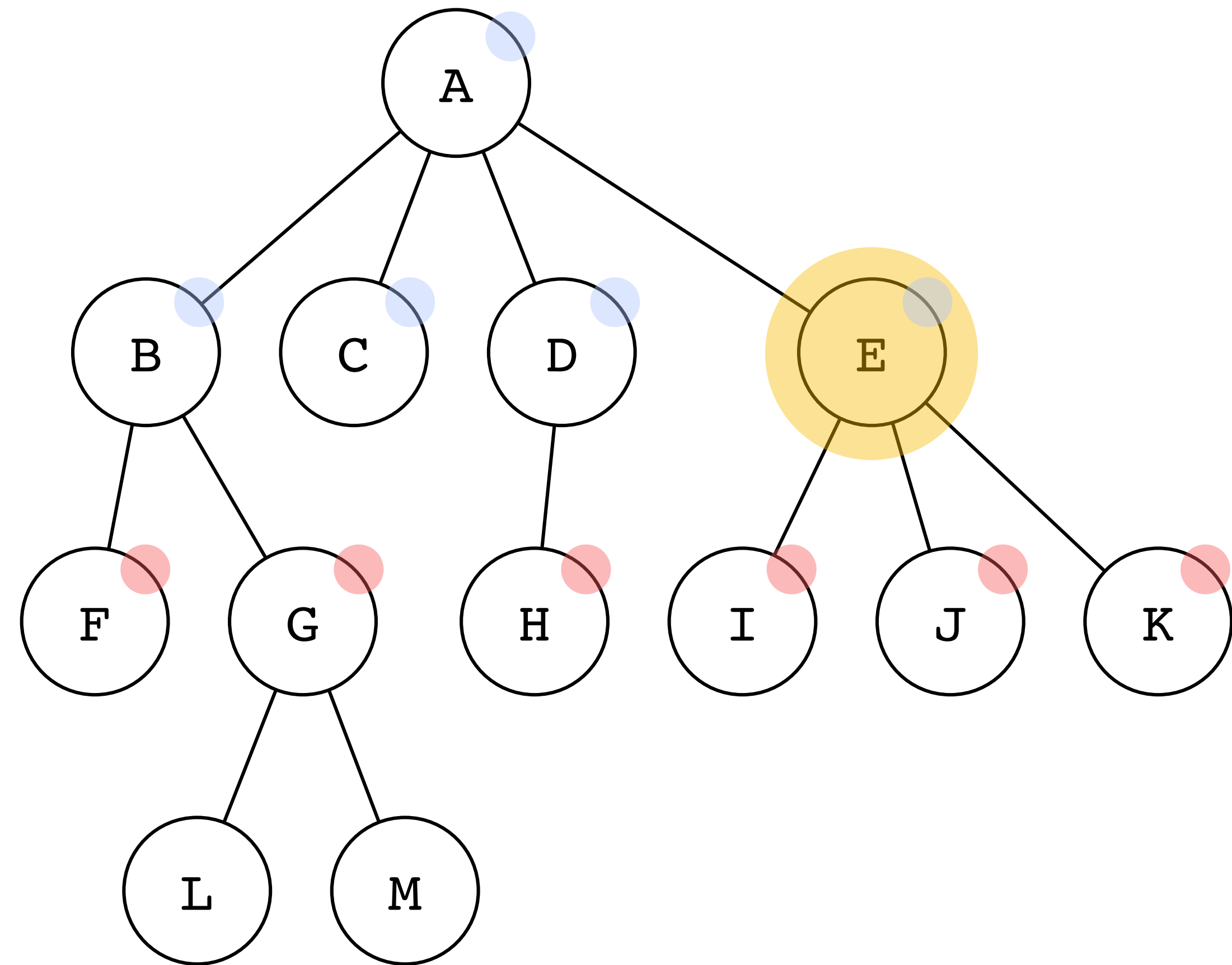
- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

F G H I J K

Ordre de visite ●

A B C D E





À chaque noeud visité :

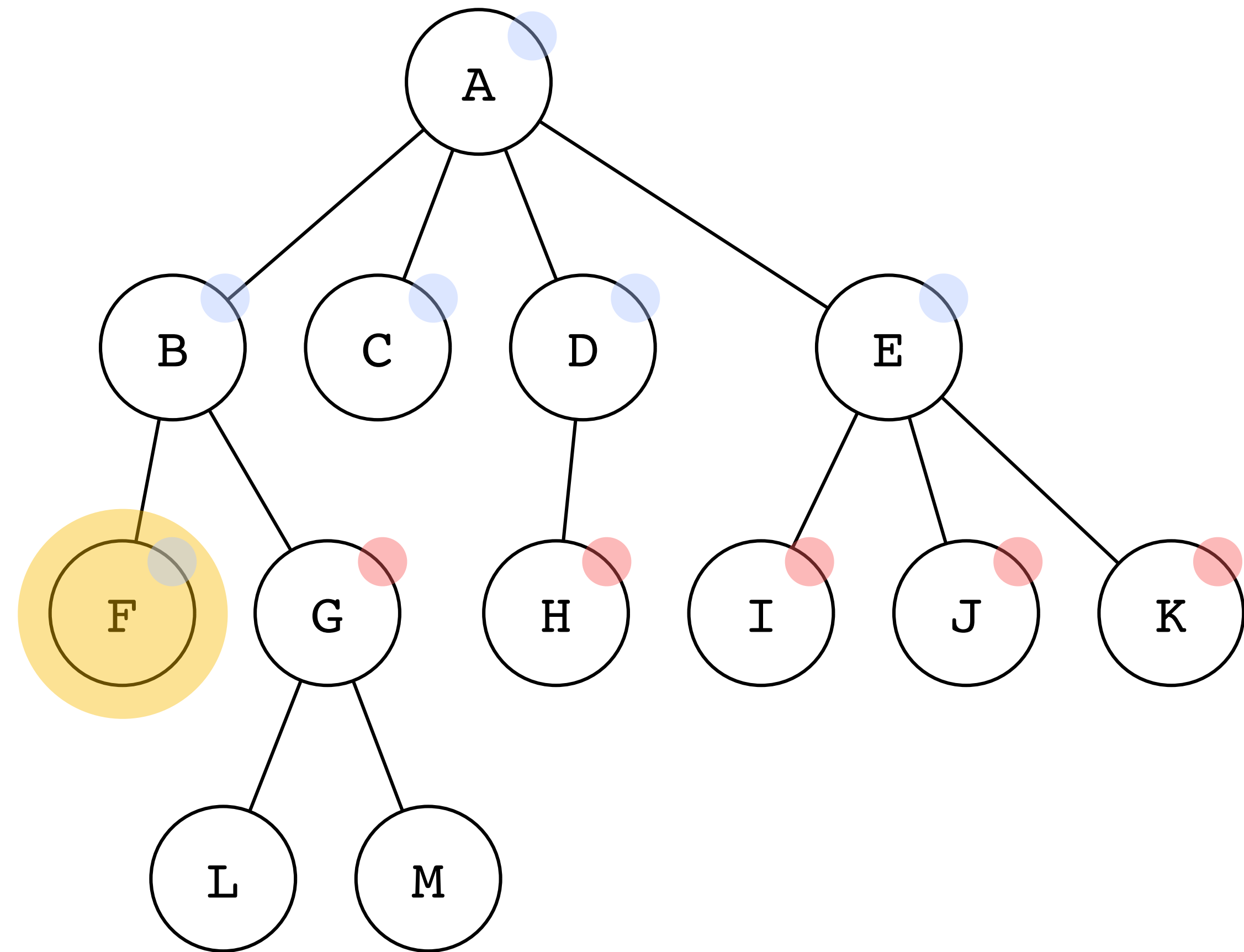
- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

G H I J K

Ordre de visite ●

A B C D E F





À chaque noeud visité :

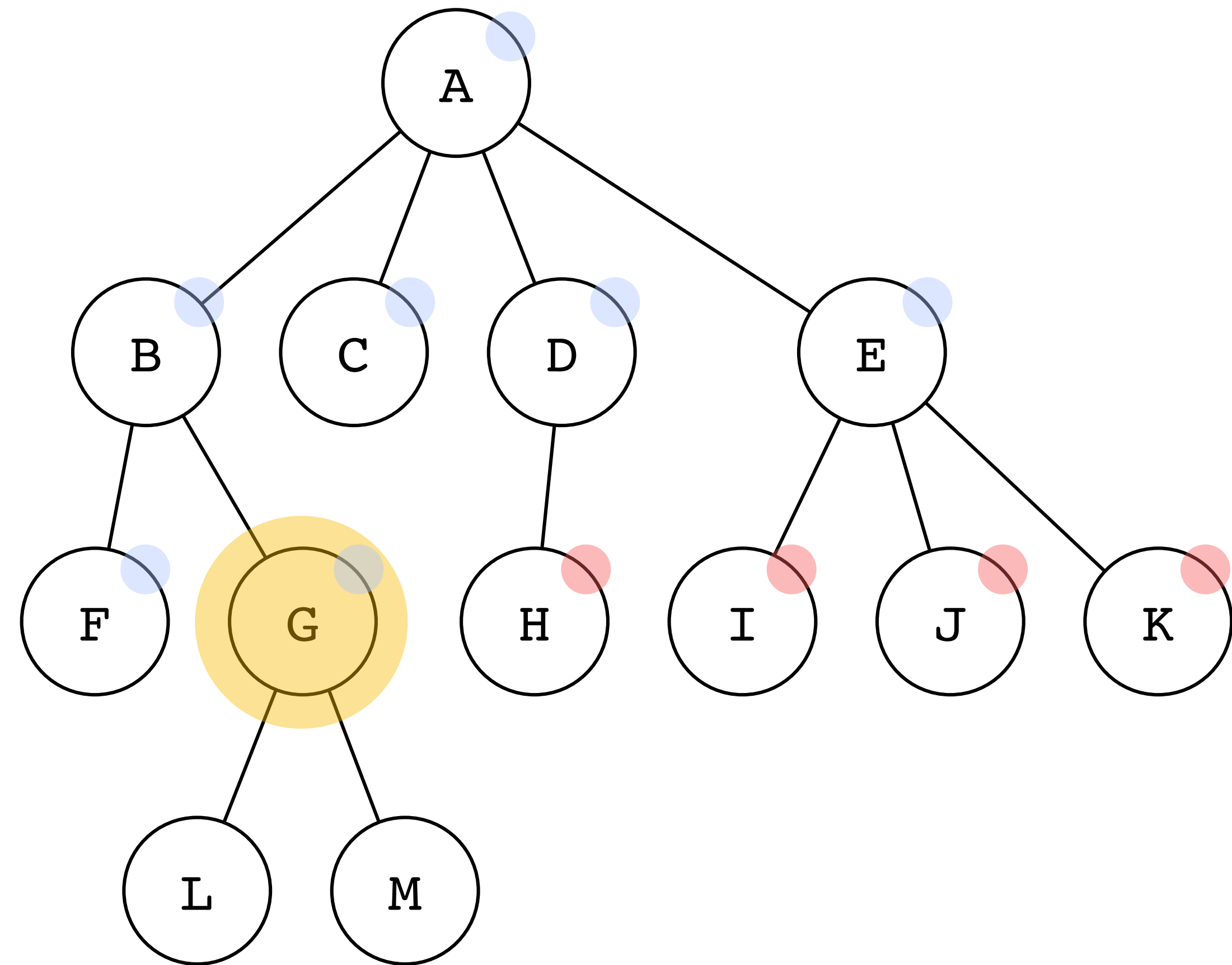
- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

H I J K

Ordre de visite ●

A B C D E F G





À chaque noeud visité :

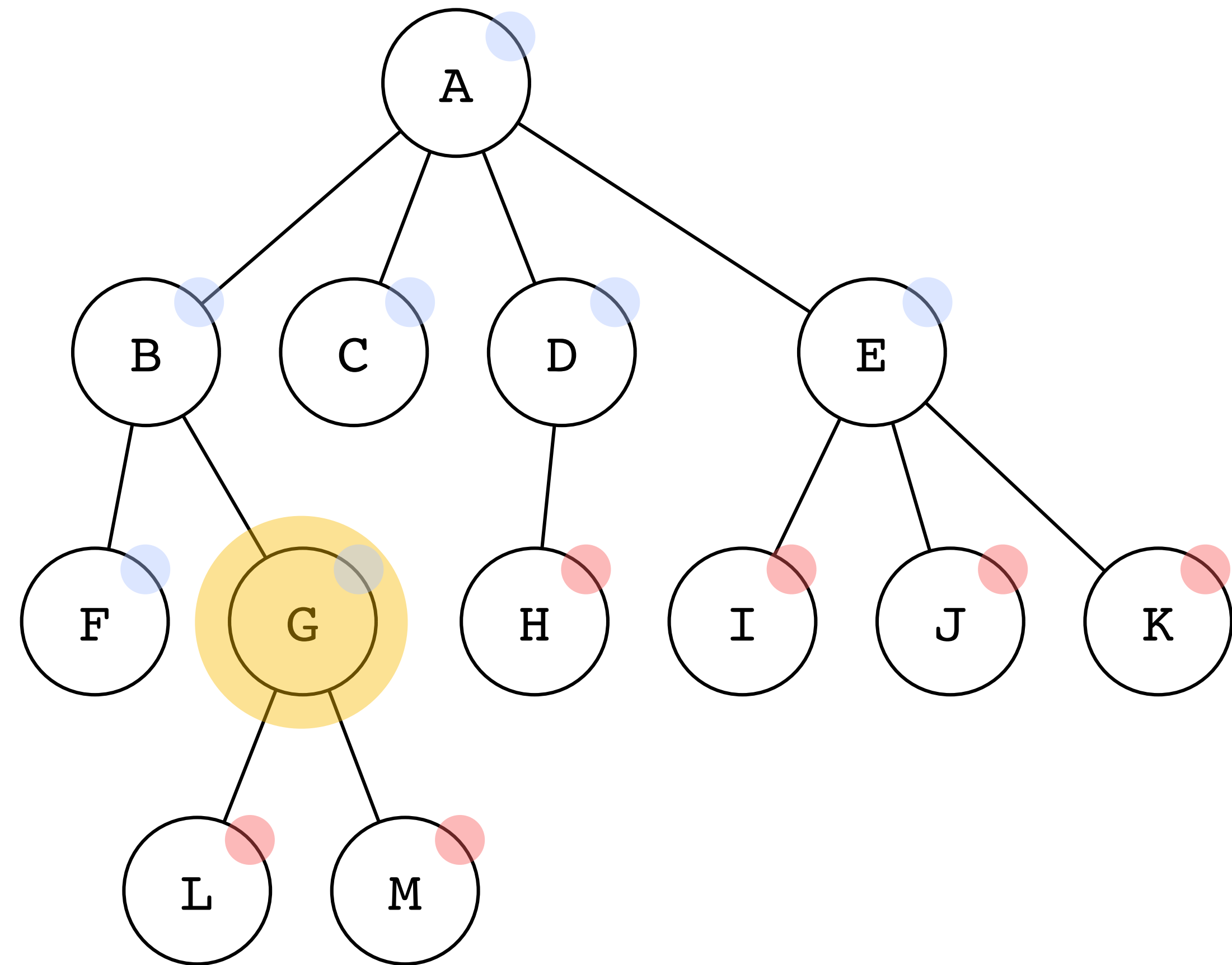
- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

H I J K L M

Ordre de visite ●

A B C D E F G





À chaque noeud visité :

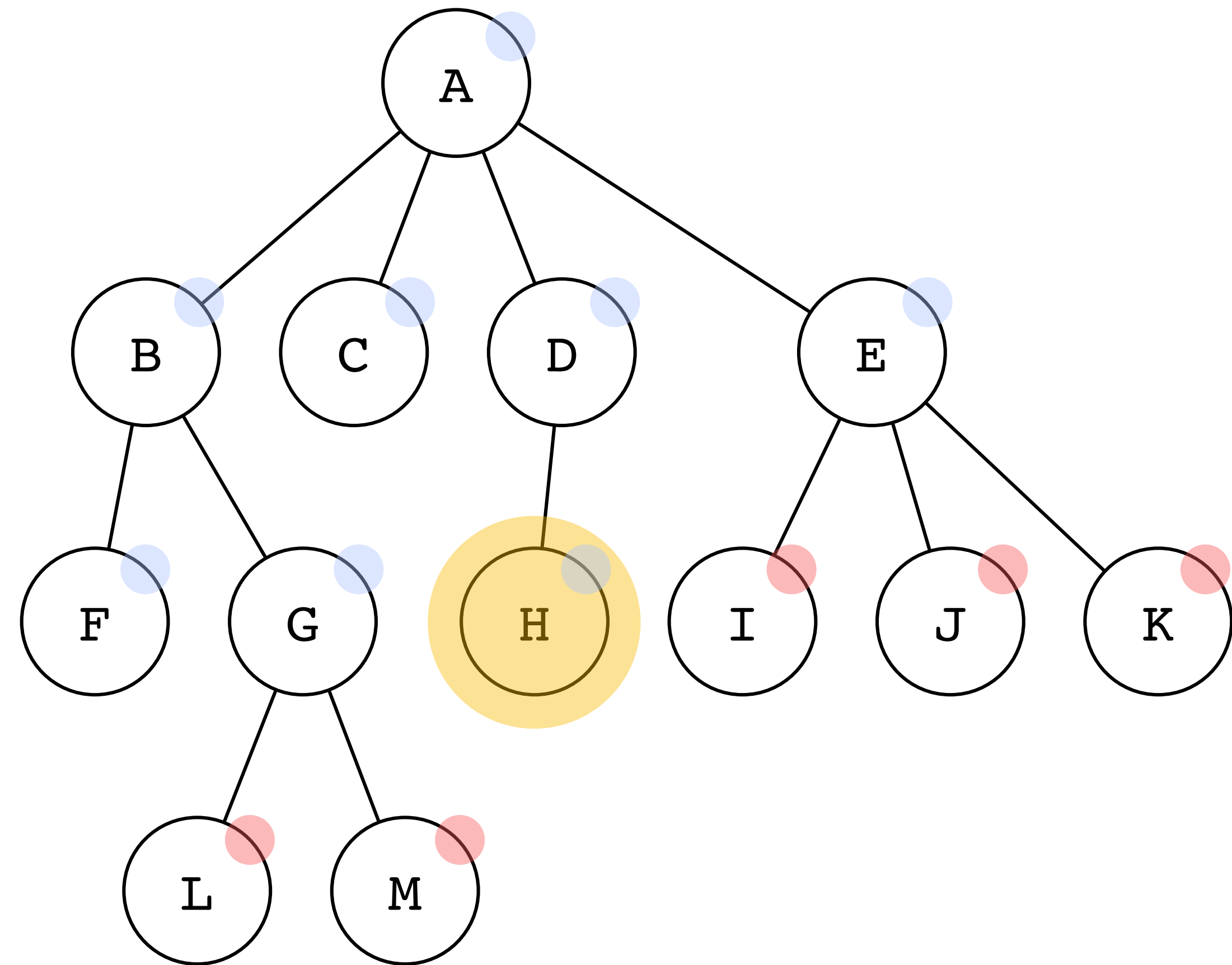
- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

I J K L M

Ordre de visite ●

A B C D E F G H





À chaque noeud visité :

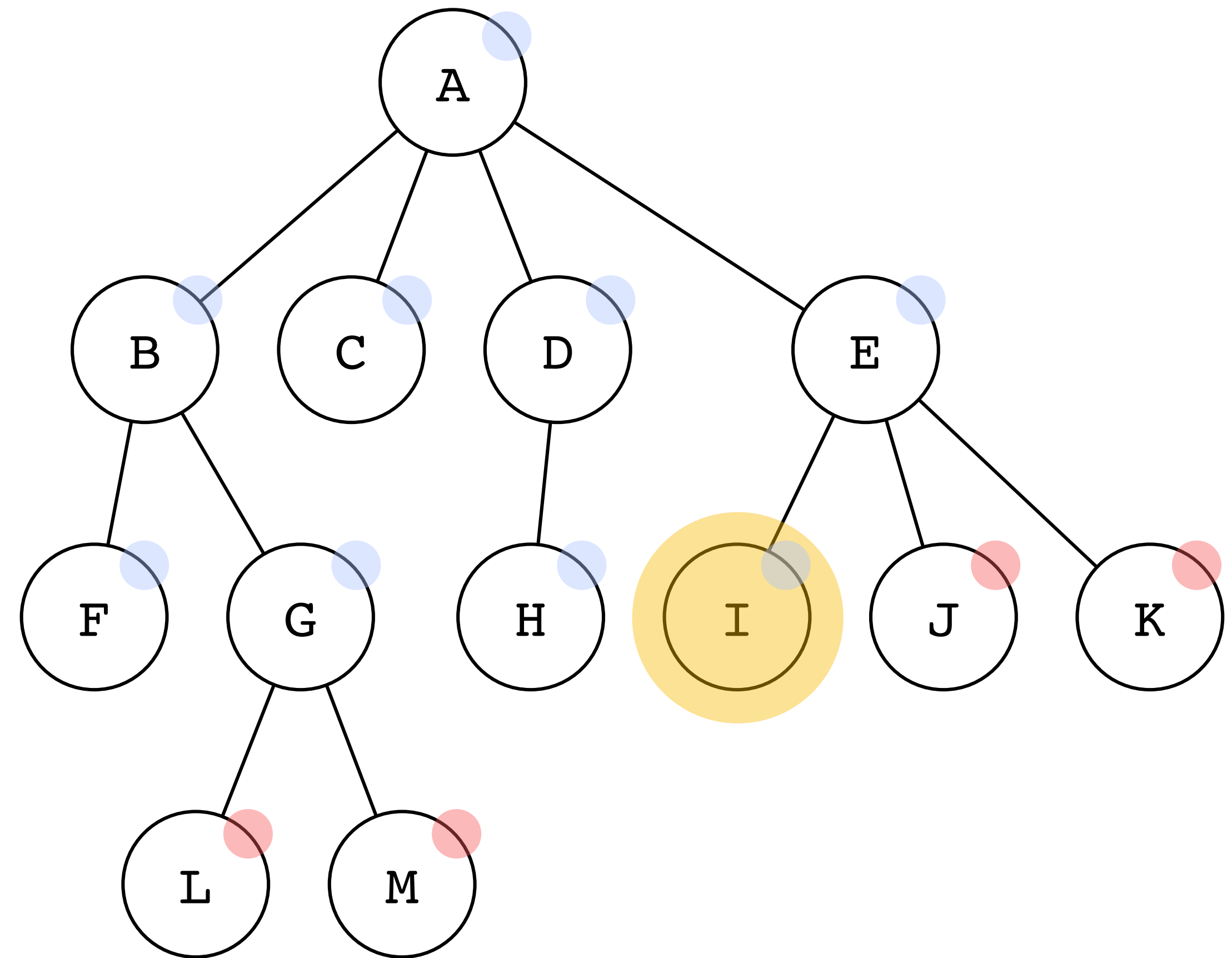
- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

J K L M

Ordre de visite ●

A B C D E F G H I





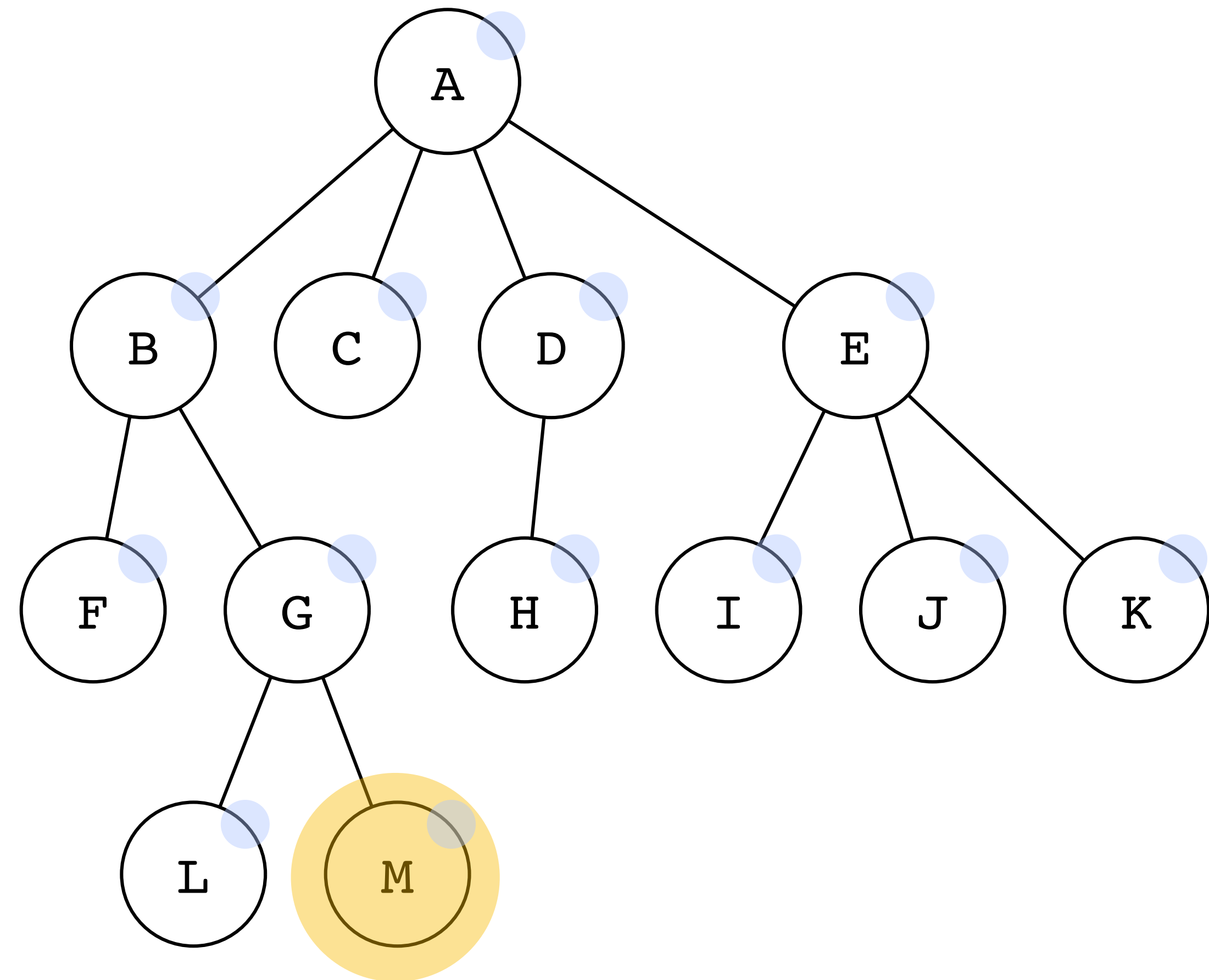
À chaque noeud visité :

- Chaque enfant est mis en file d'attente.
- Le prochain noeud visité est le suivant dans la file.

File d'attente ●

Ordre de visite ●

A B C D E F G H I J K L M



Parcours en largeur



Comment visiter tous les noeuds d'un arbre
niveau par niveau ?

On utilise une FIFO

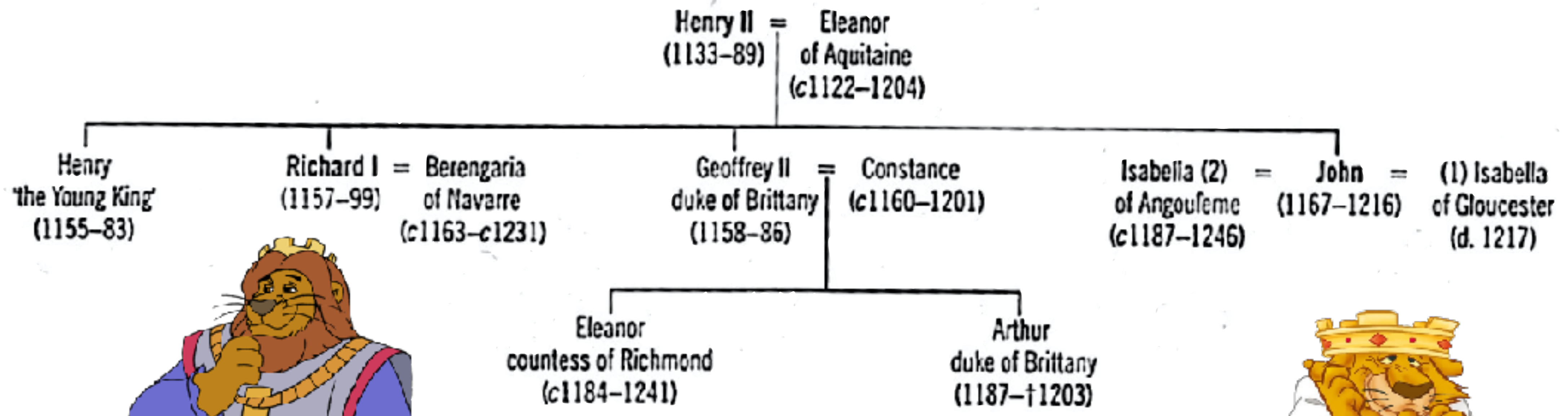
- Le prochain noeud à visiter est le prochain dans la file
- Chaque enfant d'un noeud visité est ajouté à la file

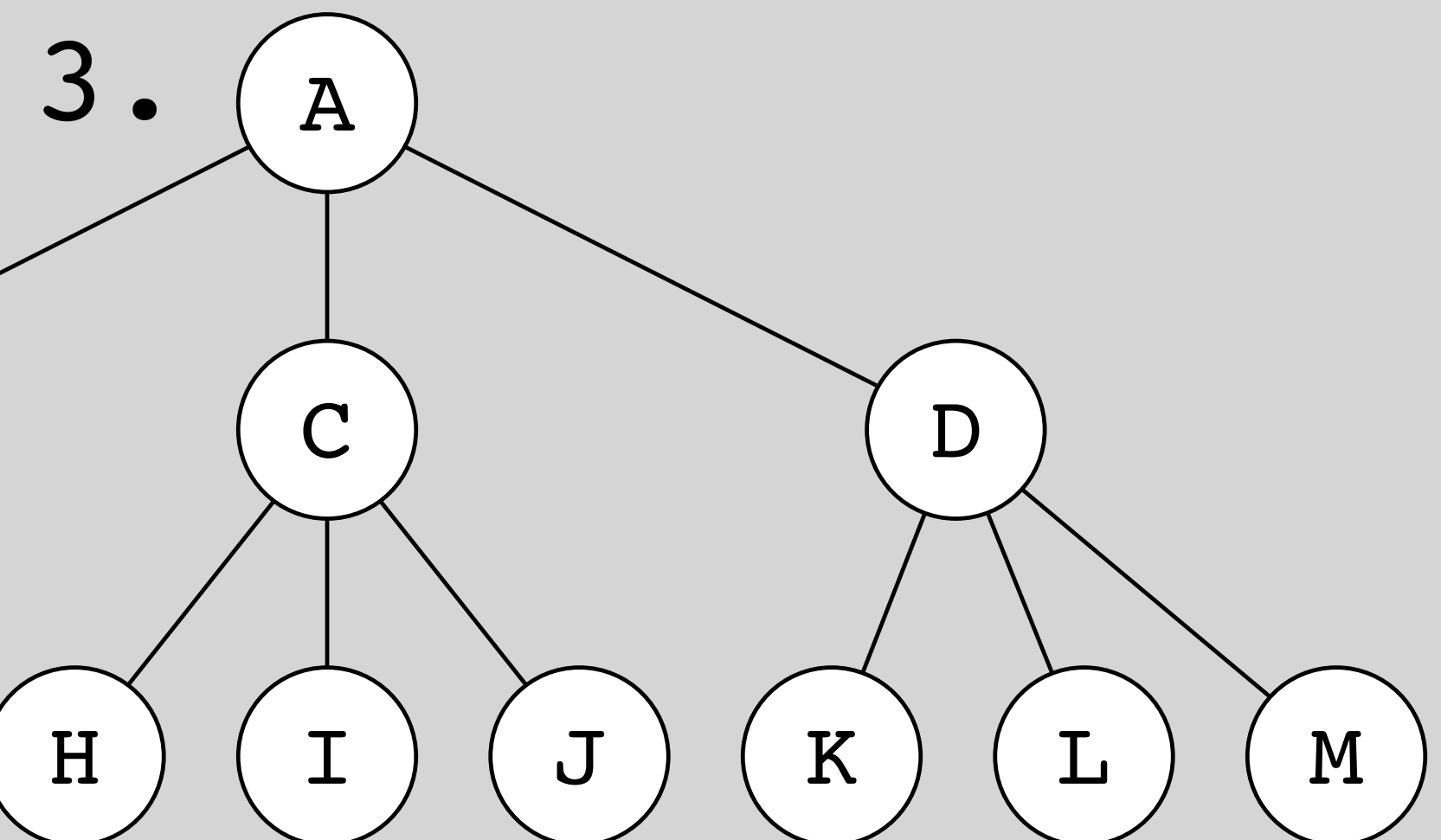
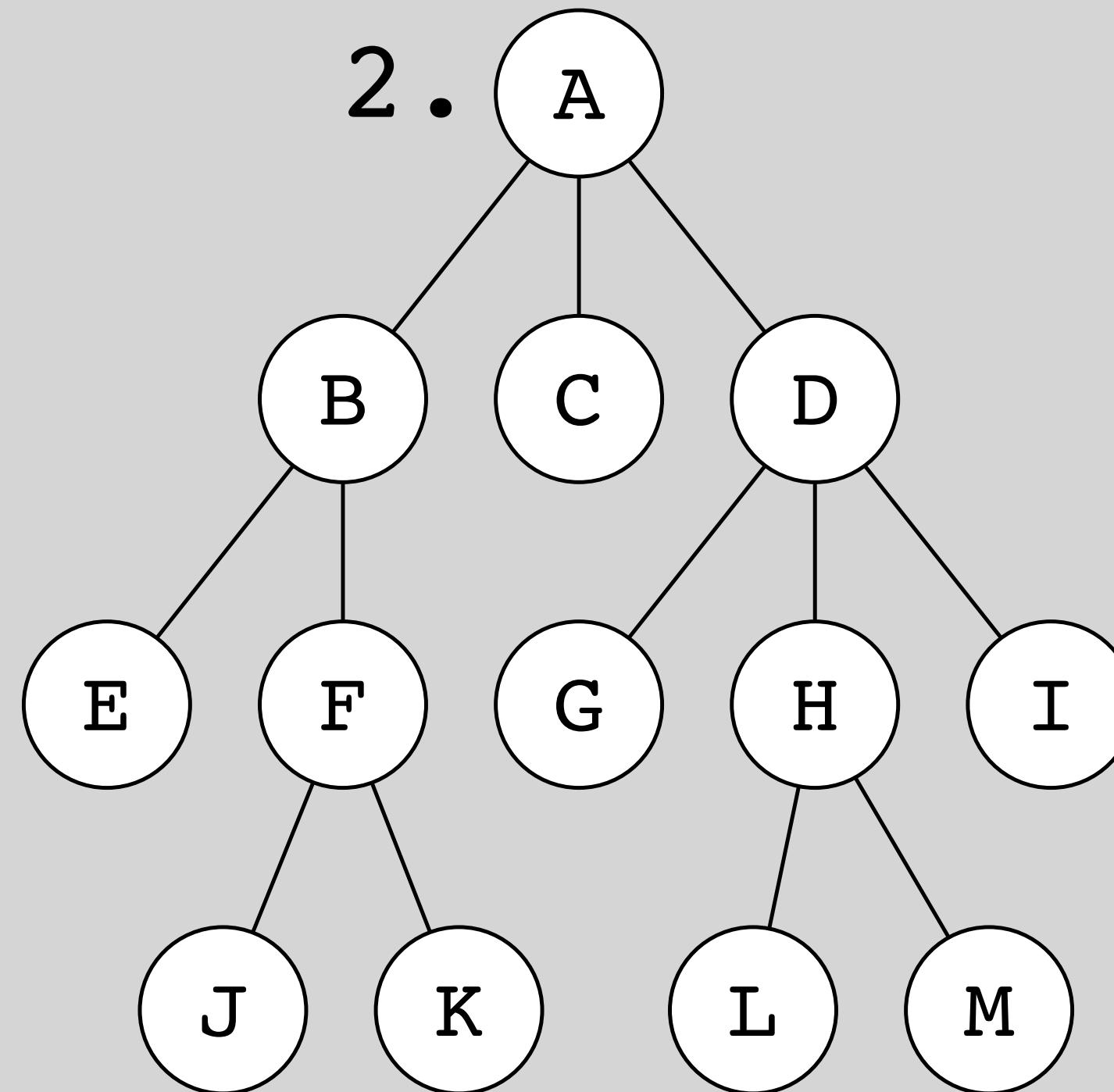
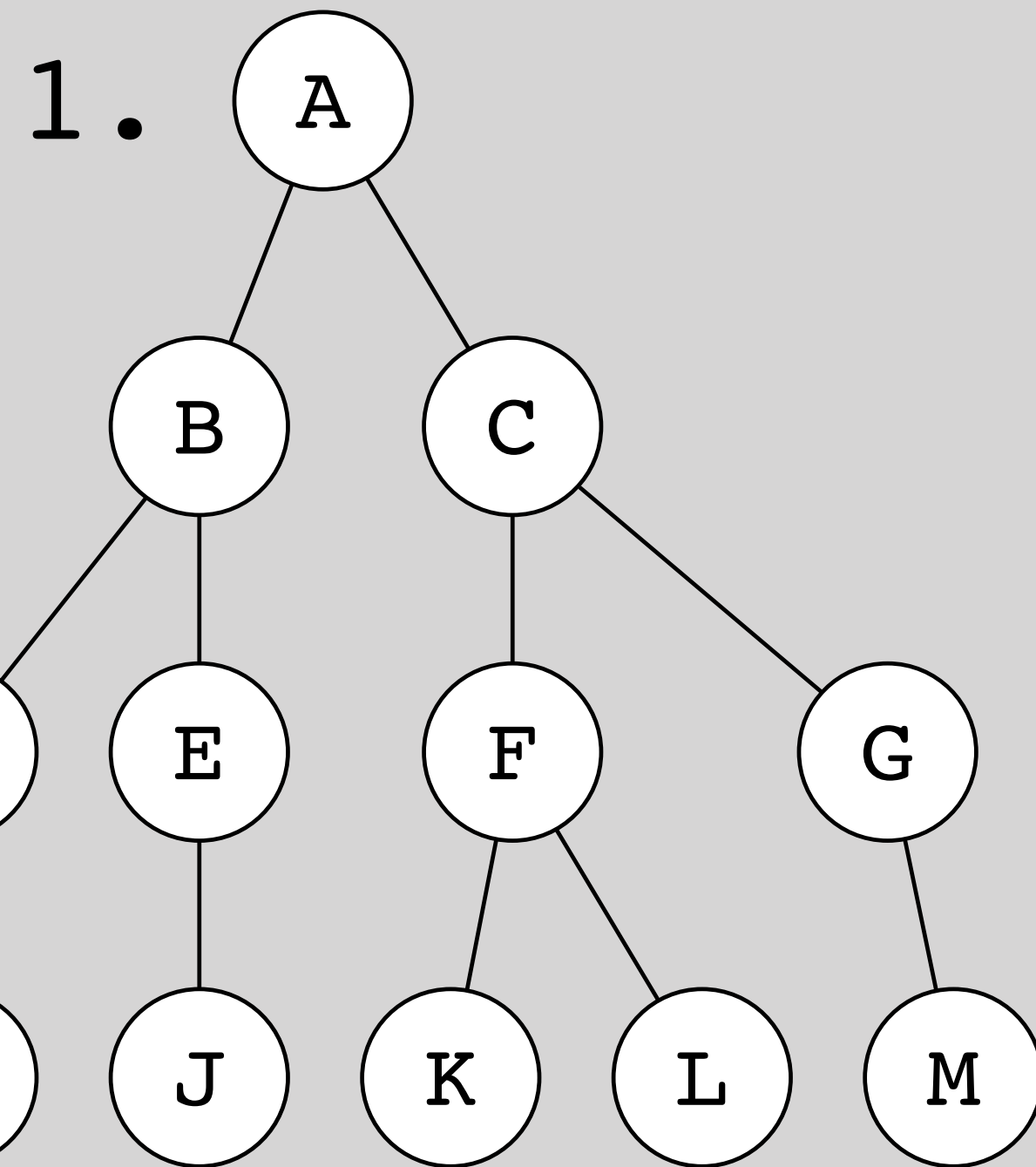
```
fonction largeur (r, fn)
    Initialiser File Q
    Q.push(r)
    tant que pas Q.vide()
        r = Q.front(); Q.pop()
        fn(r)
        pour tout enfant e de r
            Q.push(e)
```

(Succession de Richard Coeur de Lion)

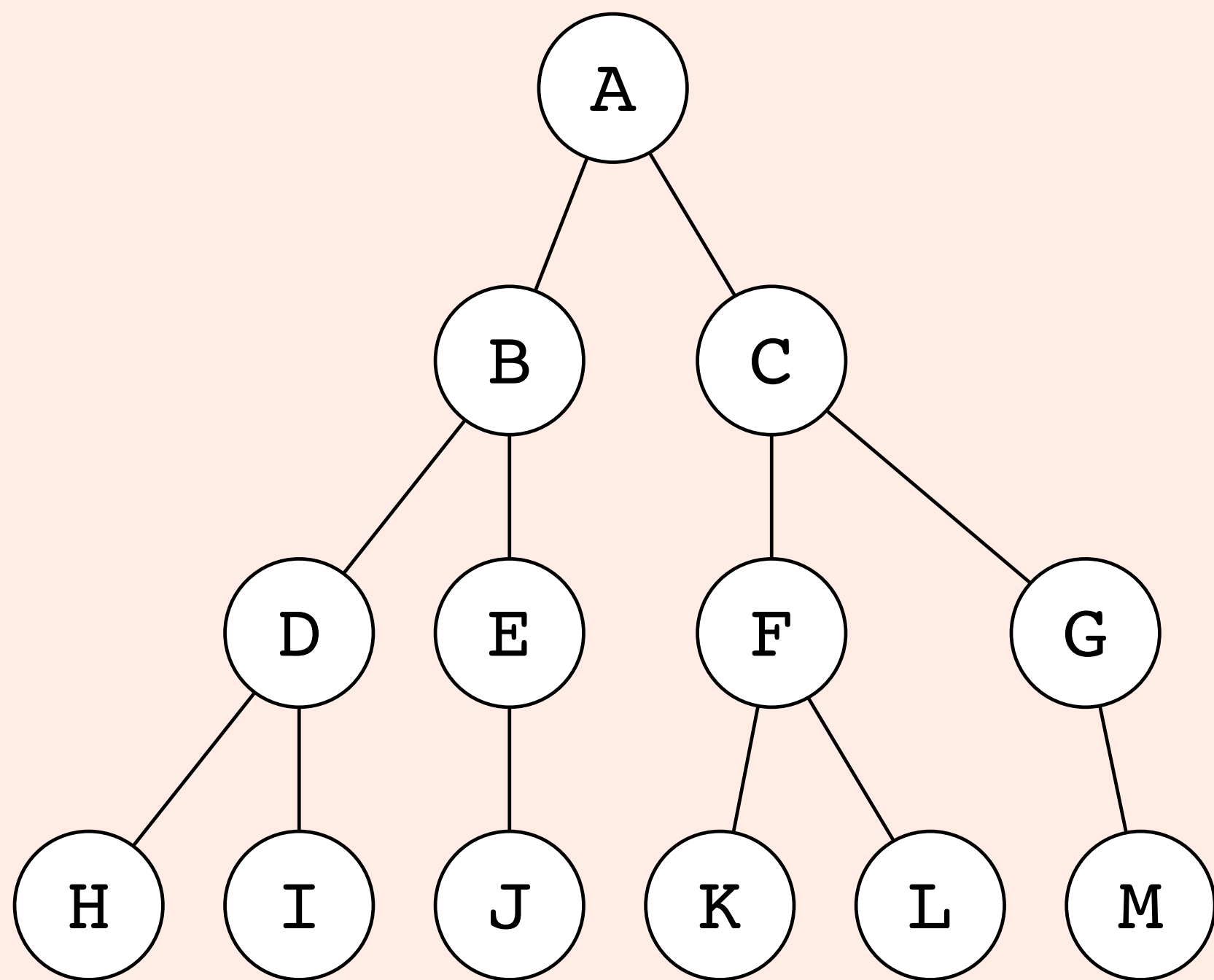


The House of Anjou/ Plantagenet





Parcours (1)

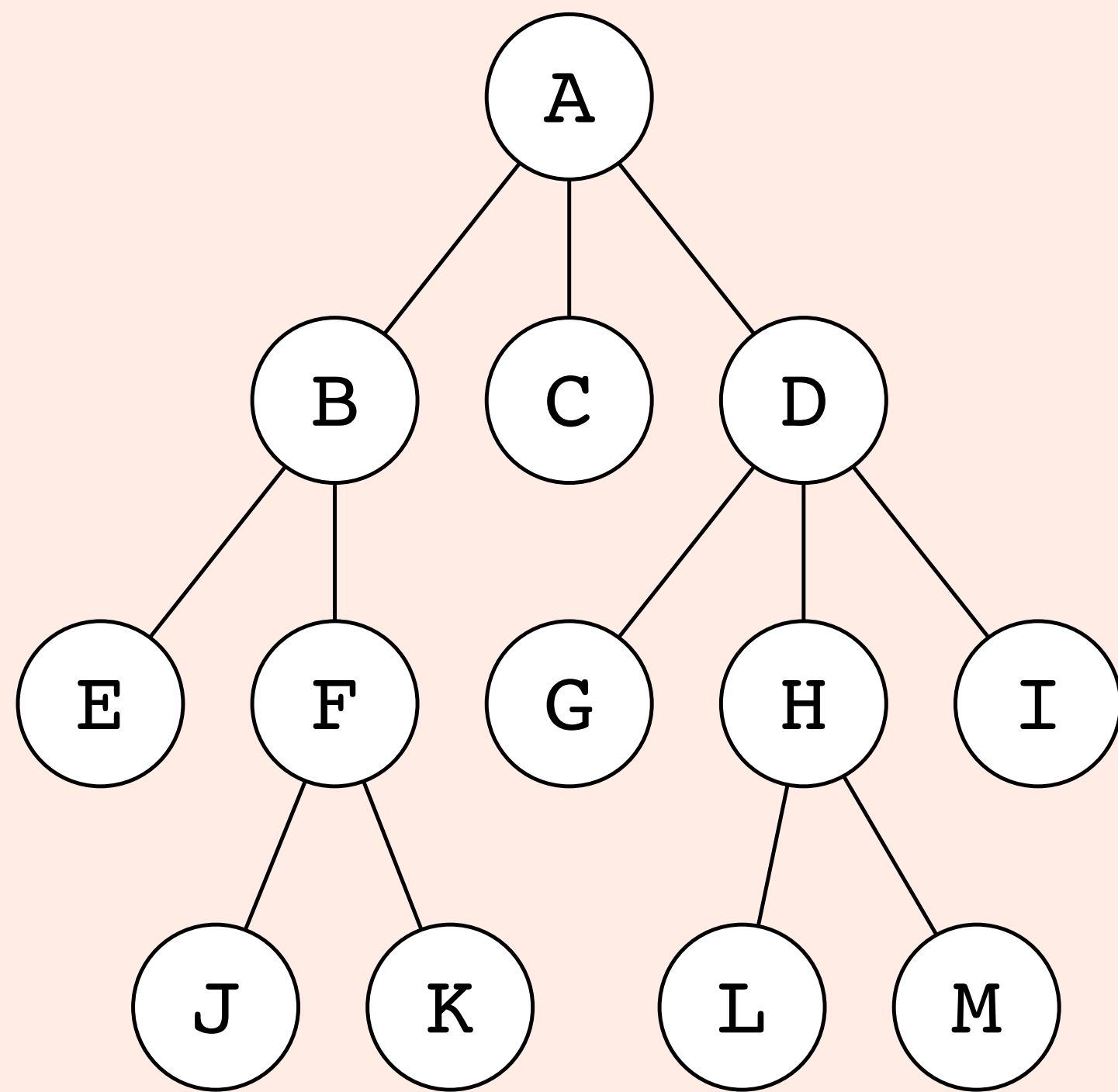


Pré: ABDHIEJCFKLGM

Post: HIDJEBKLFMGCA

Largeur: ABCDEFGHIJKLM

Parcours (2)



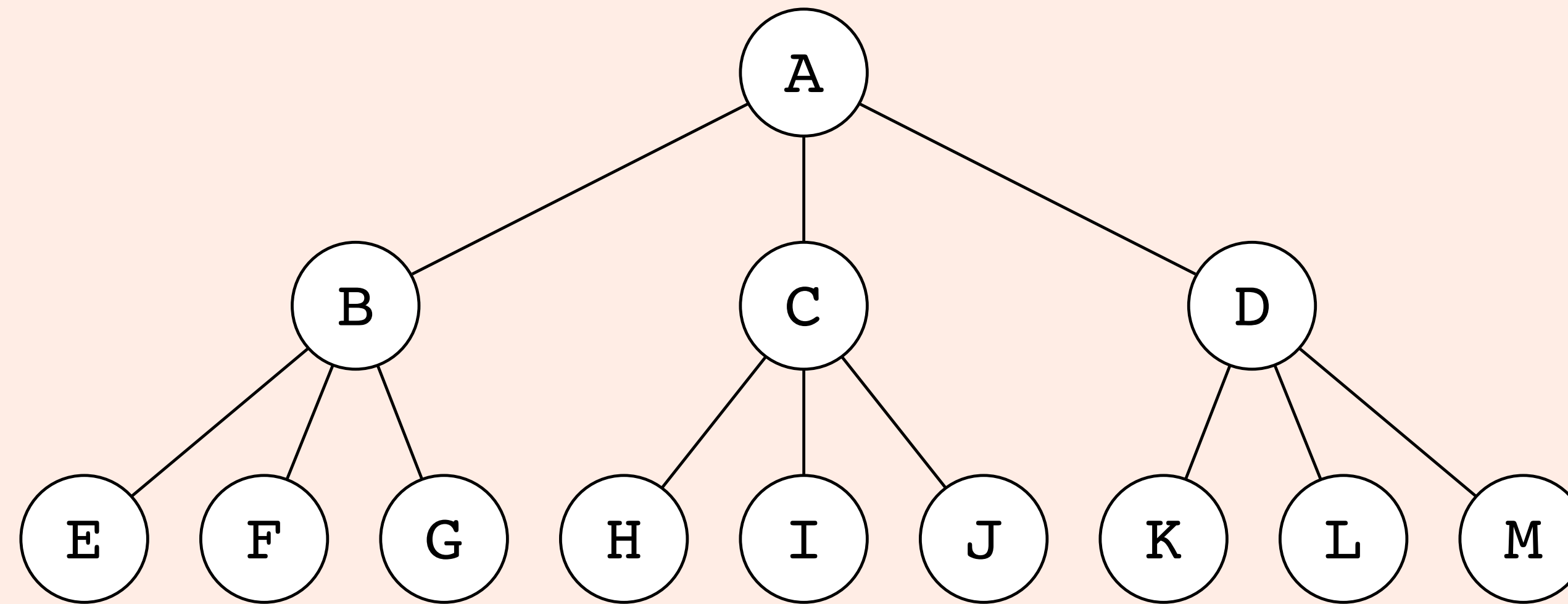
Pré: ABEFJKCDGHLMI

Post: EJKFBCGLMHIDA

Largeur: ABCDEFGHIJKLM



Parcours (3)



Pré: AB EFGCH IJDKLM

Post: EFGBH IJCKLMDA

Largeur: ABCDEFGH IJKLM

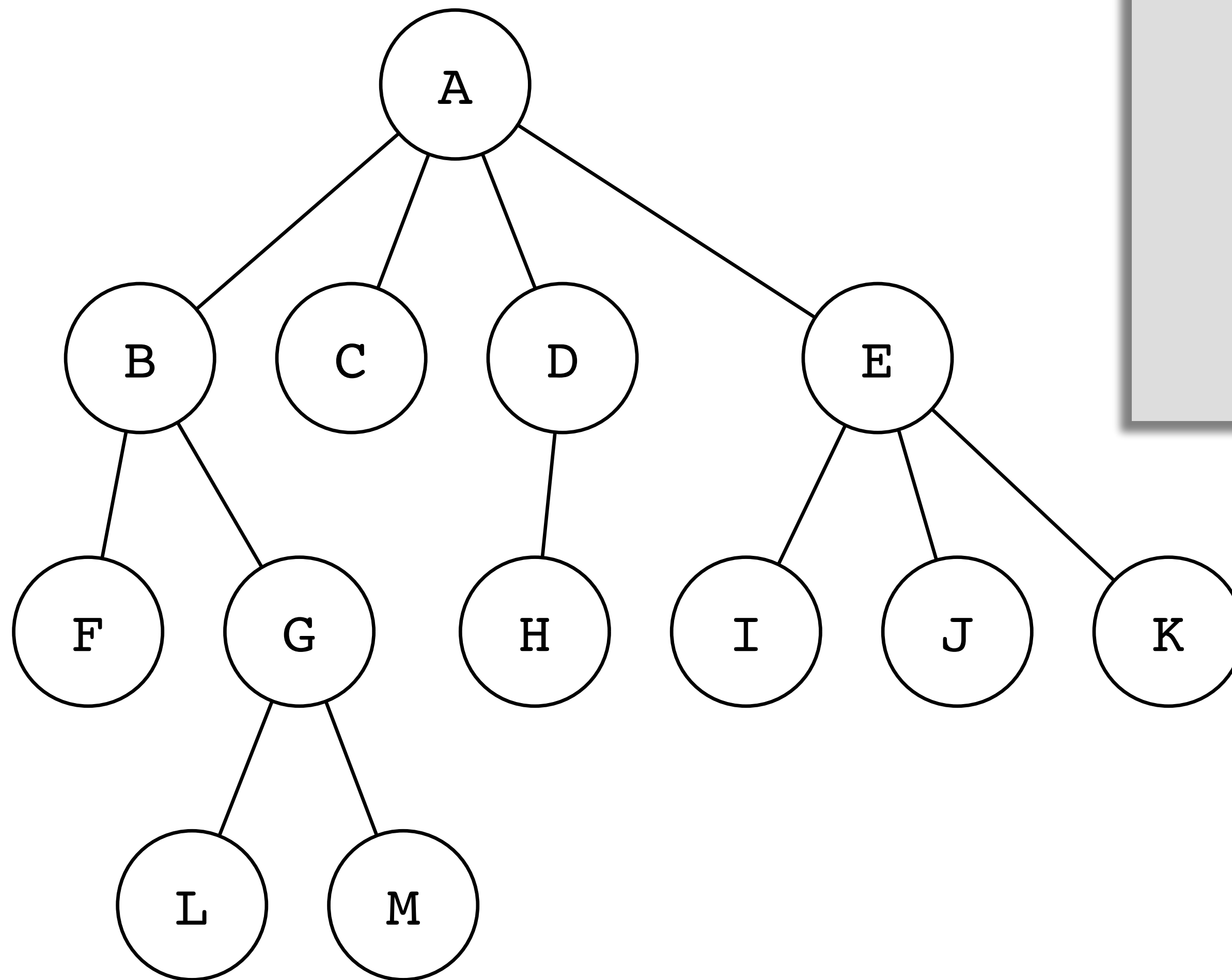
4. Représentations linéaires



Représentation indentée (1)



```
fonction indenter(r, n)
  si r != ∅
    Afficher n blancs
    Afficher r.etiquette
    Afficher saut de ligne
    pour tout enfant e de r
      indenter(e, n+1)
```



A

B

F

G

L

M

C

D

H

E

I

J

K

B

C

D

E

F

G

H

I

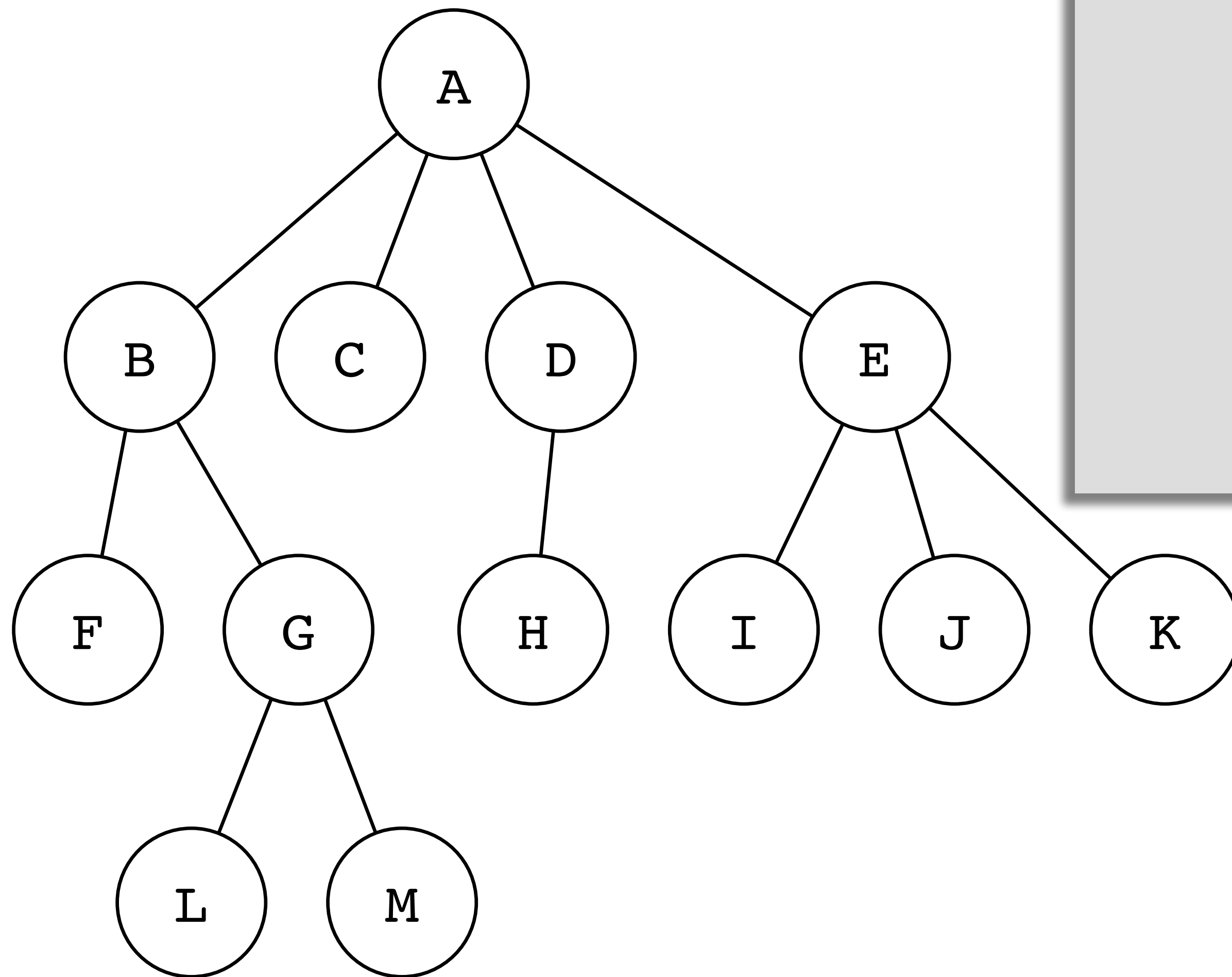
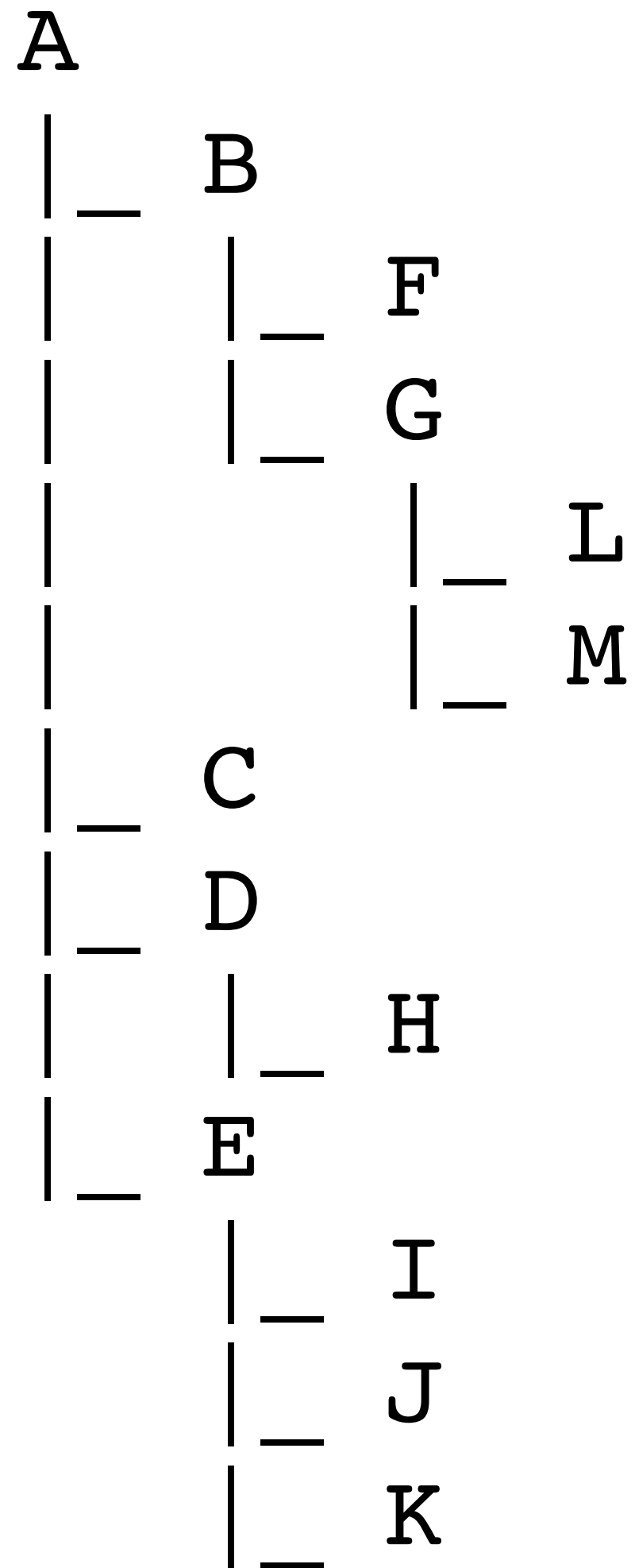
J

K

L

M

Représentation indentée (2)



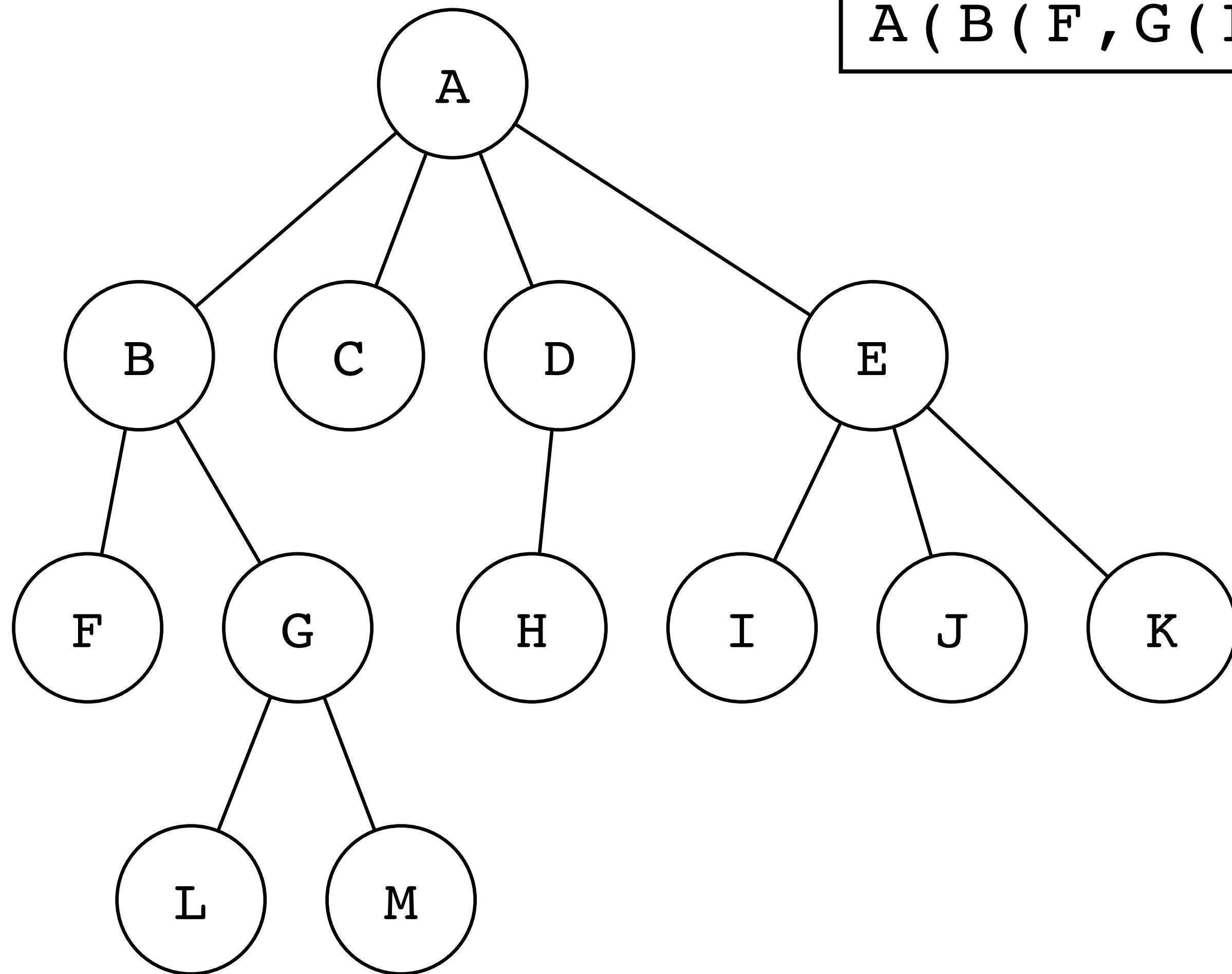
```

fonction indenter(r, préfixe)
  si r != ∅
    Afficher préfixe
    Afficher r.etiquette
    Afficher saut de ligne
    pour tout enfant e de r
      pe ← préfixe modifié
      indenter(r,pe)
  
```

Listes imbriquées



$A(B(F, G(L, M)), C, D(H), E(I, J, K))$



```
fonction imbriquer(r)
  si r != ∅
    Afficher r.etiquette
    si degré(r) > 0
      Afficher "("
      pour tout enfant e de r
        imbriquer(e)
      si e n'est pas le dernier
        Afficher ","
      Afficher ")"
```

Arbre depuis un liste imbriquée



```
void from_string(std::istream &s, Noeud *parent, Noeud *&out) {
    out = new Noeud{read_label(s),
                    parent, nullptr, nullptr};
```

```
    Noeud *n = out;
```

```
    char c;
```

```
    while (s.get(c)) {
```

```
        switch (c) {
```

```
            case ',':
```

```
                n->puine = new Noeud{read_label(s),
                                    parent, nullptr, nullptr};
```

```
                n = n->puine;
```

```
                break;
```

```
            case '(':
```

```
                from_string(s, n, n->aine);
```

```
                break;
```

```
            case ')':
```

```
                return;
```

```
            default :
```

```
                throw bad_expression();
```

```
        }
```

A (B (F , G) , C , D (H) , E (I))

```
struct Noeud {
    std::string etiquette;
    Noeud *parent;
    Noeud *aine;
    Noeud *puine;
};
```

```
Arbre::Arbre(std::string const &imbriquees) {
    std::istream s(imbriquees);
    from_string(s, nullptr, racine);
}
```



- Dessinez les graphes représentés par les listes suivantes

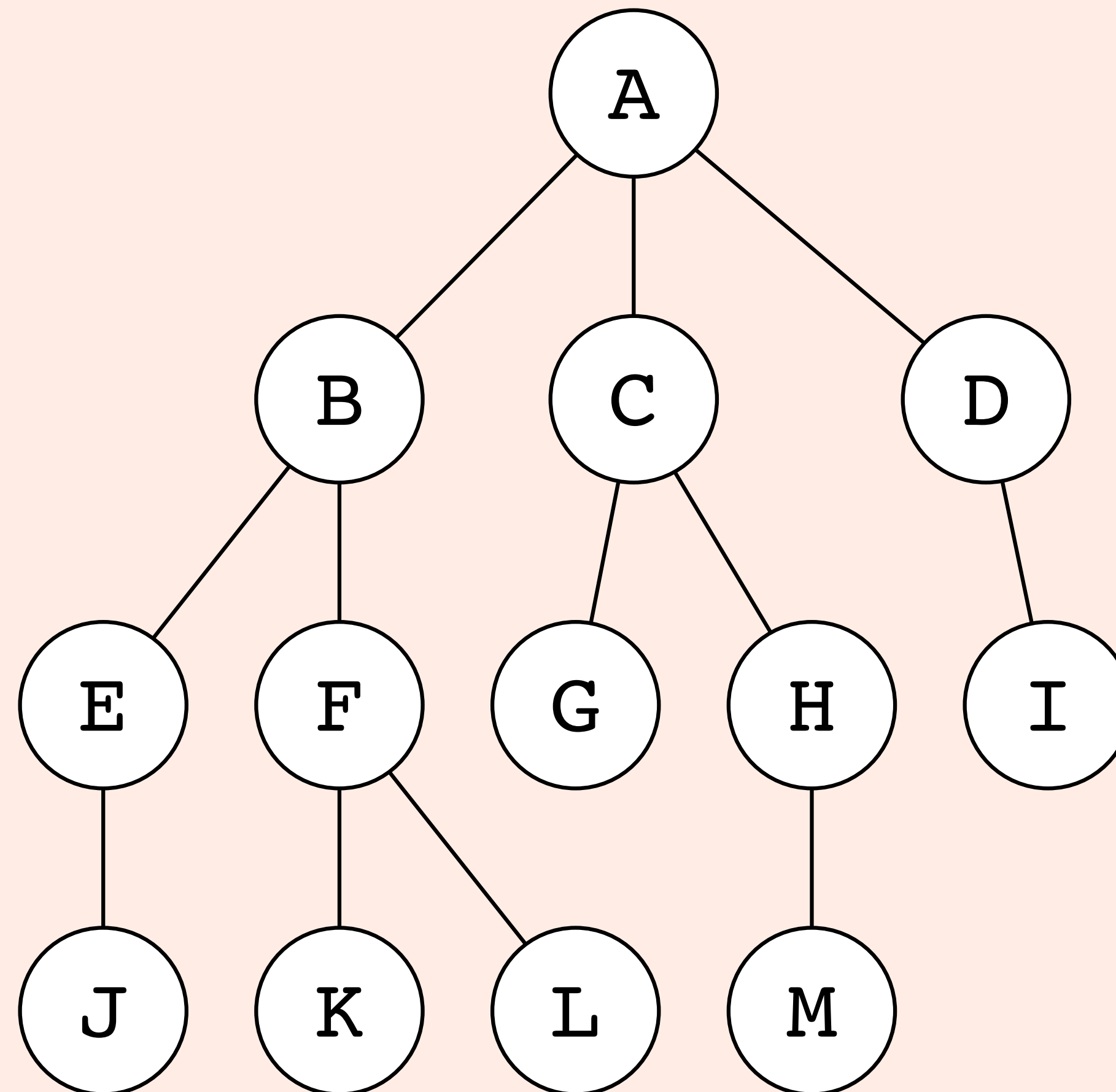
$A(B(E(J), F(K, L)), C(G, H(M)), D(I))$

$A(B(D(L), F(I), G(H, K)), C(E(M), J))$



Solution (1)

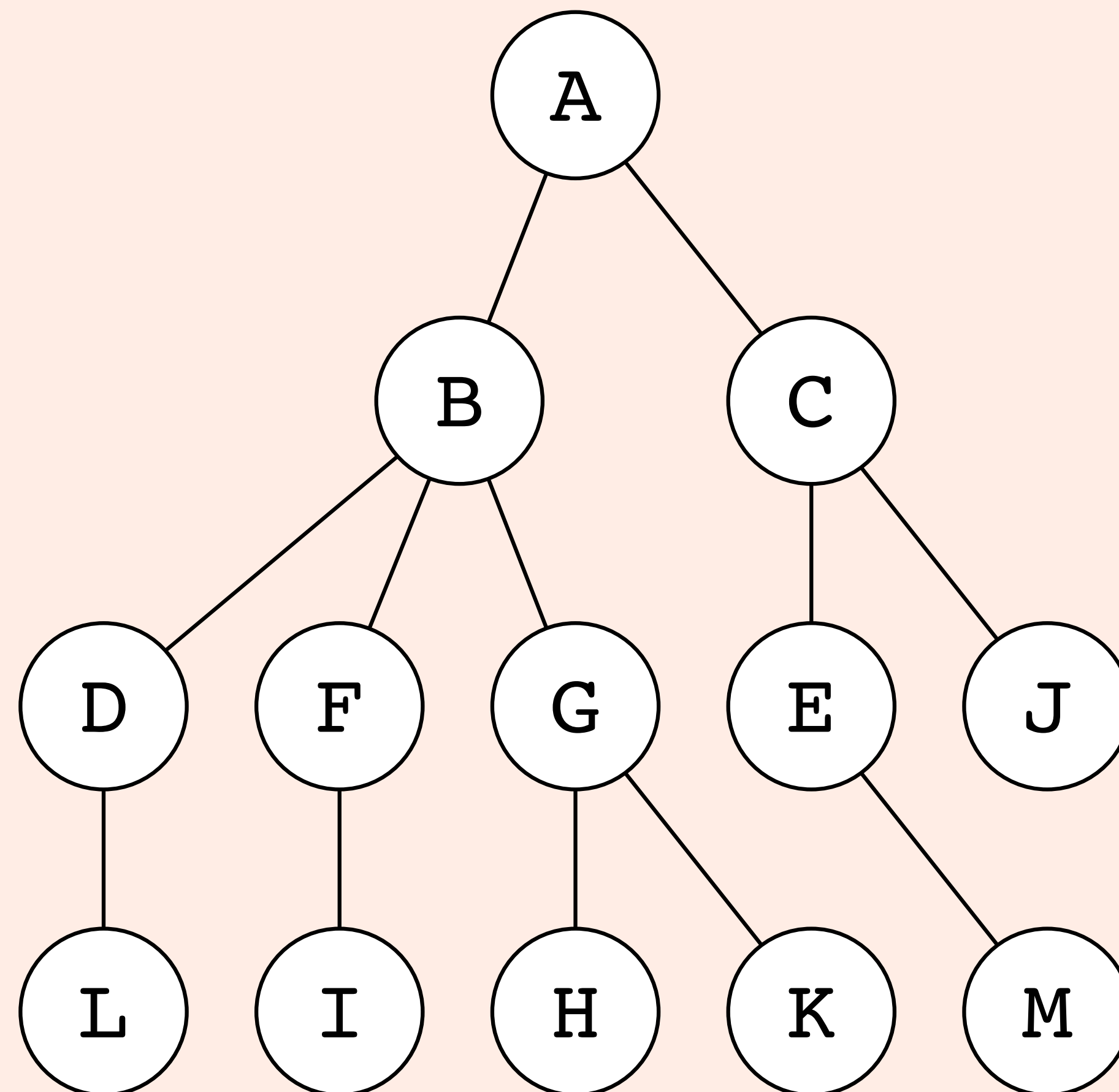
$A(B(E(J), F(K, L)), C(G, H(M)), D(I))$



Solution (2)



$A(B(D(L), F(I), G(H, K)), C(E(M), J))$



Itération sur un arbre

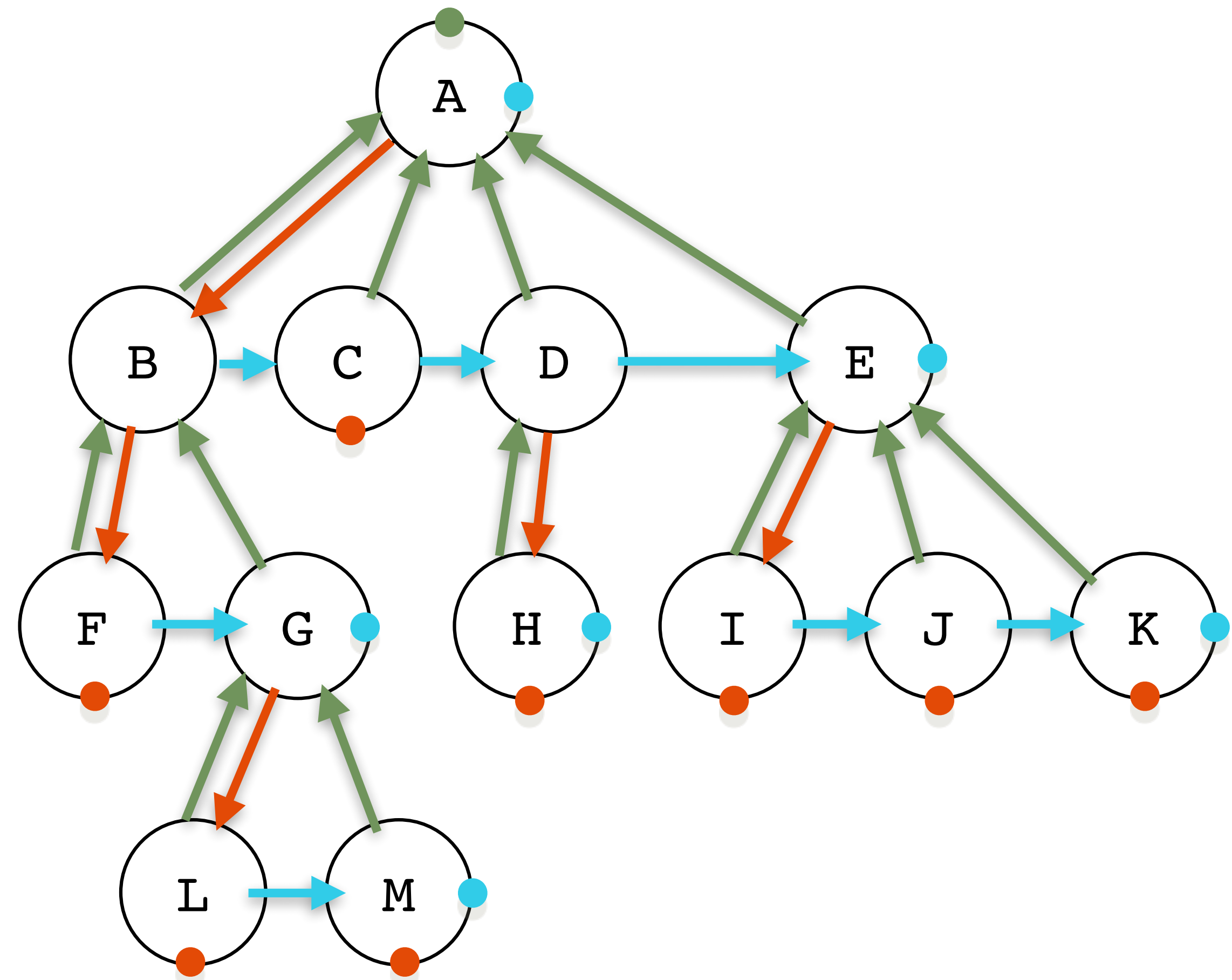




- Itérer = parcourir avec **début**, **suivant**, **fin**

```
fonction pour_tous (r, fn)
n ← début(r)
tant que n != fin(n)
    fn(n)
    n ← suivant(n)
```

```
structure Noeud<T>
    T étiquette
    Noeud* parent
    Noeud* aîné
    Noeud* puiné
```



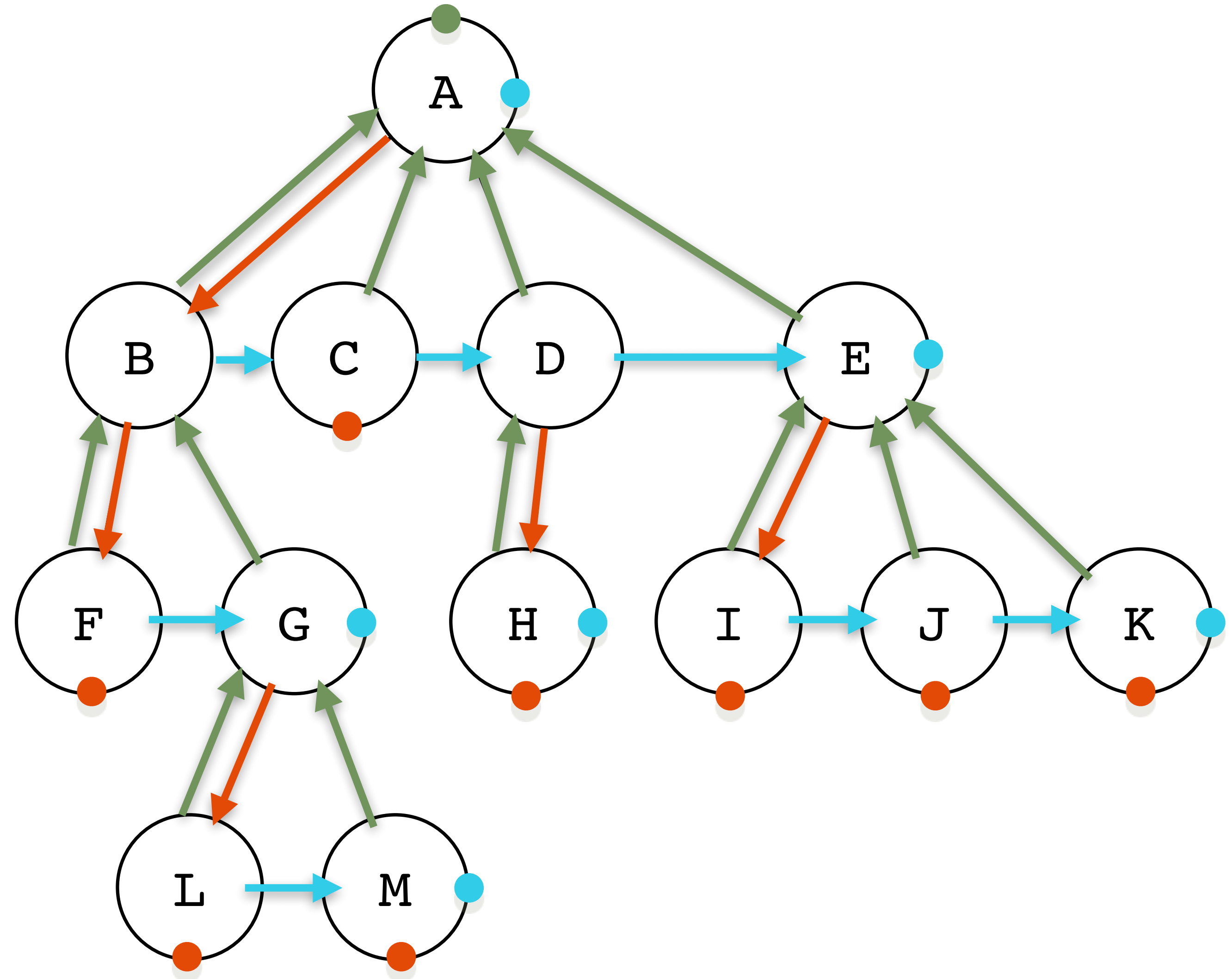
Itération en pré-ordre



```
fonction début(r)
    retourner r
```

```
fonction fin(r)
    retourner ∅
```

```
fonction suivant(r)
    si r.ainé != ∅
        retourner r.ainé
    sinon
        tant que r.puiné == ∅
            et r.parent != ∅
                r ← r.parent
        retourner r.puiné
```



Itération en post-ordre



```

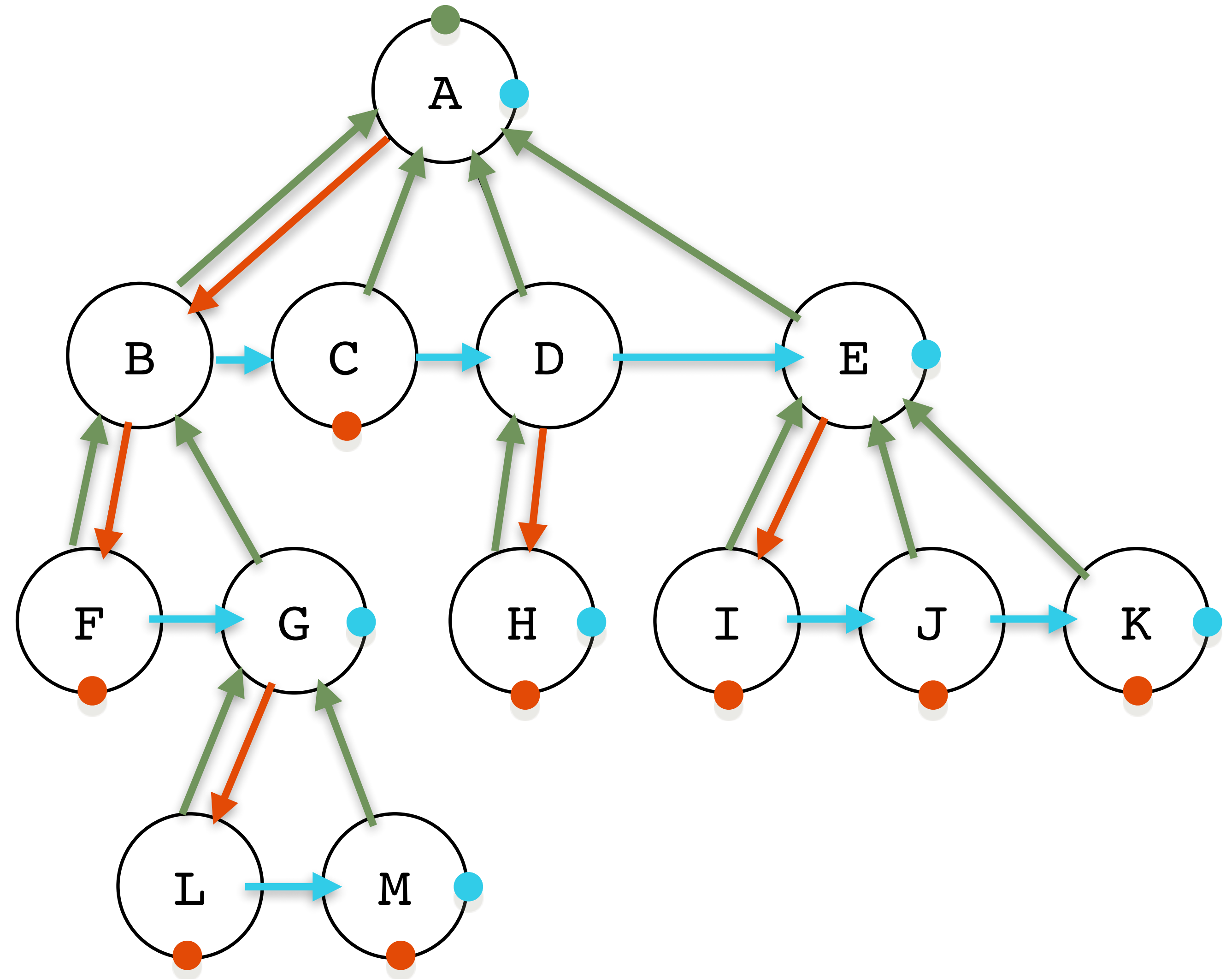
fonction début (r)
  si r == ∅, retourner ∅
  tant que r.ainé != ∅
    r ← r.ainé
  retourner r
  
```

```

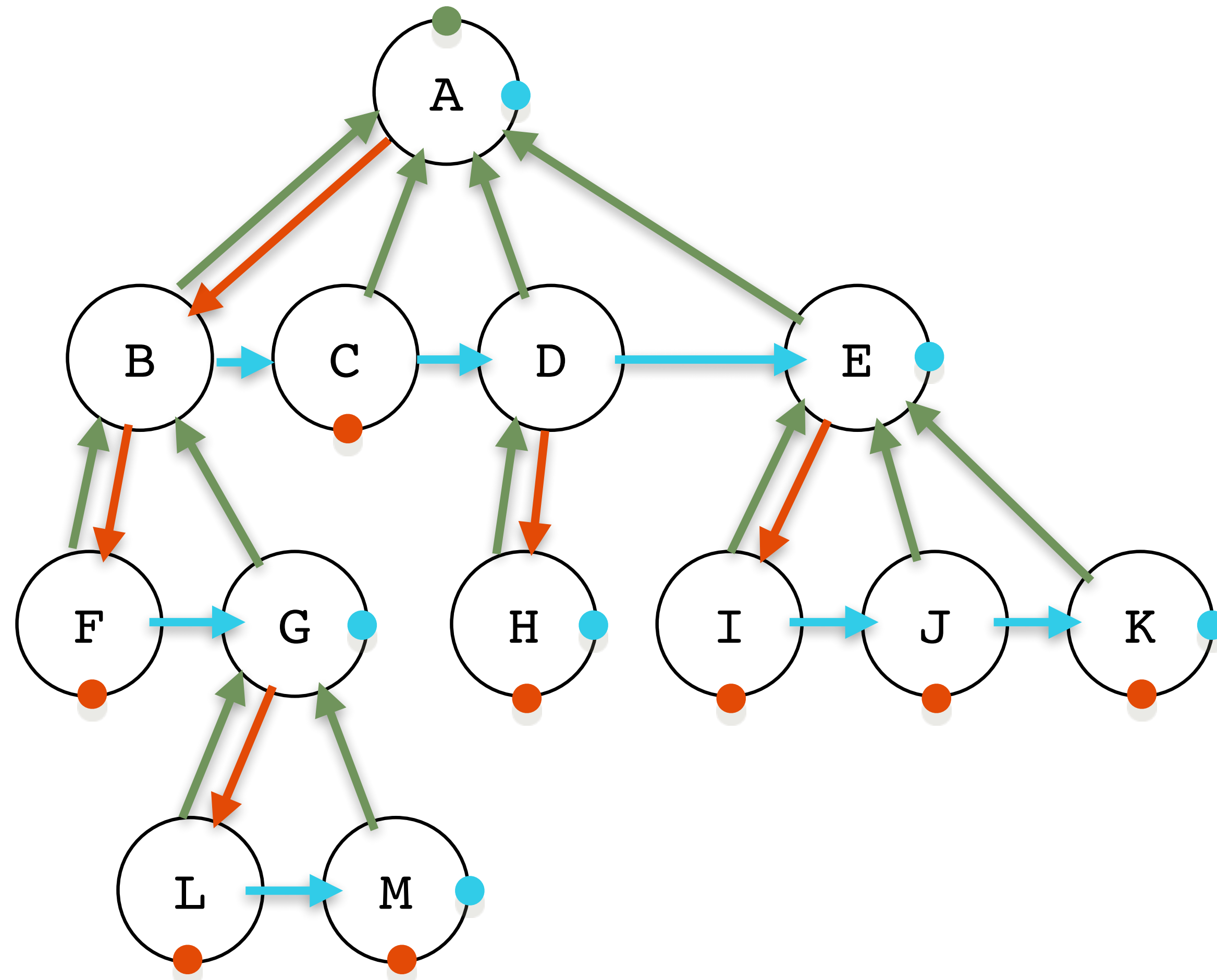
fonction fin(r)
  retourner ∅
  
```

```

fonction suivant(r)
  si r.puiné != ∅
    retourner début(r.puiné)
  sinon
    retourner r.parent
  
```



Traçons ces 2 parcours



Pré: A-B-F-G-L-M-G-B-C-D-H-D-E-I-J-K-E-A

Post: A-B-F-G-L-M-G-B-C-D-H-D-E-I-J-K-E-A

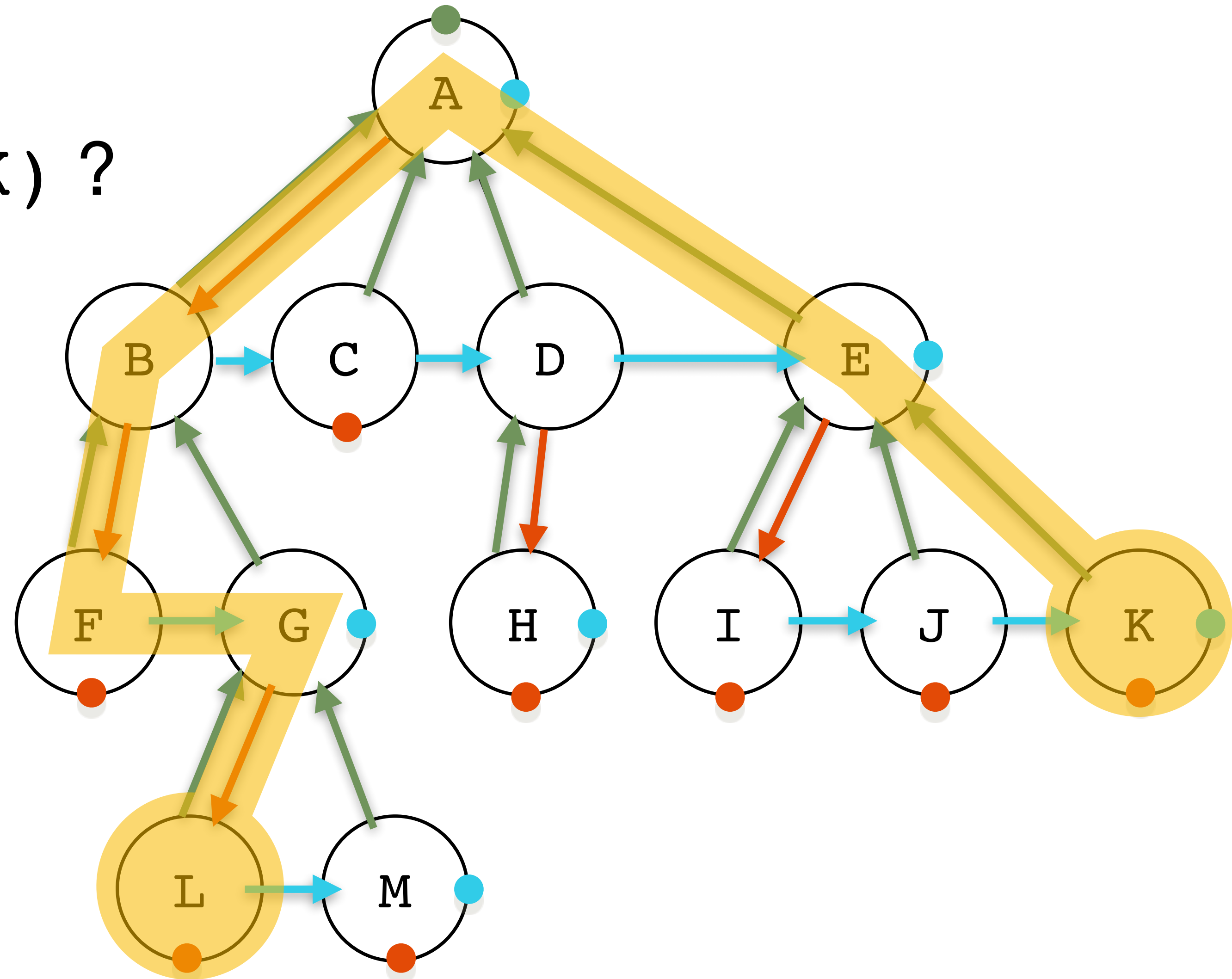


Comment calculer suivant (K) ?

On n'a accès qu'à K, donc pas de queue dans laquelle se souvenir de L...

Il faut utiliser les liens à disposition...

K-E-A-B-F-G-L



Itération en largeur (2)



(niv.1) A—

(niv.2) B—

C—D—

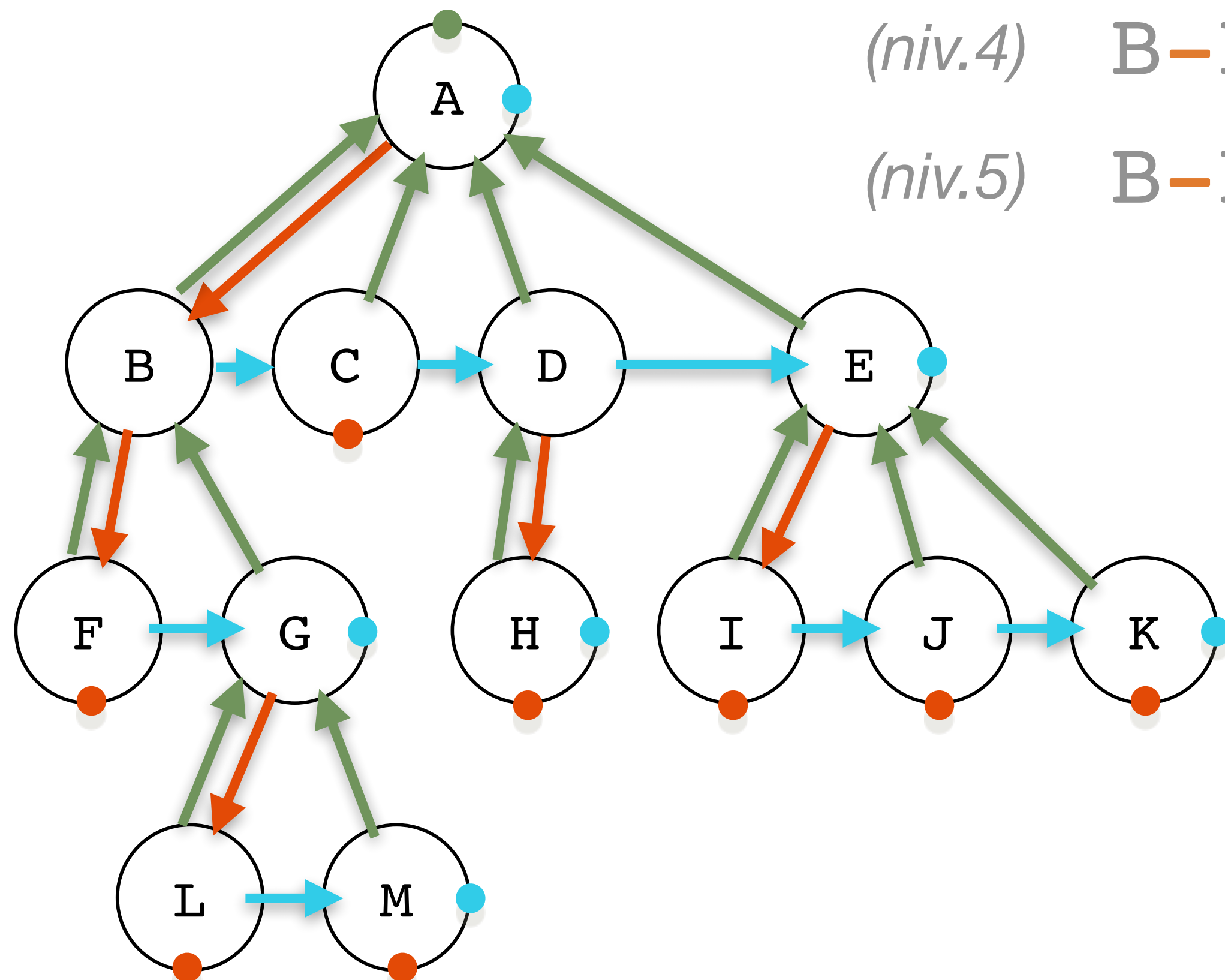
E—A—

(niv.3) B—F—

G—B—C—D—H—D—E—I—J—K—E—A—

(niv.4) B—F—G—L—M—G—B—C—D—H—D—E—I—J—K—E—A—

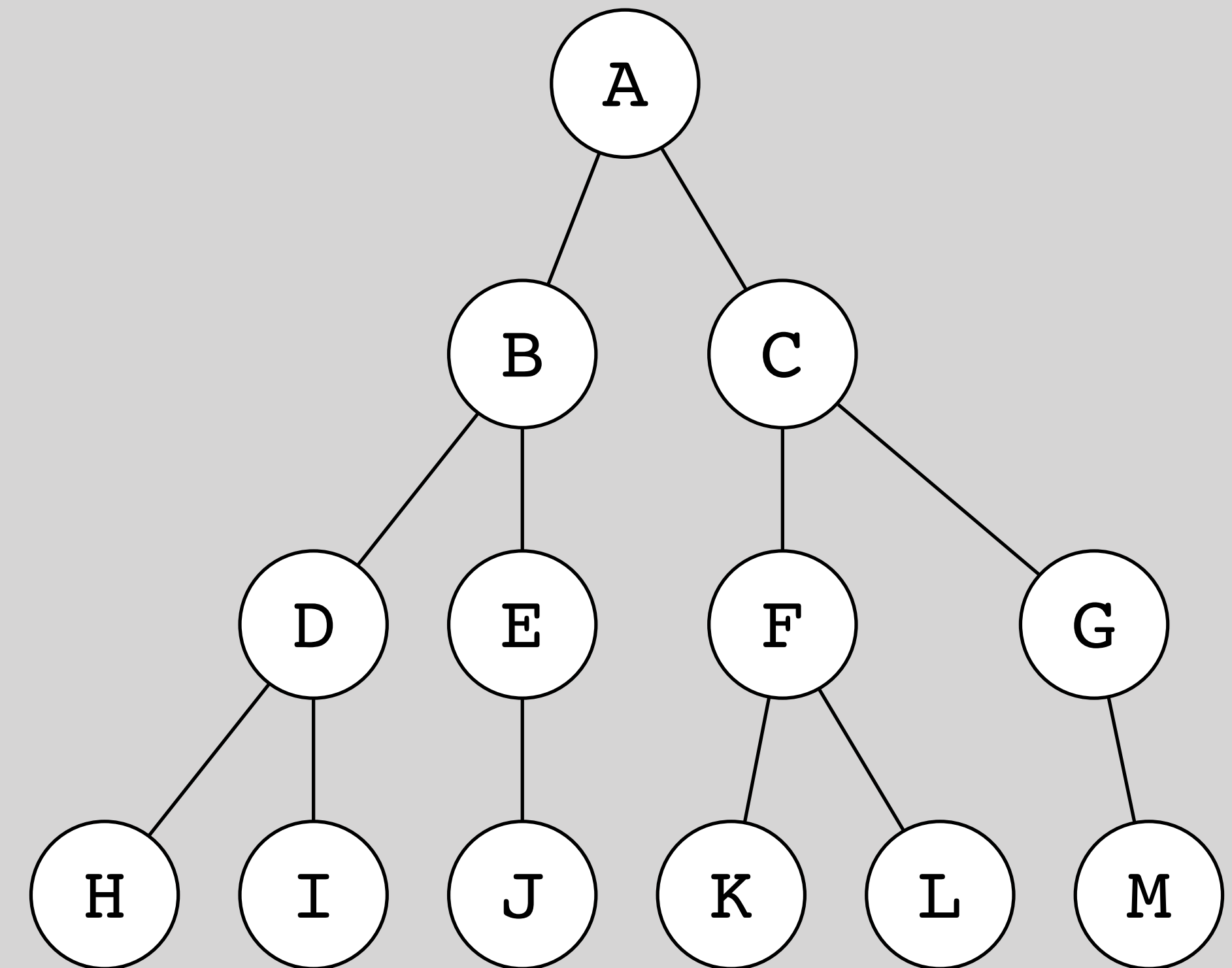
(niv.5) B—F—G—L—M—G—B—C—D—H—D—E—I—J—K—E—A



Très inefficace.
On évite.



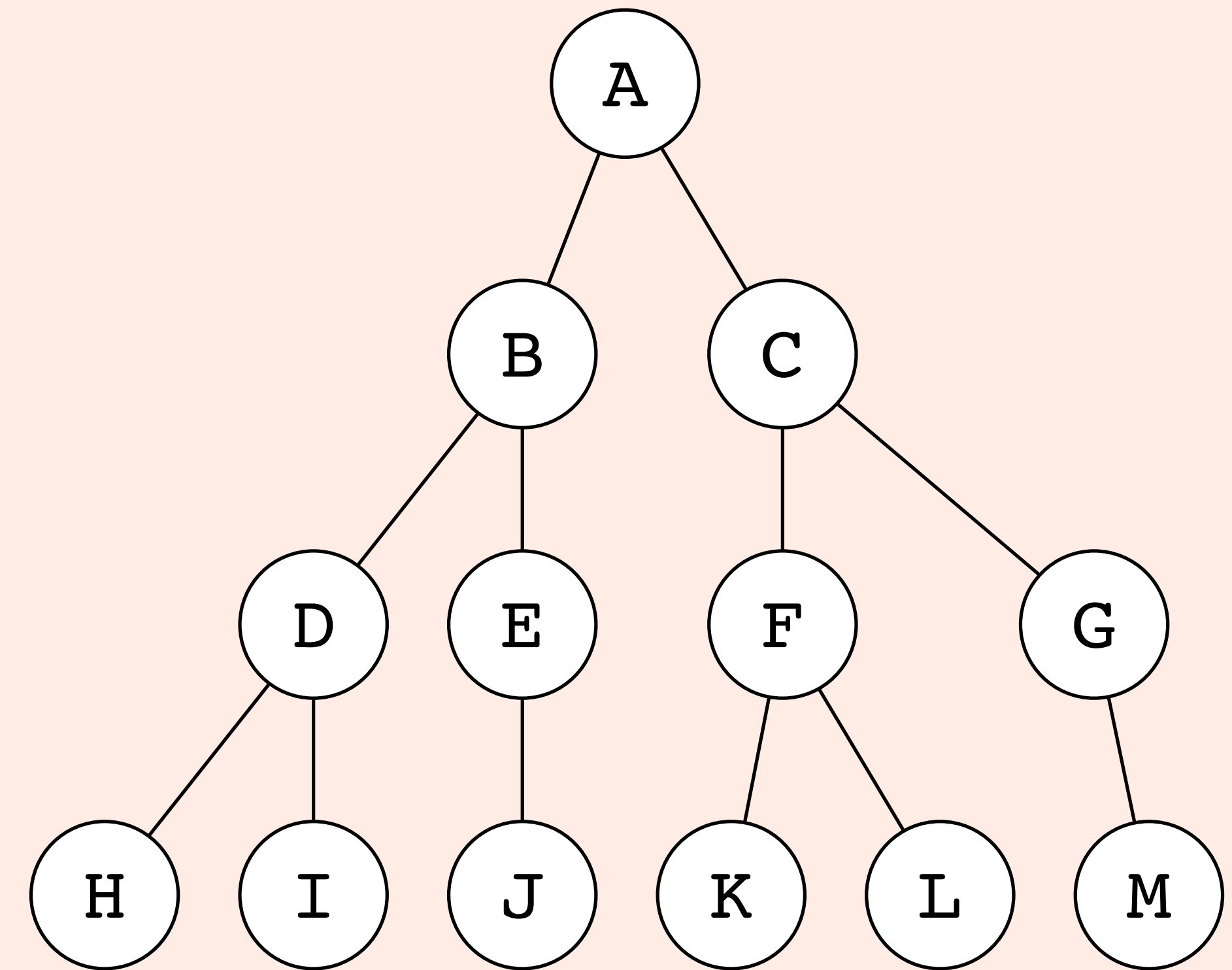
- Quel noeud suit J en pré-ordre ?
- Quel noeud suit K en pré-ordre ?
- Quel noeud suit J en post-ordre ?
- Quel noeud suit B en post-ordre ?



- Tracez les noeuds par lesquels passe l'itération en pré-ordre sur cet arbre



- Quel noeud suit J en pré-ordre ? **C**
- Quel noeud suit K en pré-ordre ? **L**
- Quel noeud suit J en post-ordre ? **E**
- Quel noeud suit B en post-ordre ? **K**



- Tracez les noeuds par lesquels passe l'itération en pré-ordre sur cet arbre **A-B-D-H-I-D-E-J-E-B-C-F-K-L-F-G-M-G-C-A**

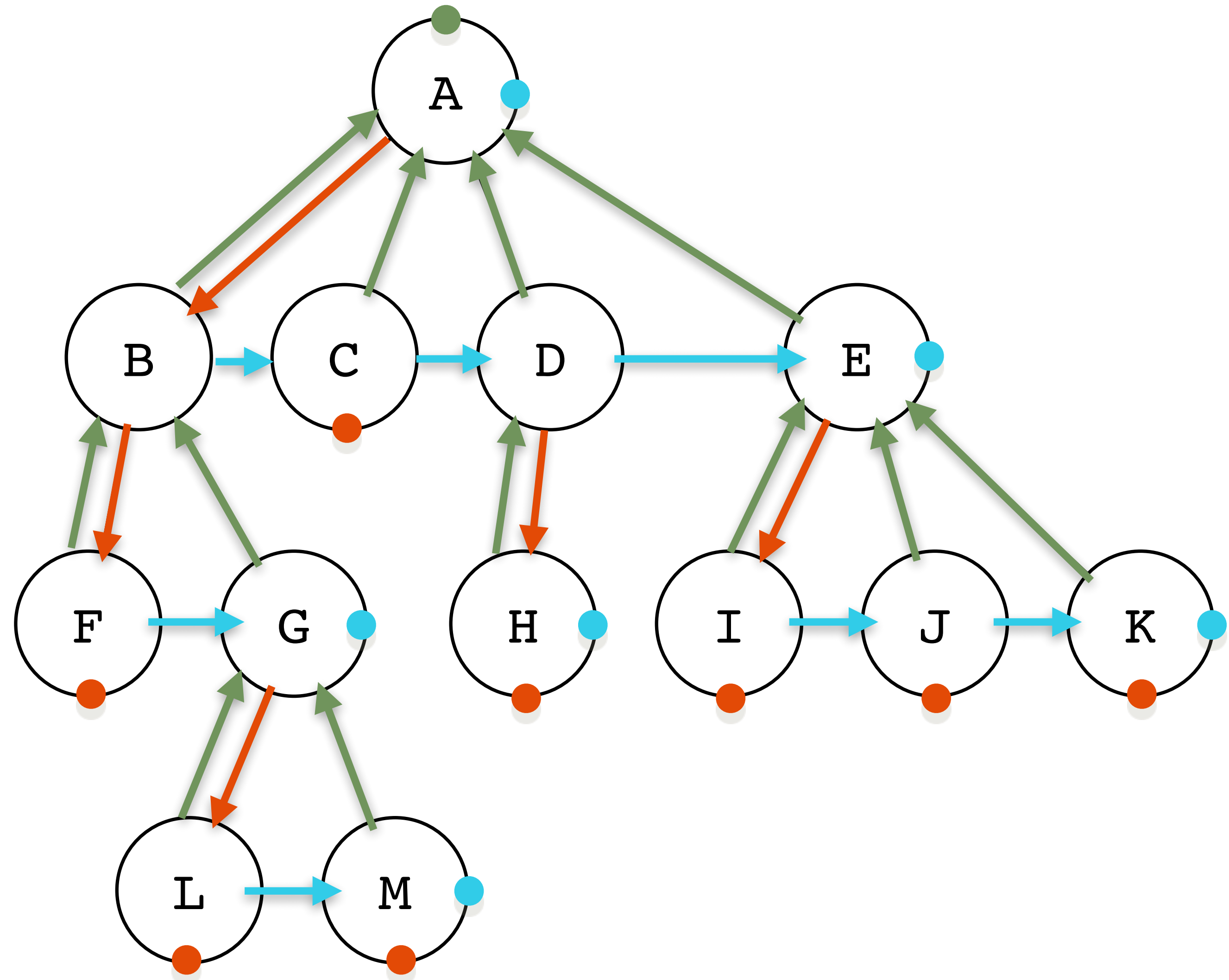
6. et encore...





- Dépendent fortement de la structure de Noeud utilisée
- En cas de suppression, que fait-on des enfants du noeud supprimé ?

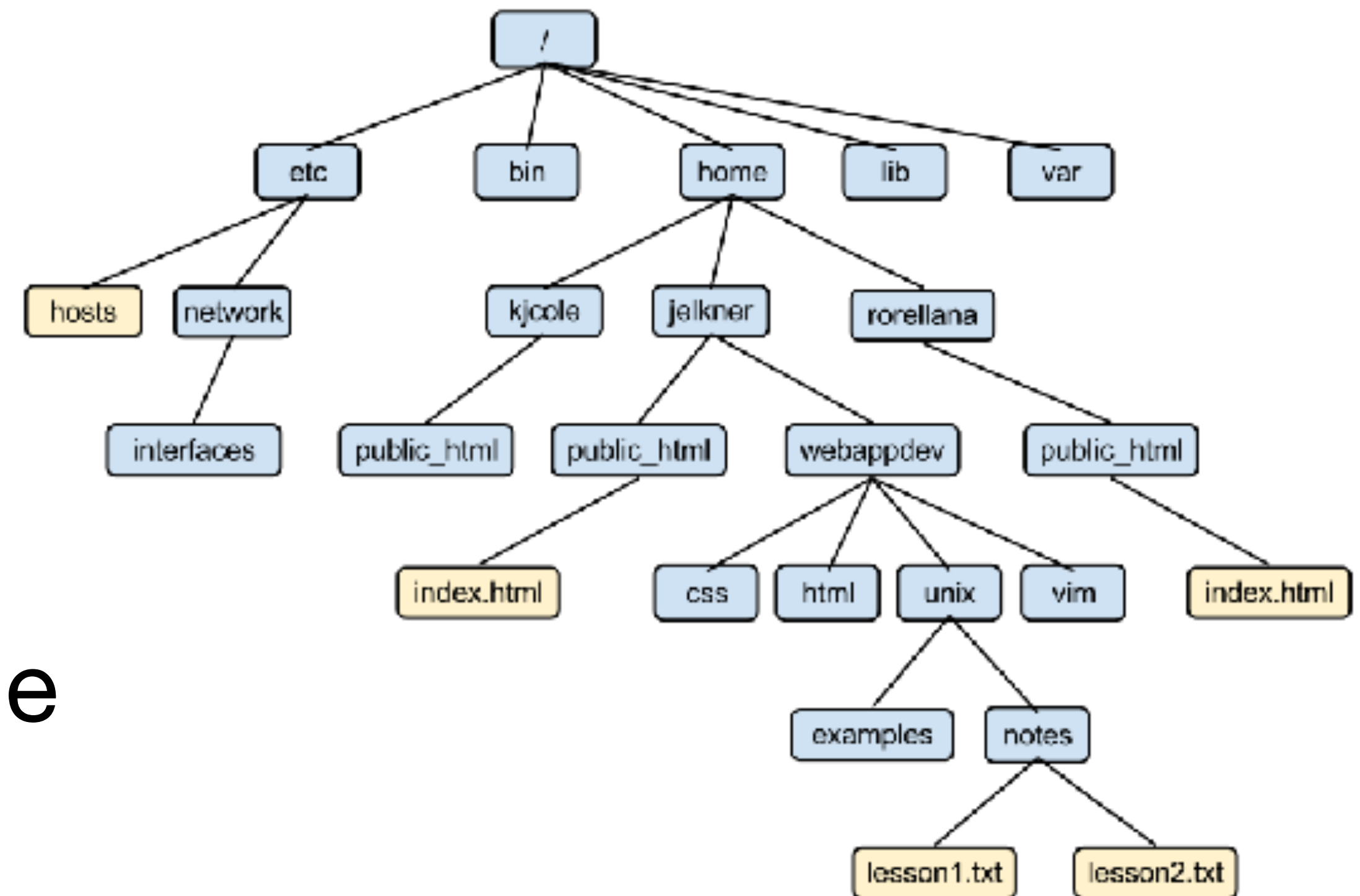
(Sera couvert plus tard pour des arbres particuliers)



Noeuds de types différents

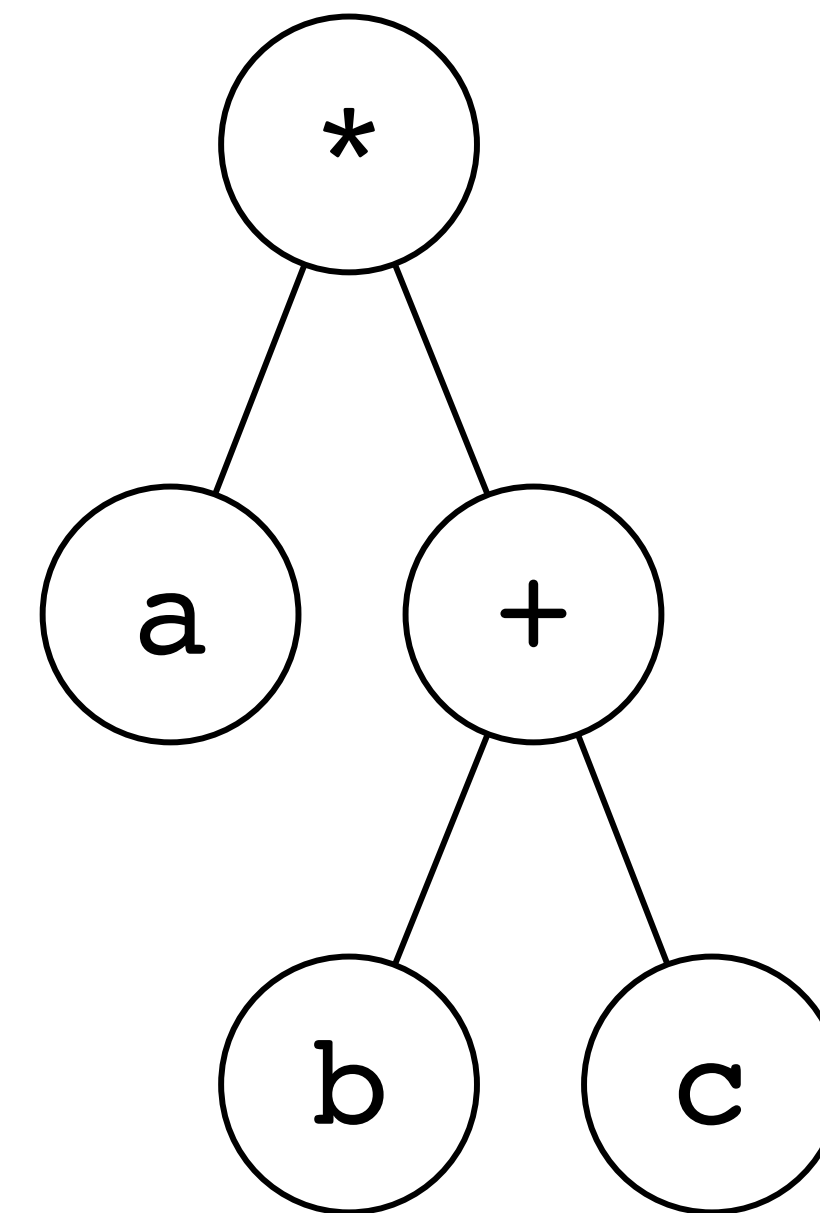


- Dans l'arbre d'un système de fichier, on a 2 types de noeuds
- Répertoires $\Leftrightarrow$ Noeud interne
- Fichiers $\Leftrightarrow$ feuille





- Dans l'arbre représentant une expression arithmétique
- Opérateur $\Leftrightarrow$ Noeuds interne
- Opérandes $\Leftrightarrow$ Feuille
- Pas de noeud de degré 1



$(a * (b + c))$



- En deuxième année
- Programmation Orientée Objet
- Modèles de Conception Réutilisables
- Génie Logiciel

