



LICENCE INFORMATIQUE

# ALGORITHMES & STRUCTURES DE DONNÉES

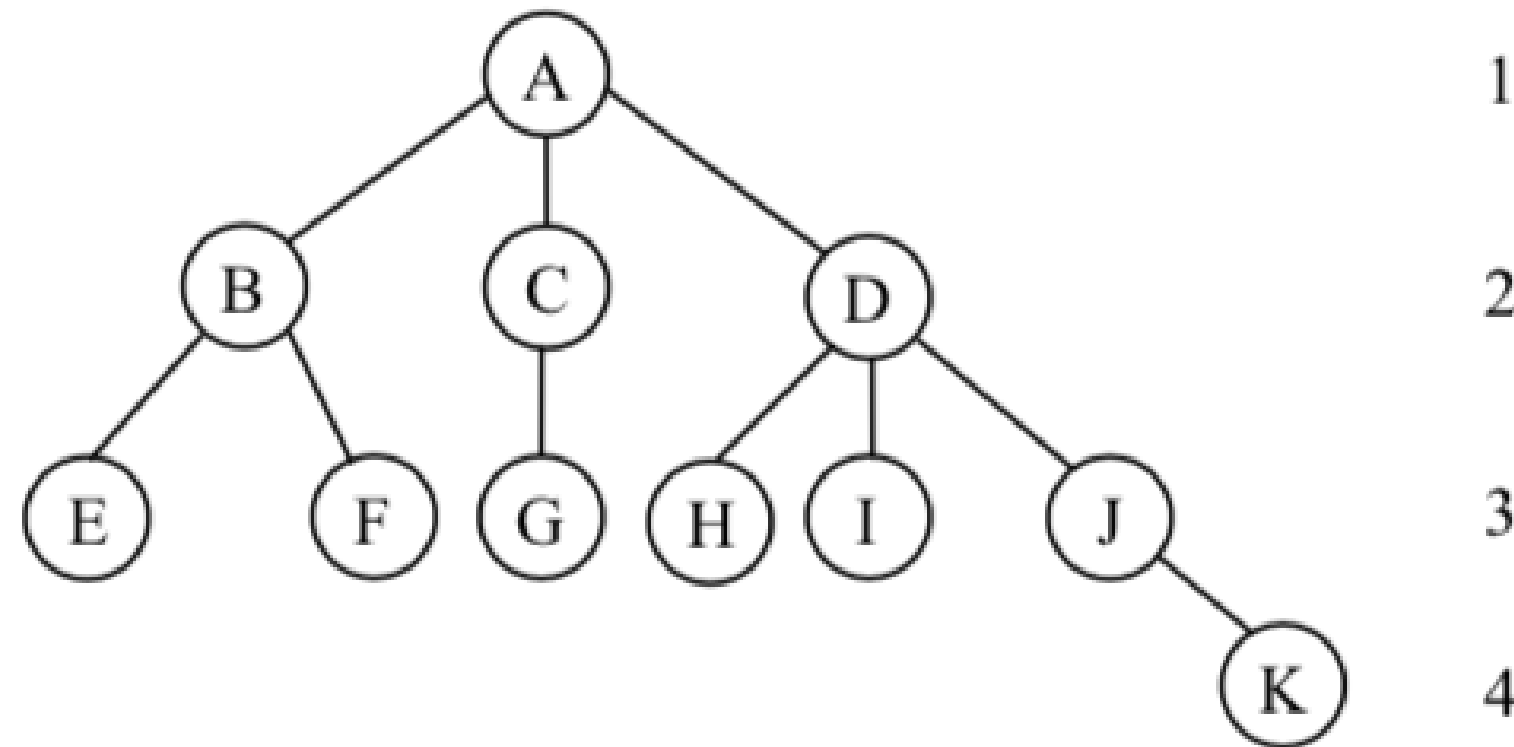
## Arbres

Présenté par BABACAR DIOP

# Les arbres

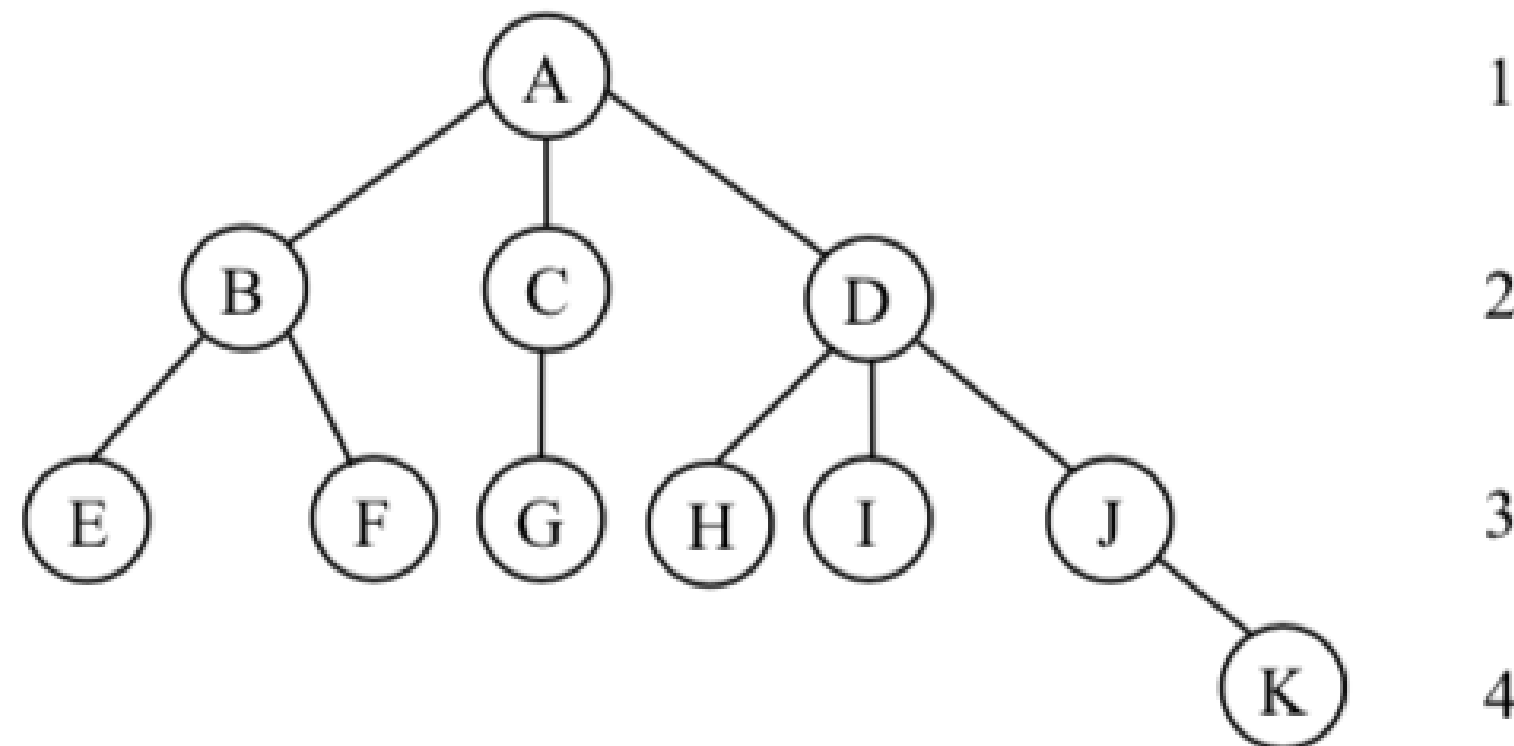
**Définition:** un arbre peut être défini comme un ensemble fini de nœuds non vide, tel que:

1. Il existe un nœud spécialement désigné appelé racine,
2. Les nœuds restants sont partitionnés en zéro ou plusieurs arbres disjoints  $T_1, T_2 \dots T_n$  sont appelés les sous-arbres de la racine  $R$



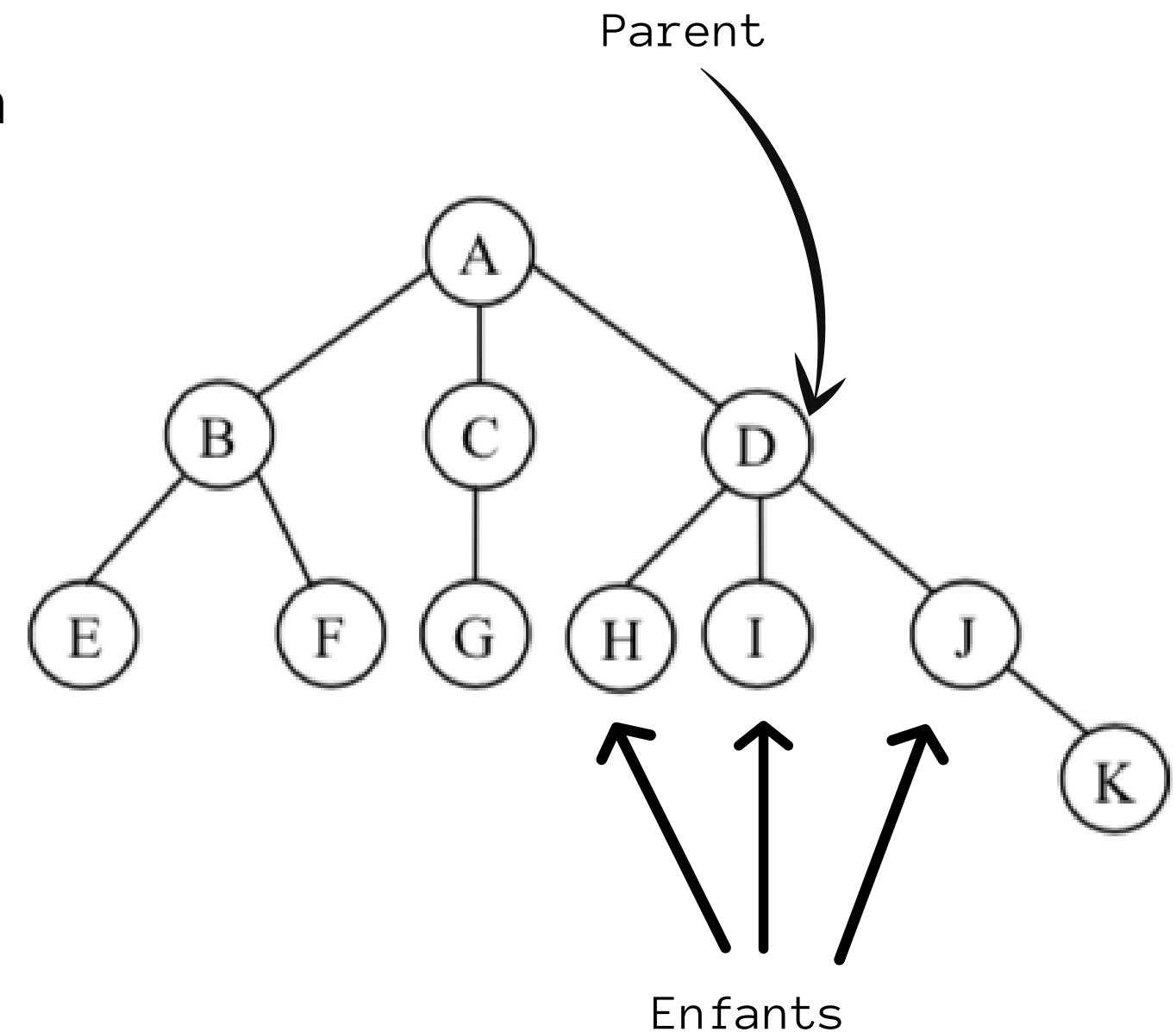
# Terminologies

- **Noeud** (ou sommet): Un noeud représente l'élément d'information avec les branches vers d'autres éléments. Cet arbre a 11 noeuds (A, B, C, D, E, F, G, H, I, J et K).
- **Racine**: un noeud sans parent est appelé noeud racine. Dans la figure, A est le noeud racine.



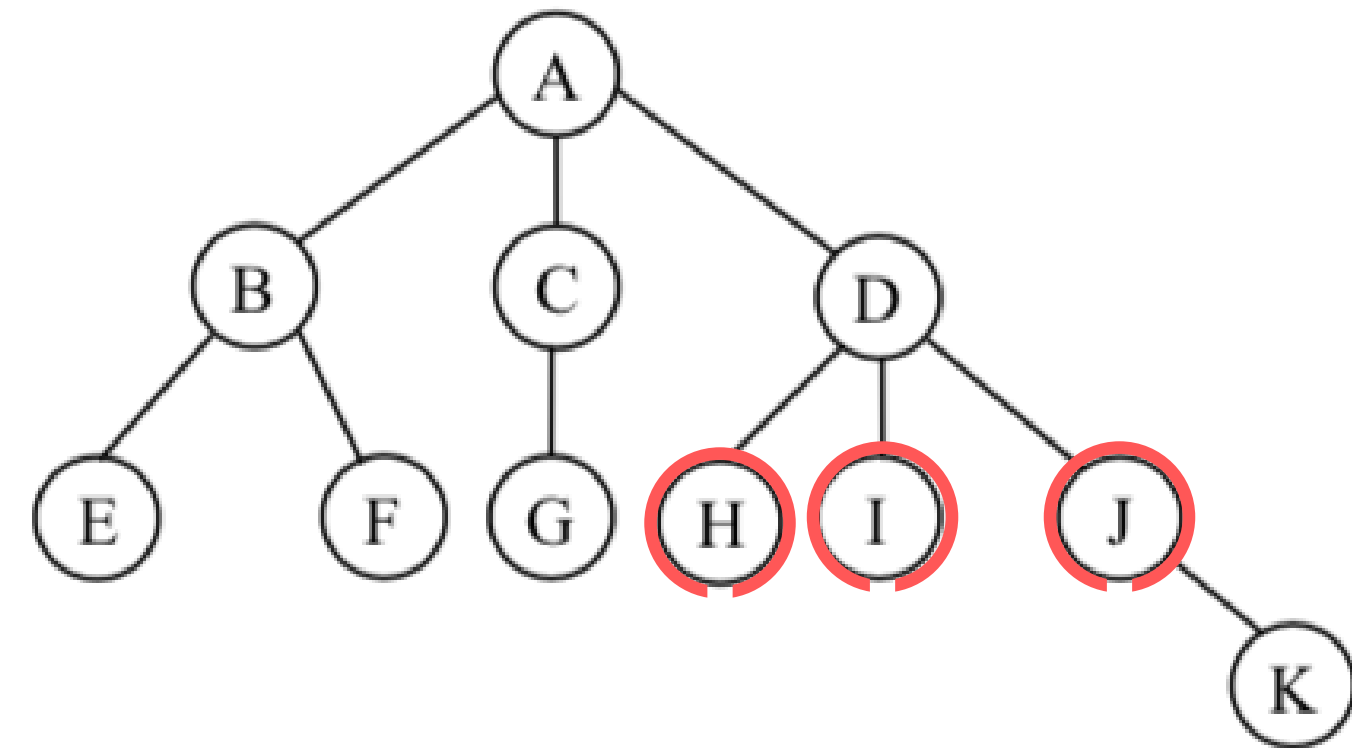
# Terminologies

- **Nœud parent** (ou prédécesseur): Supposons que  $N$  est un nœud dans un arbre avec les successeurs  $s_1, s_2 \dots s_n$  alors  $N$  est appelé le parent (ou prédécesseur) des successeurs. Chaque nœud de l'arbre, à l'exception de la racine, a un parent unique.
- **Enfants**: Les successeurs d'un nœud  $N$  sont appelés enfants de  $N$ . Les successeurs gauche et droit du nœud  $N$  sont respectivement appelés enfant gauche et enfant droit du nœud  $N$ .



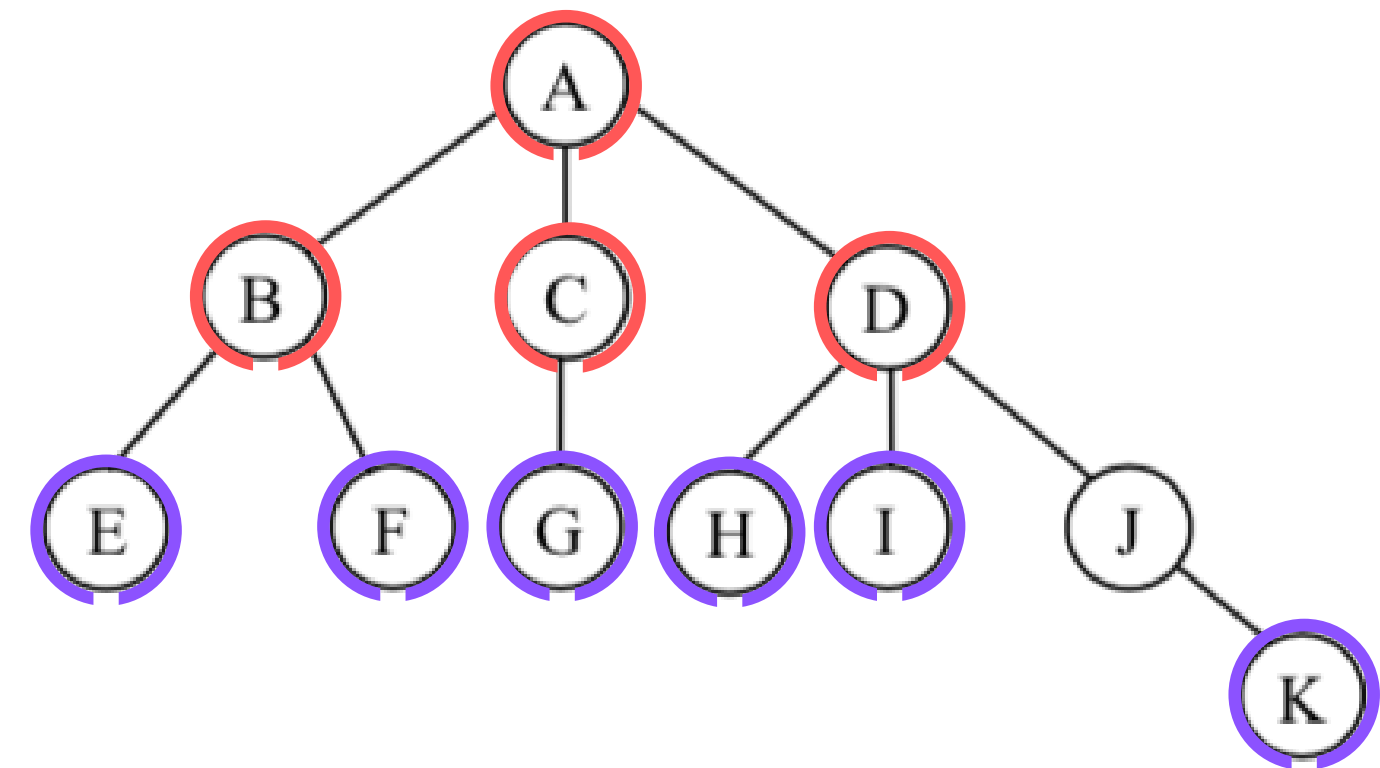
# Terminologies

- **Frères et sœurs**: les enfants (ou les nœuds) du même parent sont considérés comme des frères et sœurs. Dans la figure ci-dessus, H, I et J sont frères et sœurs.
- **Degré d'un nœud**: Le nombre de sous-arbres (ou enfants) d'un nœud est appelé son degré.
- **Degré d'un arbre**: Le degré d'un arbre est le degré maximum des nœuds de l'arbre.



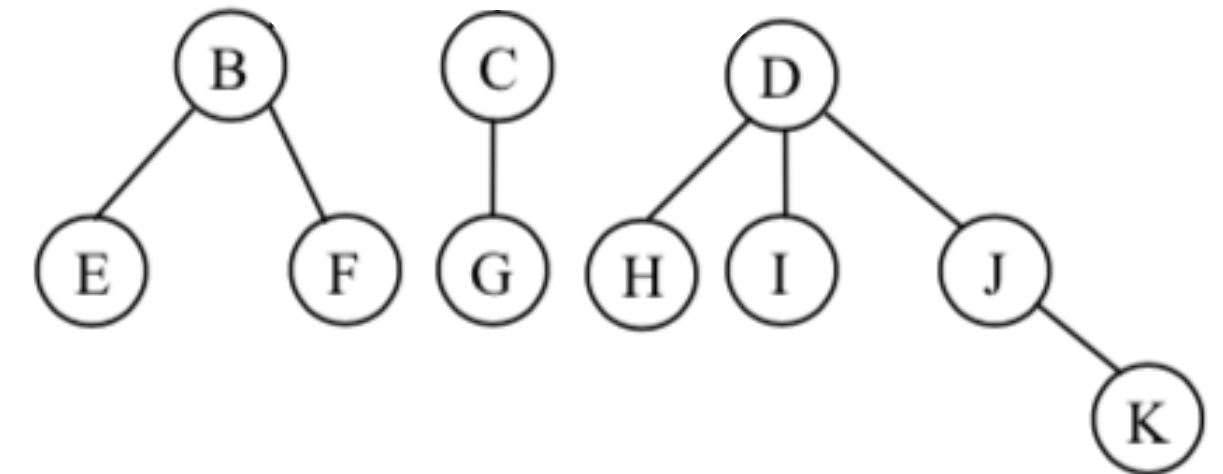
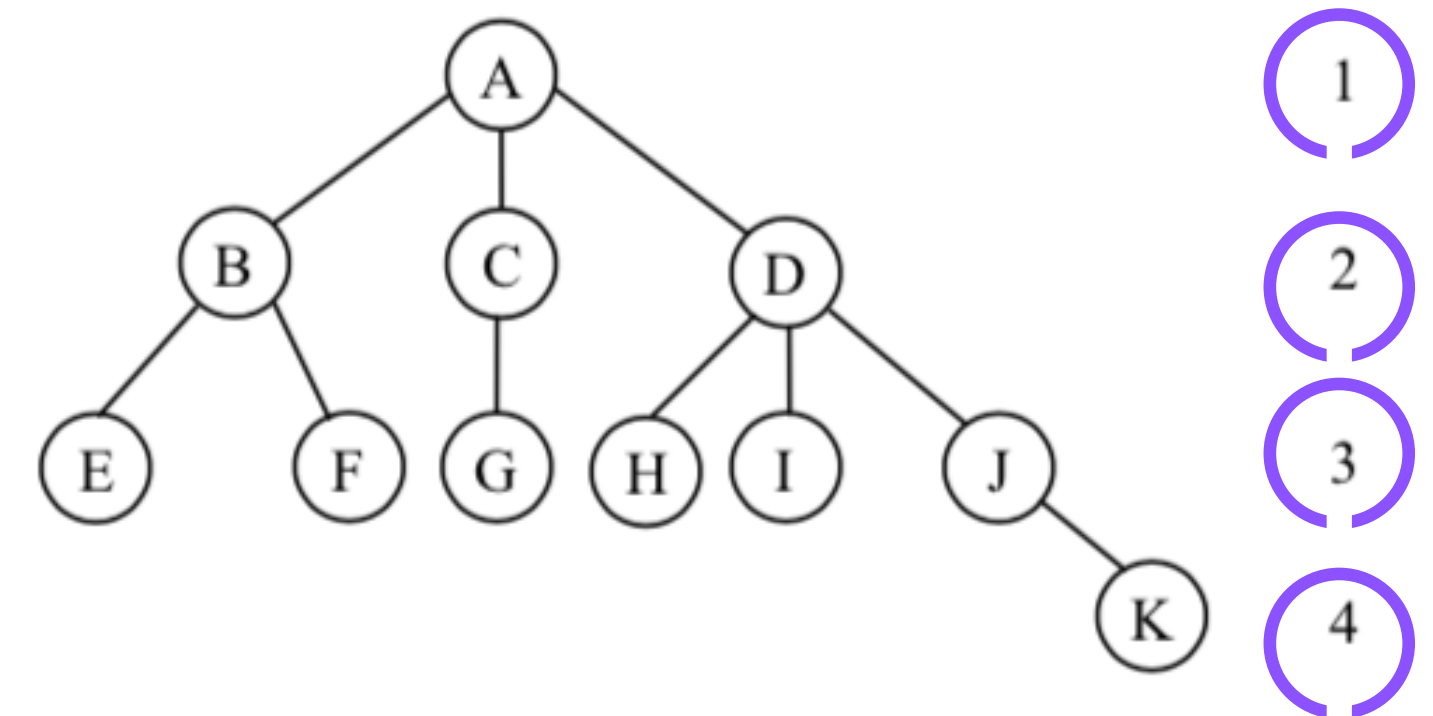
# Terminologies

- **Noeud interne** (ou noeud non terminal): le noeud avec au moins un enfant est appelé noeud interne.
- **Noeuds externes** (ou noeud feuille): Les noeuds qui ont le degré zéro sont appelés noeuds externes ou noeuds feuilles ou terminaux.
- **Niveau**: Le niveau d'un nœud est défini comme suit:
  - i) La racine de l'arbre est au niveau un.
  - ii) Si un nœud est au niveau L, alors ses enfants sont au niveau  $L + 1$ .



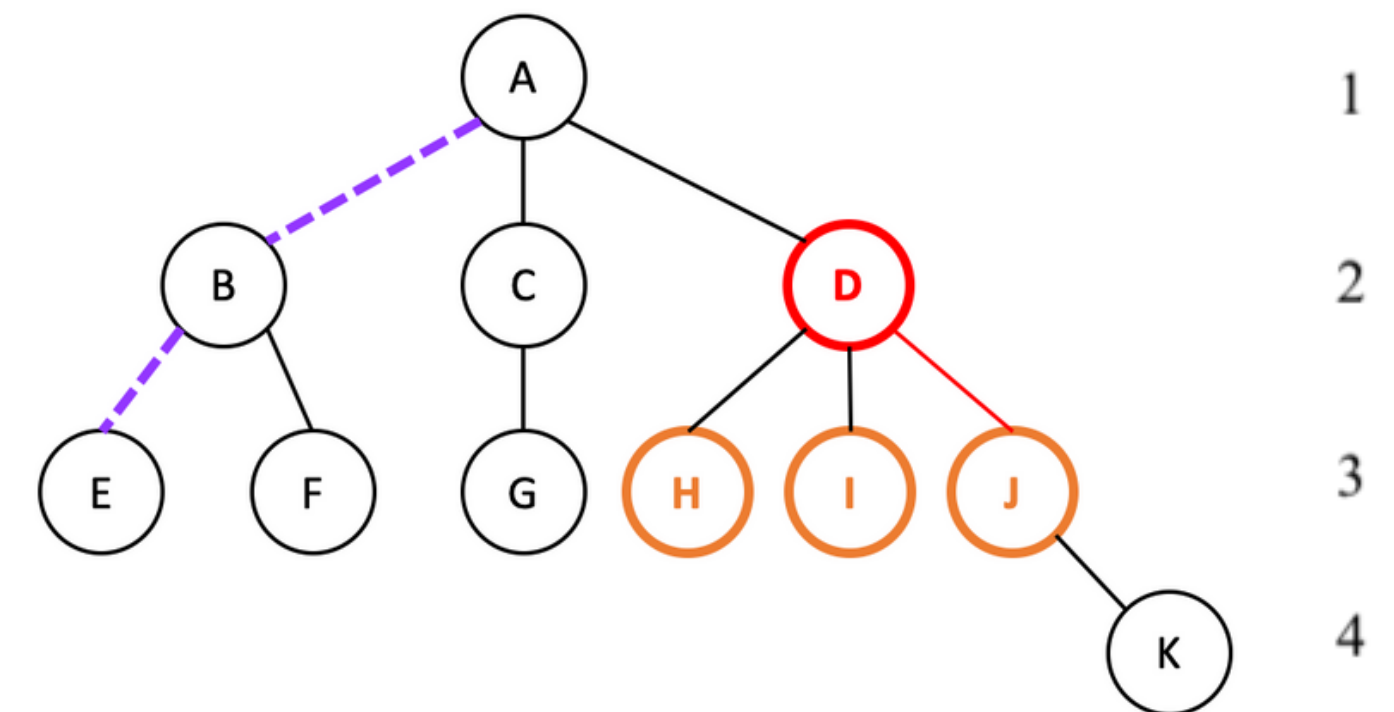
# Terminologies

- **Hauteur** (ou profondeur): la hauteur ou la profondeur d'un arbre est définie comme le niveau maximum de n'importe quel nœud de l'arbre. Dans la figure, la hauteur de l'arbre est de 4.
- **Forêt**: Une forêt est un ensemble de zéro ou plusieurs arbres disjoints. La suppression du nœud racine d'un arbre se traduit par une forêt.



# Terminologies

- **Descendant**: Un nœud  $M$  dans l'arbre est appelé un descendant d'un autre nœud  $N$ , si  $M$  est accessible à partir de  $N$  en procédant de façon répétée du parent à l'enfant.
- **Ancêtre et descendant**: Un nœud  $N$  est appelé un ancêtre du nœud  $M$  si  $N$  est soit le parent de  $M$ , soit le parent d'un ancêtre de  $M$ ; le nœud  $M$  est appelé descendant du nœud  $N$ .
- **Arête**: la ligne allant d'un nœud  $N$  dans l'arbre à un successeur est appelée arête.



 Descendant

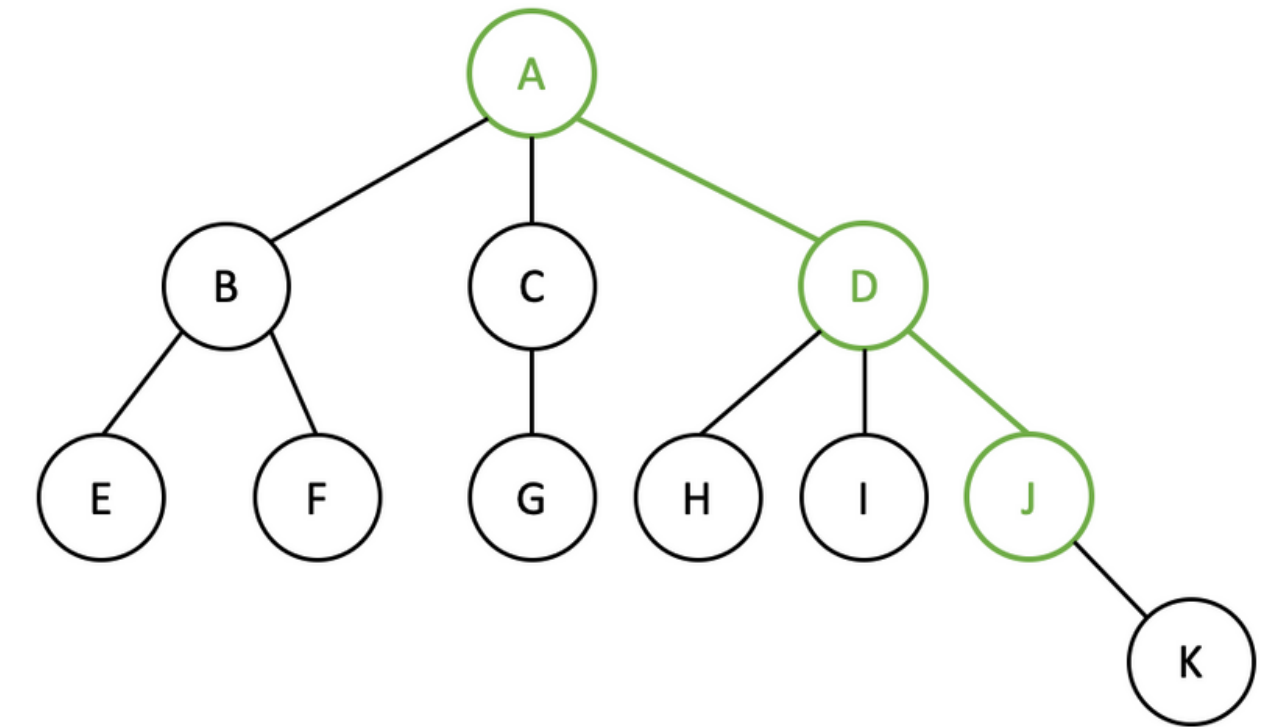
 Ancêtre

 Arête

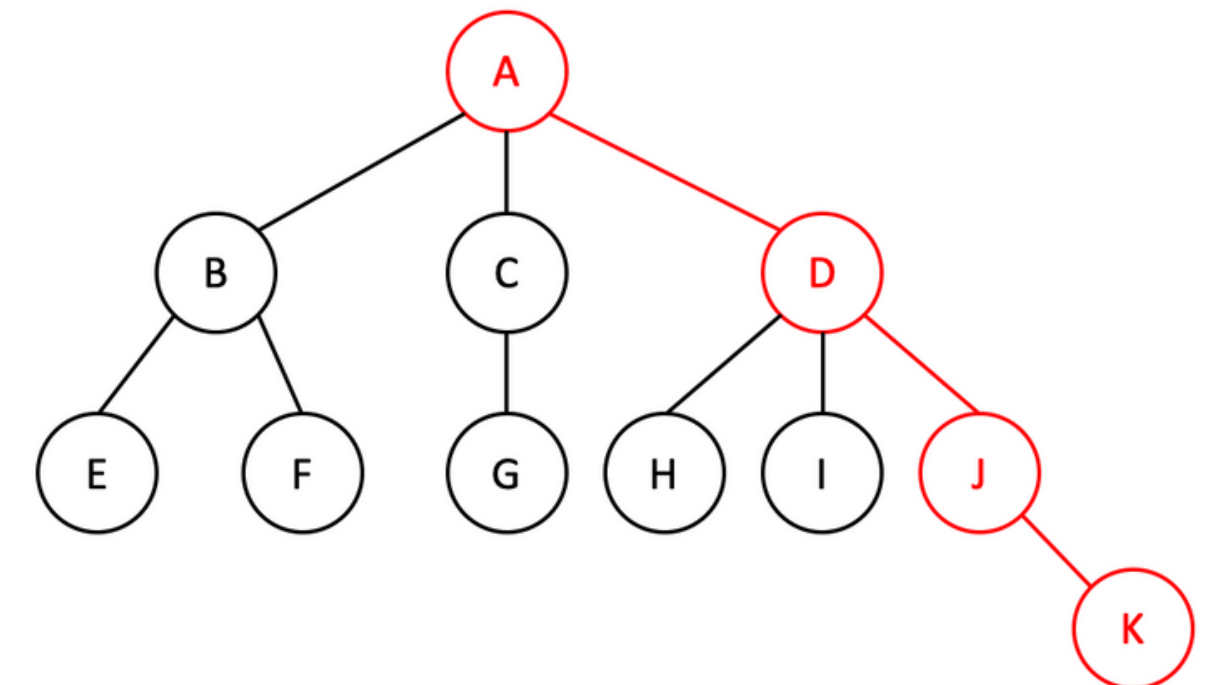


# Terminologies

- **Chemin et longueur de chemin:** une séquence de noeuds consécutifs du nœud source au nœud de destination est appelée chemin. Le nombre d'arêtes d'un chemin correspond à la longueur du chemin. Dans la figure ci-dessus,  $A \rightarrow D \rightarrow J$  est un chemin de longueur 2.



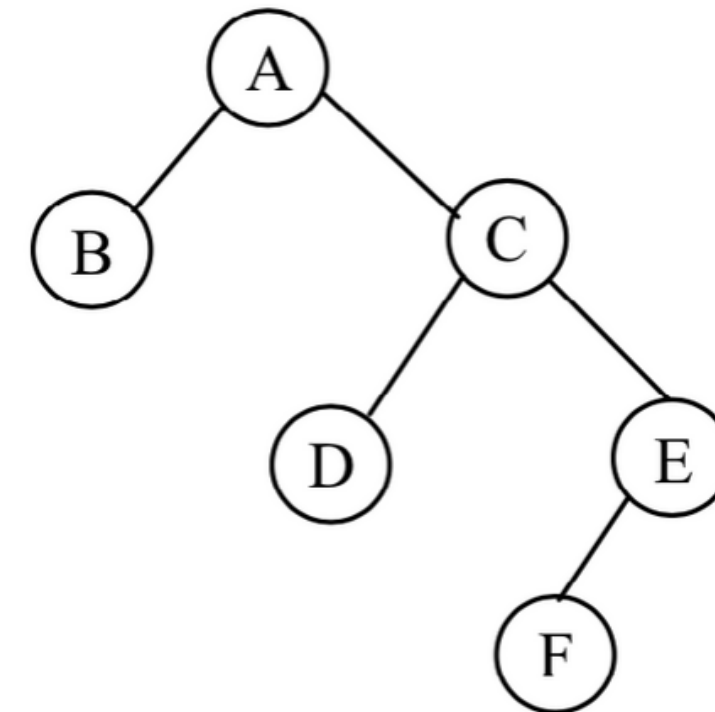
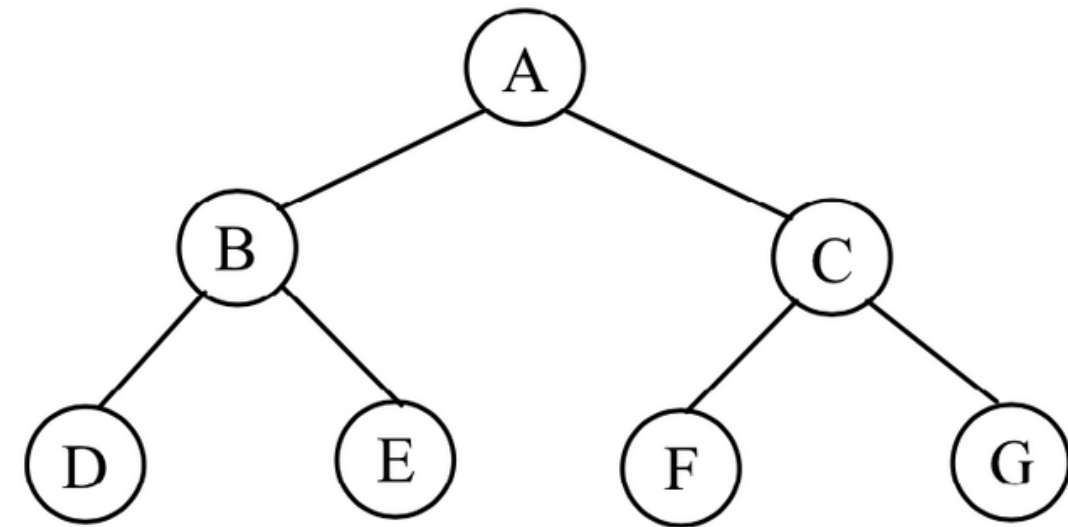
- **Branche:** un chemin se terminant par un nœud feuille est appelé branche. Dans la figure ci-dessus,  $A \rightarrow D \rightarrow J \rightarrow K$  est une branche.



# Arbres binaires

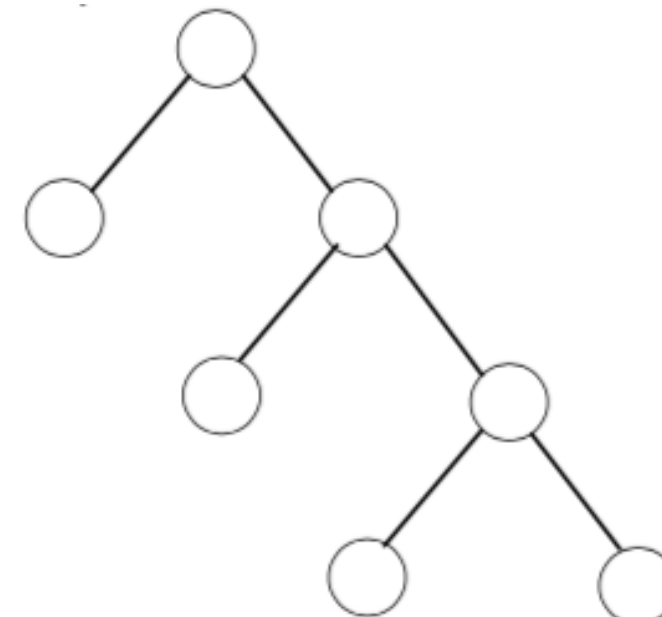
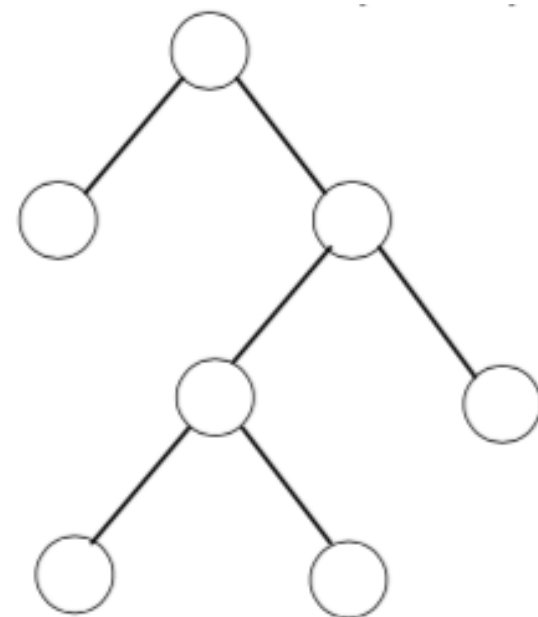
Un arbre binaire est un ensemble fini de nœuds, qui est :

- soit vide,
- soit constitué d'une racine et de deux arbres binaires disjoints appelés sous-arbre gauche et sous-arbre droit.



# Arbre strictement binaire

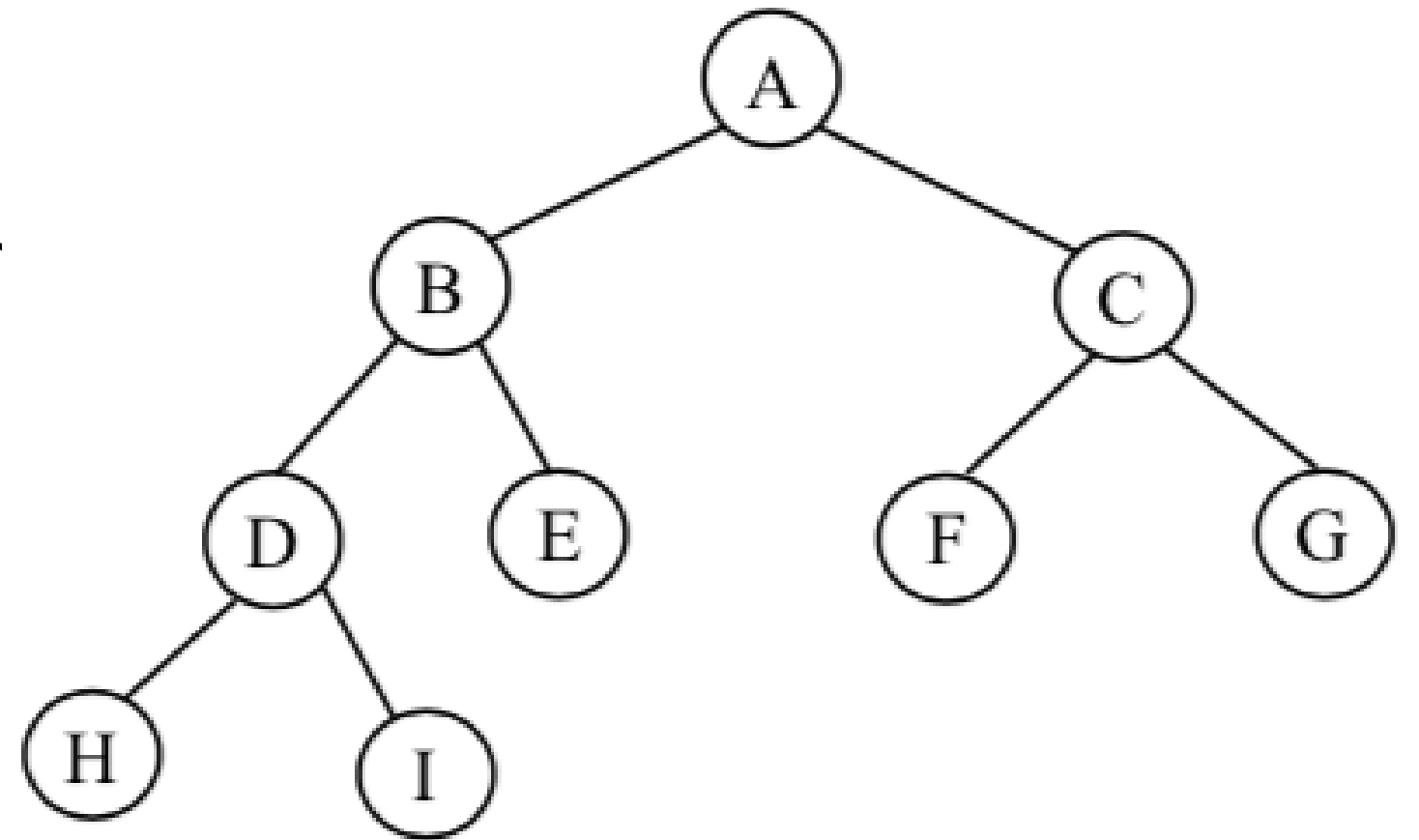
- Un arbre binaire est appelé arbre strictement binaire si chaque nœud non terminal a un sous-arbre gauche et droit non vide.
- Tous les nœuds non terminaux doivent toujours avoir exactement deux enfants non vides. Dans l'arbre strictement binaire, le degré de n'importe quel nœud est soit zéro soit deux, jamais de degré un.
- Un arbre strictement binaire avec  $n$  feuilles contient toujours  $2n - 1$  nœuds.





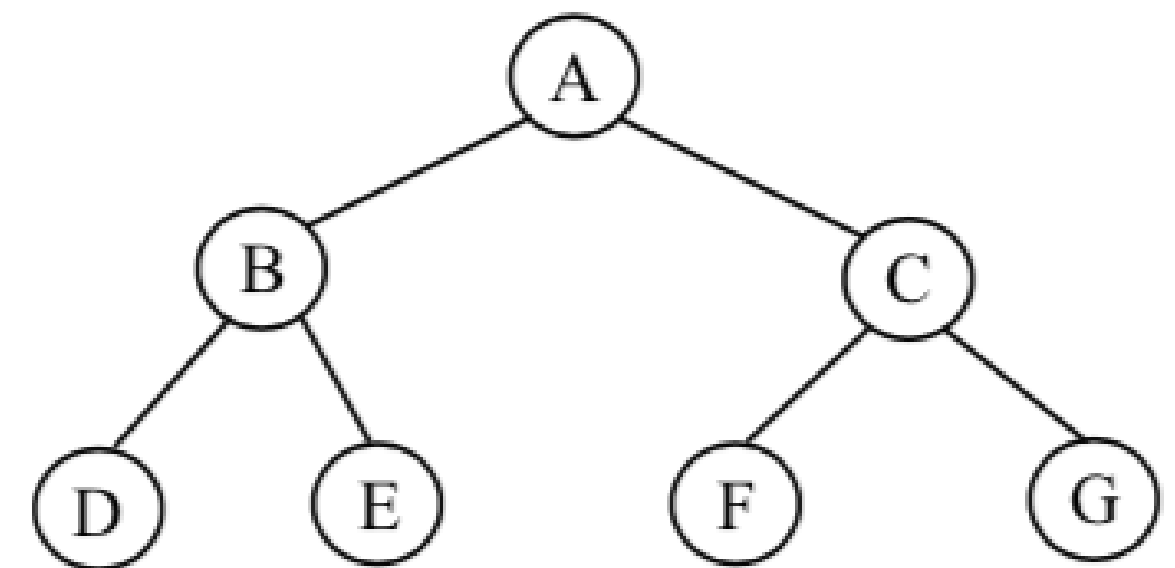
# Arbre complet

Un arbre binaire est appelé un arbre binaire complet dans lequel tous les niveaux sont remplis et le dernier niveau peut être partiellement rempli de gauche à droite et certaines feuilles les plus à droite peuvent manquer. L'arbre binaire complet utilise au maximum l'espace.



# Arbre parfait

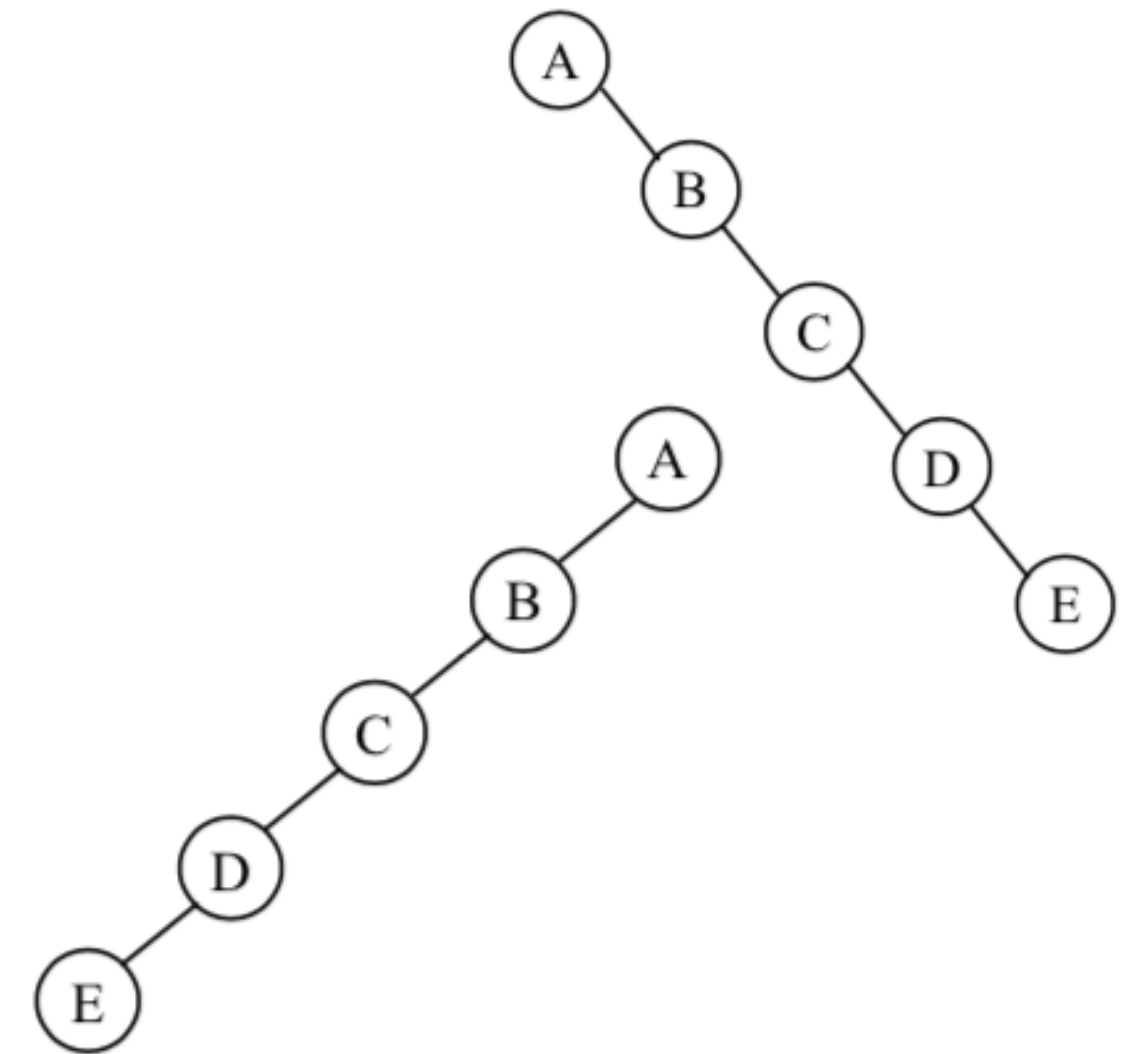
- Un arbre binaire de profondeur  $k \geq 1$ , contenant  $(2^k)-1$  nœuds, est appelé arbre binaire parfait.
- Notez qu'un arbre binaire peut avoir comme nombre maximal de nœuds  $(2^k)-1$ .
- Dans un arbre binaire complet, tous les nœuds internes ont deux enfants et toutes les feuilles sont au même niveau.
- Tout arbre binaire complet est un arbre binaire parfait, mais tous les arbres binaires complets ne sont pas un arbre binaire parfait.



Arbre binaire complet  
de profondeur 3.

# Arbre asymétrique

- Un arbre binaire, qui est dominé uniquement par des nœuds enfants gauches ou des nœuds enfants droits, est appelé un arbre binaire asymétrique.
- La hauteur d'un arbre binaire asymétrique de  $n$  nœuds est  $n$ .
- Les arbres binaires asymétriques sont les moins performants dans tous les types d'arbres; ces arbres sont exécutés de la même manière que la liste chaînée.



# Arbre d'expression

Un arbre d'expression binaire est un arbre strictement binaire, qui est utilisé pour représenter une expression mathématique. Deux types courants d'expressions qu'un arbre d'expression peut représenter sont algébriques et booléens.

Il n'est cependant pas nécessaire que l'arbre d'expression soit toujours un arbre binaire.



# Arbre d'expression

## **Applications des arbres d'expression**

- Évaluation d'une expression.
- Effectuer des opérations mathématiques symboliques sur une expression.
- Génération du code de compilateur correct pour réellement calculer la valeur d'une expression au moment de l'exécution.

## **Propriétés de l'arbre d'expression:**

- L'arbre d'expression ne contient pas de parenthèses.
- Les nœuds terminaux contiennent les opérandes tels que les constantes ou les variables.
- Les nœuds non-feuilles contiennent les opérateurs.

# Arbre d'expression

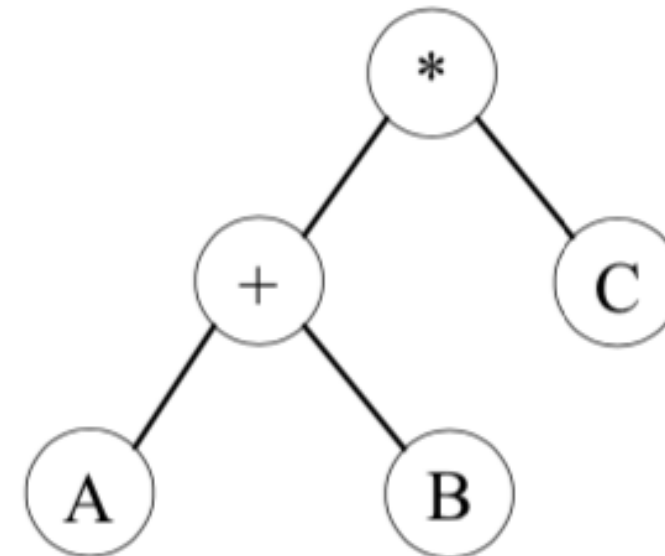
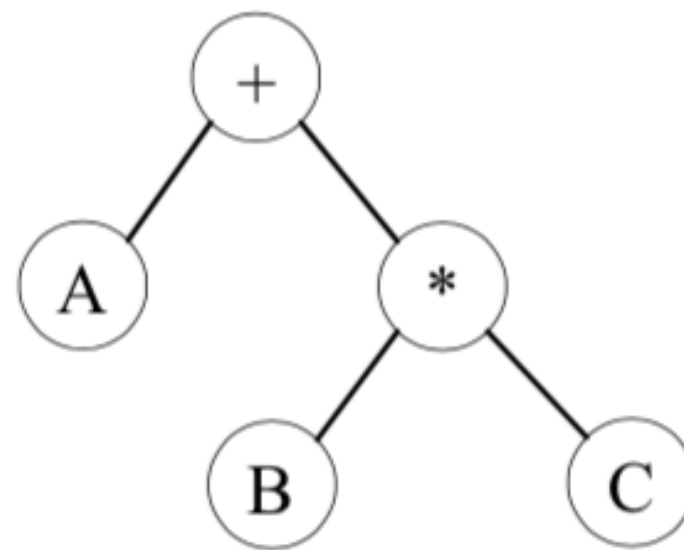
## Parcours ??

- Une traversée (ou parcours) dans l'ordre de l'arbre d'expression produit une forme **infixe** de l'expression sans parenthèse.
- Lorsque l'arbre d'expression est parcouru en pré-ordre, la forme **préfixée** de l'expression est obtenue.
- De même, lorsque l'arbre d'expression est parcouru en post-ordre, la forme **postfixée** de l'expression est obtenue. Un préfixe ou un suffixe correspond exactement à un arbre d'expressions.
- Un parcours **infixé** peut correspondre à plusieurs arbres d'expression. Par conséquent, il ne convient pas à l'évaluation d'une expression.

# Arbre d'expression

## Parcours ??

Considérez l'expression infixée  $A + B * C$ . L'expression est ambiguë car elle produit plus de plusieurs arbres d'expression avec la même forme d'**infixée**.



Expression	Prefix form	Infix form	Postfix form
$A + (B * C)$	$+ A * BC$	$A + B * C$	$ABC * +$
$(A + B) * C$	$* + ABC$	$A + B * C$	$AB + C *$



# Arbre d'expression

## Parcours ??

**Algorithme:** CREATION\_ARBRE\_EXPRESSION (P)

**Sortie:** arbre d'expression arithmétique

1. **Répéter** jusqu'à la fin de l'expression P
2. Lire l'expression postfix un symbole S à la fois
3. **Si** S est un opérande, **alors**
4.       créer un nœud pour l'opérande
5.       faire avancer le pointeur vers le nœud créé sur une pile
6. **Si** S est un opérateur binaire, **alors**
7.       créer un nœud pour l'opérateur
8.       T1 = dépiler de la pile un pointeur sur un opérande
9.       T2 = dépiler de la pile un pointeur vers un opérande
10.       faire de T2 le sous-arbre gauche et T1 le sous-arbre droit du nœud opérateur
11.       faire avancer le pointeur vers le nœud opérateur sur la pile
12. **[Fin de répéter]**
13. **T** = dépiler de la pile un pointeur vers l'arbre d'expression
14. retourner T

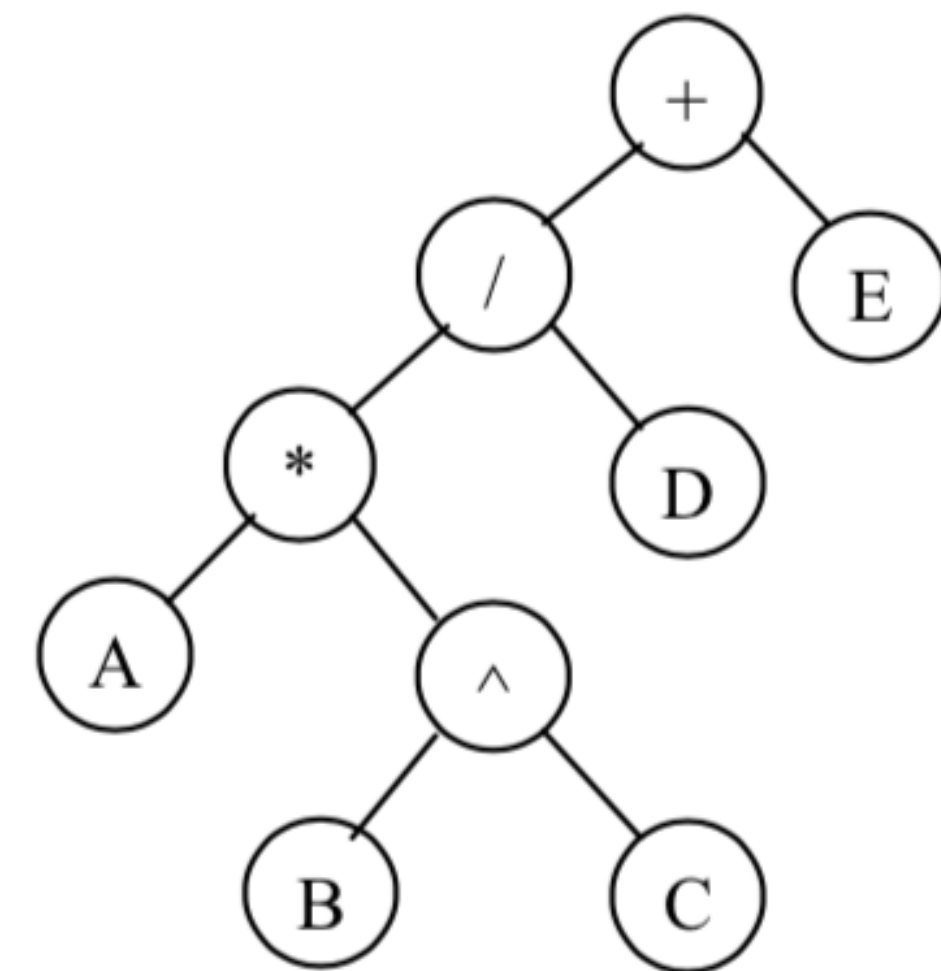
# Arbre d'expression

Parcours ??

**Dessinez un arbre d'expression à partir de l'expression suivante:**  $A * B ^ C / D + E$

Maintenant, convertir d'abord l'expression ci-dessus en son expression **postfixée**  
équivalente:  $A B C ^ * D / E +$

La traversée en pré-commande de l'arbre d'expression ci-dessus est  $+ /* A ^ B C D E$ , c'est la forme préfixe de l'expression. La traversée post-ordre de l'arbre d'expression ci-dessus est  $A B C ^ * D / E +$ , c'est la forme postfixe de l'expression.



# Arbre d'expression

## Évaluation

Une expression postfixée peut être convertie en arbre d'expression en utilisant l'algorithme ci-dessus.

**Algorithm:** EVALUATION (T)

1. **Si** T est feuille **alors**
2.     **Retourner** la valeur de l'opérande de T
3. **Else**
4.     Operator = T.Element
5.     Operand1 = EVALUATION(T.Left)
6.     Operand2 = EVALUATION(T.Right)
7.     **Retourner** (Operation (Operand1, Operator, Operand2))
8. **[Fin de Si]**
9. **Fin**

# Arbre équilibré

Il existe principalement deux types d'arbres binaires équilibrés.

- i) Arbre binaire à poids équilibré
- ii) Arbre binaire équilibré en hauteur

# Arbre équilibré

## Arbre binaire à poids équilibré

Un arbre binaire à poids équilibré est un arbre de recherche binaire si, **pour chaque nœud, le nombre de nœuds internes dans le sous-arbre gauche et le nombre de nœuds internes dans le sous-arbre droit diffèrent d'au plus un.** C



# Arbre équilibré

## Arbre binaire à poids équilibré

L'équilibre de l'arbre binaire à poids équilibré est basé sur les tailles (nombre d'éléments) des sous-arbres dans chaque nœud. La taille du nœud feuille est nulle. La taille des nœuds internes est la somme des tailles de ses deux enfants plus un.

$$\text{taille}[n] = \text{taille}[n.\text{left}] + \text{taille}[n.\text{right}] + 1$$

En fonction de la taille, on définit le poids soit comme égal à la taille, soit comme  $\text{poids}[n] = \text{taille}[n] + 1$ .

# Arbre équilibré

## Arbre binaire équilibré en hauteur

Un arbre binaire équilibré en hauteur a la hauteur minimale pour les nœuds feuilles.

Un arbre binaire est équilibré en hauteur si la hauteur de l'arbre est  $O(\sqrt{n})$ , où  $n$  est le nombre de nœuds.

Une structure d'arbre équilibrée en hauteur connue est une structure d'arbre binaire dans laquelle les sous-arbres gauche et droit de chaque nœud ne diffèrent pas en hauteur de plus d'un.

# Arbre équilibré

## Arbre binaire équilibré en hauteur

Par exemple,

- l'arbre AVL (Adel'son-Vel'skii et E. M. Landis, 1962) maintient la hauteur  $O(\log(n))$  en s'assurant que la différence entre les hauteurs des sous-arbres gauche et droit est d'au plus  $\pm 1$ .
- Les arbres rouge-noir (Guibas et Sedgewick, 1978) maintiennent la hauteur  $O(\log(n))$  en s'assurant que le nombre de nœuds noirs sur chaque trajet racine-feuille est le même et qu'il n'y a pas de nœuds rouges adjacents. Les arbres de recherche binaire équilibrés sont performantes car elles fournissent du temps  $O(\log(n))$  pour les opérations de recherche, d'insertion et de suppression.

# Arbre cousu

## Arbre cousu

Un arbre binaire fileté est un arbre binaire dans lequel il y a une boucle. Dans un arbre binaire, la plupart des entrées du champ de lien contiendront des éléments nuls. Ces entrées nulles sont remplacées par des pointeurs spéciaux, qui pointent vers des nœuds plus élevés dans l'arborescence. Ces pointeurs spéciaux sont appelés **threads** et les arbres binaires avec de tels pointeurs sont appelés arbre cousu.

# Arbre couse

Pour optimiser les pointeurs nuls, le concept du thread est utilisé.

Il existe différents types d'arbres couse:

- les arbres binaires couse en ordre,
- les arbres binaires couse en pré-ordre
- et les arbres binaires couse en post-ordre correspondant aux traversées en ordre, en pré-ordre et en post-ordre.

Chaque type d'arbres binaires couse peut avoir deux représentations:

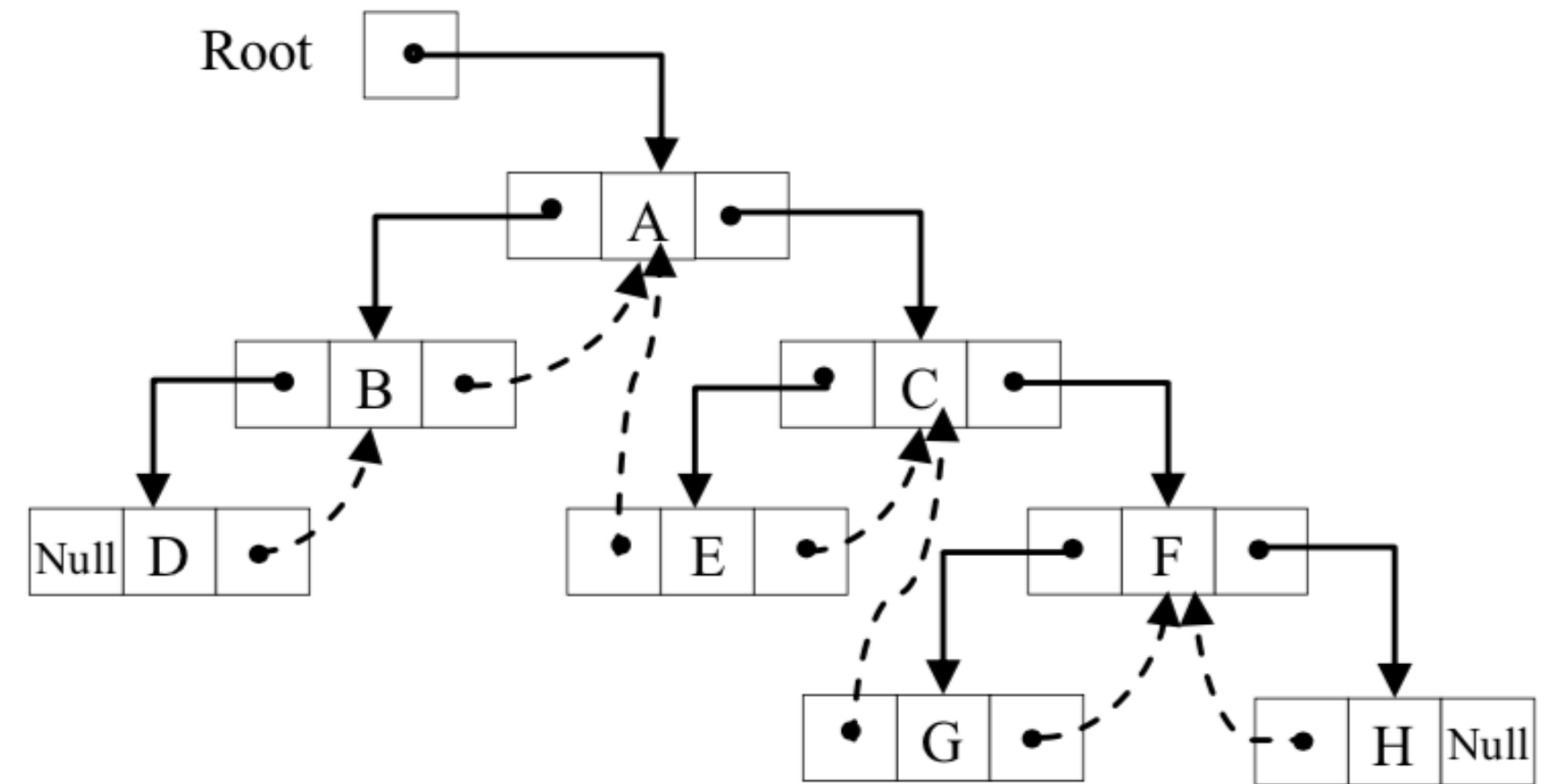
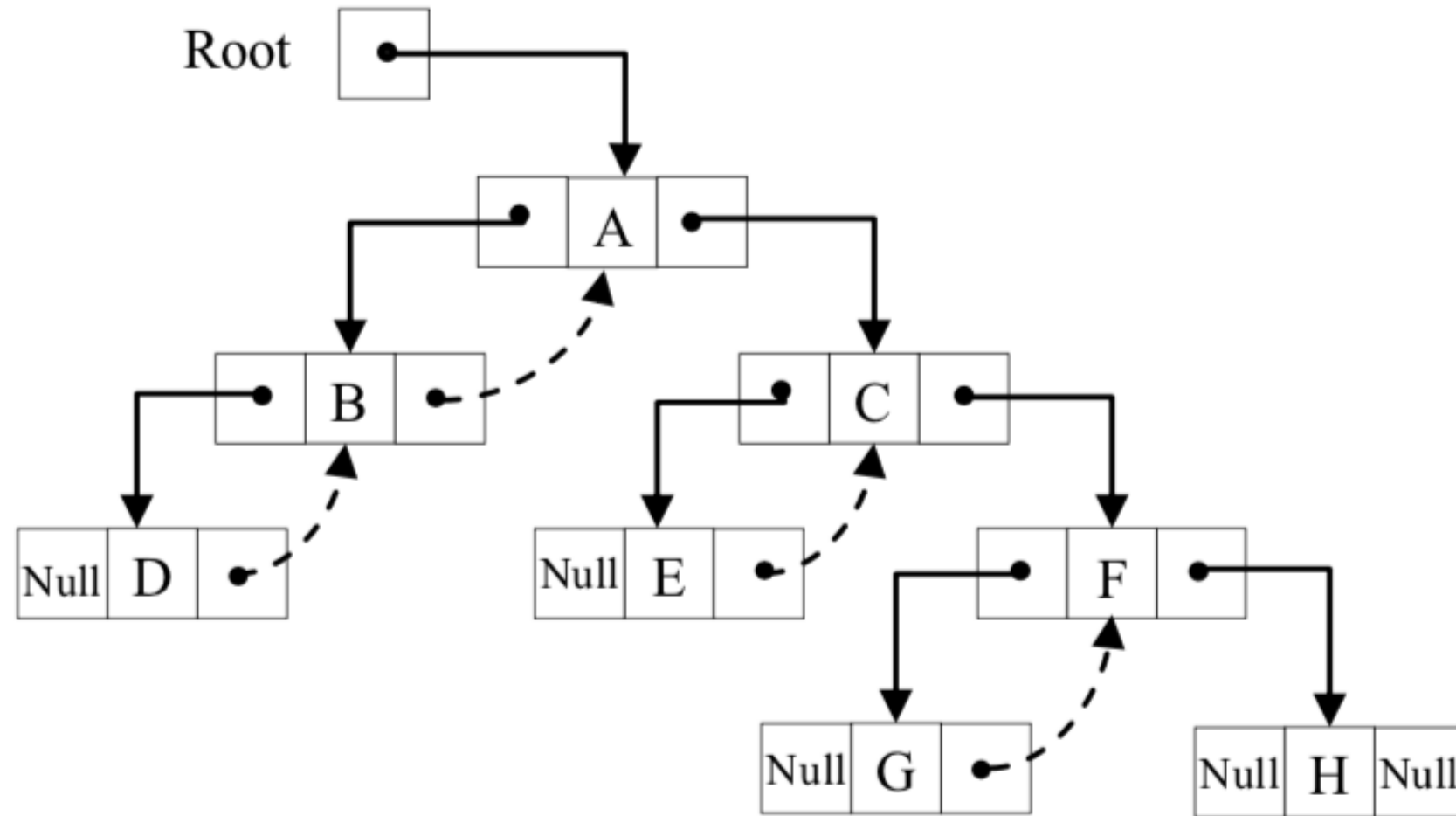
- la couture unidirectionnelle
- et la couture bidirectionnelle.

# Arbre cousu

Dans l'arbre binaire cousu unidirectionnelle, un seul pointeur nul droit utilisé comme un fil qui pointera vers le nœud suivant lors du parcours infixé (c'est-à-dire successeur inordre). Cependant, le pointeur nul droit du dernier nœud reste inutilisé.

Dans l'arbre binaire cousu bidirectionnelle, le pointeur nul gauche agit comme un thread qui pointera vers le nœud précédent dans la traversée inorder (c'est-à-dire le prédécesseur inorder) et le pointeur nul droit agit comme un thread qui pointe vers le nœud suivant dans le parcours en ordre (c'est-à-dire successeur de l'ordre). Toutefois, le pointeur nul gauche du premier nœud et le pointeur nul droit du dernier nœud restent inutilisés.

# Arbre coustu



```
struct Node {  
    struct Node *left;  
    char lthread;  
    int info;  
    struct Node *right;  
    char rthread;  
};
```

# Propriétés des arbres binaires

- Lemme 1: Un arbre binaire avec  $n$  nœuds a exactement  $n - 1$  arête (comme tout arbre normal).
- Lemme 2: Le nombre maximum de nœuds au niveau  $i$  d'un arbre binaire est  $2^{i-1}$ ,  $i \geq 1$
- Lemme 3: Le nombre maximum de nœuds dans un arbre binaire de profondeur  $k$  est  $2^k - 1$ ,  $k \geq 1$ .
- Lemme 4: Pour tout arbre binaire non vide  $T$ , si  $n_0$  est le nombre de feuilles (nœuds terminaux) et  $n_2$  le nombre de nœuds de degré 2 alors  $n_0 = n_2 + 1$ .
- Lemme 5: Si  $n$  est le nombre total de nœuds dans un arbre binaire complet de hauteur  $h$ , alors  $h = \lfloor \log_2(n) \rfloor + 1$ .



# Représentation

L'arbre est un type de données abstrait largement utilisé car il est défini en termes d'opérations sur celui-ci. Par conséquent, nous pouvons implémenter un arbre en utilisant un tableau ou une liste chaînée. L'arbre binaire peut également être représenté de deux manières différentes.

- Utilisation du tableau
- Utilisation de la liste chaînée

# Représentation

## Représentation d'arbre binaire avec un tableau

L'arbre binaire peut être représenté par un tableau. Il existe deux façons différentes de représenter un arbre binaire avec un tableau.

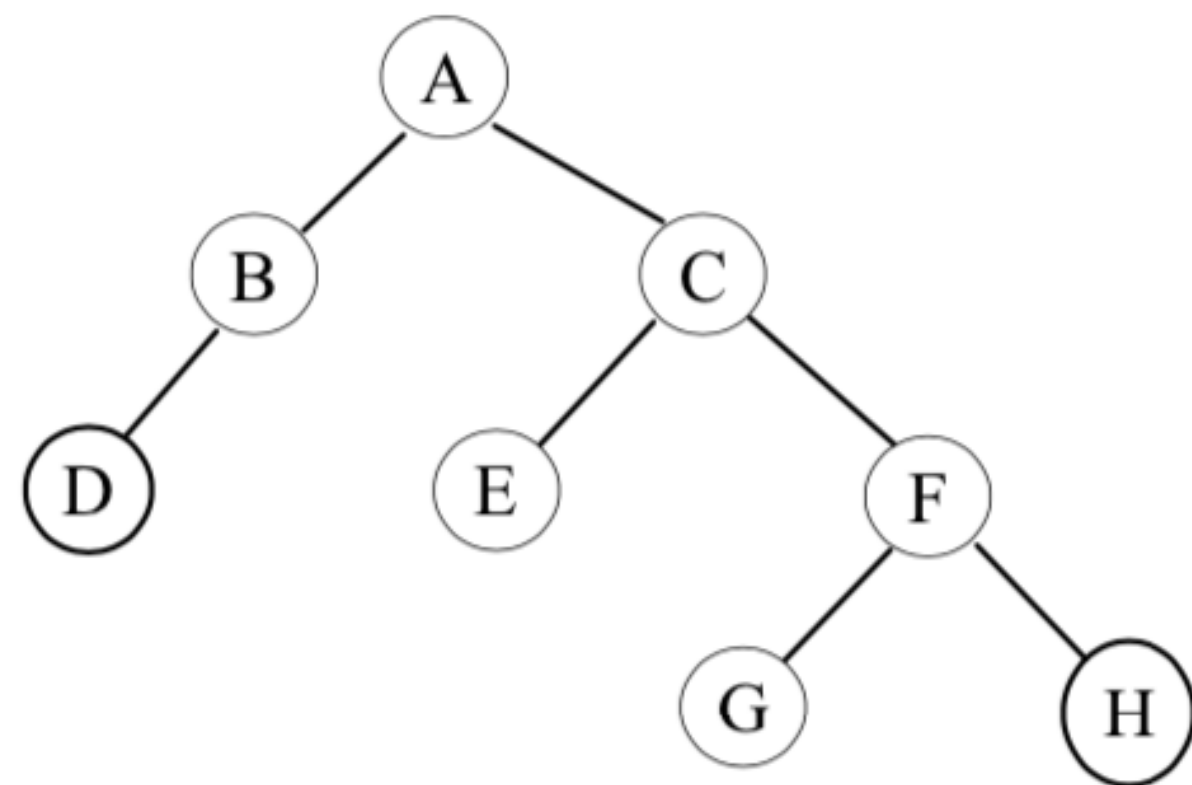
- □ Représentation liée
- □ Représentation séquentielle

# Représentation liée

Dans la représentation liée, un arbre binaire peut être stocké dans la mémoire de l'ordinateur en utilisant trois tableaux parallèles, DATA, LCHILD et RCHILD et une variable de pointeur ROOT. Maintenant, chaque nœud N de l'arbre binaire T correspondra à un emplacement k tel que,

- i) DATA [K] contient les données au nœud N
- ii) LCHILD [K] contient l'emplacement de l'enfant gauche du nœud N
- iii) RCHILD [K] contient la position de l'enfant droit du nœud N

# Représentation liée



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DATA	E		A		G	C		B		F		D			H
LCHILD	0		8		0	1		12		5		0			0
RCHILD	0		6		0	10		0		15		0			0

# Représentation séquentielle

Dans la représentation séquentielle, un arbre binaire  $T$  peut être représenté en utilisant un seul arbre linéaire unique, tel que

- i) La racine de  $T$  est stockée dans  $TREE [1]$ .
- ii) Lorsqu'un nœud  $N$  stocke dans  $TREE [K]$ , son enfant gauche sera stocké dans  $TREE [2*K]$  et l'enfant droit sera stocké dans  $TREE [2*K+1]$ .

Par conséquent, si  $K$  est l'index du nœud actuel, le nœud parent est alors stocké dans le  $FLOOR (K/2)$ .

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	D		E	F							G	H

# Représentation séquentielle

Pour un tableau à base zéro,

i) La racine de T est stockée dans TREE [0]

ii) Lorsqu'un nœud N stocke dans TREE [K], son enfant gauche sera stocké dans TREE [2 \* K + 1] et

l'enfant de droite sera stocké dans TREE [2 \* K + 2].

Par conséquent, si K est l'index d'un nœud actuel, le nœud parent est stocké dans FLOOR ((K-1) / 2).

# Représentation séquentielle

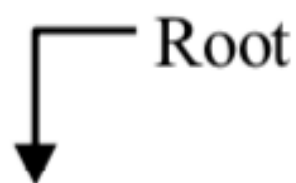


Pour un arbre binaire complet, la représentation séquentielle est parfaite car aucun espace n'est gaspillé. Cependant, c'est un gaspillage pour de nombreux autres arbres binaires. Pour un arbre binaire asymétrique, moins de la moitié du tableau peut être utilisée. Dans le pire des cas, un arbre binaire asymétrique de profondeur  $k$  nécessitera un espace mémoire de  $2^k - 1$ . De plus, l'insertion ou la suppression d'un nœud au milieu de l'arbre nécessite le déplacement de nombreux nœuds. Ces problèmes peuvent être résolus en utilisant la représentation de liste chaînée.

# Représentation séquentielle

## Représentation d'arbre binaire cousu avec un tableau

L'arbre binaire cousu T peut être stocké dans la mémoire de l'ordinateur à l'aide d'une représentation liée. Ici, le thread peut être représenté par une valeur négative de l'emplacement et le pointeur ordinaire peut être représenté par la valeur positive de l'emplacement.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DATA	E		A		G	C		B		F		D			H
LCHILD	-3		8		-6	1		12		5		0			-10
RCHILD	-6		6		-10	10		-3		15		-8			0



# Représentation avec les listes chaînées

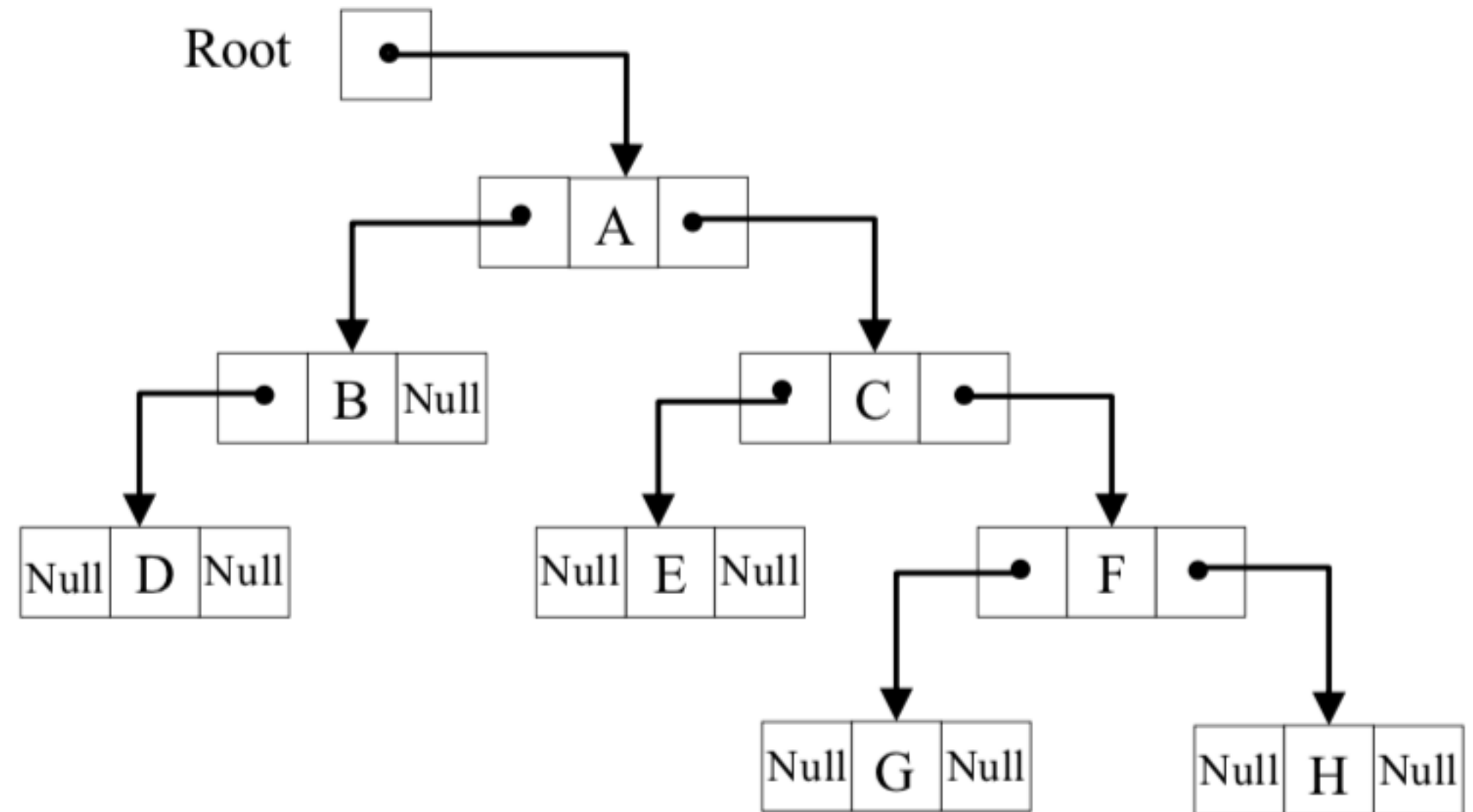
L'arbre binaire peut également être représenté avec une liste chaînée.

Dans cette représentation, chaque nœud d'un arbre binaire se compose de trois champs tels que

- Le premier champ contient le pointeur qui pointe vers l'enfant gauche.
- Le deuxième champ contient les données.
- Le troisième champ contient le pointeur qui pointe vers le enfant droite.

# Représentation avec les listes chaînées

```
struct Node {  
    struct Node *left;  
    int info;  
    struct Node *right;  
};  
typedef struct Node BTreeNode;
```



# Traversée d'arbres binaires

La traversée de l'arbre consiste à visiter chaque nœud de l'arbre exactement une fois. Il existe principalement deux traversées:

- i) Recherche en profondeur (DFS)
- ii) Recherche étendue (BFS)

# Recherche en profondeur

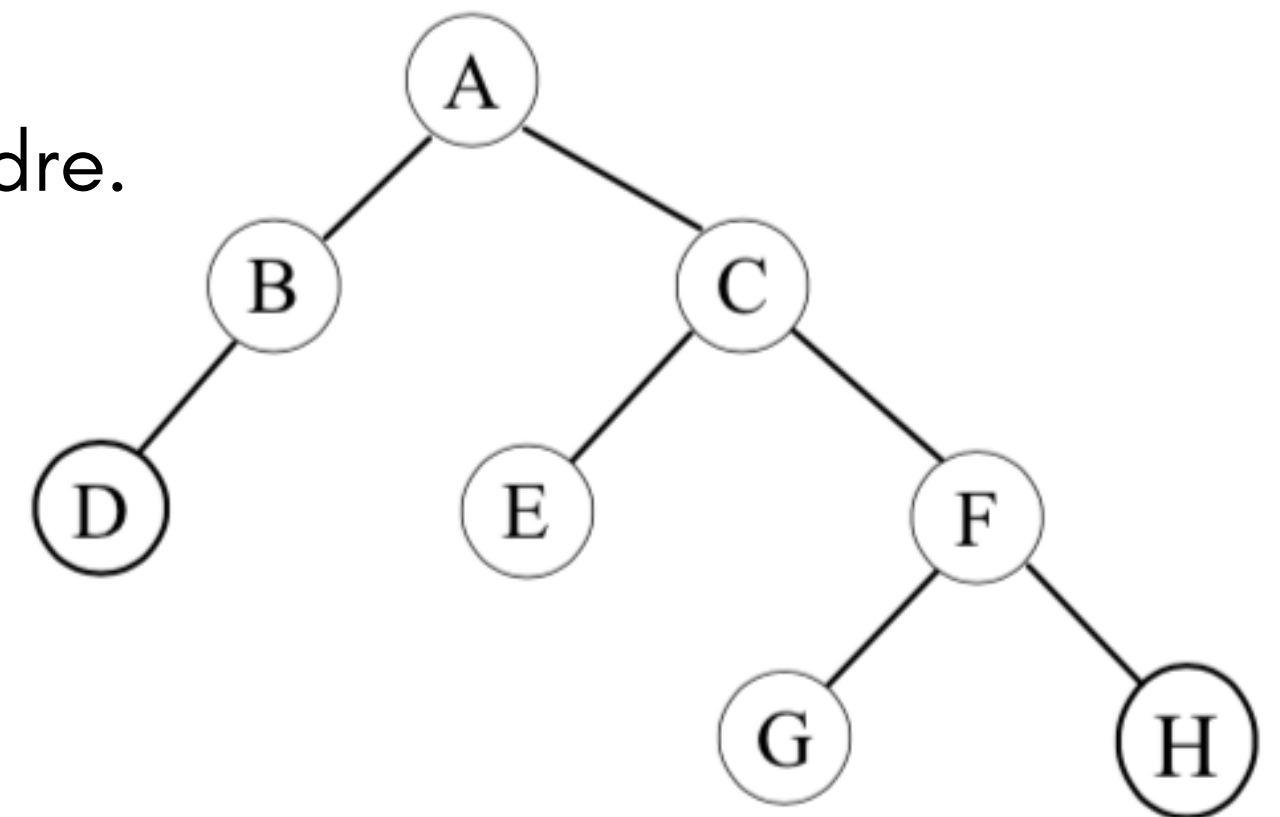
- La recherche en profondeur peut être implémentée facilement à l'aide d'une pile, y compris la récursivité.
- En partant de la racine de l'arborescence binaire, trois étapes principales peuvent être effectuées. Ces étapes se déplacent à gauche (L), visitent le nœud (D) et se déplacent à droite (R), puis il y a six combinaisons possibles de traversée: LDR, LRD, DLR, DRL, RDL et RLD.
- Maintenant, si nous traversons gauche avant droite, il ne reste que trois traversées: LDR, LRD et DLR; LDR est connu comme inorder, LRD est postorder et DLR est preorder, ces trois ne sont que des traversées d'arbres binaires standard.

# Recherche en profondeur

## Traversée dans l'ordre (infixe)

La traversée inordonnée d'un arbre binaire non vide est définie comme suit, à partir du nœud racine:

- i) Traverser le sous-arbre gauche de la racine dans l'ordre.
- ii) Visiter le nœud racine.
- iii) Traverser le sous-arbre droit de la racine dans l'ordre.

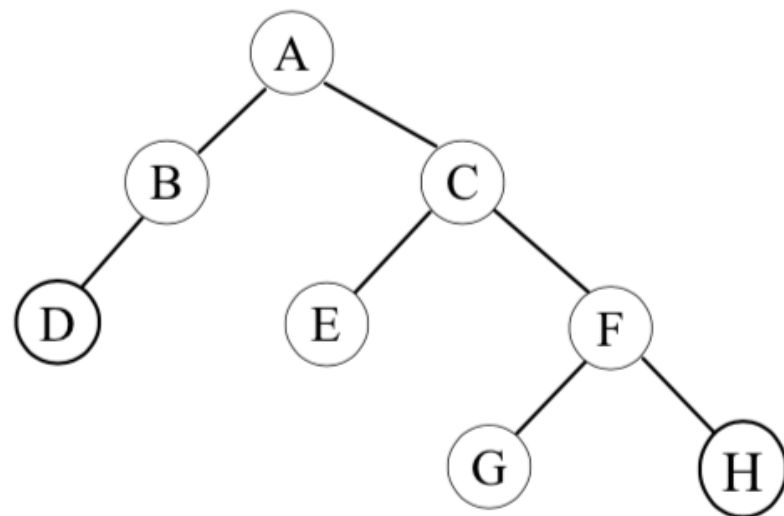


# Recherche en profondeur

Algorithm: **INORDER** (ROOT)

1. **IF** ROOT  $\neq$  NULL **THEN**
2.     INORDER(ROOT  $\rightarrow$  LCHILD)
3.     PRINT: ROOT  $\rightarrow$  INFO
4.     INORDER(ROOT  $\rightarrow$  RCHILD)
5. **RETURN**

Récuratif



L'ordre de l'arbre binaire ci-dessus  
est: DBAECGFH

Algorithm: **INORDER** (ROOT)

1. P = ROOT
2. Initialize Stack
3. **Repeat while** Stack is not empty or P  $\neq$  null
4.     **Repeat while** P  $\neq$  null
5.         a) PUSH(Stack, P)
6.         b) P = P  $\rightarrow$  LCHILD
7.     **[Fin de Repeat]**
8.     **If** Stack is not empty **then**
9.         a) P = POP(Stack)
10.        b) Print: P  $\rightarrow$  Info
11.        c) P = P  $\rightarrow$  RCHILD
12. **[Fin de Repeat]**
13. **RETURN**

Itératif

# Recherche en profondeur

## **Traversée en pré-ordre**

La traversée en pré-ordre d'un arbre binaire non vide est définie comme suit, à partir du nœud racine:

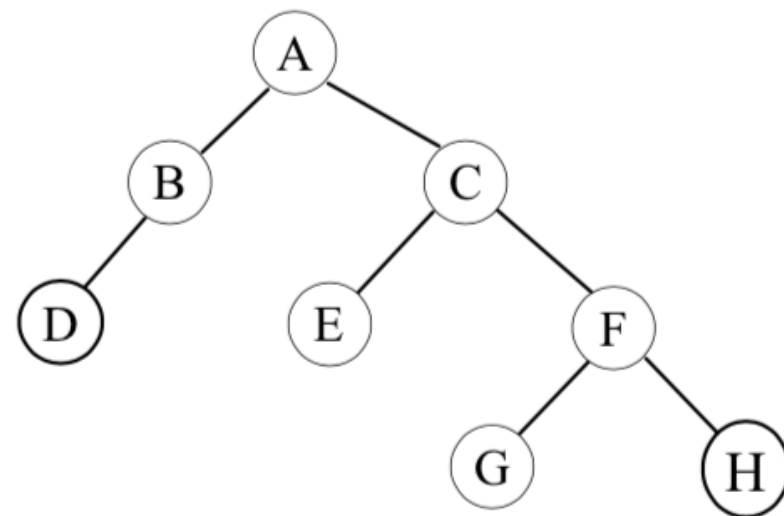
- i) Visiter le nœud racine.
- ii) Traverser le sous-arbre gauche de la racine en précommande.
- iii) Traverser le sous-arbre droit de la racine en précommande.

# Recherche en profondeur

Algorithm: **PREORDER**(ROOT)

1. **IF** ROOT  $\neq$  NULL **THEN**
2.   a) PRINT: ROOT  $\rightarrow$  INFO
3.   b) PREORDER(ROOT  $\rightarrow$  LCHILD)
4.   c) PREORDER(ROOT  $\rightarrow$  RCHILD)
5. **RETURN**

Récuratif



Le pré-ordre de l'arbre binaire ci-dessus est: ABDCEFGH

Algorithm: **PREORDER**(ROOT)

1. Initialize Stack
2. PUSH(Stack, ROOT)
3. **Repeat** while Stack is not empty
4.   a) P = Pop(Stack)
5.   b) **If** P  $\neq$  null **then**
6.       i) Print: P  $\rightarrow$  Info
7.       ii) Push(Stack, P  $\rightarrow$  RCHILD)
8.       iii) Push(Stack, P  $\rightarrow$  LCHILD)
9. **[End of loop]**
10. **Return**

Itératif



# Recherche en profondeur

## **Traversée en post-ordre**

La traversée post-ordre d'un arbre binaire non vide est définie comme suit:  
à partir du nœud racine:

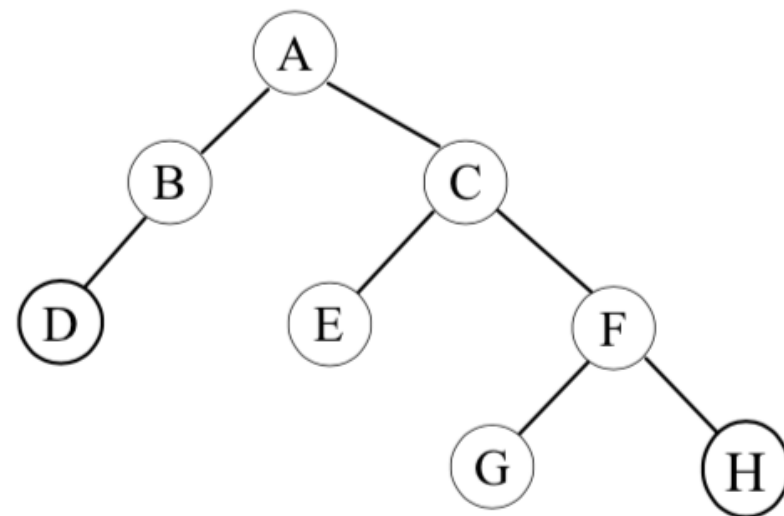
- i) Traverser le sous-arbre gauche de la racine en post-ordre.
- ii) Traverser le sous-arbre droit de la racine en post-ordre.
- iii) Visitez le nœud racine.

# Recherche en profondeur

Algorithm: **POSTORDER** (ROOT)

1. **IF** ROOT  $\neq$  NULL **THEN**
2.   a) POSTORDER(ROOT  $\rightarrow$  LCHILD)
3.   b) POSTORDER(ROOT  $\rightarrow$  RCHILD)
4.   c) PRINT: ROOT  $\rightarrow$  INFO
5. **RETURN**

Récurif



Le post-ordre de l'arbre binaire ci-dessus est: DBEGHFCA

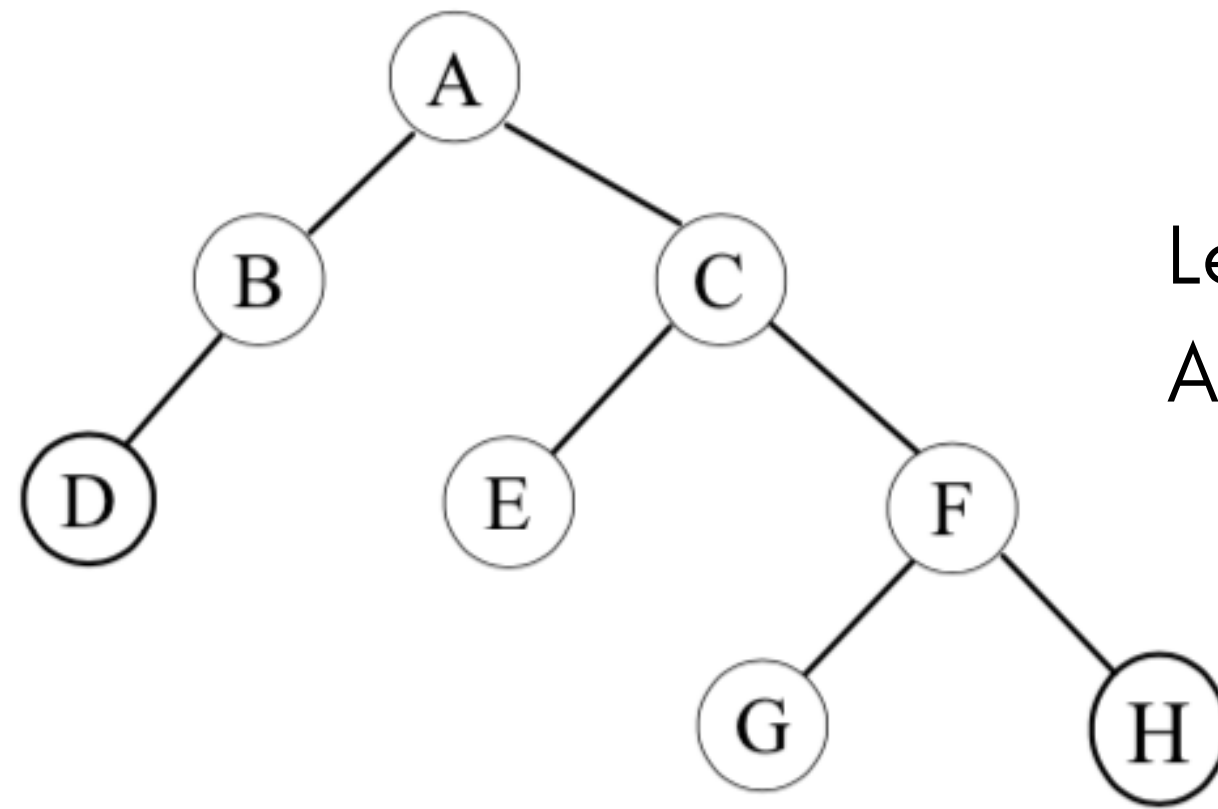
Algorithm: **POSTORDER** (ROOT)

1. Initialize Stack
2. P = ROOT
3. **Repeat** while Stack is not empty
4.   **Repeat** while P  $\neq$  null
5.     Push(Stack, P)
6.     **If** P  $\rightarrow$  RCHILD  $\neq$  null
7.       Push(Stack, null)
8.     P = P  $\rightarrow$  LCHILD
9.   **[End of Repeat]**
10.   Q = Pop(Stack)
11.   **If** Q  $\neq$  null
12.     PRINT: Q  $\rightarrow$  INFO
13.   **Else**
14.     Q = Pop(Stack)
15.     P = Q  $\rightarrow$  RCHILD
16.     Push(Stack, Q)
17.   **[End of If]**
18. **[End of loop]**
19. **Return**

Itératif

# Recherche en largeur

Les arbres binaires peuvent également être parcourus dans l'ordre des niveaux, où chaque nœud est visité à un niveau avant le niveau suivant. Cette recherche est connue sous le nom de recherche en largeur (BFS). La première traversée de recherche en largeur peut être implémentée facilement à l'aide d'une file d'attente.



Le parcours en largeur de cet arbre donne:  
ABCDEFGH

# Reconstruction d'un arbre binaire

Un arbre original ne peut pas être reconstruit par son parcours en ordre ou en pré-ordre ou post-ordre seul.

Cependant, un arbre binaire unique peut être reconstruit soit par des parcours en ordre et en pré-ordre, soit par des parcours en ordre et post-ordre.

Cependant, les traversées pré-ordre et post-ordre donnent une certaine ambiguïté dans la structure arborescente.

- Le premier nœud visité dans une traversée en pré-ordre d'un arbre binaire est la racine, puis les sous-arbres de gauche et les sous-arbres de droite sont parcourus.
- Dans la traversée post-ordre d'un arbre binaire, le sous-arbre gauche et le sous-arbre droit sont parcourus, puis le dernier nœud visité est la racine.
- Dans la traversée en ordre d'un arbre binaire, le sous-arbre gauche est d'abord traversé, puis le nœud racine est visité, enfin le sous-arbre droit est traversé.

# Reconstruction d'un arbre binaire

## **Exemple:**

La séquence de parcours en ordre et pré-ordre des nœuds dans un arbre binaire est donnée ci-dessous:

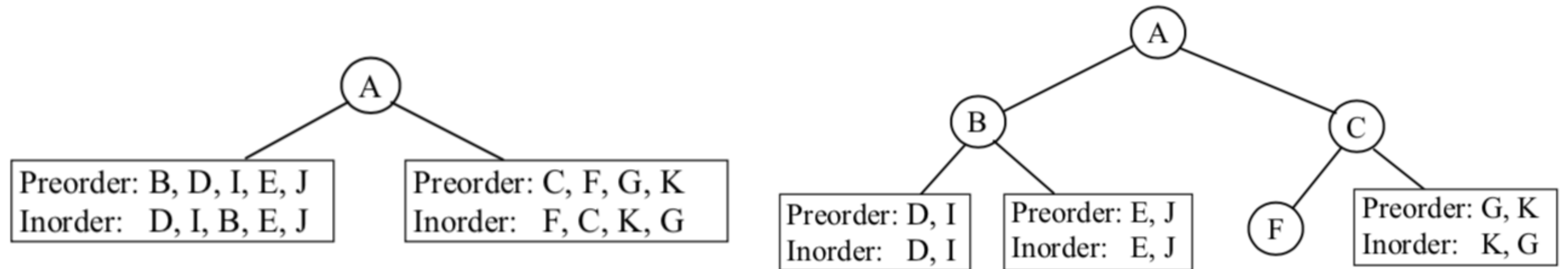
- Pré-ordre: A, B, D, I, E, J, C, F, G, K
- En ordre: D, I, B, E, J, A, F, C, K, G

# Reconstruction d'un arbre binaire

Les étapes suivantes sont utilisées pour reconstruire l'arbre binaire:

- Dans la traversée en pré-ordre d'un arbre binaire, le nœud racine est visité en premier, puis le sous-arbre gauche est parcouru, et enfin, le sous-arbre droit est parcouru.
- Dans la traversée en ordre d'un arbre binaire, le sous-arbre gauche est parcouru en premier, puis le nœud racine est visité, enfin, le sous-arbre droit est parcouru. Par conséquent, dans la traversée de pré-ordre, le premier nœud A doit être Racine.
- Maintenant, en recherchant le nœud A dans la traversée en ordre, nous pouvons découvrir que tous les éléments du côté gauche de A sont une traversée en ordre du sous-arbre gauche et les éléments de droite sont une traversée en ordre du sous-arbre droit. Par conséquent, une traversée en ordre et en pré-ordre du sous-arbre gauche et du sous-arbre droit peut être obtenue.

# Reconstruction d'un arbre binaire



De même, le nœud B est la racine du sous-arbre gauche et le nœud C est la racine du sous-arbre droit. La traversée en ordre et en pré-ordre du sous-arbre gauche et du sous-arbre droit de B et C peut être trouvée.

Le nœud F est l'enfant évidemment laissé du nœud C.

# Reconstruction d'un arbre binaire

De même, les nœuds D et E sont les racines du sous-arbre gauche et le sous-arbre droit du nœud B. le nœud G est le sous-arbre droit du nœud C. En outre, le nœud I est l'enfant droit du nœud D, le nœud J est le droit enfant du nœud E et le nœud K est l'enfant gauche du nœud K.

