Parcourir un arbre binaire

Préfixe, infixe, postfixe et parcours en largeur par FIFO

4R-IN1 – ESIEE Paris

Denière modification 19 novembre 2026

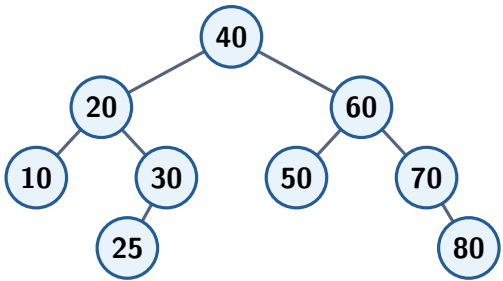
Objectifs de la séance

- Comprendre ce que signifie **visiter tous les nœuds** d'un arbre.
- Différencier les trois parcours **en profondeur** : préfixe, infixe, postfixe.
- Décrire le parcours **en largeur** à l'aide d'une **FIFO de pointeurs de nœuds**.
- Écrire ces parcours en C en réutilisant la structure de l'ABR et la FIFO du TP précédent.

Point important

Les quatre parcours s'appliquent à tout **arbre binaire**. Le parcours infixe donne une suite triée **uniquement si l'arbre est un ABR**.

Un même arbre pour tous les exemples



Arbre de référence

9 nœuds, 4 niveaux.

C'est un **ABR** : toutes les clés à gauche d'un nœud lui sont inférieures, et celles à droite lui sont supérieures.

Question : dans quel ordre visiter ces 9 nœuds ?

Parcourir un arbre

Effectuer une opération `visiter(nœud)` **une seule fois** sur chacun des nœuds, selon un ordre déterminé.

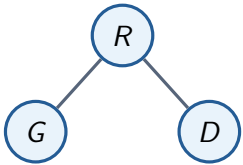
En profondeur (DFS)

- suivre d'abord une branche ;
- exploiter la **récursion** ou une pile ;
- trois ordres : préfixe, infixe, postfixe.

En largeur (BFS)

- visiter les nœuds niveau par niveau ;
- utiliser une **file FIFO** ;
- retirer en tête, ajouter en queue.

En profondeur : position de la visite



Parcours	**Schéma de visite**	**Moment de visiter** R
Préfixe	$R - G - D$	avant les deux sous-arbres
Infixe	$G - R - D$	entre les deux sous-arbres
Postfixe	$G - D - R$	après les deux sous-arbres

Règle commune

Les sous-arbres sont traités récursivement : **gauche avant droite** dans les trois cas.

Parcours préfixe : racine, gauche, droite

Principe

On visite la racine **avant** d'explorer récursivement ses sous-arbres.

Ordre

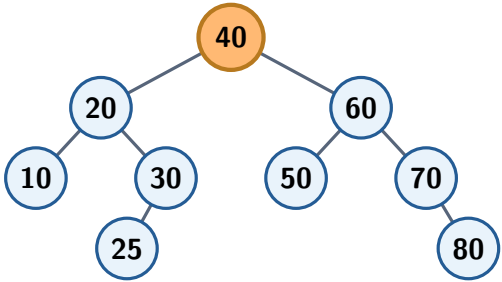
$R \to G \to D$

Implémentation en C

```
void prefixe(BST t) {
    if (t == NULL) return;
    visiter(t->data);
    prefixe(t->ls);
    prefixe(t->rs);
}
```

À observer : le nœud parent est traité avant ses descendants.

Préfixe : animation pas à pas



Étape courante

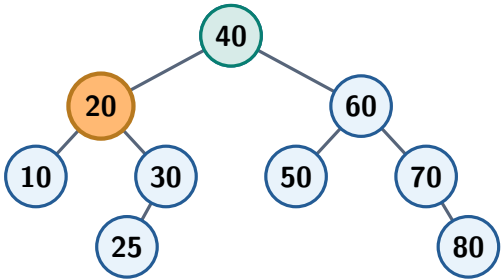
1 / 9 : visiter **40**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : 40

On visite une racine avant de parcourir son fils gauche, puis son fils droit.

Préfixe : animation pas à pas



Étape courante

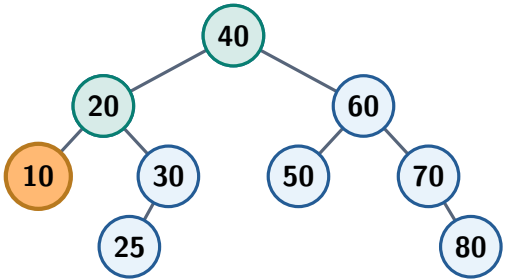
2 / 9 : visiter **20**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : 40 – 20

On visite une racine avant de parcourir son fils gauche, puis son fils droit.

Préfixe : animation pas à pas



Étape courante

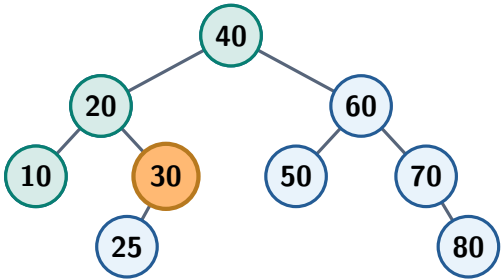
3 / 9 : visiter **10**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `40 - 20 - 10`

On visite une racine avant de parcourir son fils gauche, puis son fils droit.

Préfixe : animation pas à pas



Sortie construite : `40 - 20 - 10 - 30`

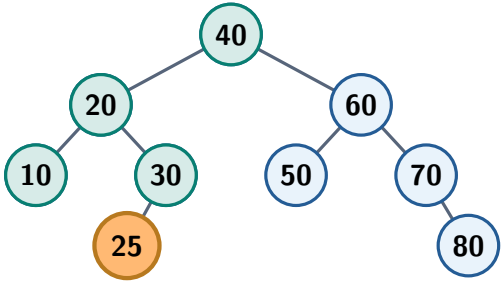
Étape courante

4 / 9 : visiter **30**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

On visite une racine avant de parcourir son fils gauche, puis son fils droit.

Préfixe : animation pas à pas



Sortie construite : `40 - 20 - 10 - 30 - 25`

parcourir son fils gauche, puis son fils droit.

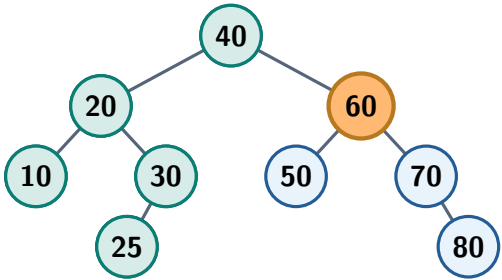
Étape courante

5 / 9 : visiter **25**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

On visite une racine avant de

Préfixe : animation pas à pas



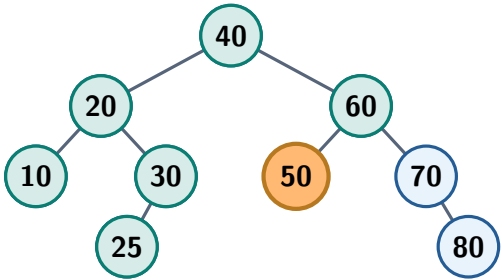
Étape courante

6 / 9 : visiter **60**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `40 - 20 - 10 - 30 - 25 - 60` On visite une racine avant de parcourir son fils gauche, puis son fils droit.

Préfixe : animation pas à pas



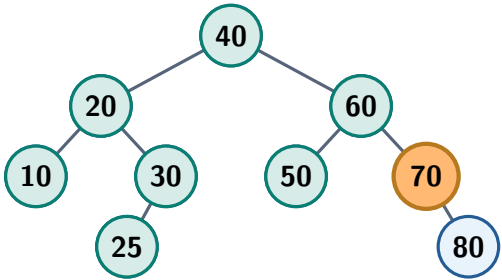
Étape courante

7 / 9 : visiter **50**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : 40 – 20 – 10 – 30 – 25 – 60 – 50 On visite une racine avant de parcourir son fils gauche, puis son fils droit.

Préfixe : animation pas à pas



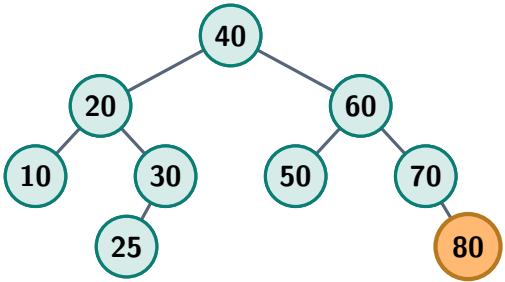
Étape courante

8 / 9 : visiter **70**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `40 - 20 - 10 - 30 - 25 - 60 - 50 - 70` On visite une racine avant de parcourir son fils gauche, puis son fils droit.

Préfixe : animation pas à pas



Étape courante

9 / 9 : visiter **80**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : 40 – 20 – 10 – 30 – 25 – 60 – 50 – 70 – 80 On visite une racine avant de parcourir son fils gauche, puis son fils droit.

Parcours infixe : gauche, racine, droite

Principe

On visite la racine **entre** les explorations des sous-arbres gauche et droit.

Ordre

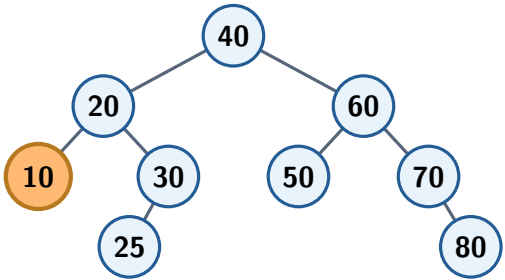
$G \to R \to D$

Implémentation en C

```c
void infixe(BST t) {
    if (t == NULL) return;
    infixe(t->ls);
    visiter(t->data);
    infixe(t->rs);
}
```

Propriété ABR : l'infixe affiche les clés dans l'ordre croissant.

Infixe : animation pas à pas



Étape courante

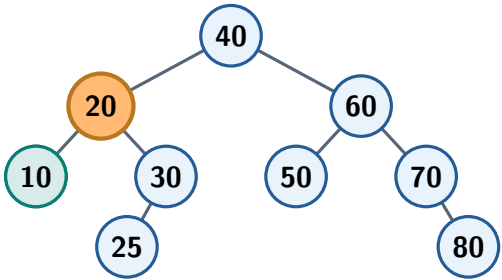
1 / 9 : visiter **10**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : 10

Dans cet ABR, lórdre produit est croissant : la propriété ne vaut pas pour un arbre binaire quelconque.

Infixe : animation pas à pas



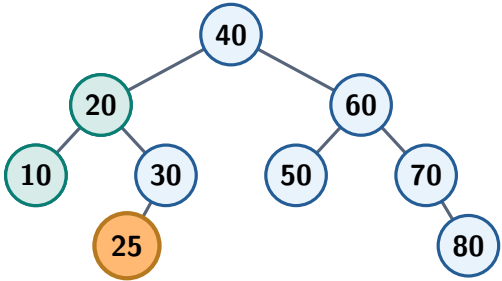
Étape courante

2 / 9 : visiter **20**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `10 - 20` Dans cet ABR, lórdre produit est croissant : la propriété ne vaut pas pour un arbre binaire quelconque.

Infixe : animation pas à pas



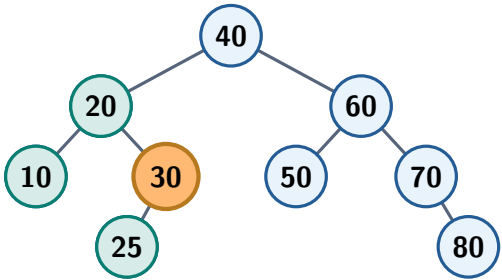
Étape courante

3 / 9 : visiter **25**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `10 - 20 - 25` Dans cet ABR, lórdre produit est croissant : la propriété ne vaut pas pour un arbre binaire quelconque.

Infixe : animation pas à pas



Étape courante

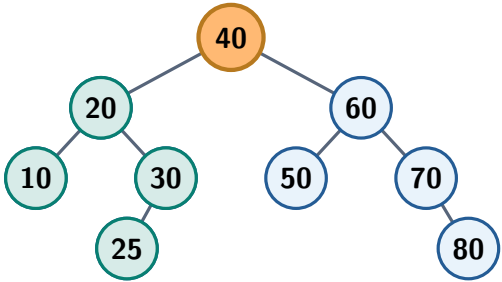
4 / 9 : visiter **30**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `10 – 20 – 25 – 30`

Dans cet ABR, lórdre produit est croissant : la propriété ne vaut pas pour un arbre binaire quelconque.

Infixe : animation pas à pas



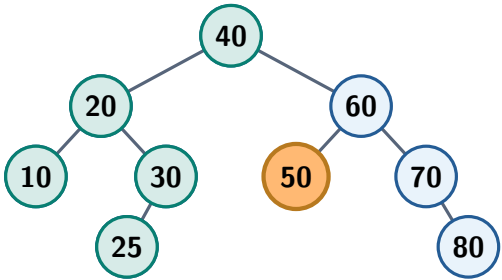
Étape courante

5 / 9 : visiter **40**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `10 – 20 – 25 – 30 – 40` Dans cet ABR, lórdre produit est croissant : la propriété ne vaut pas pour un arbre binaire quelconque.

Infixe : animation pas à pas



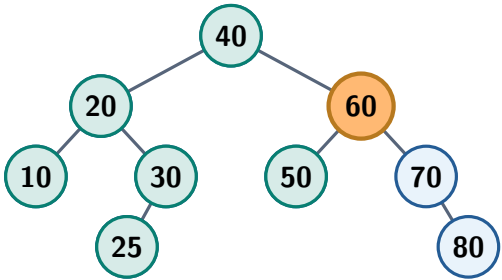
Étape courante

6 / 9 : visiter **50**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `10 - 20 - 25 - 30 - 40 - 50` Dans cet ABR, lórdre produit est croissant : la propriété ne vaut pas pour un arbre binaire quelconque.

Infixe : animation pas à pas



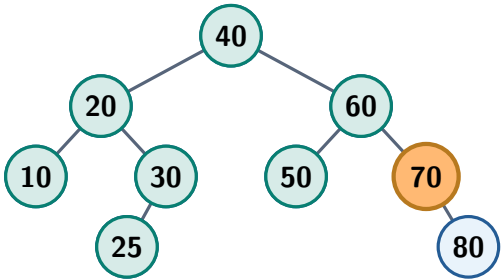
Étape courante

7 / 9 : visiter **60**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : 10 – 20 – 25 – 30 – 40 – 50 – 60 Dans cet ABR, lórdre produit est croissant : la propriété ne vaut pas pour un arbre binaire quelconque.

Infixe : animation pas à pas



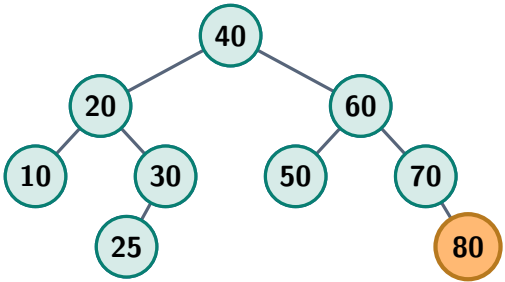
Étape courante

8 / 9 : visiter **70**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : 10 – 20 – 25 – 30 – 40 – 50 – 60 – 70 Dans cet ABR, lórdre produit est croissant : la propriété ne vaut pas pour un arbre binaire quelconque.

Infixe : animation pas à pas



Étape courante

9 / 9 : visiter **80**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `10 – 20 – 25 – 30 – 40 – 50 – 60 – 70 – 80` Dans cet ABR, lórdre produit est croissant : la propriété ne vaut pas pour un arbre binaire quelconque.

Parcours postfixe : gauche, droite, racine

Principe

On visite la racine **après** avoir parcouru ses deux sous-arbres.

Ordre

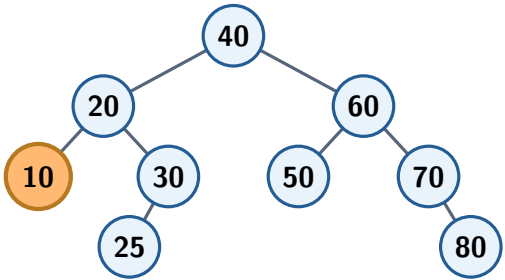
$G \to D \to R$

Implémentation en C

```
void postfixe(BST t) {
    if (t == NULL) return;
    postfixe(t->ls);
    postfixe(t->rs);
    visiter(t->data);
}
```

Usage : libérer les nœuds d'un arbre : les enfants sont traités avant le parent.

Postfixe : animation pas à pas



Étape courante

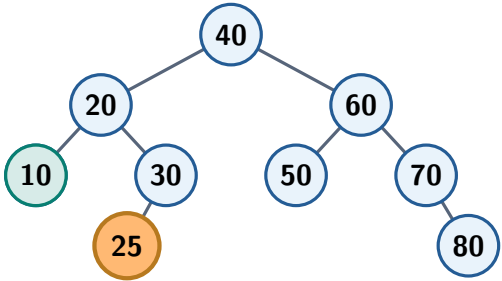
1 / 9 : visiter **10**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : 10

On traite les descendants avant leur parent : pratique pour libérer un sous-arbre sans perdre ses fils.

Postfixe : animation pas à pas



Étape courante

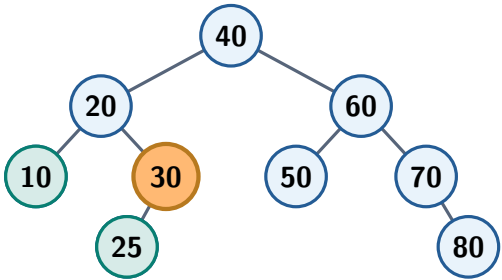
2 / 9 : visiter **25**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `10 - 25`

On traite les descendants avant leur parent : pratique pour libérer un sous-arbre sans perdre ses fils.

Postfixe : animation pas à pas



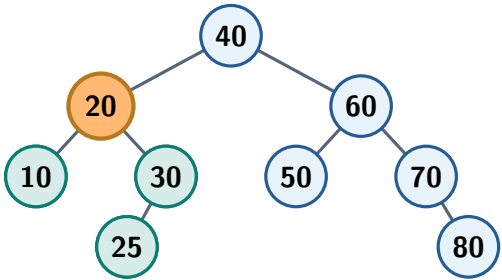
Étape courante

3 / 9 : visiter **30**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `10 - 25 - 30` On traite les descendants avant leur parent : pratique pour libérer un sous-arbre sans perdre ses fils.

Postfixe : animation pas à pas



Étape courante

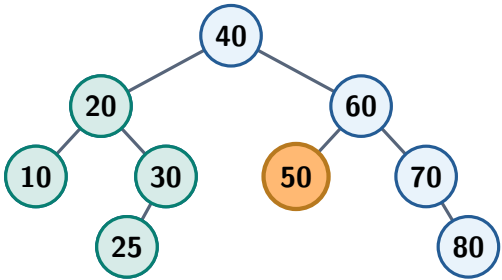
4 / 9 : visiter **20**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : 10 – 25 – 30 – 20

On traite les descendants avant leur parent : pratique pour libérer un sous-arbre sans perdre ses fils.

Postfixe : animation pas à pas



Étape courante

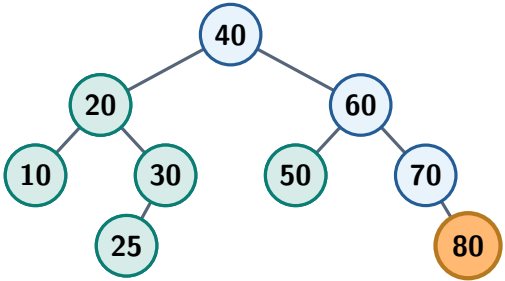
5 / 9 : visiter **50**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `10 – 25 – 30 – 20 – 50`

On traite les descendants avant leur parent : pratique pour libérer un sous-arbre sans perdre ses fils.

Postfixe : animation pas à pas



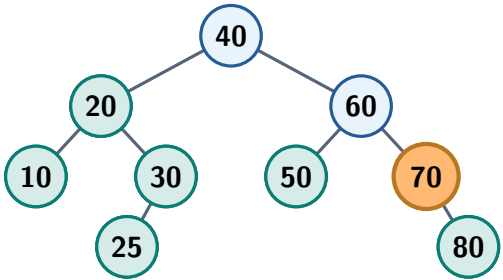
Étape courante

6 / 9 : visiter **80**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `10 – 25 – 30 – 20 – 50 – 80` On traite les descendants avant leur parent : pratique pour libérer un sous-arbre sans perdre ses fils.

Postfixe : animation pas à pas



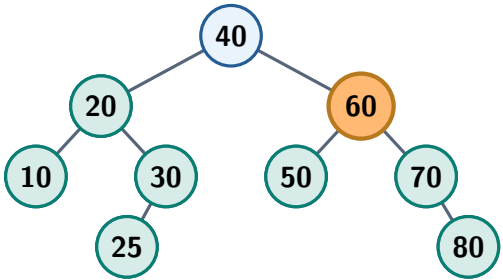
Étape courante

7 / 9 : visiter **70**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : 10 – 25 – 30 – 20 – 50 – 80 – 70 On traite les descendants avant leur parent : pratique pour libérer un sous-arbre sans perdre ses fils.

Postfixe : animation pas à pas



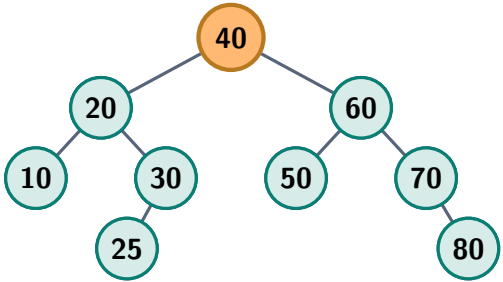
Étape courante

8 / 9 : visiter **60**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : 10 – 25 – 30 – 20 – 50 – 80 – 70 – 60 On traite les descendants avant leur parent : pratique pour libérer un sous-arbre sans perdre ses fils.

Postfixe : animation pas à pas



Étape courante

9 / 9 : visiter **40**.

- **Orange** : nœud visité à cette étape.
- **Vert** : nœuds déjà visités.

Sortie construite : `10 - 25 - 30 - 20 - 50 - 80 - 70 - 60 - 40` On traite les descendants avant leur parent : pratique pour libérer un sous-arbre sans perdre ses fils.

Comparer les parcours en profondeur

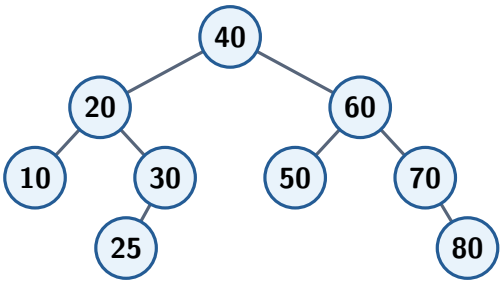
Préfixe	`40, 20, 10, 30, 25, 60, 50, 70, 80`
Infixe	`10, 20, 25, 30, 40, 50, 60, 70, 80`
Postfixe	`10, 25, 30, 20, 50, 80, 70, 60, 40`

Complexités

Chaque nœud est visité une fois : $O(n)$ **en temps**. La pile des appels récursifs occupe $O(h)$ en mémoire auxiliaire, h étant la hauteur de l'arbre.

Question : quel parcours utiliser pour afficher les clés triées d'un ABR ? Pour libérer ses nœuds ?

Parcours en largeur : niveau par niveau



- Visiter d'abord la racine.
- Puis tous les nœuds du niveau suivant, **de gauche à droite**.
- Répéter jusqu'aux feuilles.

Ordre obtenu

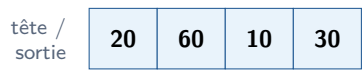
`40, 20, 60, 10, 30, 50, 70, 25, 80`

Pourquoi une FIFO ?

Invariant de la file

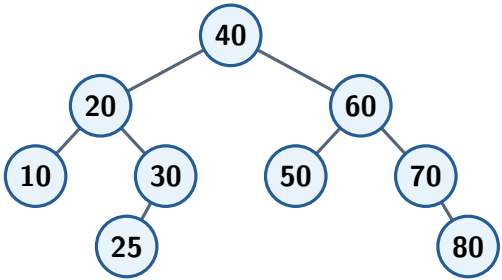
La FIFO contient les **nœuds découverts mais pas encore visités**, dans l'ordre où ils devront être traités.

1. Au départ : on **enfile la racine**.
2. On **défile le premier nœud**, puis on le visite.
3. On enfile son **fils gauche**, puis son **fils droit**, s'ils existent.
4. On recommence tant que la FIFO n'est pas vide.



Retrait à gauche (tête), insertion à droite (fin) : **FIFO = First In, First Out**.

Parcours en largeur : animation de la FIFO



Opération effectuée :

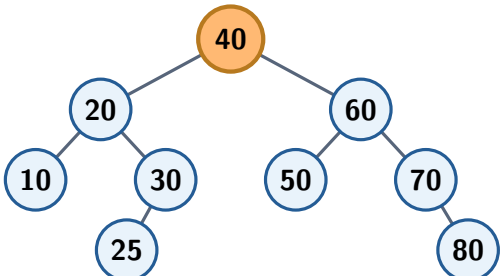
Initialisation : enfiler la racine 40.

État de la FIFO après l'étape :

tête / sortie | **40**

Nœuds déjà visités : *aucun*

Parcours en largeur : animation de la FIFO



Opération effectuée :

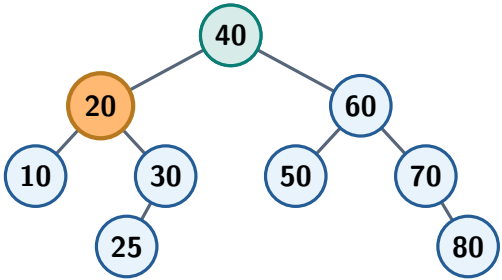
Défiler 40 ; visiter 40 ; enfiler 20 puis 60.

État de la FIFO après l'étape :

tête / sortie		
	20	60

Nœuds déjà visités :40

Parcours en largeur : animation de la FIFO



Opération effectuée :

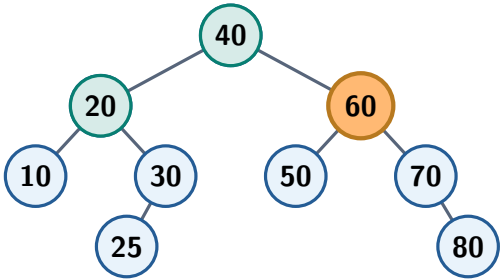
Défiler 20 ; visiter 20 ; enfiler 10 puis 30.

État de la FIFO après l'étape :

tête / sortie			
	60	**10**	**30**

Nœuds déjà visités :40 - 20

Parcours en largeur : animation de la FIFO



Opération effectuée :

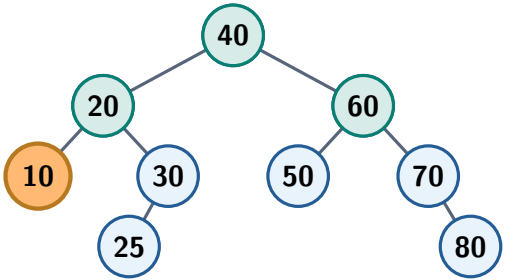
Défiler 60 ; visiter 60 ; enfiler 50 puis 70.

État de la FIFO après l'étape :

tête / sortie				
	10	30	50	70

Nœuds déjà visités :40 - 20 - 60

Parcours en largeur : animation de la FIFO



État de la FIFO après l'étape :

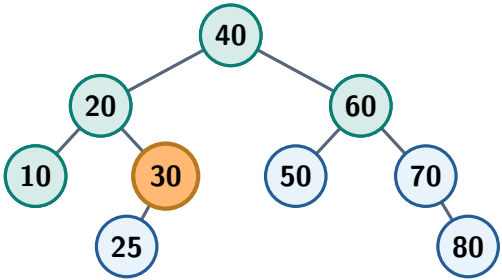
tête / sortie			
	30	50	70

Nœuds déjà visités :40 - 20 - 60 - 10

Opération effectuée :

Défiler 10 ; visiter 10 ; aucun fils à enfiler.

Parcours en largeur : animation de la FIFO



Opération effectuée :

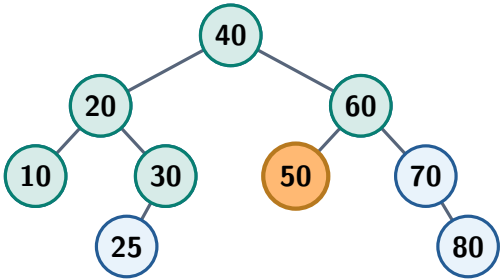
Défiler 30 ; visiter 30 ; enfiler 25.

État de la FIFO après l'étape :

tête / sortie			
	50	70	25

Nœuds déjà visités :40 - 20 - 60 - 10 - 30

Parcours en largeur : animation de la FIFO



Opération effectuée :

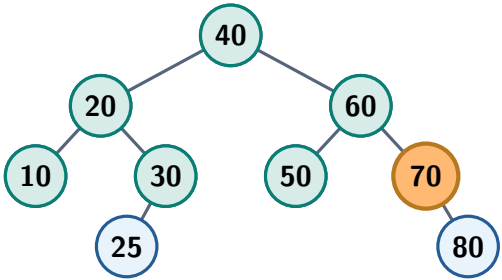
Défiler 50 ; visiter 50 ; aucun fils à enfiler.

État de la FIFO après l'étape :

tête / sortie		
	70	25

Nœuds déjà visités : 40 - 20 - 60 - 10 - 30 - 50

Parcours en largeur : animation de la FIFO



Opération effectuée :

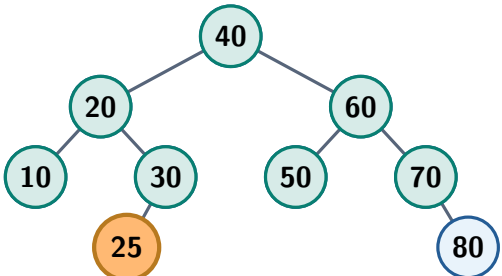
Défiler 70 ; visiter 70 ; enfiler 80.

État de la FIFO après l'étape :

tête / sortie		
	25	80

Nœuds déjà visités : 40 - 20 - 60 - 10 - 30 - 50 - 70

Parcours en largeur : animation de la FIFO



Opération effectuée :

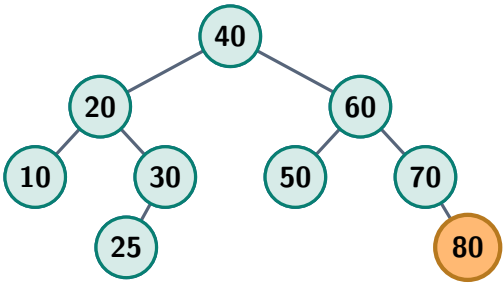
Défiler 25 ; visiter 25 ; aucun fils à enfiler.

État de la FIFO après l'étape :

tête / sortie | **80**

Nœuds déjà visités : 40 - 20 - 60 - 10 - 30 - 50 - 70 - 25

Parcours en largeur : animation de la FIFO



Opération effectuée :

Défiler 80 ; visiter 80 ; FIFO vide : terminé.

État de la FIFO après l'étape :

tête / sortie

File vide

Nœuds déjà visités :40 - 20 - 60 - 10 - 30 - 50 - 70 - 25 - 80

Zoom : une itération de la FIFO (traitement de 20)

État de départ : la racine 40 a déjà été visitée ; la file vaut $[20, 60]$.

1. **Défiler** la tête : on obtient le nœud 20.

Évolution de la FIFO

tête / sortie	**60**	20 vient d'être retiré

Résultat partiel : 40 , 20.

Zoom : une itération de la FIFO (traitement de 20)

État de départ : la racine 40 a déjà été visitée ; la file vaut $[20, 60]$.

1. **Défiler** la tête : on obtient le nœud 20.
2. **Visiter** 20 (une seule fois).

Évolution de la FIFO

tête / sortie	**60**	20 est visité

Résultat partiel : 40 , 20.

Zoom : une itération de la FIFO (traitement de 20)

État de départ : la racine 40 a déjà été visitée ; la file vaut $[20, 60]$.

1. **Défiler** la tête : on obtient le nœud 20.
2. **Visiter** 20 (une seule fois).
3. **Enfiler** son fils gauche 10, puis son fils droit 30.

Évolution de la FIFO

tête / sortie	**60**	**10**	fils gauche ajouté

Résultat partiel : 40 , 20. Les fils attendent leur tour dans la file.

Zoom : une itération de la FIFO (traitement de 20)

État de départ : la racine 40 a déjà été visitée ; la file vaut $[20, 60]$.

1. **Défiler** la tête : on obtient le nœud 20.
2. **Visiter** 20 (une seule fois).
3. **Enfiler** son fils gauche 10, puis son fils droit 30.

Évolution de la FIFO

tête / sortie	**60**	**10**	**30**	puis fils droit ajouté

Résultat partiel : 40 , 20. Les fils attendent leur tour dans la file.

Parcours en largeur : pseudo-code

```
largeur(racine):
    si racine == NULL : terminer
    f = FIFO vide
    enfiler(f, racine)
    tant que f n'est pas vide :
        n = defiler(f)
        visiter(n)
        si n.gauche != NULL : enfiler(f, n.gauche)
        si n.droite != NULL : enfiler(f, n.droite)
```

Pourquoi enfiler des pointeurs et non des valeurs ?

La FIFO mémorise les **nœuds à explorer** : après avoir retiré un nœud, il faut encore accéder à ses deux fils.

En C : réutiliser la FIFO du TP précédent

```c
void largeur(BST t) {
    Fifo f = create_fifo();
    if (t == NULL) return;
    fifo_add(&f, t);
    while (!fifo_is_empty(f)) {
        BST n = fifo_get(&f);
        visiter(n->data);
        if (n->ls != NULL) fifo_add(&f, n->ls);
        if (n->rs != NULL) fifo_add(&f, n->rs);
    }
}
```

Adaptation au TP3 : `fifo_add` stocke un `void*` vers un nœud ; `fifo_get` retire et libère la cellule de file, *pas* le nœud de l'arbre. Pour la production, traiter également les éventuels échecs d'allocation.

Complexités et ressources

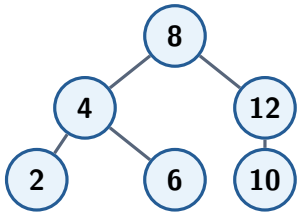
Parcours	Temps	Mémoire auxiliaire
Préfixe / infixe / postfixe	$O(n)$	$O(h)$: pile des appels récursifs
Largeur	$O(n)$	$O(w)$: nœuds présents dans la FIFO

- n : nombre total de nœuds.
- h : hauteur de l'arbre.
- w : largeur maximale de l'arbre (nombre de nœuds sur un même niveau).

À retenir

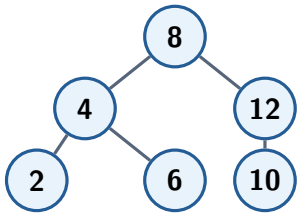
Un arbre très profond coûte davantage en pile pour un parcours récursif ; un arbre très large peut nécessiter une FIFO plus importante.

Exercice : prédire les quatre parcours



1. Préfixe ?
2. Infixe ?
3. Postfixe ?
4. Largeur ?

Exercice : prédire les quatre parcours



1. Préfixe ?
2. Infixe ?
3. Postfixe ?
4. Largeur ?

Correction

Préfixe : 8, 4, 2, 6, 12, 10.
Infixe : 2, 4, 6, 8, 10, 12.
Postfixe : 2, 6, 4, 10, 12, 8.
Largeur : 8, 4, 12, 2, 6, 10.

Synthèse

Parcours	Ordre	Structure ou technique
Préfixe	Racine – Gauche – Droite	Récursion / pile
Infixe	Gauche – Racine – Droite	Récursion / pile
Postfixe	Gauche – Droite – Racine	Récursion / pile
Largeur	Niveau par niveau	**FIFO**

Deux invariants à conserver

- Chaque nœud est **visité exactement une fois**.
- La structure de l'arbre reste **inchangée** pendant les quatre parcours.

Lien avec le TP4 : l'infixe permet de vérifier l'ordre des clés après les rotations ; le parcours en largeur réutilise la FIFO du TP3.