Algorithmique et structure de données

Structures de données

Rohan Fossé

11 Octobre 2021

Algorithmes et structures de données

La plupart des bons algorithmes fonctionnent grâce à une méthode astucieuse pour organiser les données. Nous allons étudier cinq classes de structures de données :

- les tableaux;
- les piles;
- les listes;
- les files;
- les arbres.

Structure séquentielles: les tableaux

Structure de donnée séquentielle : tableau

En anglais : array, vector.

Définition

Un **tableau** est une structure de donnée T qui permet de stocker un certain nombre d'éléments $T[i]$ repérés par un index i. Les tableaux vérifient généralement les propriétés suivantes :

- tous les éléments ont le même type de base;
- le nombre d'éléments stockés est fixé;
- l'accès et la modification de l'élément numéro i est en temps constant $\mathcal{O}(1)$, indépendant de i et du nombre d'éléments dans le tableau.

Opérations de base

Hypothèses :

- tableau de taille *capacite* alloué;
- éléments $0 \leq i \leq taille \leq capacite$ initialisés

A retenir

- accès au premier élément : $\mathcal{O}(1)$;

Opérations de base

Hypothèses :

- tableau de taille *capacite* alloué;
- éléments $0 \leq i \leq taille \leq capacite$ initialisés

A retenir

- accès au premier élément : $\mathcal{O}(1)$;
- accès à l'élément numéro i : $\mathcal{O}(1)$;

Opérations de base

Hypothèses :

- tableau de taille *capacite* alloué;
- éléments $0 \leq i \leq taille \leq capacite$ initialisés

A retenir

- accès au premier élément : $\mathcal{O}(1)$;
- accès à l'élément numéro i : $\mathcal{O}(1)$;
- accès au dernier élément : $\mathcal{O}(1)$;

Opérations de base

Hypothèses :

- tableau de taille *capacite* alloué;
- éléments $0 \leq i \leq taille \leq capacite$ initialisés

A retenir

- accès au premier élément : $\mathcal{O}(1)$;
- accès à l'élément numéro i : $\mathcal{O}(1)$;
- accès au dernier élément : $\mathcal{O}(1)$;
- insertion/suppression d'un élément au début : $\mathcal{O}(taille)$;

Opérations de base

Hypothèses :

- tableau de taille *capacite* alloué;
- éléments $0 \leq i \leq \mathit{taille} \leq \mathit{capacite}$ initialisés

A retenir

- accès au premier élément : $\mathcal{O}(1)$;
- accès à l'élément numéro i : $\mathcal{O}(1)$;
- accès au dernier élément : $\mathcal{O}(1)$;
- insertion/suppression d'un élément au début : $\mathcal{O}(\mathit{taille})$;
- insert./suppr. d'un elt en position i : $\mathcal{O}(\mathit{taille} - i)$

Opérations de base

Hypothèses :

- tableau de taille *capacite* alloué;
- éléments $0 \leq i \leq \mathit{taille} \leq \mathit{capacite}$ initialisés

A retenir

- accès au premier élément : $\mathcal{O}(1)$;
- accès à l'élément numéro i : $\mathcal{O}(1)$;
- accès au dernier élément : $\mathcal{O}(1)$;
- insertion/suppression d'un élément au début : $\mathcal{O}(\mathit{taille})$;
- insert./suppr. d'un elt en position i : $\mathcal{O}(\mathit{taille} - i)$
- insert./suppr. d'un elt à la fin : $\mathcal{O}(1)$;

Les piles

Définition

définition

Une pile est une liste linéaire d'objets où les consultations, les insertions et les suppressions se font du même côté. On dit que c'est le dernier arrivé, premier servi.

Définition

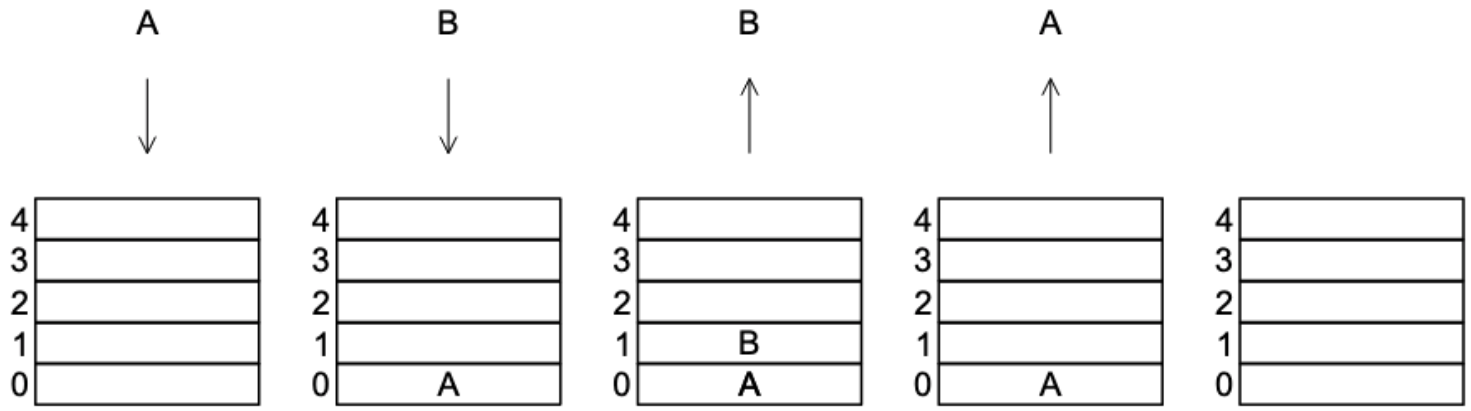
définition

Une pile est une liste linéaire d'objets où les consultations, les insertions et les suppressions se font du même côté. On dit que c'est le dernier arrivé, premier servi.

Définition

définition

Une pile est une liste linéaire d'objets où les consultations, les insertions et les suppressions se font du même côté. On dit que c'est le dernier arrivé, premier servi.



Définition

définition

Une pile est une liste linéaire d'objets où les consultations, les insertions et les suppressions se font du même côté. On dit que c'est le dernier arrivé, premier servi.

Vocabulaire

Le vocabulaire est basé sur une représentation verticale et est essentiellement en anglais, entre parenthèses ici:

- Dernier arrivé, premier servi (*Last-in-first-out* ou *LIFO*);
- pile (*stack*);
- empiler (*push*);
- dépiler (*pop*);
- sommet (*top*);
- pile vide (*empty*)

Opérations sur les piles

Les opérations sur la pile peuvent impliquer l'initialisation de la pile, son utilisation et sa dé-initialisation. En dehors de ces opérations de base, une pile est utilisée pour les deux opérations principales suivantes

- *push()* - Pousser (stocker) un élément sur la pile;
- *pop()* - Retirer (accéder à) un élément de la pile.

Pour utiliser efficacement une pile, nous devons également vérifier l'état de la pile, les fonctionnalités suivantes sont ajoutées aux piles :

- peek() - obtient l'élément de données supérieur de la pile, sans le supprimer;
- isFull() - Vérifie si la pile est pleine;
- isEmpty() - vérifie si la pile est vide.

Le pointeur top fournit la valeur supérieure de la pile sans la supprimer réellement.

Première opération : Peek

Définition

La fonction *peek*() permet d'obtenir l'élément de données supérieur de la pile, sans le supprimer.

```
1: function peek
2:     return stack[top]
```

Deuxième opération : isFull

Définition

La fonction *isFull*() permet de vérifier si la pile est pleine.

```
1: function isFull
2:     if top == MAXSIZE then
3:         return true
4:     else
5:         return false
```

Troisième opération : isEmpty

Définition

La fonction *isEmpty*() permet de vérifier si la pile est vide.

```
1: function isEmpty
2:     if top < 1 then
3:         return true
4:     else
5:         return false
```

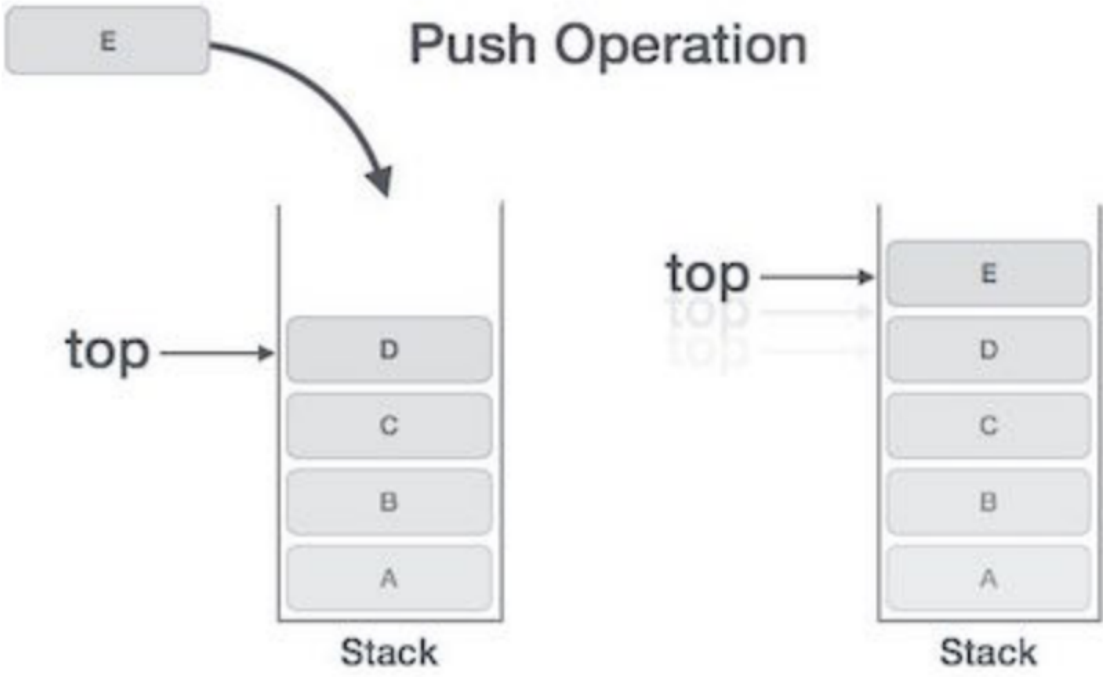
Opération push

Le processus consistant à placer un nouvel élément de données sur la pile est connu sous le nom d'opération push. L'opération push comporte une série d'étapes:

1. on vérifie si la pile est pleine;
2. si la pile est pleine, on retourne une erreur;
3. si la pile n'est pas pleine, on incrémente *top* jusqu'au prochain espace vide;
4. Ajoute un élément à l'emplacement de la pile où *top* pointe.

Opération push

Le processus consistant à placer un nouvel élément de données sur la pile est connu sous le nom d'opération push. L'opération push comporte une série d'étapes:



Opération push

Le processus consistant à placer un nouvel élément de données sur la pile est connu sous le nom d'opération push. L'opération push comporte une série d'étapes:

```
1: function push(data)
2:     if stack.isfull() then
3:         return null
4:     top ← top + 1
5:     stack[top] ← data
```

Opération pop

Accéder au contenu tout en le retirant de la pile est connu sous le nom d'opération pop. La fonction change suivant l'implémentation de la pile:

- Si elle a été implémenté avec un tableau, l'élément n'est pas réellement supprimé, mais le sommet est décrémenté à une position inférieure dans la pile pour pointer vers la valeur suivante;
- si elle a été implementé avec une liste chaînée, l'opération *pop*() supprime réellement l'élément de données et désalloue l'espace mémoire.

Opération pop

Accéder au contenu tout en le retirant de la pile est connu sous le nom d'opération pop. La fonction change suivant l'implémentation de la pile:

- Si elle a été implémenté avec un tableau, l'élément n'est pas réellement supprimé, mais le sommet est décrémenté à une position inférieure dans la pile pour pointer vers la valeur suivante;
- si elle a été implementé avec une liste chaînée, l'opération *pop*() supprime réellement l'élément de données et désalloue l'espace mémoire.

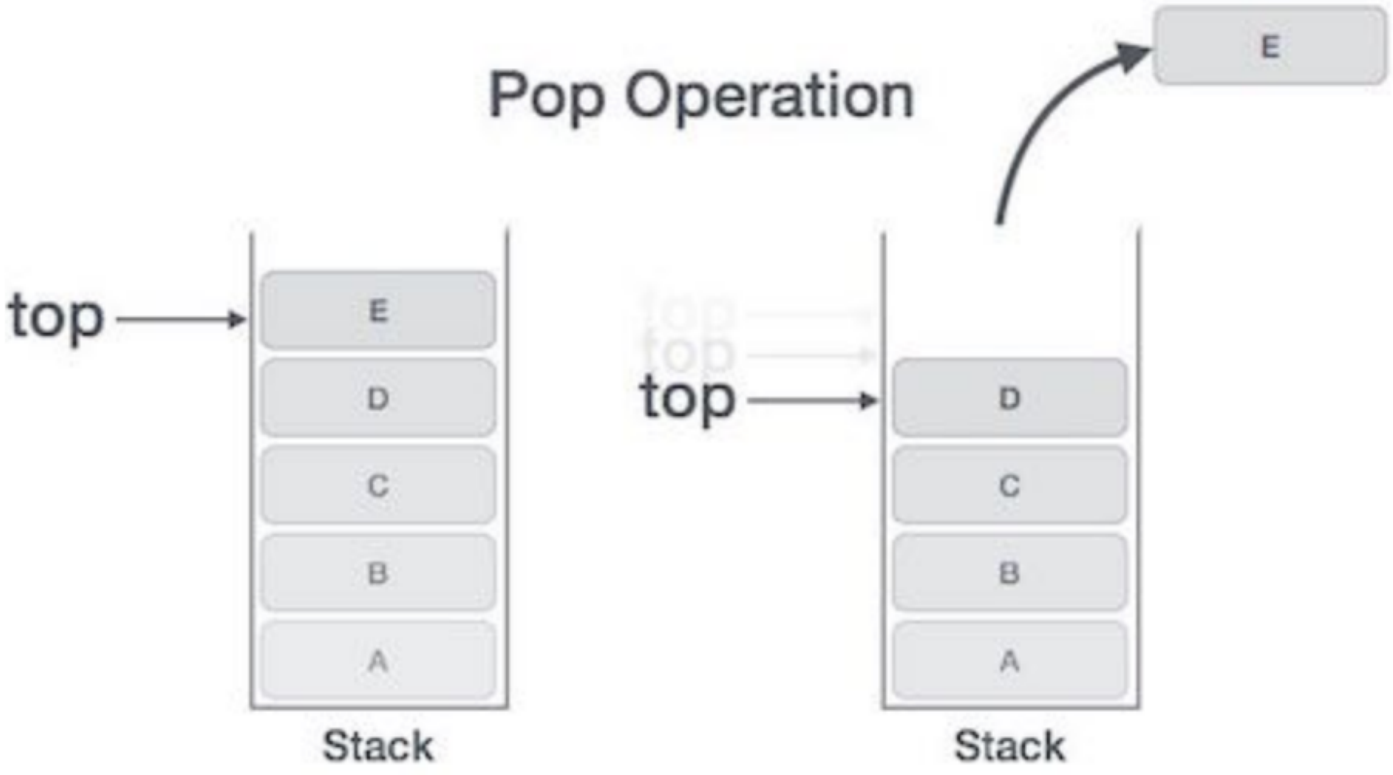
Opération pop

L'opération pop comporte une série d'étapes:

1. On vérifie si la pile est vide;
2. si elle est effectivement vide, on retourne une erreur;
3. sinon, on accède à l'élément *data* où *top* pointe;
4. on décroit la valeur de *top* de 1;
5. on retourne *data*

Opération pop

Le processus consistant à récupérer un élément de données sur la pile est connu sous le nom d'opération pop. L'opération pop comporte une série d'étapes:



Opération pop

Le processus consistant à récupérer un élément de données sur la pile est connu sous le nom d'opération pop. L'opération pop comporte une série d'étapes:

```
1: function pop
2:     if stack.isEmpty() then
3:         return null
4:     data ← stack[top]
5:     top ← top – 1
6:     return data
```

Petit exercice sur les piles

On suppose que une séquence d'opérations *push* et *pop* sont faites sur une pile P. On suppose que les opérations *push* empilent les entiers $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ dans l'ordre croissant et que toute valeur dépilée avec *pop* est affichée. On souhaite afficher toutes les valeurs.

Exemple

Considérons la séquence d'opérations suivante :

$$push(1) \rightarrow push(2) \rightarrow push(3) \rightarrow pop() \rightarrow pop() \rightarrow push(2)$$

Cette séquence va afficher :

Petit exercice sur les piles

On suppose que une séquence d'opérations *push* et *pop* sont faites sur une pile P. On suppose que les opérations *push* empilent les entiers $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ dans l'ordre croissant et que toute valeur dépilée avec *pop* est affichée. On souhaite afficher toutes les valeurs.

Exemple

Considérons la séquence d'opérations suivante :

$$push(1) \rightarrow push(2) \rightarrow push(3) \rightarrow pop() \rightarrow pop() \rightarrow push(2)$$

Cette séquence va afficher :

3 2

Petit exercice sur les piles

On suppose que une séquence d'opérations *push* et *pop* sont faites sur une pile P. On suppose que les opérations *push* empilent les entiers $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ dans l'ordre croissant et que toute valeur dépilée avec *pop* est affichée. On souhaite afficher toutes les valeurs.

Exercice

Dire quelle séquence de valeurs affichées n'est pas possible. Si elle l'est, donner la séquence d'opérations permettant de l'obtenir.

1. 4 3 2 1 0 9 8 7 6 5
2. 4 6 8 7 5 3 2 9 0 1
3. 2 5 6 7 4 8 9 3 1 0
4. 4 3 2 1 0 5 6 7 8 9

Solution de l'exercice

1. 4 3 2 1 0 9 8 7 6 5 ✓
2. 4 6 8 7 5 3 2 9 0 1
3. 2 5 6 7 4 8 9 3 1 0
4. 4 3 2 1 0 5 6 7 8 9

Solution

$$push(0-4) \rightarrow pop(4-0) \rightarrow push(5-9) \rightarrow pop(9-5)$$

Solution de l'exercice

1. 4 3 2 1 0 9 8 7 6 5
2. 4 6 8 7 5 3 2 9 0 1 ✗
3. 2 5 6 7 4 8 9 3 1 0
4. 4 3 2 1 0 5 6 7 8 9

Solution

$$push(0-4) \rightarrow pop(4) \rightarrow push(5-6) \rightarrow pop(6) \rightarrow push(7-8) \rightarrow pop(8)$$

$$\rightarrow pop(7) \rightarrow pop(5) \rightarrow pop(3) \rightarrow pop(2) \rightarrow push(9) \rightarrow pop(9) \rightarrow \text{✗}$$

Solution de l'exercice

1. 4 3 2 1 0 9 8 7 6 5
2. 4 6 8 7 5 3 2 9 0 1
3. 2 5 6 7 4 8 9 3 1 0 ✓
4. 4 3 2 1 0 5 6 7 8 9

Solution

$$push(0-2) \to pop(2) \to push(3-5) \to pop(5) \to push(6)$$

$$\to pop(6) \to push(7) \to pop(7) \to pop(4) \to push(8) \to pop(9)$$

$$\to push(9) \to pop(9) \to pop(3) \to pop(1) \to pop(0)$$

Solution de l'exercice

1. 4 3 2 1 0 9 8 7 6 5
2. 4 6 8 7 5 3 2 9 0 1
3. 2 5 6 7 4 8 9 3 1 0
4. 4 3 2 1 0 5 6 7 8 9 ✓

Solution

$$push(0-4) \to pop(4-0) \to push(5) \to pop(5) \to push(6) \to$$

$$pop(6) \to push(7) \to pop(7) \to push(8) \to pop(8) \to push(9) \to pop(9)$$

Exemples d'utilisation des piles

- Les algorithmes récursifs utilisent une pile d'appel;
- Dans un navigateur web, une pile sert à mémoriser les pages Web visitées;
- La fonction « Annuler la frappe » (en anglais Undo) d'un traitement de texte mémorise les modifications apportées au texte dans une pile;
- Par exemple, on peut inverser un tableau ou une chaîne de caractères en utilisant une pile;
- L'évaluation des expressions mathématiques en notation *post-fixée* (ou polonaise inverse) utilise une pile. Qu'est-ce-que c'est ?

Les différentes notations

La manière d'écrire une expression arithmétique est connue sous le nom de notation. Une expression arithmétique peut être écrite dans trois notations différentes mais équivalentes, c'est-à-dire sans changer la sortie d'une expression. Ces notations sont les suivantes:

- Notation infixe;
- Notation préfixe (polonaise);
- Notation postfixe (polonaise inversée).

Notation infixe

Nous écrivons naturellement les expressions en notation infixe (par exemple a - b + c), où les opérateurs sont utilisés entre les opérandes.
Il est facile pour nous, humains, de lire, d'écrire et de parler en notation infixe, mais ça n'est pas aussi simple pour un ordinateur.
Un algorithme pour traiter la notation infixe pourrait être difficile et coûteux en termes de temps et d'espace.

Notation préfixe

Dans la notation préfixe, l'opérateur précède les deux opérandes.

infixe	**prefixe**
2+3	+ 2 3
p-q	- p q
a + b * c	+ a * b c

Notation postfixe (notation polonaise inversée)

Dans la notation préfixe, l'opérateur est écrit après les deux opérandes.

infixe	**prefixe**	**postfixe**
2+3	+ 2 3	2 3 +
p-q	- p q	p q -
a + b * c	+ a * b c	a b c * +

Notation postfixe (notation polonaise inversée)

Dans la notation préfixe, l'opérateur est écrit après les deux opérandes.

infixe	**prefixe**	**postfixe**
2+3	+ 2 3	2 3 +
p-q	- p q	p q -
a + b * c	+ a * b c	a b c * +

A quoi ça sert ?

Conversion d'une expression infixe à postfixe

Supposons que l'on ait l'expression :

$$a * b + c * d - e$$

Conversion d'une expression infixe à postfixe

Supposons que l'on ait l'expression :

$$a * b + c * d - e$$

De façon équivalente, on peut l'écrire de la façon suivante :

$$[(a * b) + (c * d)] - e$$

Conversion d'une expression infixe à postfixe

Supposons que l'on ait l'expression :

$$a * b + c * d - e$$

On peut maintenant changer l'expression en expression posfixe :

$$[(ab*) + (cd*)] - e$$

Conversion d'une expression infixe à postfixe

Supposons que l'on ait l'expression :

$$a * b + c * d - e$$

On peut maintenant changer l'expression en expression posfixe :

$$[(ab*)(cd*)+] - e$$

Conversion d'une expression infixe à postfixe

Supposons que l'on ait l'expression :

$$a * b + c * d - e$$

On peut maintenant changer l'expression en expression posfixe :

$$[(ab*)(cd*)+]e-$$

Conversion d'une expression infixe à postfixe

Supposons que l'on ait l'expression :

$$a * b + c * d - e$$

On peut maintenant changer l'expression en expression posfixe :

$$[(ab*)(cd*)+]e-$$

On obtient donc à la fin :

$$ab * cd * +e-$$

Évaluation d'une expression postfixe

Reprenons notre expression postfixe :

$$ab * cd * +e-$$

Supposons que l'on a :

$$a = 2, b = 3, c = 5, d = 4, e = 9$$

On obtient donc :

$$23 * 54 * +9-$$

Évaluation d'une expression postfixe

Reprenons notre expression postfixe :

$$ab * cd * +e-$$

Supposons que l'on a :

$$a = 2, b = 3, c = 5, d = 4, e = 9$$

On obtient donc :

$$654 * +9-$$

Évaluation d'une expression postfixe

Reprenons notre expression postfixe :

$$ab * cd * +e-$$

Supposons que l'on a :

$$a = 2, b = 3, c = 5, d = 4, e = 9$$

On obtient donc :

$$620 + 9-$$

Évaluation d'une expression postfixe

Reprenons notre expression postfixe :

$$ab * cd * +e-$$

Supposons que l'on a :

$$a = 2, b = 3, c = 5, d = 4, e = 9$$

On obtient donc :

$$269-$$

Évaluation d'une expression postfixe

Reprenons notre expression postfixe :

$$ab * cd * +e-$$

Supposons que l'on a :

$$a = 2, b = 3, c = 5, d = 4, e = 9$$

On obtient donc :

$$17$$

Quel est l'algorithme ?

Rappel

- En mathématiques, Un opérateur binaire est un opérateur qui opère sur deux opérandes (comme l'addition;
- les opérateurs unaires agissent sur un seul opérande pour produire une nouvelle valeur.

On utilise pour cela une pile initialement vide. On lit successivement chacun des termes de l'expression et on applique l'une des trois règles suivantes selon la nature de ce terme:

1. C'est un opérande, on l'empile.
2. C'est un opérateur binaire $\triangle$, on dépile la valeur y au sommet de la pile puis la valeur x, puis on empile le résultat de l'opération x $\triangle$ y
3. C'est un opérateur unaire $\bigstar$, on dépile la valeur x au sommet de la pile, puis on empile le résultat de l'opération $\bigstar$ x;

Prenons un exemple

Rappel des règles

1. C'est un opérande, on l'empile.
2. C'est un opérateur binaire $\triangle$, on dépile la valeur y au sommet de la pile puis la valeur x, puis on empile le résultat de l'opération x $\triangle$ y
3. C'est un opérateur unaire $\bigstar$, on dépile la valeur x au sommet de la pile, puis on empile le résultat de l'opération $\bigstar$ x;

$$1.2 \quad 5 * 3 * 3 \quad 1 - 2 * 3 * +$$

Prenons un exemple

Rappel des règles

1. C'est un opérande, on l'empile.
2. C'est un opérateur binaire $\triangle$, on dépile la valeur y au sommet de la pile puis la valeur x, puis on empile le résultat de l'opération x $\triangle$ y
3. C'est un opérateur unaire $\bigstar$, on dépile la valeur x au sommet de la pile, puis on empile le résultat de l'opération $\bigstar$ x;

Considérons l'expression postfixe suivante :

$$1.2 \quad 5 * 3 * 3 \quad 1 - 2 * 3 * +$$

Prenons un exemple

Rappel des règles

1. C'est un opérande, on l'empile.
2. C'est un opérateur binaire $\triangle$, on dépile la valeur y au sommet de la pile puis la valeur x, puis on empile le résultat de l'opération x $\triangle$ y
3. C'est un opérateur unaire $\bigstar$, on dépile la valeur x au sommet de la pile, puis on empile le résultat de l'opération $\bigstar$ x;

$$1.2 \quad 5 * 3 * 3 \quad 1 - 2 * 3 * +$$

							1		2		3			
		5		3		3	3	2	2	4	4	12		
	1.2	1.2	6	6	18	18	18	18	18	18	18	18	30	
∅	1.2	5	*	3	*	3	1	-	2	*	3	*	+	∅

Quel est l'algorithme ?

```
1:  function evaluationpostfix(E)
2:      stack ← createStack()
3:      for chaque caractère ch dans l'expression postixe E do
4:          if  ch est un opérateur★ then
5:              a ← stack.pop()
6:              b ← stack.pop()
7:              res ← b★a
8:          if ch est un opérateur△ then
9:              a ← stack.pop()
10:             res ← △a
11:         else
12:             stack.push(ch)
13:     return top
```

Les listes

Liste chaînée : définition

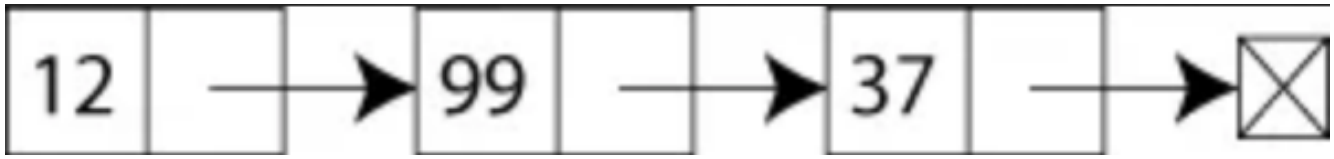
Définition

Une liste chaînée désigne en informatique une structure de données représentant une collection ordonnée et de taille arbitraire d'éléments de même type.

L'accès aux éléments d'une liste se fait de manière séquentielle : chaque élément permet l'accès au suivant (contrairement au cas du tableau dans lequel l'accès se fait de manière absolue, par adressage direct de chaque cellule du dit tableau).

Le principe

Le principe de la liste chaînée est que chaque élément possède, en plus de la donnée, des pointeurs vers les éléments qui lui sont logiquement adjacents dans la liste.



De ce fait, l'usage d'une liste est préconisé pour des raisons de rapidité de traitement, lorsque les insertions et suppressions d'élément en tout point sont relativement plus fréquentes que les accès simples.

En effet, une insertion ou une suppression se font en temps constant car elles ne demandent au maximum que deux accès en écriture. En revanche, l'accès à un élément aléatoirement positionné nécessite le parcours de chaque élément qui sépare l'index de l'élément choisi.

Types de listes chaînées

Il existe plusieurs types de listes chaînées :

Liste simplement chaînée
Liste que l'on ne peut traverser que dans un seul sens.

Liste simplement chaînée circulaire
Liste simplement chaînée et le dernier élément pointe sur le premier

Liste doublement chaînée
Chaque élément pointe vers l'élément devant et dernier lui.

Liste doublement chaînée
Liste doublement chaînée et le dernier élément pointe sur le premier

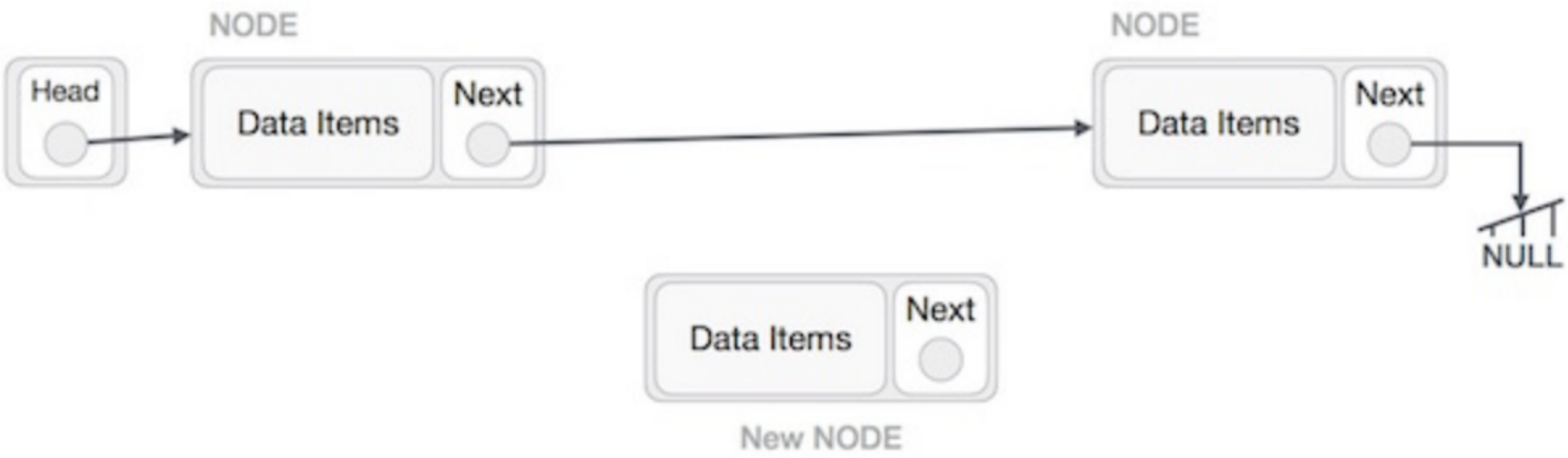
Opérations de base

Voici les différentes opérations principales que l'on peut appliquer aux listes chaînées.

- **Insertion** : on ajoute un élément au début de la liste;
- **Suppression** : on supprime un élément au début de la liste;
- **Recherche** : On cherche un élément dans la liste;

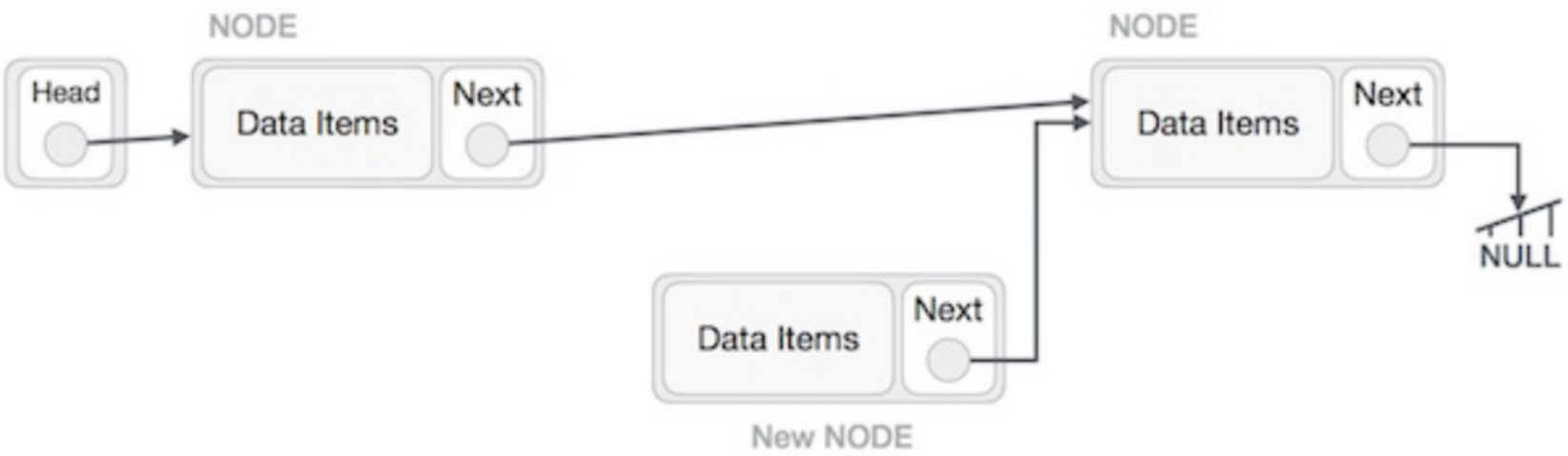
L'insertion

L'ajout d'un nouveau nœud dans une liste chaînée est une activité à plusieurs étapes. Nous allons l'apprendre ici à l'aide de diagrammes. Tout d'abord, créez un nœud en utilisant la même structure et trouvez l'emplacement où il doit être inséré.



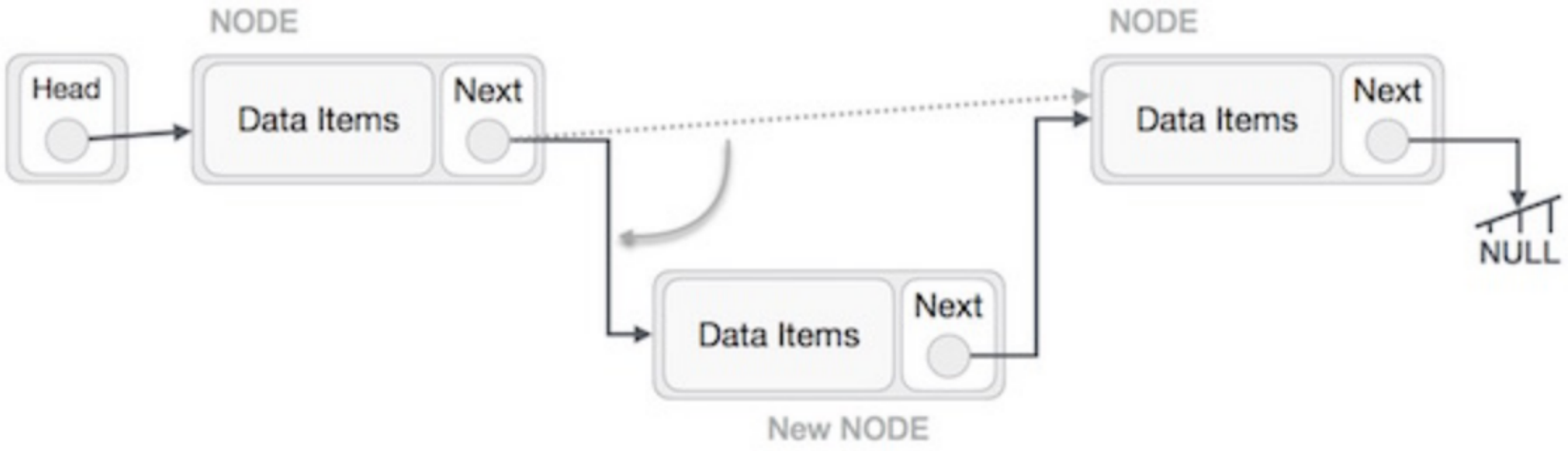
L'insertion

L'ajout d'un nouveau nœud dans une liste chaînée est une activité à plusieurs étapes. Nous allons l'apprendre ici à l'aide de diagrammes. Tout d'abord, créez un nœud en utilisant la même structure et trouvez l'emplacement où il doit être inséré.



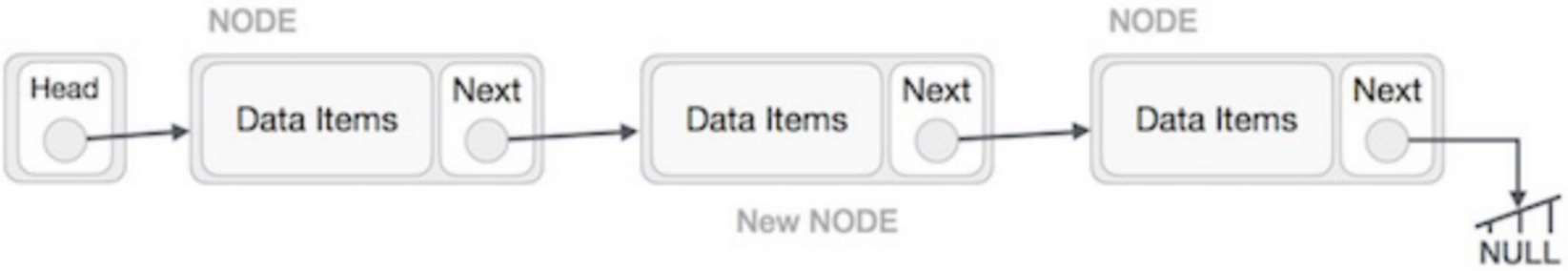
L'insertion

L'ajout d'un nouveau nœud dans une liste chaînée est une activité à plusieurs étapes. Nous allons l'apprendre ici à l'aide de diagrammes. Tout d'abord, créez un nœud en utilisant la même structure et trouvez l'emplacement où il doit être inséré.



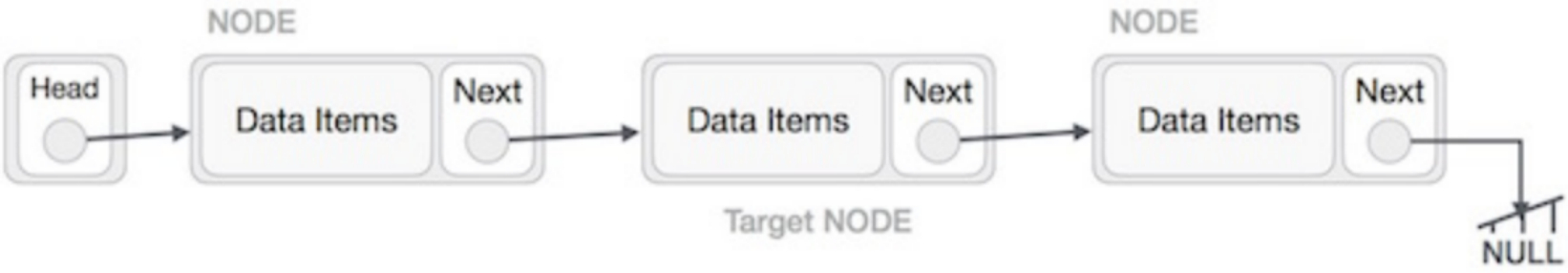
L'insertion

L'ajout d'un nouveau nœud dans une liste chaînée est une activité à plusieurs étapes. Nous allons l'apprendre ici à l'aide de diagrammes. Tout d'abord, créez un nœud en utilisant la même structure et trouvez l'emplacement où il doit être inséré.



La suppression d'un élément

La suppression est également un processus en plusieurs étapes. Nous allons l'apprendre à l'aide d'une représentation graphique. Tout d'abord, il faut localiser le nœud cible à supprimer.



La suppression d'un élément

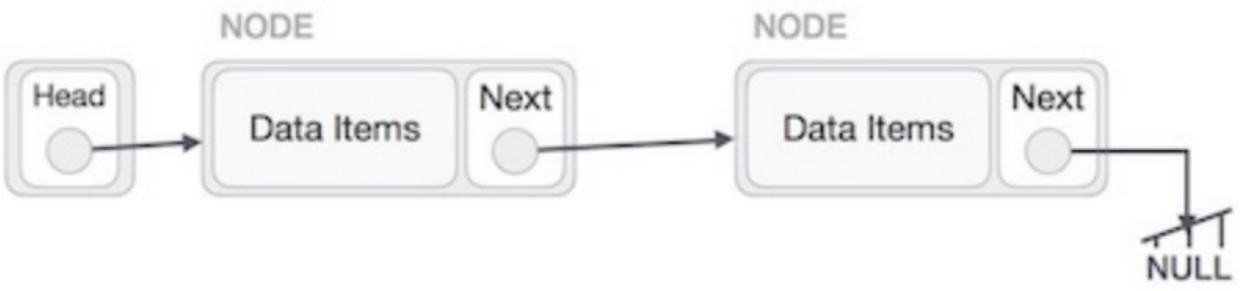
Le nœud gauche (précédent) du nœud cible doit maintenant pointer vers le nœud suivant du nœud cible.

La suppression d'un élément

Cela supprimera le lien qui pointait vers le nœud cible. Maintenant, nous allons supprimer ce vers quoi le noeud cible pointe.

La suppression d'un élément

Nous pouvons le garder en mémoire, sinon nous pouvons simplement désallouer la mémoire et effacer complètement le nœud cible.



Les files

Les files

Définition

La file est une structure de données abstraite, un peu similaire aux piles. Contrairement aux piles, une file d'attente est ouverte à ses deux extrémités.

Une extrémité est toujours utilisée pour insérer des données (*enqueue*) et l'autre est utilisée pour retirer des données (*dequeue*).

La file d'attente suit la méthodologie "*First-In-First-Out*", c'est-à-dire que l'élément de données stocké en premier sera accédé en premier.

Opérations de base

Les opérations sur les files d'attente peuvent consister à initialiser ou à définir la file d'attente, à l'utiliser, puis à l'effacer complètement de la mémoire. Nous allons essayer ici de comprendre les opérations de base associées aux files d'attente :

- **enqueue()** : ajoute (stocke) un élément à la file d'attente;
- **dequeue()** : retirer (accéder) un élément de la file d'attente;

Quelques fonctions supplémentaires sont nécessaires pour rendre l'opération de file d'attente mentionnée ci-dessus efficace.

- **peek()** - Obtient l'élément en tête de la file d'attente sans le retirer;
- **isfull()** - Vérifie si la file d'attente est pleine;
- **isempty()** - Vérifie si la file d'attente est vide.

Ajout d'un élément dans une file

Les files d'attente maintiennent deux pointeurs de données, avant et arrière. Par conséquent, leurs opérations sont plus difficiles à mettre en œuvre que celles des piles.

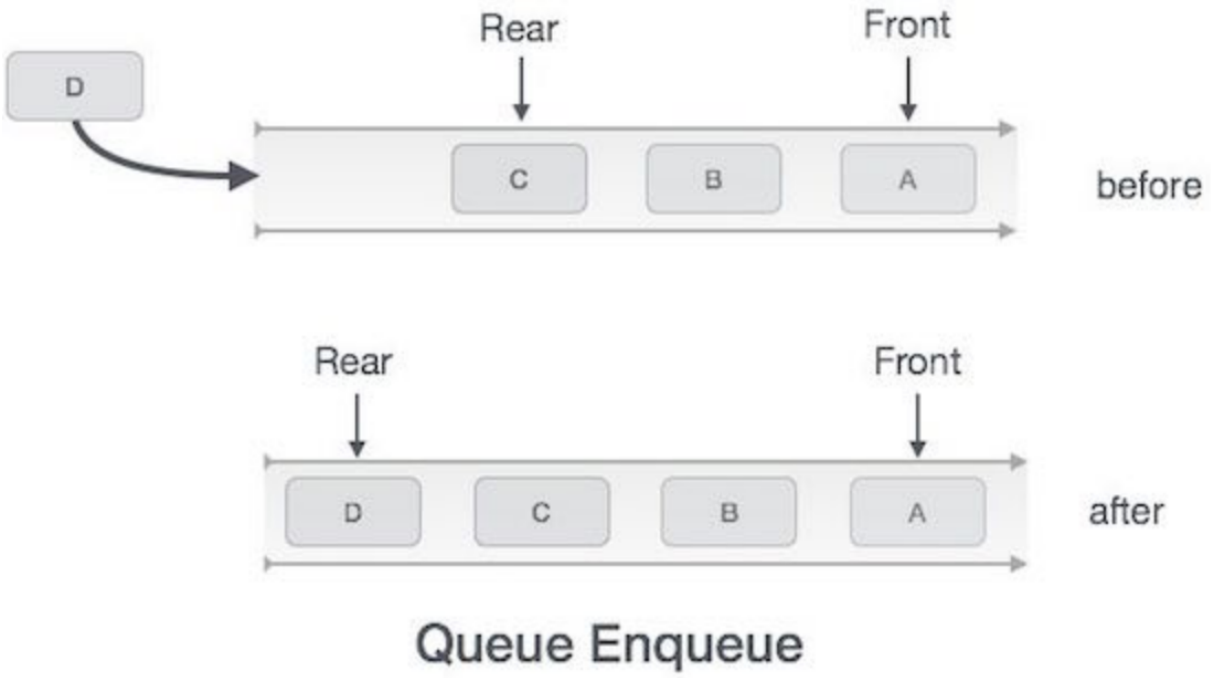
Les étapes suivantes doivent être suivies pour mettre en file d'attente (insérer) des données dans une file d'attente.

1. Étape 1 - Vérifier si la file d'attente est pleine.
2. Étape 2 - Si la file est pleine, produire une erreur de débordement et sortir.
3. Étape 3 - Si la file d'attente n'est pas pleine, incrémentez le pointeur arrière pour indiquer le prochain espace vide.
4. Étape 4 - Ajout d'un élément de données à l'emplacement de la file d'attente, où le pointeur arrière pointe.

Ajout d'un élément dans une file

Les files d'attente maintiennent deux pointeurs de données, avant et arrière. Par conséquent, leurs opérations sont plus difficiles à mettre en œuvre que celles des piles.

Les étapes suivantes doivent être suivies pour mettre en file d'attente (insérer) des données dans une file d'attente.



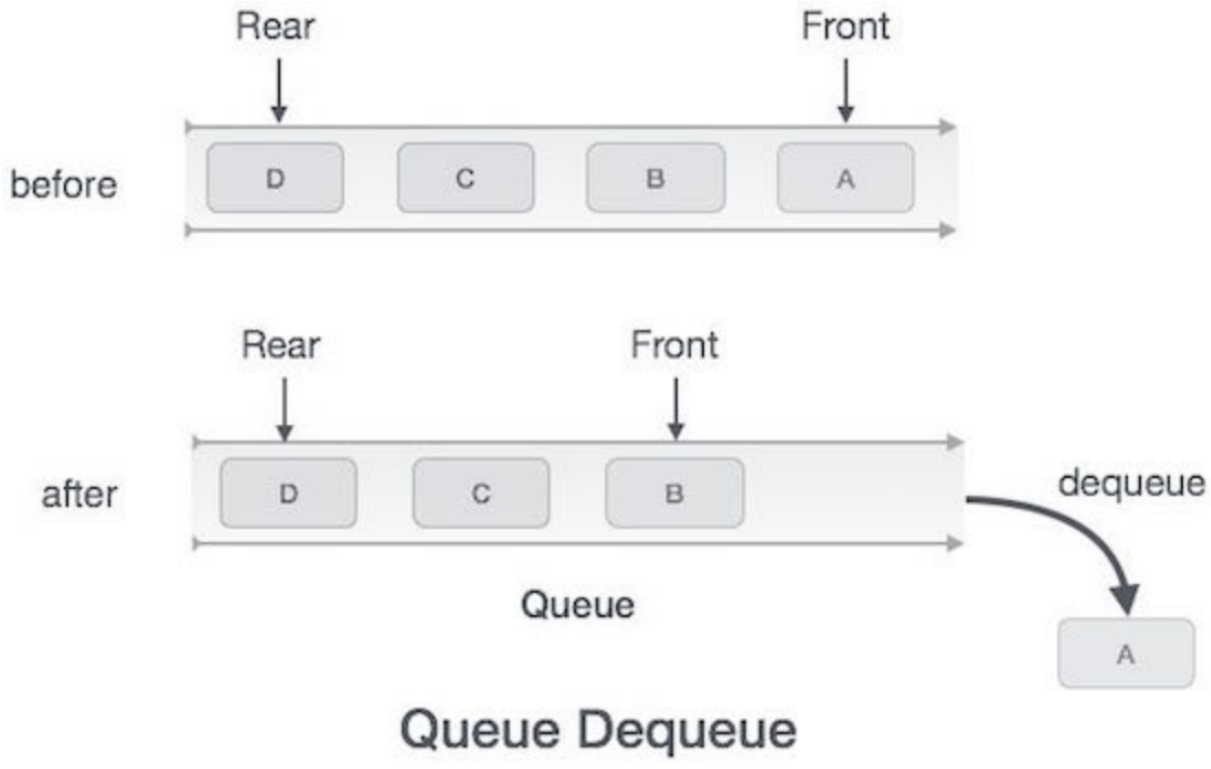
Retirer/accéder à un élément

L'accès aux données de la file d'attente est un processus composé de deux tâches - accéder aux données où le front est pointé et retirer les données après l'accès. Les étapes suivantes sont suivies pour effectuer l'opération de dequeue

1. Étape 1 - Vérifier si la file d'attente est vide.
2. Étape 2 - Si la file d'attente est vide, produire une erreur de dépassement de capacité et sortir.
3. Étape 3 - Si la file d'attente n'est pas vide, accéder aux données sur lesquelles le pointeur frontal pointe.
4. Étape 4 - Incrémente le pointeur frontal pour qu'il pointe sur le prochain élément de données disponible.

Retirer/accéder à un élément

L'accès aux données de la file d'attente est un processus composé de deux tâches - accéder aux données où le front est pointé et retirer les données après l'accès. Les étapes suivantes sont suivies pour effectuer l'opération de dequeue



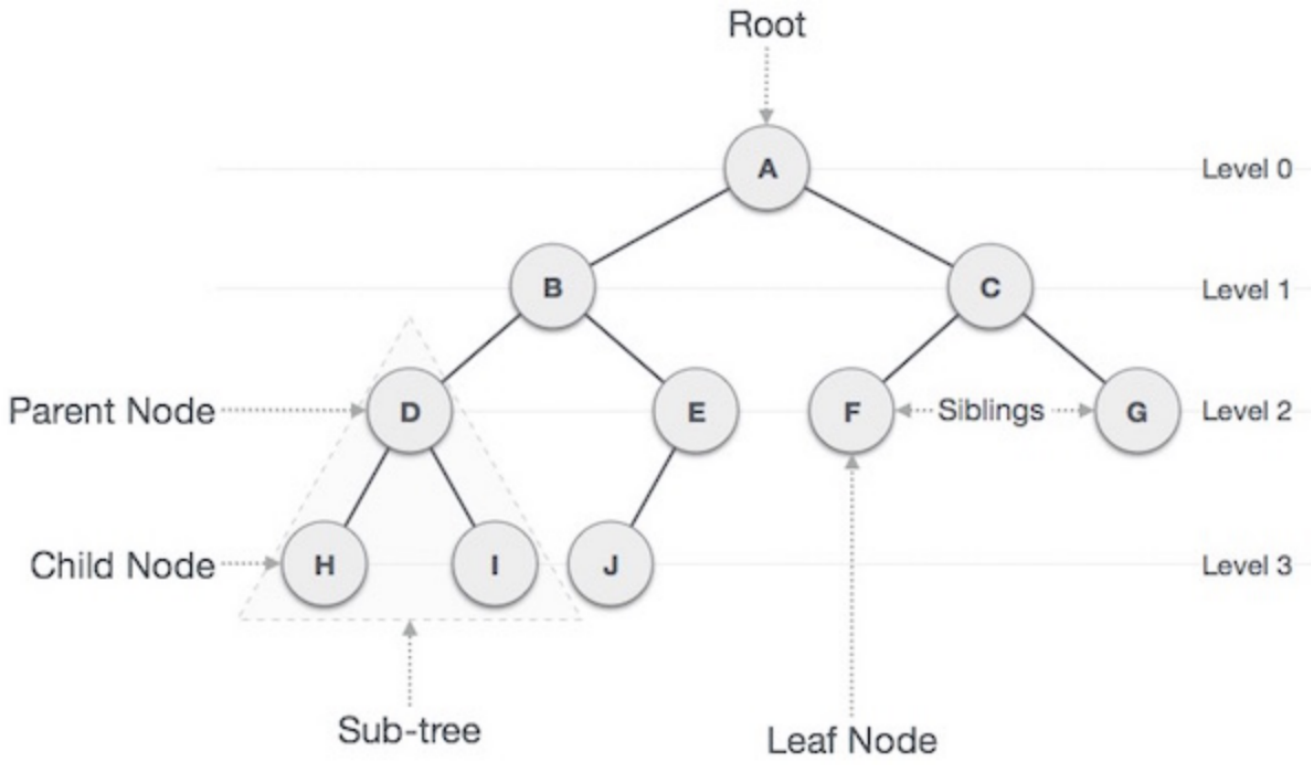
Queue Dequeue

Les arbres

Les arbres binaires

Définition

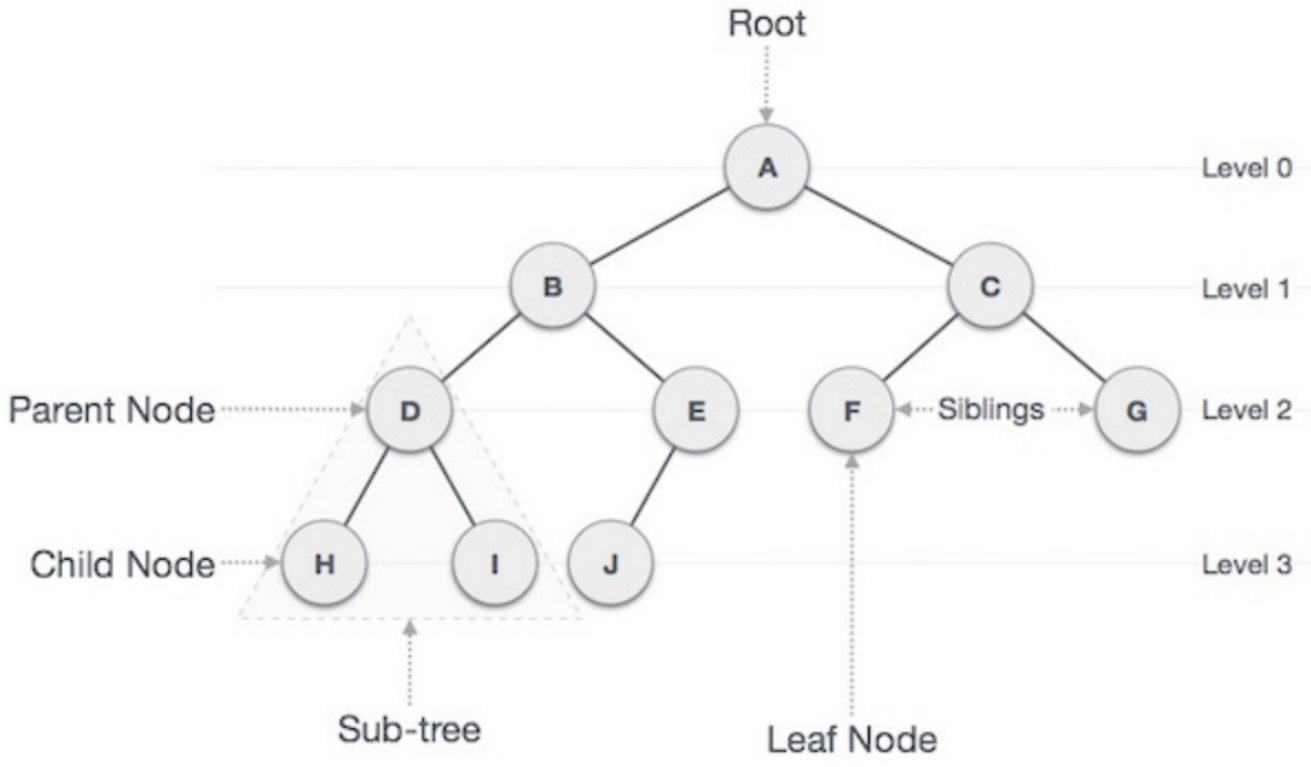
L'arbre binaire est une structure de données spéciale utilisée à des fins de stockage de données. Un arbre binaire a une condition spéciale : chaque nœud peut avoir un maximum de deux enfants.



Vocabulaire sur les arbres

Chemin

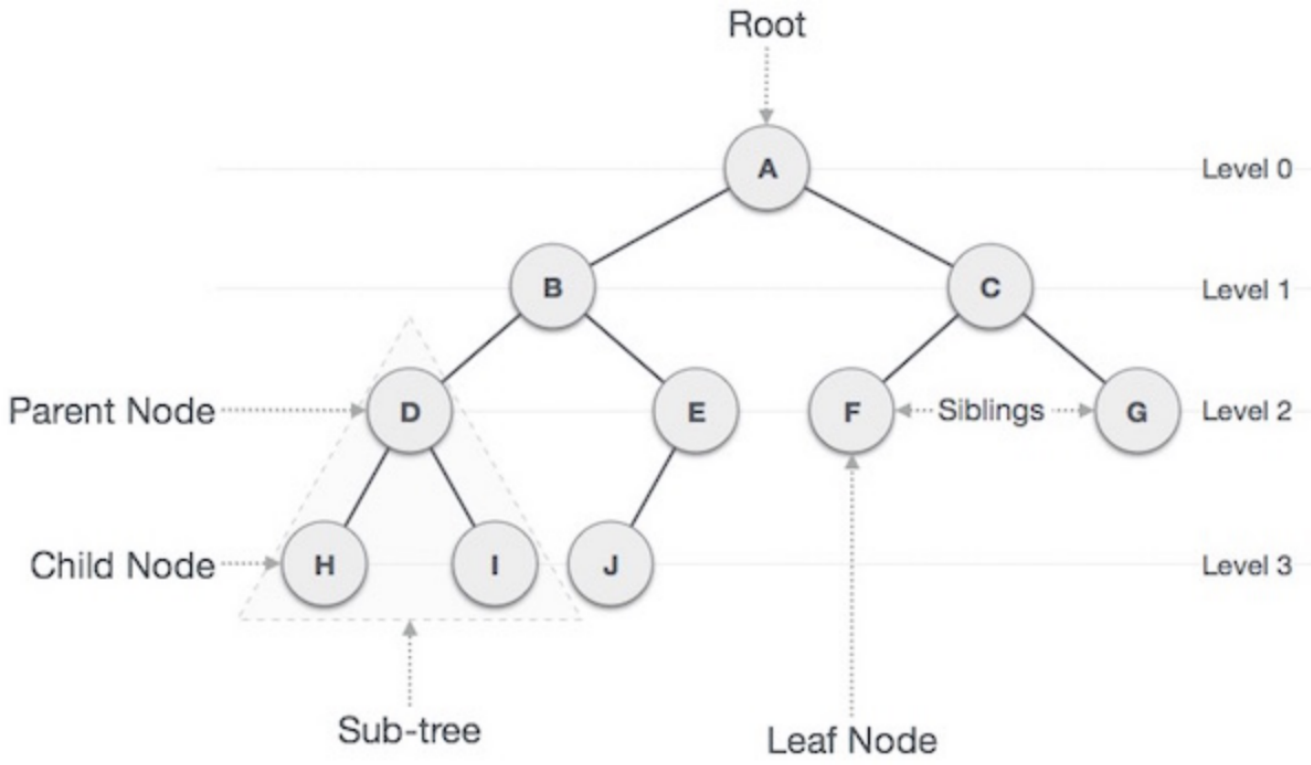
Le chemin fait référence à la séquence de nœuds le long des bords d'un arbre.



Vocabulaire sur les arbres

Racine

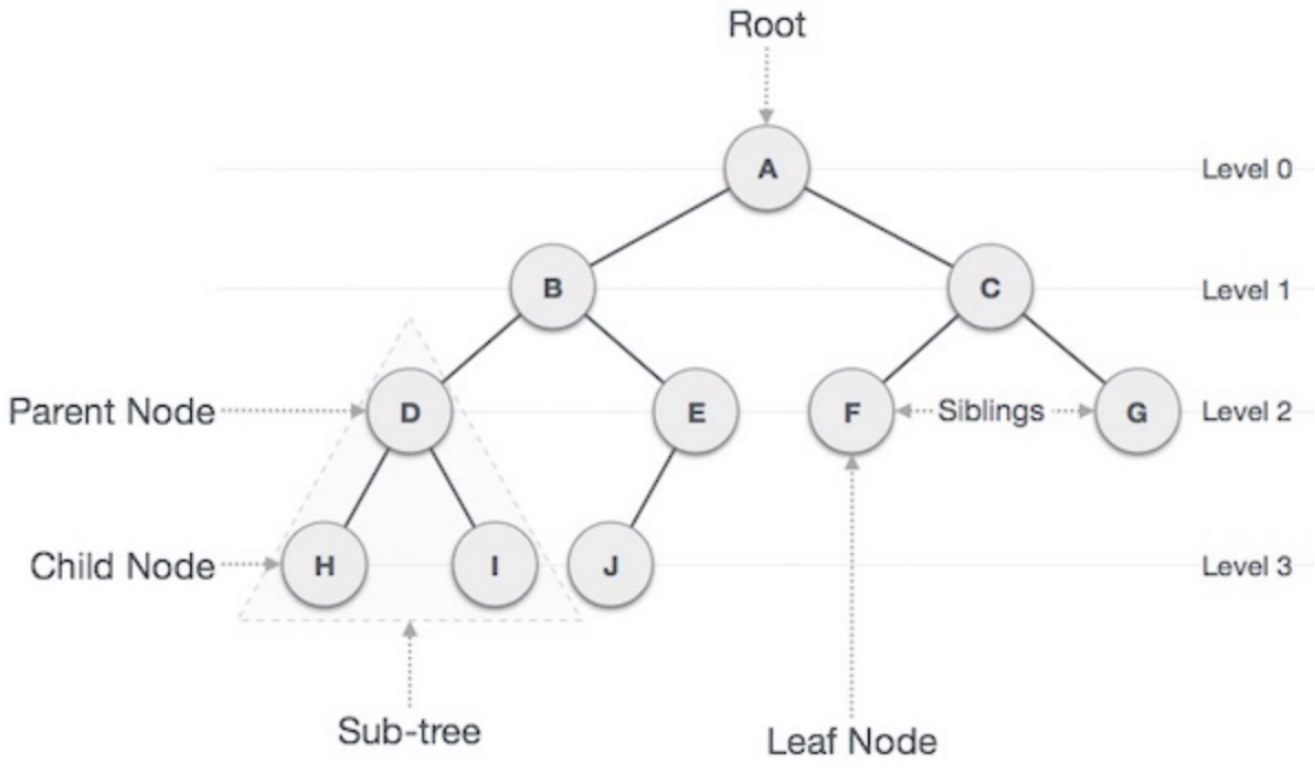
Le nœud situé au sommet de l'arbre est appelé racine. Il n'y a qu'une seule racine par arbre et un seul chemin du nœud racine vers n'importe quel nœud.



Vocabulaire sur les arbres

Parent

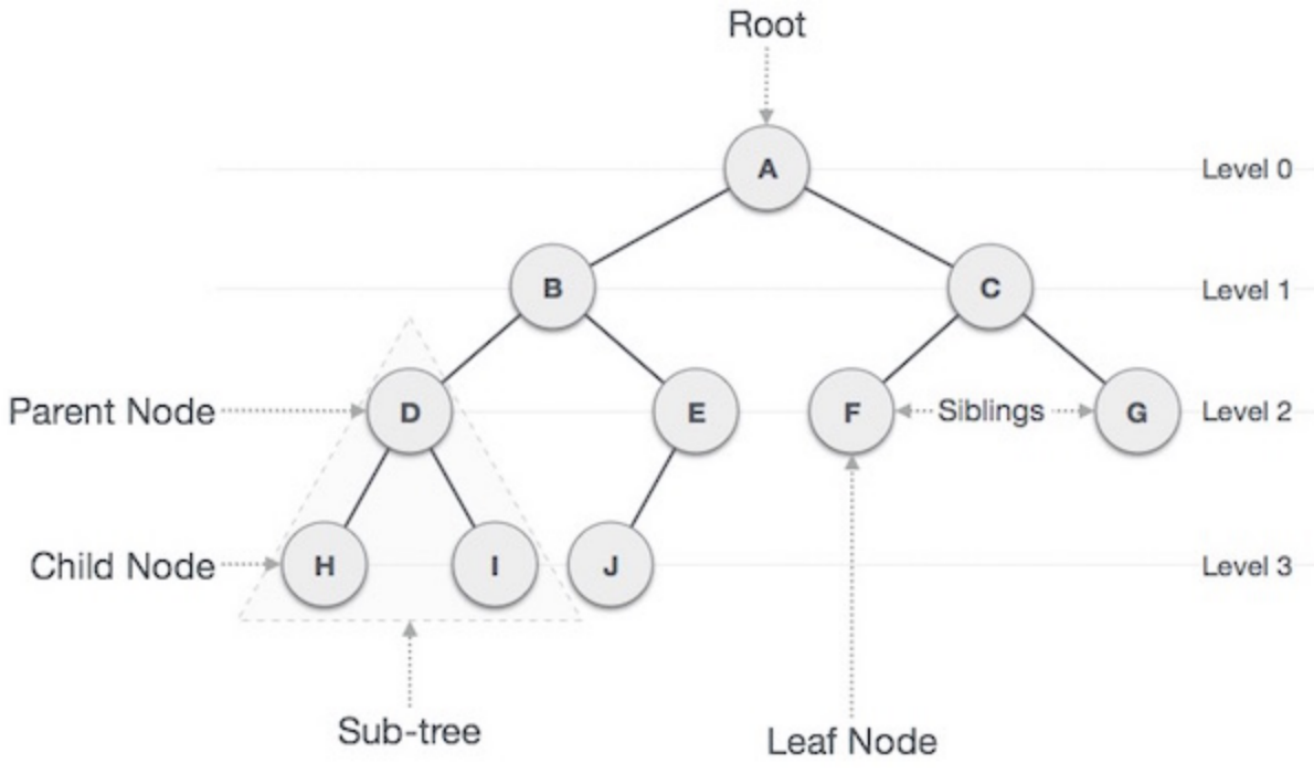
Tout nœud, à l'exception du nœud racine, possédant une arête ascendante vers un nœud appelé parent.



Vocabulaire sur les arbres

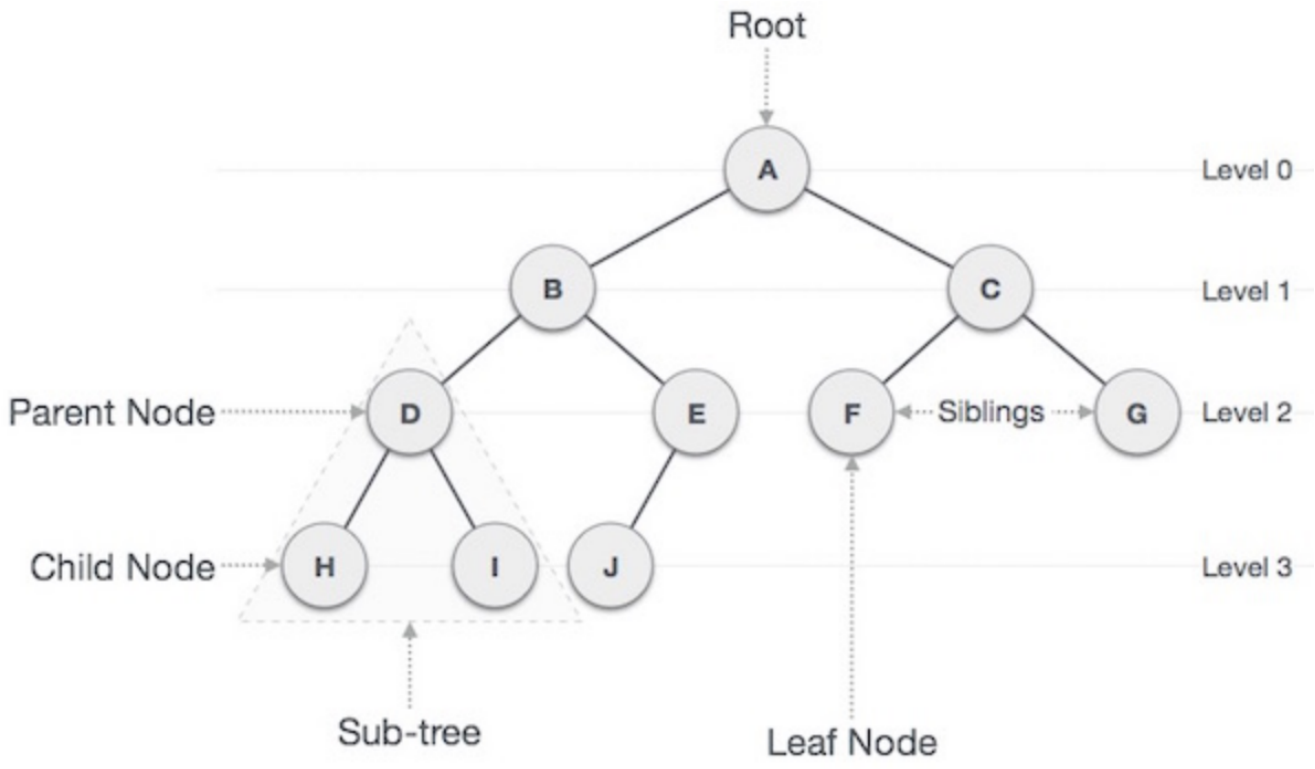
Enfant

Le nœud situé sous un nœud donné et relié par son bord descendant est appelé nœud enfant.



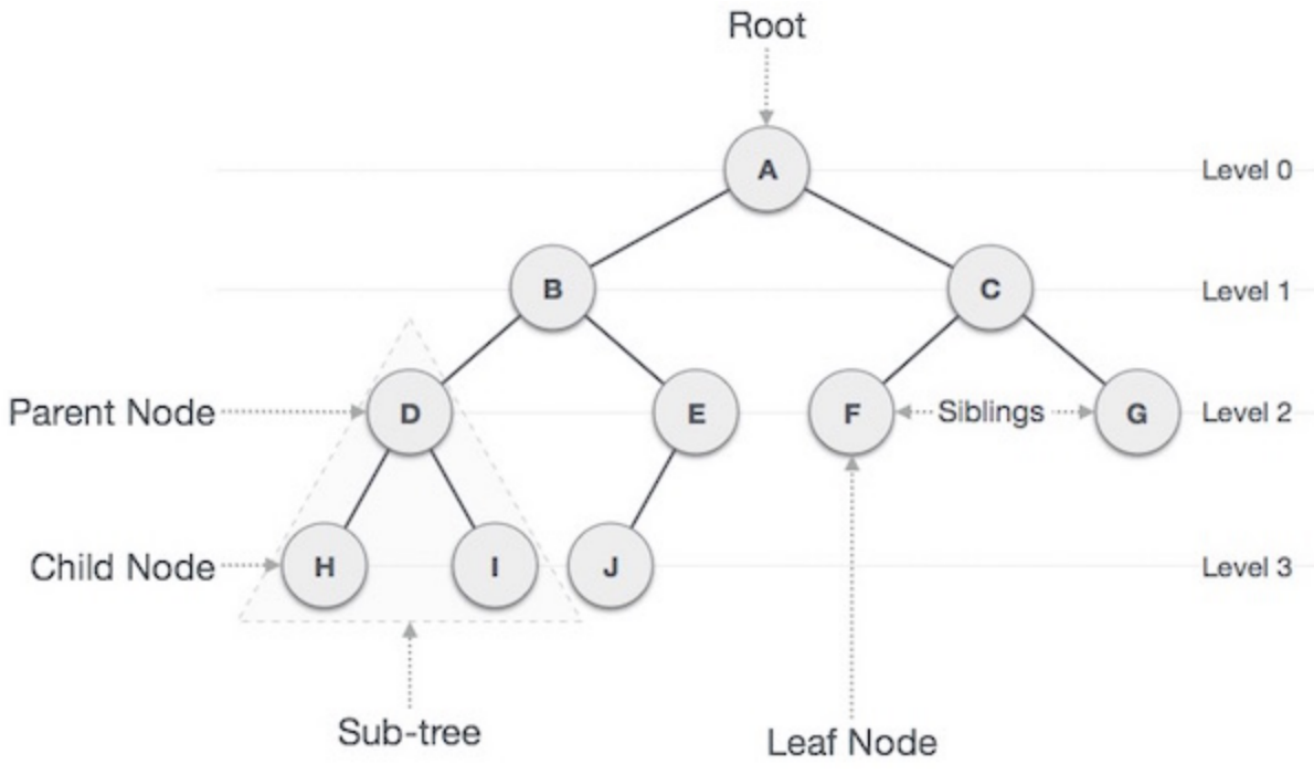
Vocabulaire sur les arbres

Feuille
Le nœud qui n'a pas de nœud enfant est appelé nœud feuille.



Vocabulaire sur les arbres

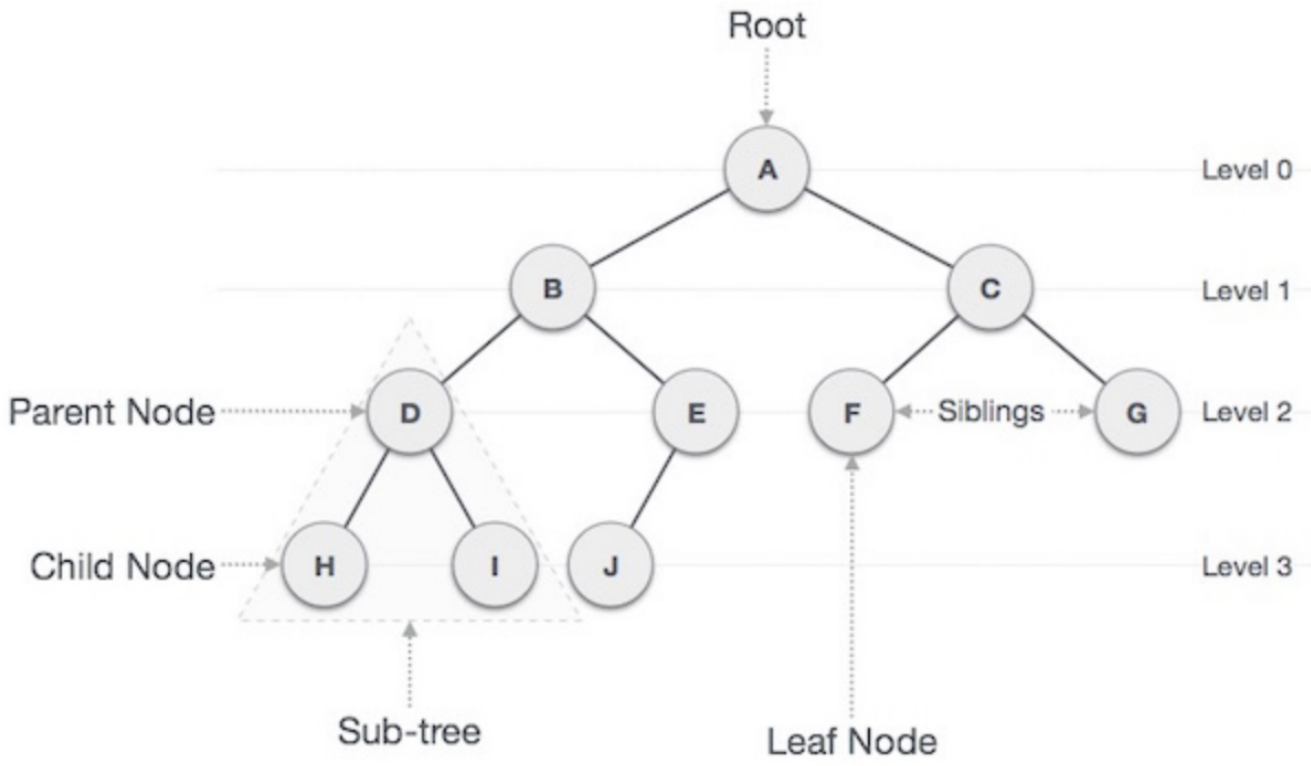
sous-arbre
Le sous-arbre représente les descendants d'un nœud.



Vocabulaire sur les arbres

Visiter

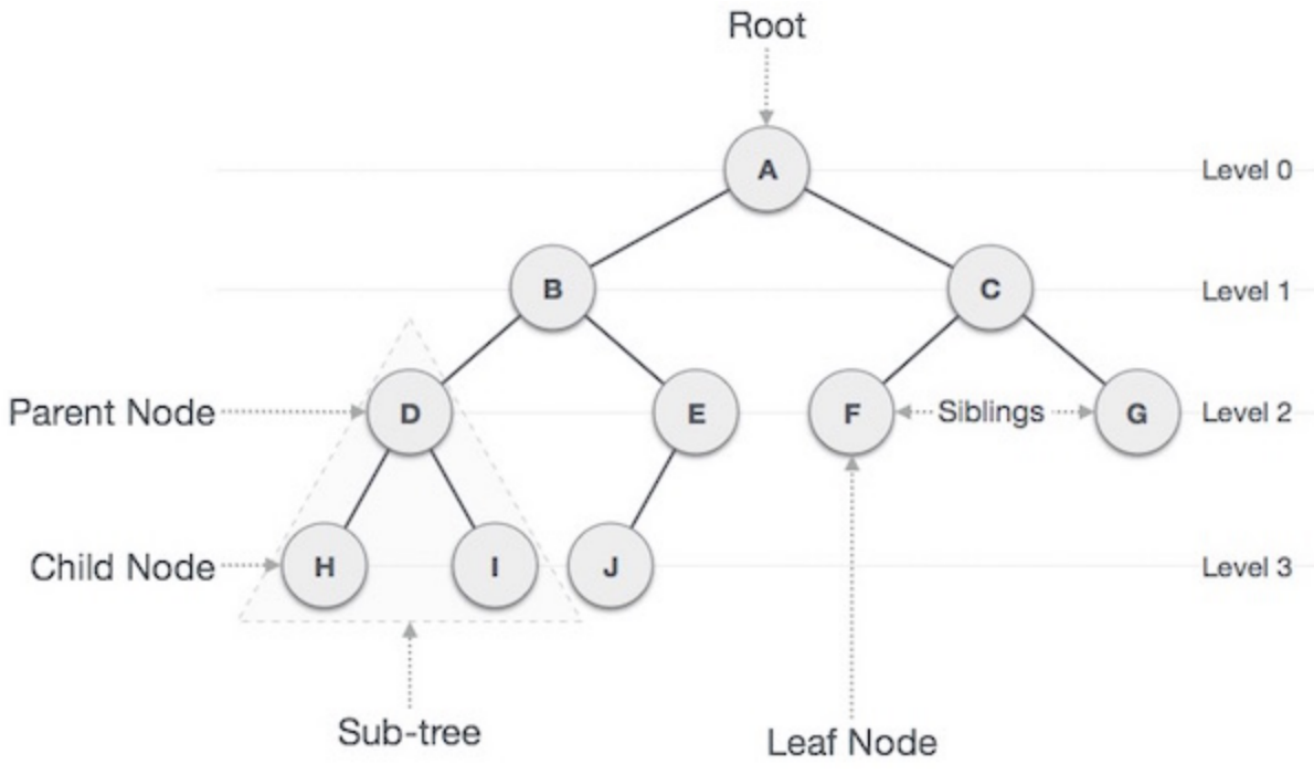
Visiter signifie vérifier la valeur d'un nœud lorsque le contrôle est sur le nœud.



Vocabulaire sur les arbres

Parcours

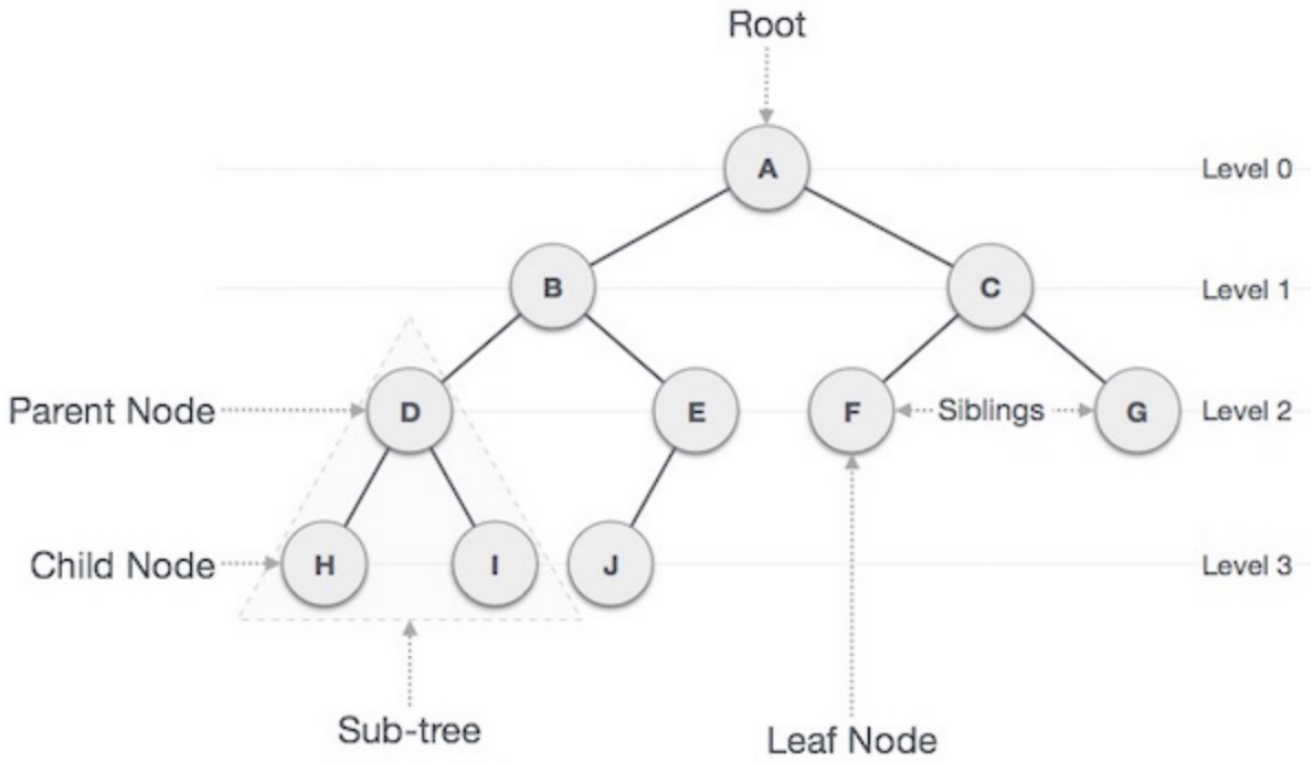
Le parcours consiste à passer par les nœuds dans un ordre spécifique.



Vocabulaire sur les arbres

Niveaux

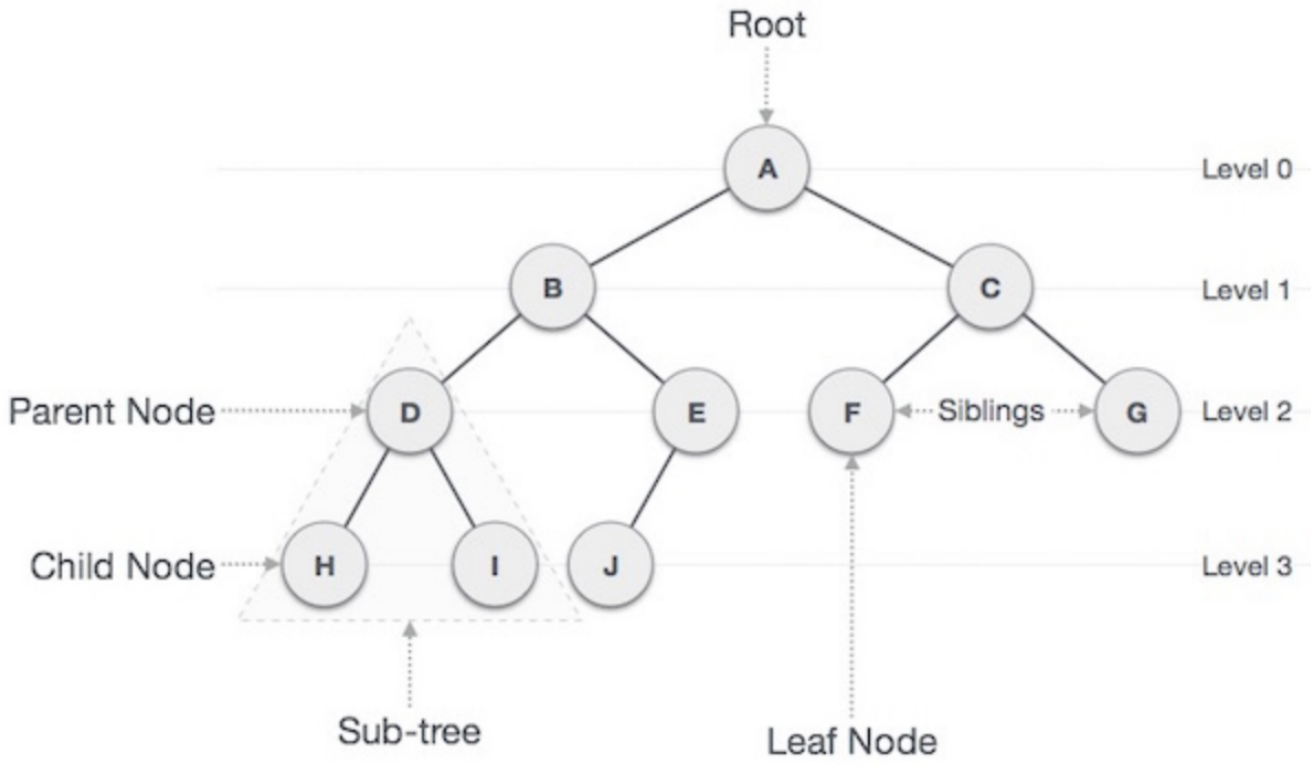
Le niveau d'un nœud représente la génération d'un nœud. Si le nœud racine se trouve au niveau 0, alors son nœud enfant suivant se trouve au niveau 1, son petit-enfant se trouve au niveau 2, et ainsi de suite.



Vocabulaire sur les arbres

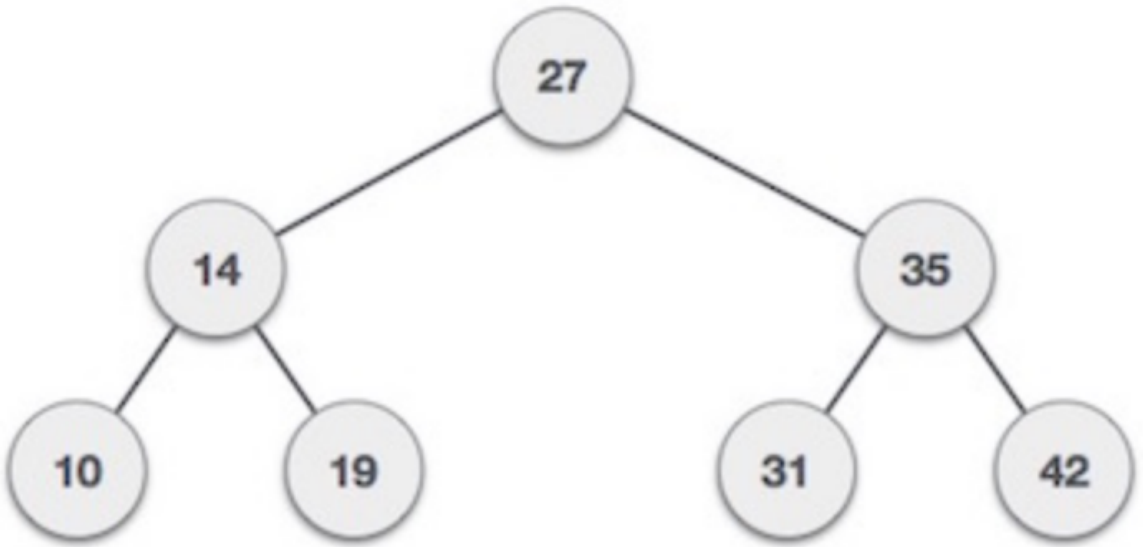
Clés

La clé représente une valeur d'un nœud sur la base de laquelle une opération de recherche doit être effectuée pour un nœud.



Représentation d'un arbre binaire de recherche

L'arbre de binaire de recherche (ou ABR) présente un comportement particulier. L'enfant gauche d'un nœud doit avoir une valeur inférieure à celle de son parent et l'enfant droit du nœud doit avoir une valeur supérieure à celle de son parent.



Opérations de base

Les opérations de base qui peuvent être effectuées sur une structure de données d'arbre de recherche binaire sont les suivantes :

- **Insertion** : Insère un élément dans un arbre/créer un arbre;
- **Recherche** : Recherche d'un élément dans un arbre;
- **Parcours préfixe** : Parcours un arbre de manière pré-ordonnée;
- **Parcours infixe** : Parcours un arbre dans l'ordre;
- **Parcours postfixe** : Parcours un arbre de manière post-ordonnée;

L'insertion d'un élément

La toute première insertion crée l'arbre. Par la suite, chaque fois qu'un élément doit être inséré, il faut d'abord localiser son emplacement approprié. Commencez la recherche à partir du noeud racine, puis si les données sont inférieures à la valeur clé, recherchez l'emplacement vide dans le sous-arbre de gauche et insérez les données. Sinon, recherchez l'emplacement vide dans le sous-arbre de droite et insérez les données.

L'insertion d'un élément

La toute première insertion crée l'arbre. Par la suite, chaque fois qu'un élément doit être inséré, il faut d'abord localiser son emplacement approprié. Commencez la recherche à partir du noeud racine, puis si les données sont inférieures à la valeur clé, recherchez l'emplacement vide dans le sous-arbre de gauche et insérez les données. Sinon, recherchez l'emplacement vide dans le sous-arbre de droite et insérez les données.

Quel serait l'algorithme ? A vous de jouer !

La recherche d'un élément

Lorsqu'un élément doit être recherché, la recherche commence à partir du nœud racine, puis si les données sont inférieures à la valeur de la clé, l'élément est recherché dans le sous-arbre de gauche. Sinon, recherchez l'élément dans le sous-arbre de droite. Suivez le même algorithme pour chaque nœud.

La recherche d'un élément

Lorsqu'un élément doit être recherché, la recherche commence à partir du nœud racine, puis si les données sont inférieures à la valeur de la clé, l'élément est recherché dans le sous-arbre de gauche. Sinon, recherchez l'élément dans le sous-arbre de droite. Suivez le même algorithme pour chaque nœud.

Quel serait l'algorithme ? A vous de jouer !

Les parcours

Parcours

Le parcours consiste à passer par les nœuds dans un ordre spécifique.

Étant donné que tous les nœuds sont reliés par des arêtes, nous commençons toujours par le nœud racine. Il y a trois façons de parcourir un arbre.

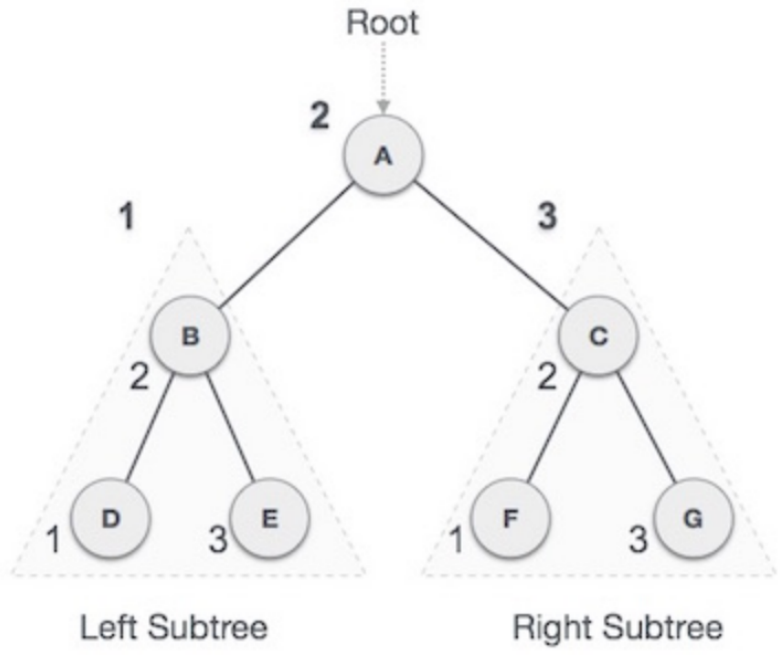
- Parcours infixe
- Parcours préfixe
- Parcours suffixe

En général, nous parcourons un arbre pour rechercher ou localiser un élément ou une clé donnée dans l'arbre ou pour imprimer toutes les valeurs qu'il contient.

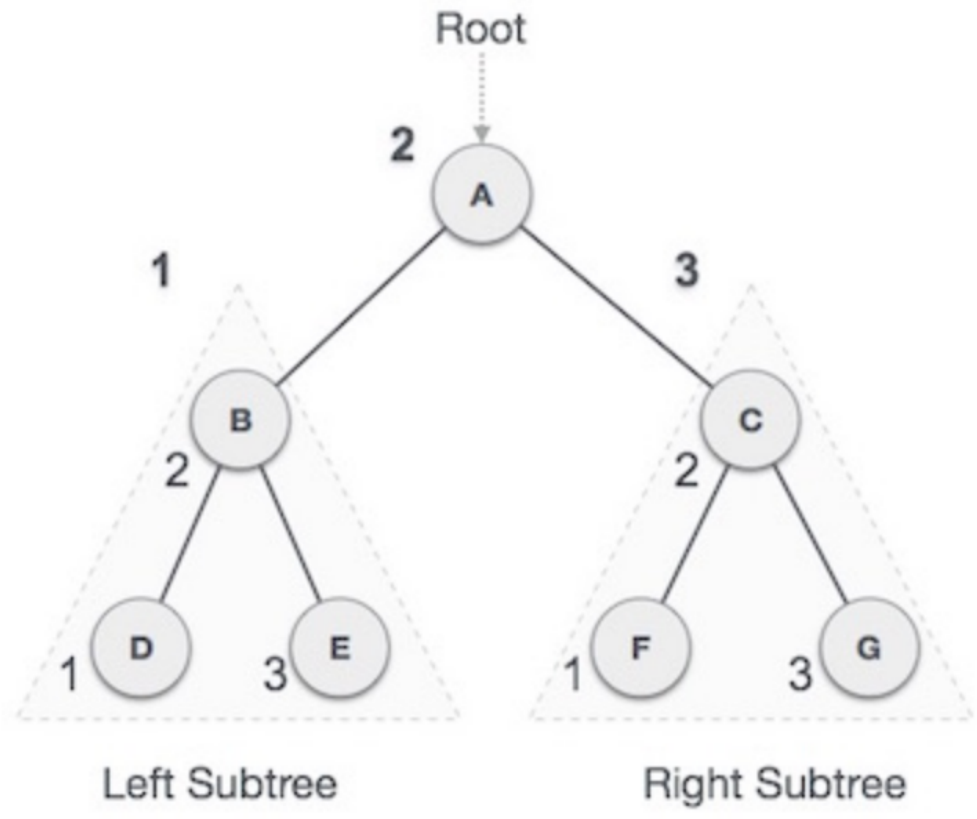
Parcours infixe

Dans ce parcours, le sous-arbre de gauche est visité en premier, puis la racine et enfin le sous-arbre de droite. Il faut toujours se rappeler que chaque nœud peut représenter un sous-arbre lui-même.

Si un arbre binaire est parcouru dans l'ordre, la sortie produira des valeurs de clés triées dans un ordre croissant.

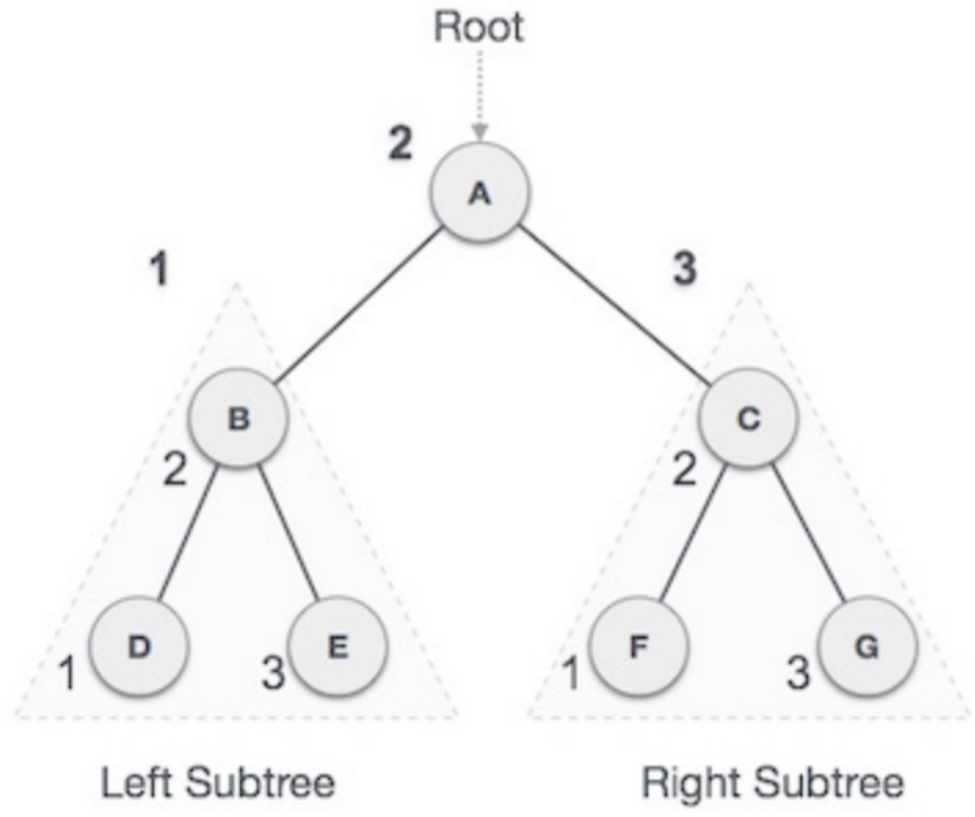


Parcours infixe



Qu'obtiendrait-on dans cet exemple ?

Parcours infixe



Qu'obtiendrait-on dans cet exemple ?

$D \to B \to E \to A \to F \to C \to G$

Ébauche d'algorithme

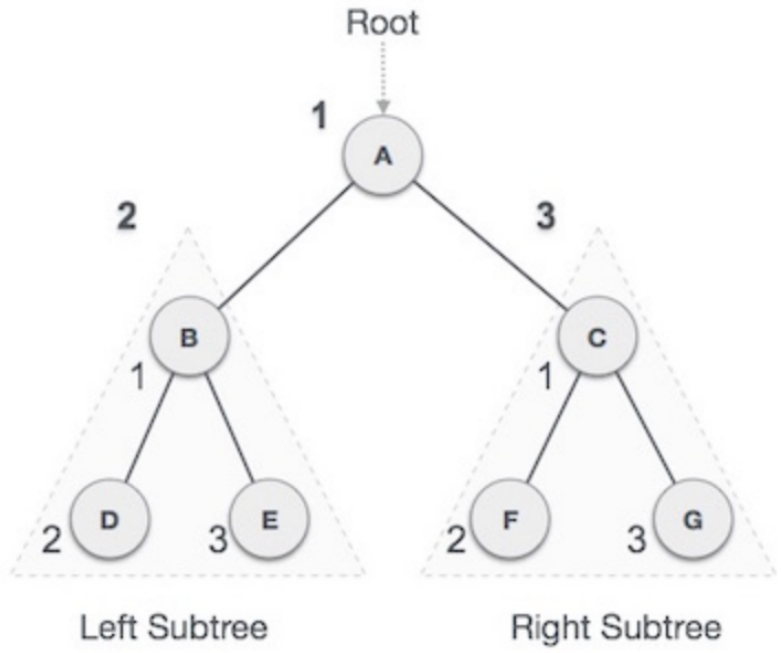
Algorithme

Tant que tout les les noeuds ne sont pas parcouru :

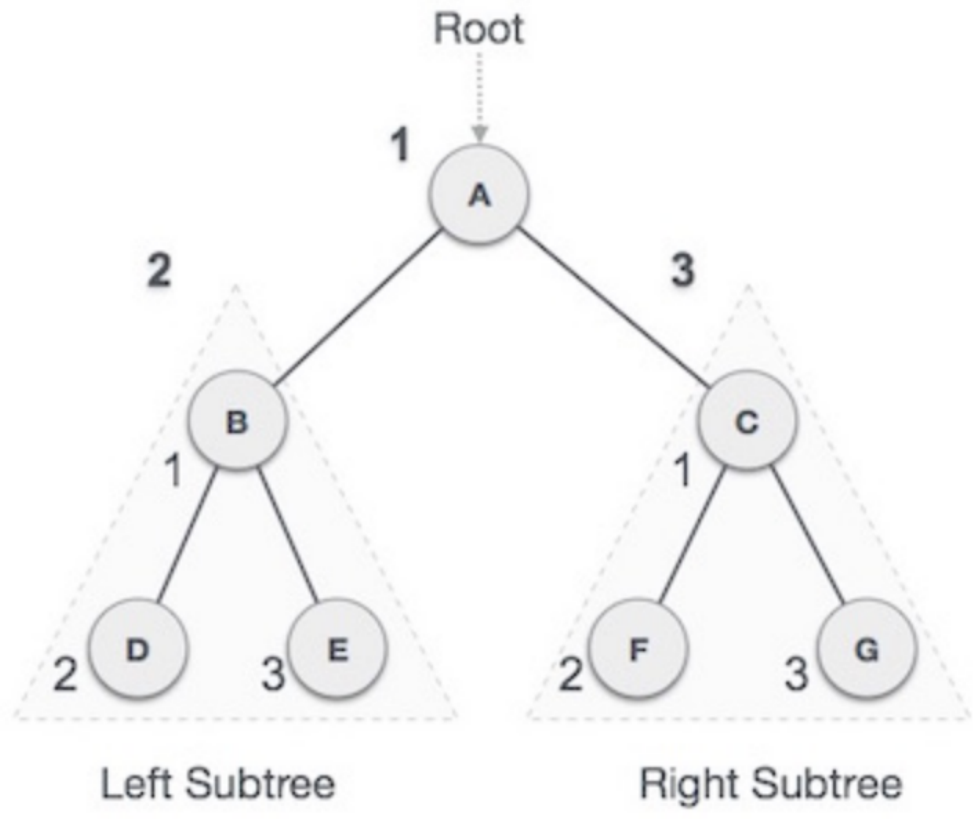
1. Étape 1 - Parcourir récursivement le sous-arbre de gauche.
2. Étape 2 - Visite du noeud racine.
3. Étape 3 - Parcourez récursivement le sous-arbre de droite.

Le parcours préfixe

Dans ce parcours, le nœud racine est visité en dernier, d'où son nom. Nous parcourons d'abord le sous-arbre de gauche, puis le sous-arbre de droite et enfin le nœud racine.

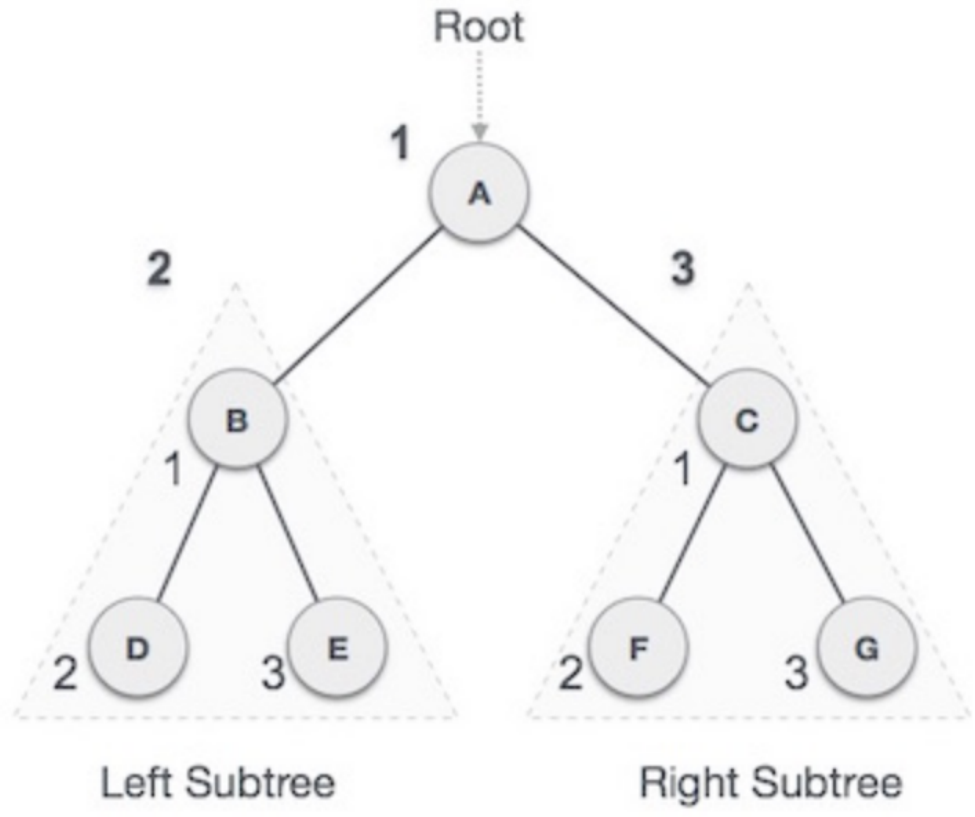


Le parcours préfixe



Qu'obtiendrait-on dans cet exemple ?

Le parcours préfixe



Qu'obtiendrait-on dans cet exemple ?

$D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$

Ébauche d'algorithme

Algorithme

Tant que tout les les noeuds ne sont pas parcouru :

1. Étape 1 - Parcourir récursivement le sous-arbre de gauche.
2. Étape 2 - Parcourez récursivement le sous-arbre de droite.
3. Étape 3 - Visite du noeud racine.