Structure de données

Master 1

Année (optionel)

© 2025 Caplogy. All rights reserved.

Plan du cour

© 2025 Caplogy. All rights reserved.

Sommaire

© 2025 Caplogy. All rights reserved.

Plan du cour

© 2025 Caplogy. All rights reserved.

Introduction aux structures de données

code Qu'est-ce qu'une structure de données ?

Une manière organisée de stocker et manipuler des données pour un accès et des modifications efficaces.

rocket Pourquoi sont-elles essentielles ?

- Optimisation des performances
- Gestion efficace de la mémoire
- Résolution de problèmes complexes
- Réutilisation du code

exclamation-triangle Importance

Le choix de la structure impacte directement l'efficacité des algorithmes !

© 2025 Caplogy. All rights reserved.

Java vs Python : Philosophies différentes

java Java

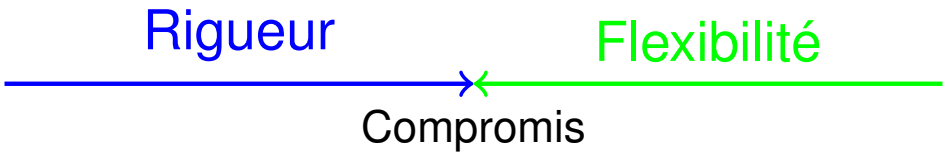
- **Typage statique**
- Compilé $\rightarrow$ JVM
- Orienté objet strict
- Performance élevée
- Verbosité contrôlée

« Write once, run anywhere »

python Python

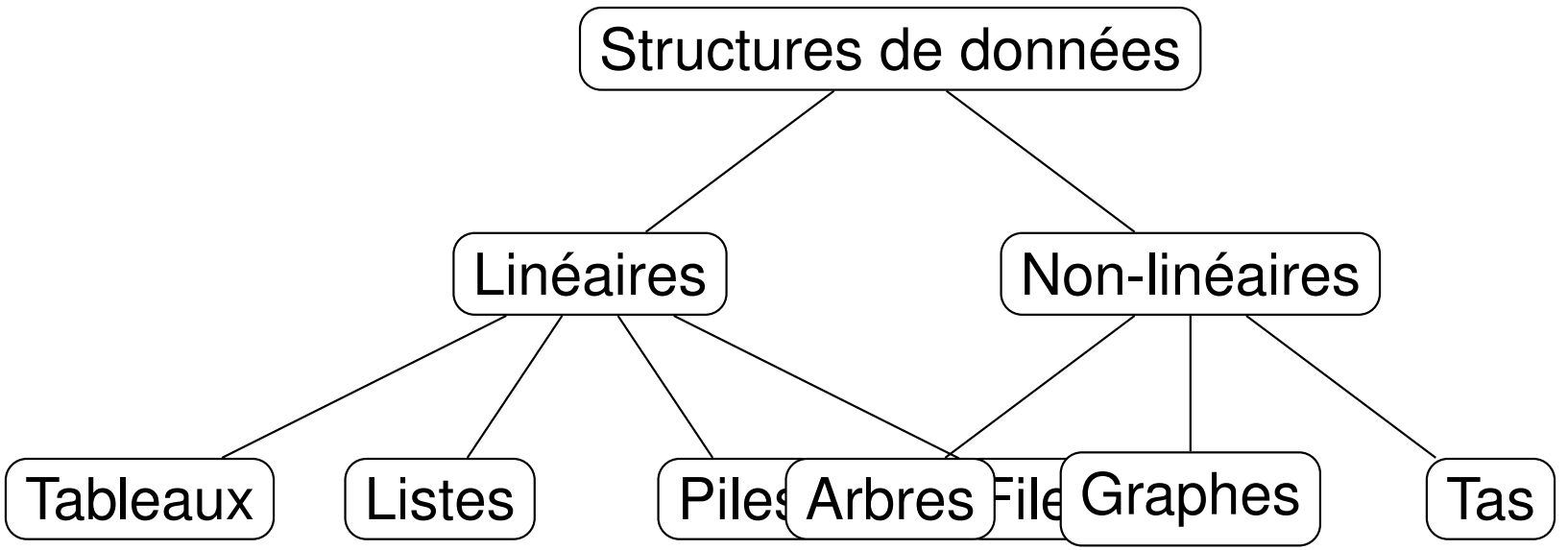
- **Typage dynamique**
- Interprété
- Multi-paradigme
- Syntaxe concise
- Productivité

« Readability counts »



© 2025 Caplogy. All rights reserved.

Classification des structures de données



© 2025 Caplogy. All rights reserved.

Plan du cour

© 2025 Caplogy. All rights reserved.

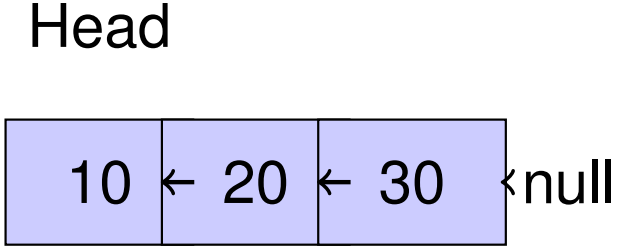
Listes chaînées : Concepts fondamentaux

Définition

Structure séquentielle où chaque élément (nœud) contient des données et une référence vers l'élément suivant.

Types principaux :

- **Simple** : arrow-right Unidirectionnelle
- **Double** : arrows-alt-h Bidirectionnelle
- **Circulaire** : sync Bouclée



© 2025 Caplogy. All rights reserved.

Implémentation Java d'une liste chaînée

```
// D finition du n ud
class ListNode {
    int data;
    ListNode next;

    ListNode(int data) {
        this.data = data;
        this.next = null;
    }
}

// Liste cha n e simple
```

© 2025 Caplogy. All rights reserved.

Implémentation Java d'une liste chaînée

```
// Liste cha n e simple
class LinkedList {
    private ListNode head;

    public void add(int data) {
        ListNode newNode = new ListNode(data);
        if (head == null) {
            head = newNode;
        } else {
            ListNode current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }
    }
    ...
}
```

© 2025 Caplogy. All rights reserved.

Implémentation Java d'une liste chaînée

```
// Liste cha n e simple
class LinkedList {
    ..
    public void display() {
        ListNode current = head;
        while (current != null) {
            System.out.print(current.data + " -> ");
            current = current.next;
        }
        System.out.println("null");
    }
}
```

© 2025 Caplogy. All rights reserved.

Implémentation Python d'une liste chaînée

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node
```

© 2025 Caplogy. All rights reserved.

Implémentation Python d'une liste chaînée

```
class LinkedList:
    ...
    def display(self):
        current = self.head
        elements = []
        while current:
            elements.append(str(current.data))
            current = current.next
        print(" -> ".join(elements) + " -> None")

    # Utilisation
    ll = LinkedList()
    ll.append(10); ll.append(20); ll.append(30)
    ll.display()  # 10 -> 20 -> 30 -> None
```

© 2025 Caplogy. All rights reserved.

Avantages et limites des listes chaînées

thumbs-up Avantages

- Insertion/suppression rapides : O(1)
- Taille dynamique
- Pas de gaspillage mémoire
- Flexibilité structurelle

thumbs-down Limites

- Accès séquentiel : O(n)
- Mémoire supplémentaire (pointeurs)
- Pas de localité mémoire
- Complexité algorithmique

© 2025 Caplogy. All rights reserved.

Avantages et limites des listes chaînées

Opération	Tableau	Liste chaînée
Accès	O(1)	O(n)
Insertion début	O(n)	O(1)
Insertion fin	O(1)	O(1)*
Suppression début	O(n)	O(1)

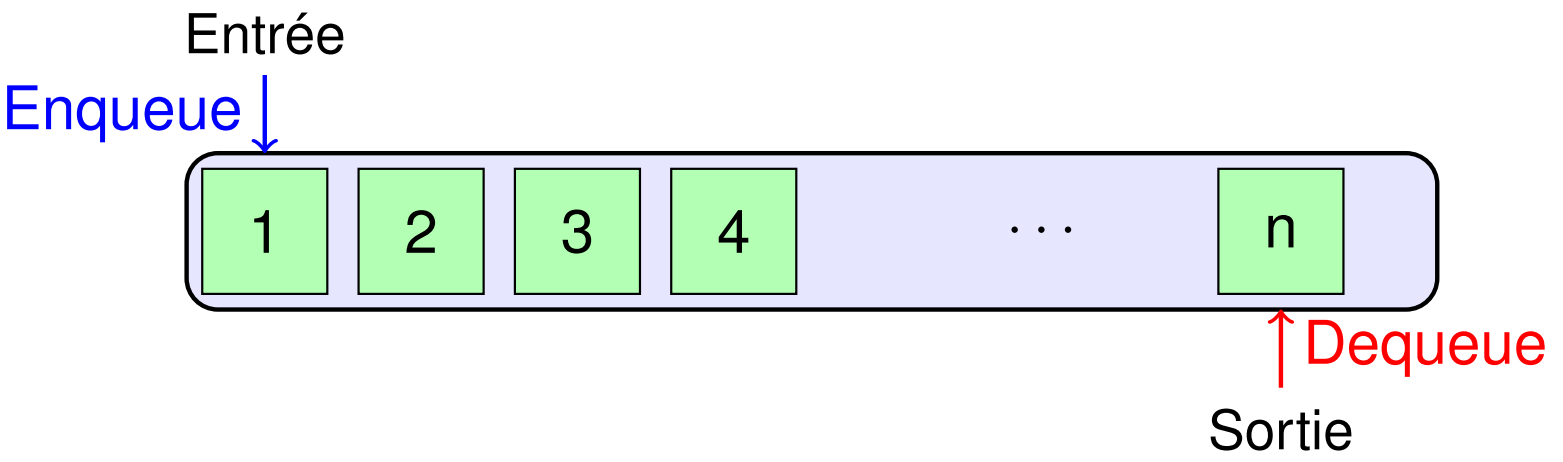
© 2025 Caplogy. All rights reserved.

Plan du cour

© 2025 Caplogy. All rights reserved.

Files : Principe FIFO

FIFO : First In, First Out



© 2025 Caplogy. All rights reserved.

Plan du cour

© 2025 Caplogy. All rights reserved.

Files : Principe FIFO

users Cas d'usage réels

- Files d'attente de processus
- Traitement de requêtes web
- Systèmes de messagerie
- Impression de documents

© 2025 Caplogy. All rights reserved.

Implémentation des files en Java

```
import java.util.LinkedList;
import java.util.Queue;
public class QueueExample {
    public static void main(String[] args) {
        // Cr ation d'une file
        Queue<Integer> queue = new LinkedList<>();

        // Enfiler des   lments   (enqueue)
        queue.offer(10);
        queue.offer(20);
        queue.offer(30);

        System.out.println("File: " + queue);
        System.out.println("T te: " + queue.peek());
        // D filer (dequeue)
        ...
    }
}
```

© 2025 Caplogy. All rights reserved.

Implémentation des files en Java

```
import java.util.LinkedList;
import java.util.Queue;
public class QueueExample {
    public static void main(String[] args) {
        ...
        // D filer (dequeue)
        System.out.println("  lment   retir : " + queue
            .poll());
        System.out.println("File apr s retrait: " +
            queue);

        // Parcours
        while (!queue.isEmpty()) {
            System.out.println("Traitement: " + queue.
                poll());
        }
    }
}
```

© 2025 Caplogy. All rights reserved.

Implémentation des files en Python

```
from collections import deque

# Cr ation d'une file
queue = deque()

# Enfiler des   lments
queue.append(10)
queue.append(20)
queue.append(30)

print(f"File: {list(queue)}")
print(f"T te: {queue[0]}")

# D filer
element = queue.popleft()
print(f"  lment   retir : {element}")
print(f"File apr s retrait: {list(queue)}")
...
```

© 2025 Caplogy. All rights reserved.

Implémentation des files en Python

```
# File avec taille limit e
bounded_queue = deque(maxlen=3)
bounded_queue.append(1)
bounded_queue.append(2)
bounded_queue.append(3)
bounded_queue.append(4)  # D place la file
print(f"File born e: {list(bounded_queue)}")  # [2, 3,
    4]

# Utilisation comme file prioritaire
import heapq
priority_queue = []
heapq.heappush(priority_queue, (2, "t che moyenne"))
heapq.heappush(priority_queue, (1, "t che urgente"))
print(heapq.heappop(priority_queue))  # (1, "t che
    urgente")
```

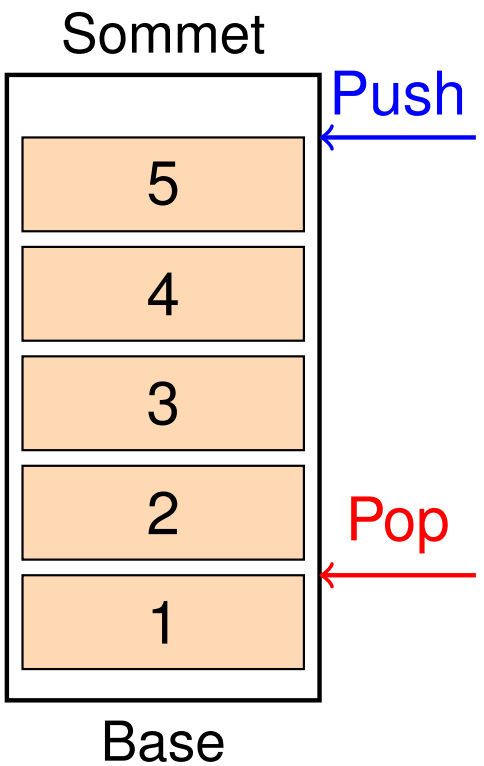
© 2025 Caplogy. All rights reserved.

Plan du cour

© 2025 Caplogy. All rights reserved.

Piles : Principe LIFO

LIFO : Last In, First Out



© 2025 Caplogy. All rights reserved.

Plan du cour

© 2025 Caplogy. All rights reserved.

Piles : Principe LIFO

layer-group Applications typiques

- Gestion de la mémoire (call stack)
- Annulation/refaire (undo/redo)
- Évaluation d'expressions
- Navigation navigateur

© 2025 Caplogy. All rights reserved.

Vérification de parenthèses équilibrées

```
def is_balanced(expression):
    stack = []
    matching = {')': '(', ']': '[', '}': '{'}

    for char in expression:
        if char in '([{':
            # Empiler les ouvrantes
            stack.append(char)
        elif char in ')]}':
            # V rifier la correspondance
            if not stack or stack.pop() != matching[char
                ]:
                return False

    # La pile doit  tre  vide    la fin
    return len(stack) == 0

# Tests
```

© 2025 Caplogy. All rights reserved.

Implémentation des piles en Java et Python

```java
// Java - Stack class
import java.util.Stack;

Stack<Integer> stack = new
    Stack<>();
stack.push(10);          //
    Empiler
stack.push(20);
stack.push(30);

System.out.println(stack.
    peek()); // 30 (sommet)

while (!stack.isEmpty()) {
    System.out.println(
        stack.pop());
    // 30, 20, 10
}
```

```python
# Python - List comme pile
stack = []
stack.append(10)   # Empiler
stack.append(20)
stack.append(30)

print(stack[-1])   # 30 (
    sommet)

while stack:
    print(stack.pop())
    # 30, 20, 10

# Python - deque (optimis
    )
```

© 2025 Caplogy. All rights reserved.

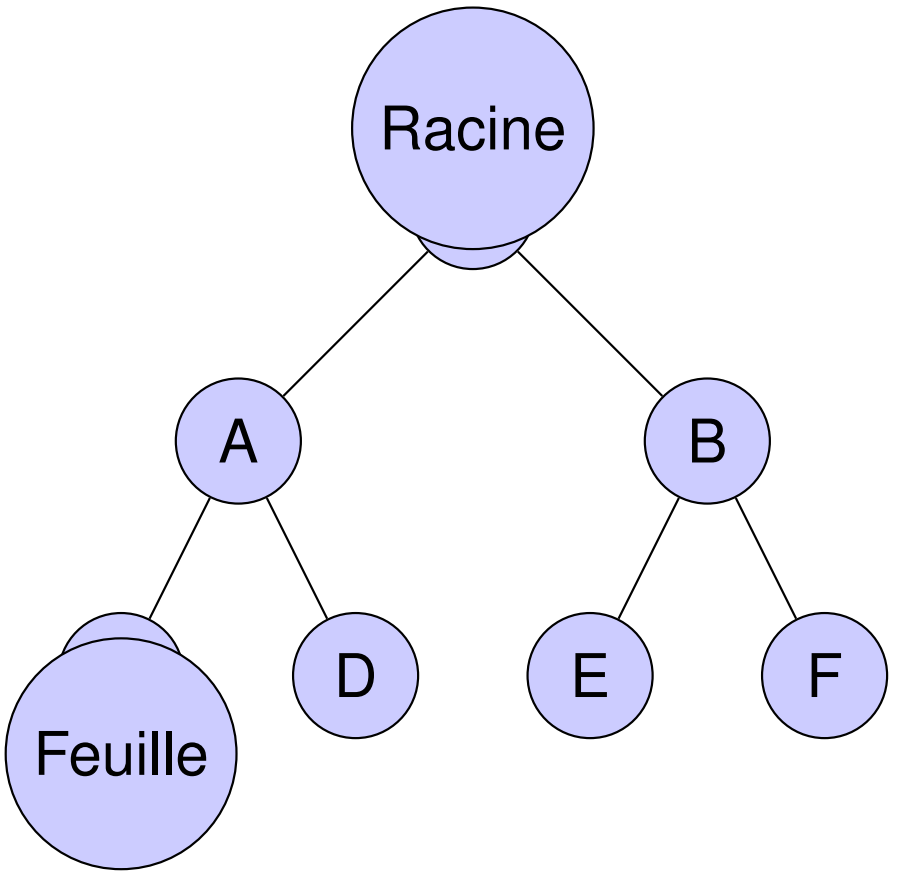
Plan du cour

© 2025 Caplogy. All rights reserved.

Arbres : Concepts fondamentaux

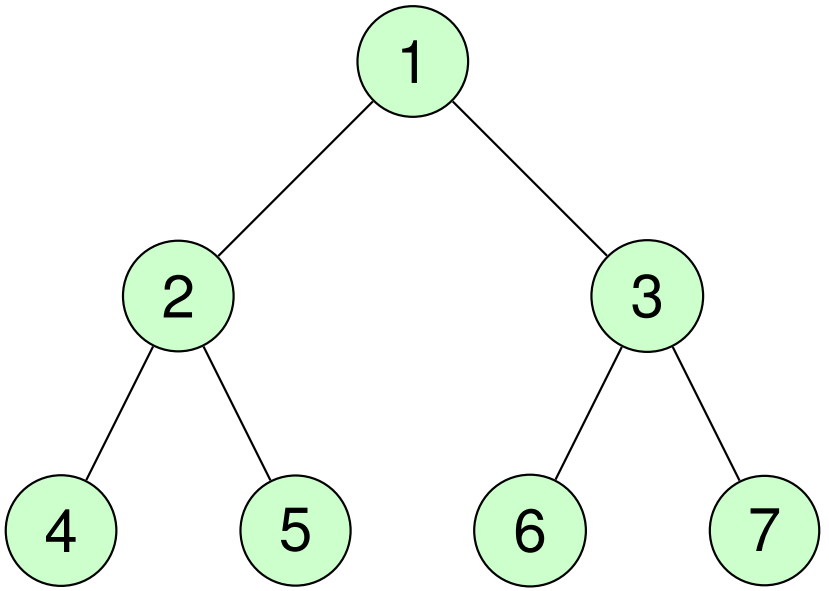
Terminologie :

- **Racine** : Nœud supérieur
- **Feuille** : Nœud sans enfants
- **Nœud interne** : Avec enfants
- **Profondeur** : Distance à la racine
- **Hauteur** : Profondeur maximale
- **Degré** : Nombre d'enfants



© 2025 Caplogy. All rights reserved.

Types de parcours d'arbres



Préfixe (RGD)
1, 2, 4, 5, 3, 6, 7

Infixe (GDR)
4, 2, 5, 1, 6, 3, 7

Postfixe (GDR)
4, 5, 2, 6, 7, 3, 1

© 2025 Caplogy. All rights reserved.

Arbre binaire de recherche (BST) en Python

```
class BSTNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

class BinarySearchTree:
    def __init__(self):
        self.root = None

    def insert(self, value):
        self.root = self._insert(self.root, value)

    def _insert(self, node, value):
        if node is None:
            return BSTNode(value)
```

© 2025 Caplogy. All rights reserved.

Arbre binaire de recherche (BST) en Python

```
        if value < node.value:
            node.left = self._insert(node.left, value)
        else:
            node.right = self._insert(node.right, value)

        return node

    def search(self, value):
        return self._search(self.root, value)

    def _search(self, node, value):
        if node is None or node.value == value:
            return node is not None

        if value < node.value:
            return self._search(node.left, value)
        else:
```

© 2025 Caplogy. All rights reserved.

Arbre binaire de recherche (BST) en Python

```
    def inorder_traversal(self):
        result = []
        self._inorder(self.root, result)
        return result

    def _inorder(self, node, result):
        if node:
            self._inorder(node.left, result)
            result.append(node.value)
            self._inorder(node.right, result)

# Utilisation
bst = BinarySearchTree()
for val in [50, 30, 70, 20, 40, 60, 80]:
    bst.insert(val)
print("Parcours infixe:", bst.inorder_traversal())
# [20, 30, 40, 50, 60, 70, 80]
```

© 2025 Caplogy. All rights reserved

Arbre binaire en Java

```
class TreeNode {
    int value;
    TreeNode left;
    TreeNode right;

    TreeNode(int value) {
        this.value = value;
        this.left = null;
        this.right = null;
    }
}

public class BinaryTree {
    TreeNode root;

    public void insert(int value) {
        root = insertRec(root, value);
    }
```

© 2025 Caplogy. All rights reserved.

Plan du cour

© 2025 Caplogy. All rights reserved.

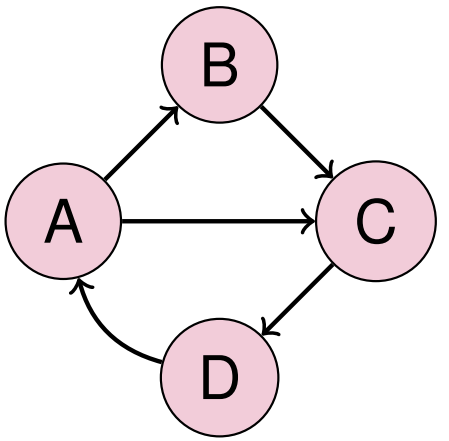
Graphes : Définitions et concepts

Définition

Ensemble de nœuds (sommets) connectés par des arêtes (liens).

Types de graphes :

- **Orienté** : Arêtes directionnelles
- **Non-orienté** : Arêtes bidirectionnelles
- **Pondéré** : Arêtes avec poids
- **Non-pondéré** : Arêtes sans poids



Graphe orienté

© 2025 Caplogy. All rights reserved.

Représentations des graphes

Liste d'adjacence

$A : [B, C]$
$B : [C]$
$C : [D]$
$D : [A]$

Avantages :

- Mémoire efficace
- Parcours voisins rapide

Matrice d'adjacence

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Avantages :

- Test lien rapide : O(1)
- Simple à implémenter

Opération	Liste	Matrice
Espace mémoire	O(V+E)	$O(V^2)$

© 2025 Caplogy. All rights reserved.

Parcours BFS (Largeur d'abord)

```
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)
    result = []

    while queue:
        node = queue.popleft()
        result.append(node)

        # Visiter tous les voisins non visit s
        for neighbor in graph.get(node, []):
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)
```

© 2025 Caplogy. All rights reserved.

Parcours DFS (Profondeur d'abord)

```
def dfs_iterative(graph, start):
    visited = set()
    stack = [start]
    result = []

    while stack:
        node = stack.pop()
        if node not in visited:
            visited.add(node)
            result.append(node)
            # Ajouter les voisins dans l'ordre inverse
            # pour un parcours coh rent
            for neighbor in reversed(graph.get(node, [])
                ):
                if neighbor not in visited:
                    stack.append(neighbor)

    return result
```

© 2025 Caplogy. All rights reserved.

Comparaison des performances

Structure	Accès	Recherche	Insertion	Suppression
Tableau	O(1)	O(n)	O(n)	O(n)
Liste chaînée	O(n)	O(n)	O(1)*	O(1)*
Pile	O(1)	O(n)	O(1)	O(1)
File	O(1)	O(n)	O(1)	O(1)
Arbre BST	O(log n)	O(log n)	O(log n)	O(log n)
Table de hachage	O(1)	O(1)	O(1)	O(1)

Complexités temporelles moyennes (cas optimal)

© 2025 Caplogy. All rights reserved.

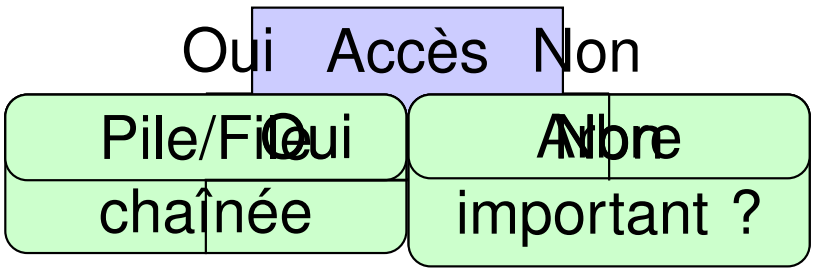
Comparaison des performances

balance-scale Compromis à considérer

- **Mémoire vs Performance**
- **Fréquence d'accès vs Modification**
- **Simplicité vs Fonctionnalités**
- **Ordre vs Rapidité**

© 2025 Caplogy. All rights reserved.

Guide de sélection des structures



© 2025 Caplogy. All rights reserved.

Structures avancées et perspectives

chevron-up Structures avancées

- **Tas (Heap)** : Files de priorité
- **Trie** : Recherche de préfixes
- **Arbres AVL/rouge-noir** : Auto-équilibrage
- **Graphes** : Réseaux complexes
- **Tables de hachage** : Dictionnaires

lightbulb Bonnes pratiques

- Comprendre les cas d'usage
- Analyser les compromis
- Profiler les performances
- Utiliser les bibliothèques standard
- Documenter les choix

© 2025 Caplogy. All rights reserved.

Structures avancées et perspectives

book Pour aller plus loin

- « **Introduction to Algorithms** » (Cormen, Leiserson, Rivest, Stein)
- « **Data Structures and Algorithms in Java/Python** »
- **Plateformes :** LeetCode, HackerRank pour la pratique

© 2025 Caplogy. All rights reserved.

Questions ?

Merci pour votre attention !

© 2025 Caplogy. All rights reserved.