

STRUCTURES DE DONNEES

Michael
X  NATIS

A quoi servent les structures de données ?



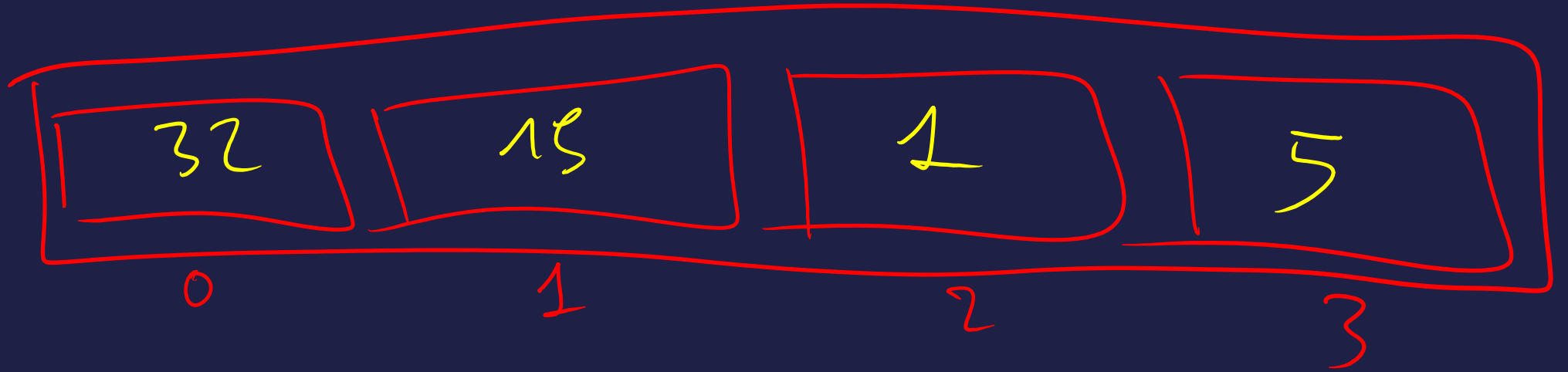


Compétence demandée :
Comprendre l'organisation des structures de
données du cours

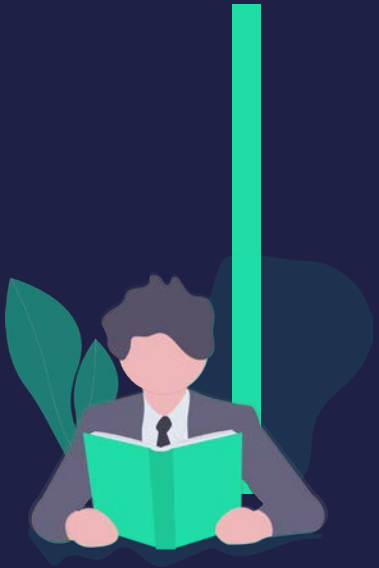
1. Tableau
2. Liste chaînée
3. Queue
4. Stack
5. Arbre binaire

1. Tableau
2. Liste chaînée
3. Queue
4. Stack
5. Arbre binaire

Tableau



En informatique, une structure de données de tableau, ou simplement un tableau, est une structure de données constituée d'un ensemble d'éléments (valeurs ou variables), chacun identifié par au moins un indice de tableau. Un tableau a une taille fixe.



Le type de structure de données le plus simple est un tableau linéaire, également appelé tableau à une dimension.

Pourquoi utiliser un tableau ?



Comment fait-on ?

- Recherche d'un élément par sa position
- Recherche d'un élément par sa valeur (recherche linéaire)
- Ajouter un élément



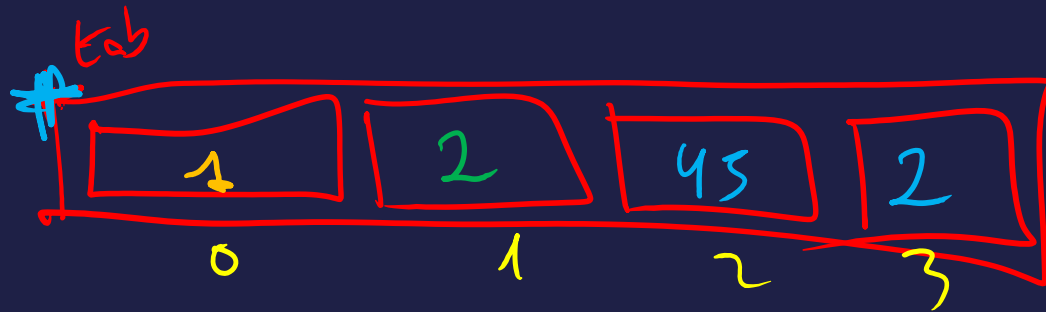
<https://www.oracle.com/java/technologies/downloads/#jdk17-windows>

tableau de int

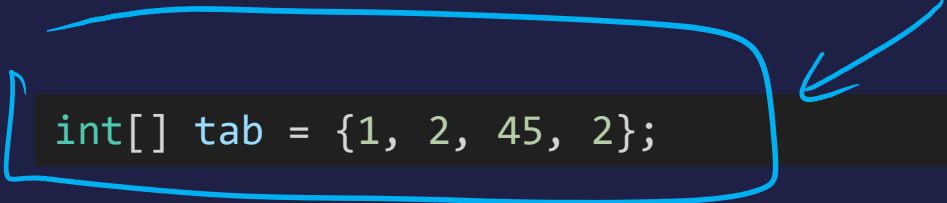
allouer

tableau de 4 int

```
1 int[] tab = new int[4];  
2 tab[0] = 1;  
3 tab[1] = 2;  
tab[2] = 45;  
tab[3] = 2;
```



```
int[] tab = {1, 2, 45, 2};
```



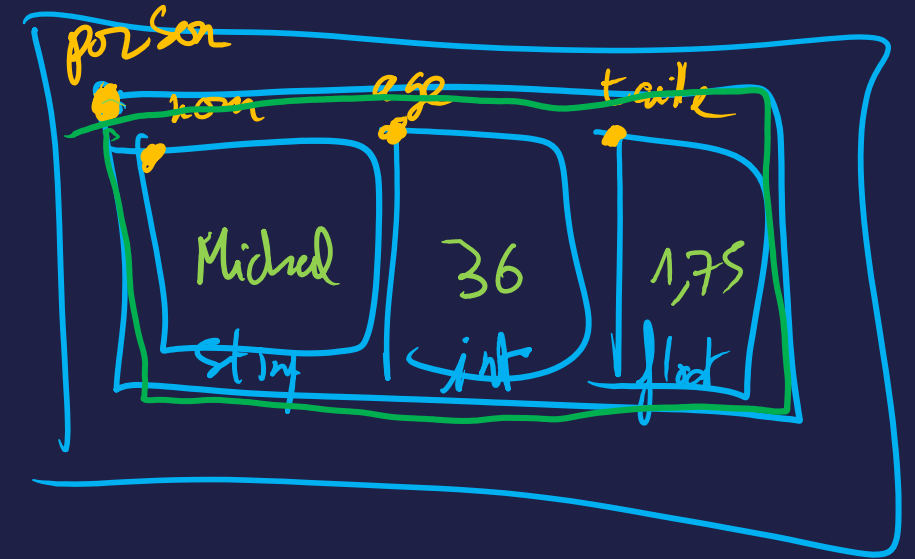
```
int tab[] = {1, 2, 45, 2};
```

```
int[] tab = new int[4];  
tab[0] = 1;  
tab[1] = 2;  
tab[2] = 45;  
tab[3] = 2;
```


Classe

Une classe, hors cadre de la POO, est un ensemble personnalisé de types existants décrivant un agencement de données. Elle permet donc de décrire une distribution des types dans un espace mémoire.

```
public class Person {  
    String nom;  
    int age;  
    float taille;  
}
```



class String {
 int taille;
 char[]

```
public class Pgm {  
    public static void main(String[] args) {  
        Person person = new Person();  
        person.nom = "Michael";  
        person.age = 36;  
        person.taille = 1.75f;  
  
        System.out.println(person.nom);  
    }  
}
```

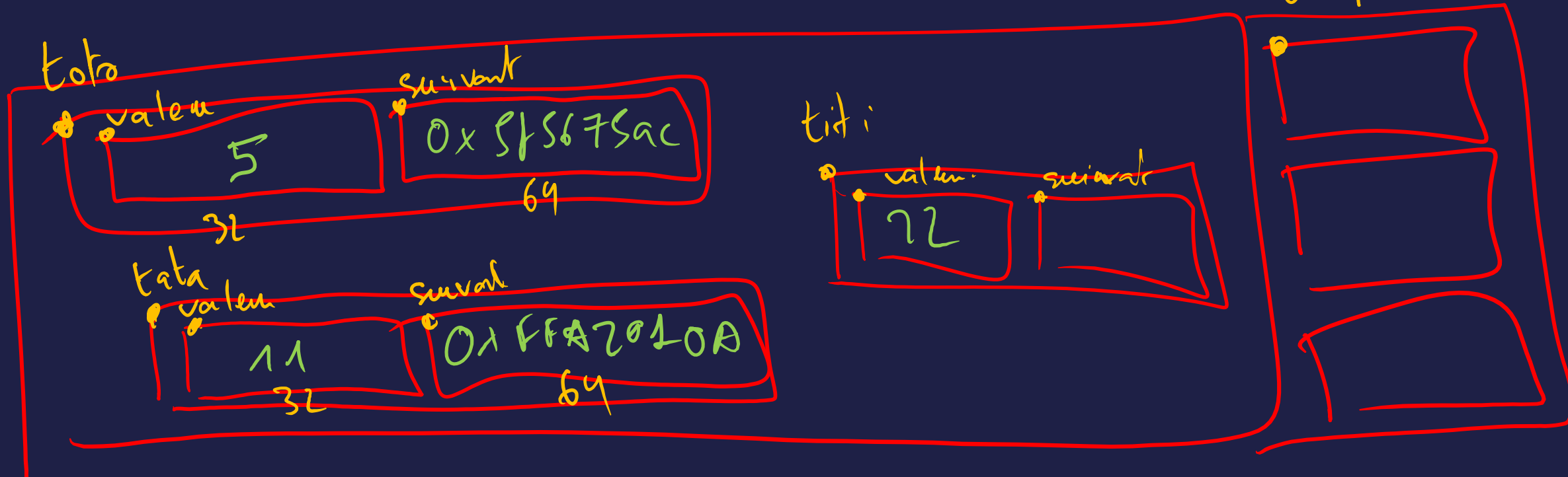
TIPS : posez-vous les questions dans cet ordre :

- types primitifs (int, long, float, boolean, char)
- tableau (pour une liste d'éléments de même type)
- classe (pour le reste)

1. Tableau
2. Liste chaînée
3. Queue
4. Stack
5. Arbre binaire

Liste chaînée

```
class Element {  
    int valeur;  
    Element suivant;  
}
```



$5 \rightarrow 11 \rightarrow 22 \rightarrow 33 \rightarrow 44 \rightarrow 2$
└──────────┘

$5 \rightarrow 6 \rightarrow 11 \rightarrow 22 \rightarrow 33 \rightarrow 44 \rightarrow 2$

Une liste chaînée désigne en informatique une structure de données représentant une collection ordonnée et de taille arbitraire d'éléments de même type, dont la représentation en mémoire de l'ordinateur est une succession de cellules faites d'un contenu et d'un pointeur vers une autre cellule.

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

Pourquoi utiliser une liste chaînée ?



Comment fait-on ?

- Recherche d'un élément par sa position
- Recherche d'un élément par sa valeur (recherche linéaire)
- Ajouter un élément




```
class linkedlist < T > {  
    T element;  
    linkedlist < T > servant;  
    void add ( T value ) ;  
}
```

```
import java.util.LinkedList;

public class Pgm {
    public static void main(String[] args) {
        LinkedList<String> cars = new LinkedList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        System.out.println(cars);
    }
}
```

1. Tableau
2. Liste chaînée
3. Queue
4. Stack
5. Arbre binaire

File / Queue - FIFO

En informatique, une file est un ensemble d'entités qui sont maintenues dans une séquence et peuvent être modifiées par l'ajout d'entités à une extrémité de la séquence et la suppression d'entités à l'autre extrémité de la séquence (mécanisme FIFO).



Par convention, la fin de la séquence à laquelle les éléments sont ajoutés est appelée l'arrière, la queue, et la fin à laquelle les éléments sont supprimés est appelée la tête

Pourquoi utiliser une queue ?




```
import java.util.LinkedList;

public class Pgm {
    public static void main(String args[]){
        LinkedList<String> queue = new LinkedList<String>();
        queue.add("Amit");
        queue.add("Vijay");
        queue.add("Karan");
        queue.add("Jai");
        queue.add("Rahul");
        System.out.println("head: " + queue.element()); // prints "head: Amit"
        System.out.println("head: " + queue.poll()); // prints "head: Amit"
        System.out.println("head: " + queue.element()); // prints "head: Vijay"
    }
}
```

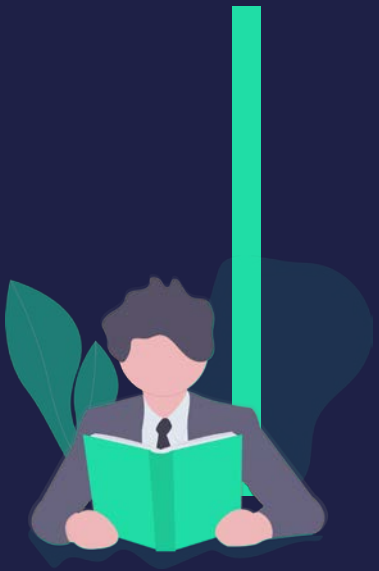
1. Tableau
2. Liste chaînée
3. Queue
4. Stack
5. Arbre binaire

Pile / Stack - LIFO

En informatique, une pile est un type de données abstrait qui sert de collection d'éléments, avec deux opérations principales :

- Push, qui ajoute un élément à la collection, et
- Pop, qui supprime l'élément le plus récemment ajouté qui n'a pas encore été supprimé.

L'ordre dans lequel les éléments sortent d'une pile donne lieu à son nom alternatif, LIFO (dernier entré, premier sorti). De plus, une opération peek ou top peut donner accès au sommet sans modifier la pile.



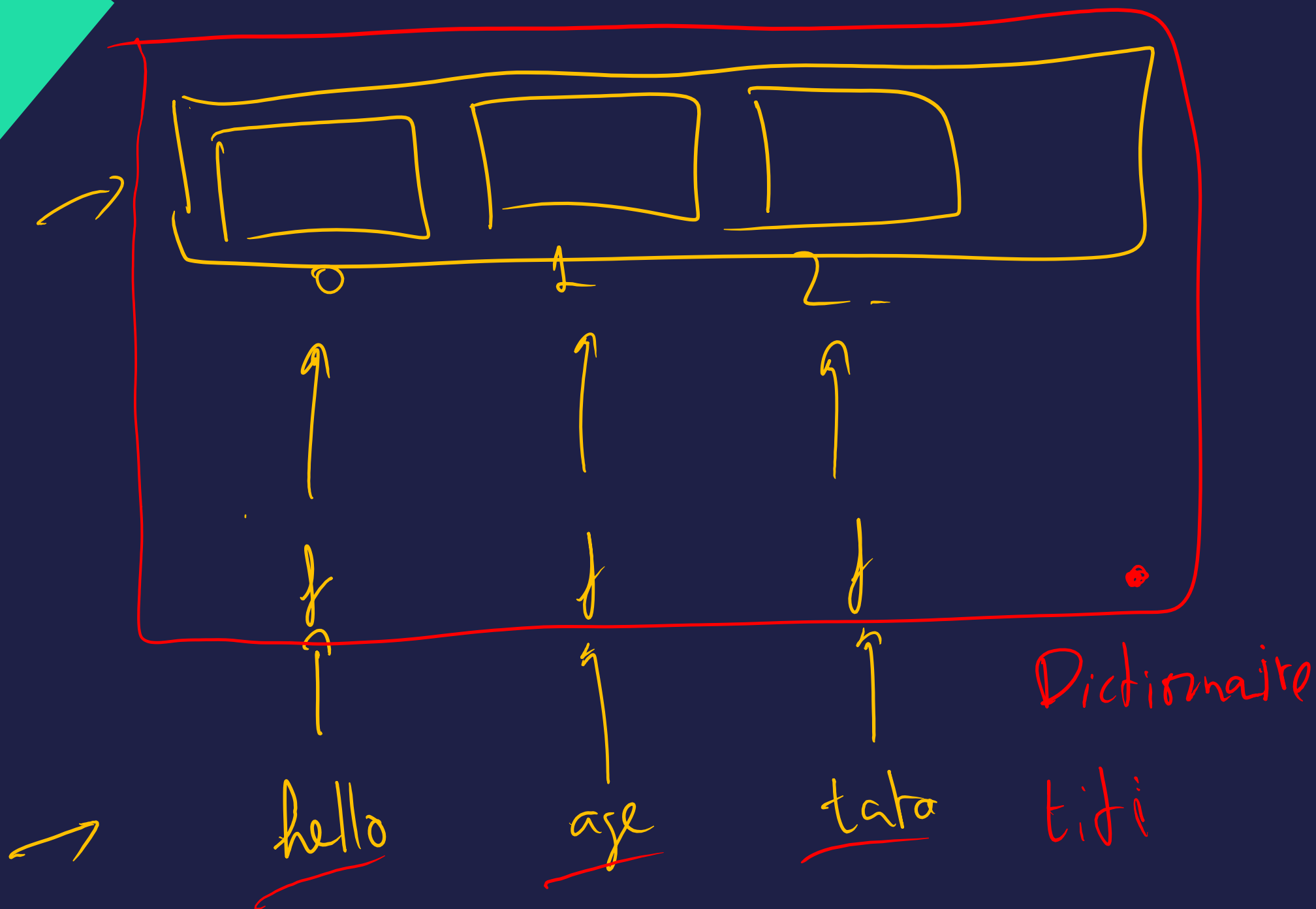
Le nom "pile" pour ce type de structure vient de l'analogie avec un ensemble d'éléments physiques empilés les uns sur les autres. Cette structure permet de retirer facilement un élément du haut de la pile, tandis que pour accéder à un élément plus profond dans la pile, il peut être nécessaire de retirer d'abord plusieurs autres éléments.

Pourquoi utiliser une stack ?




```
import java.util.Stack;

public class Pgm {
    public static void main(String args[]) {
        Stack<Integer> pile = new Stack<Integer>();
        pile.push(10);
        pile.push(15);
        pile.push(20);
        System.out.println("head: " + pile.pop()); // prints "head: 20"
        System.out.println("head: " + pile.peek()); // prints "head: 15"
    }
}
```



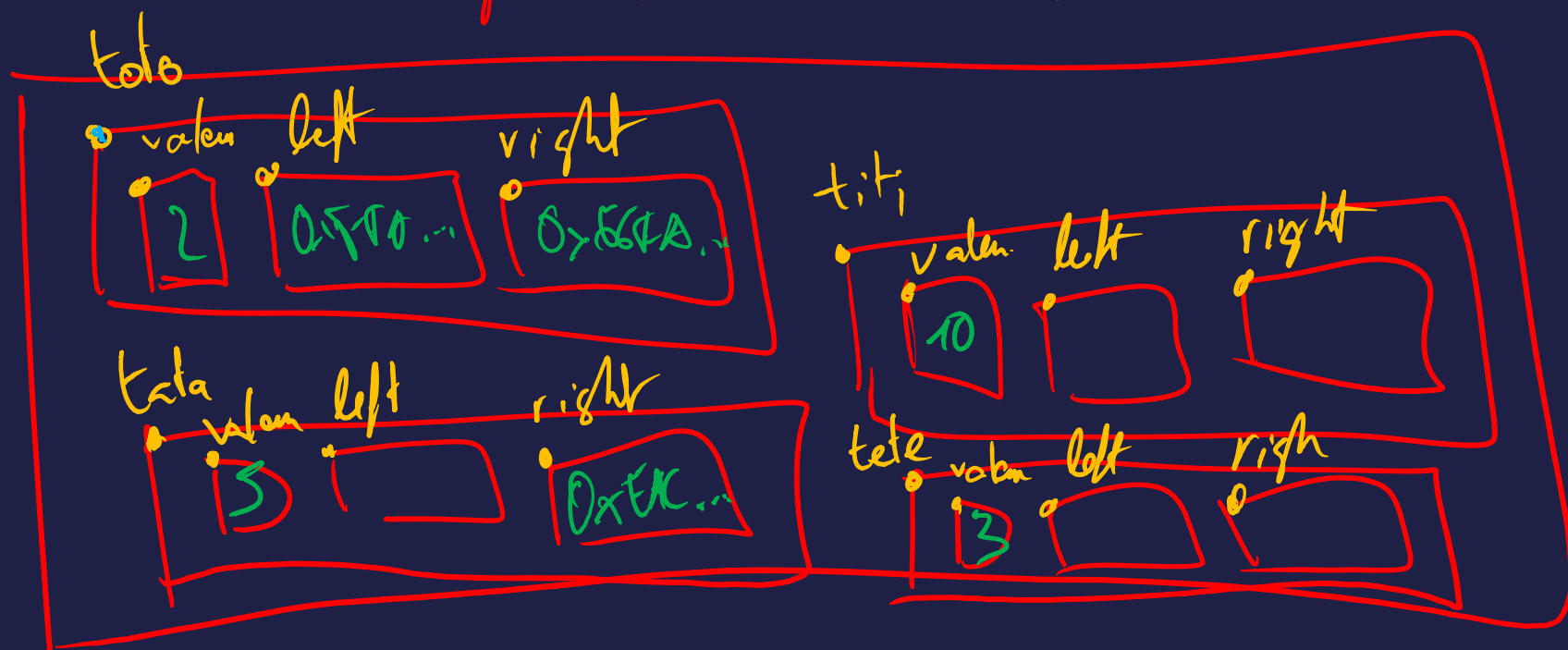
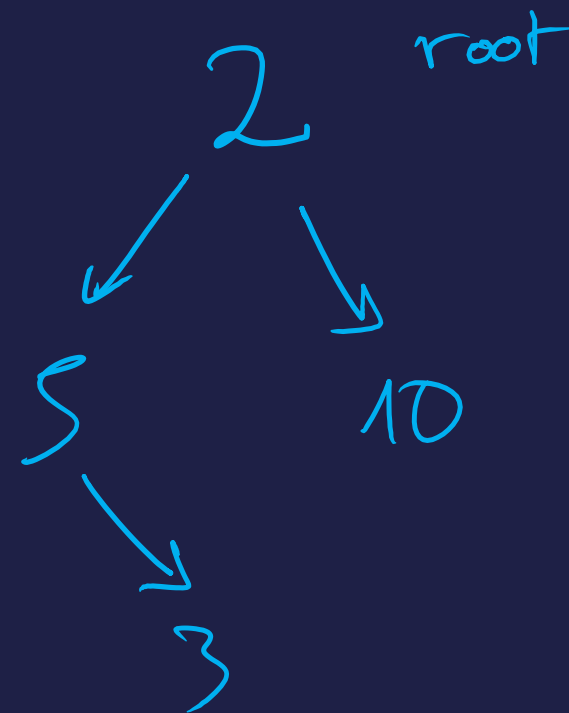
1. Tableau
2. Liste chaînée
3. Queue
4. Stack
5. Arbre binaire

Arbre binaire

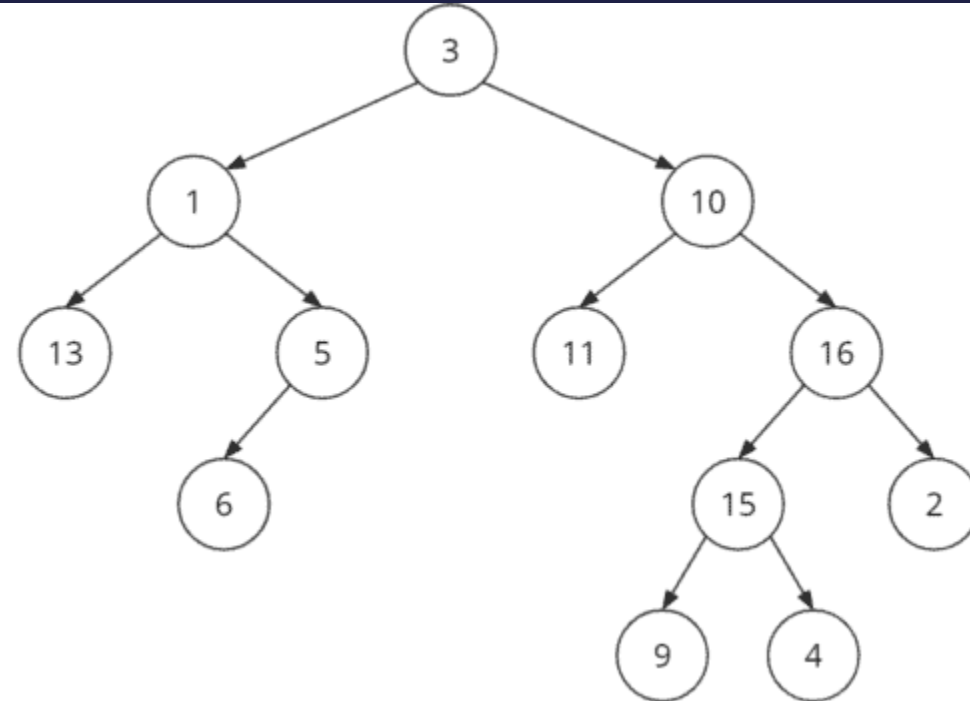

```

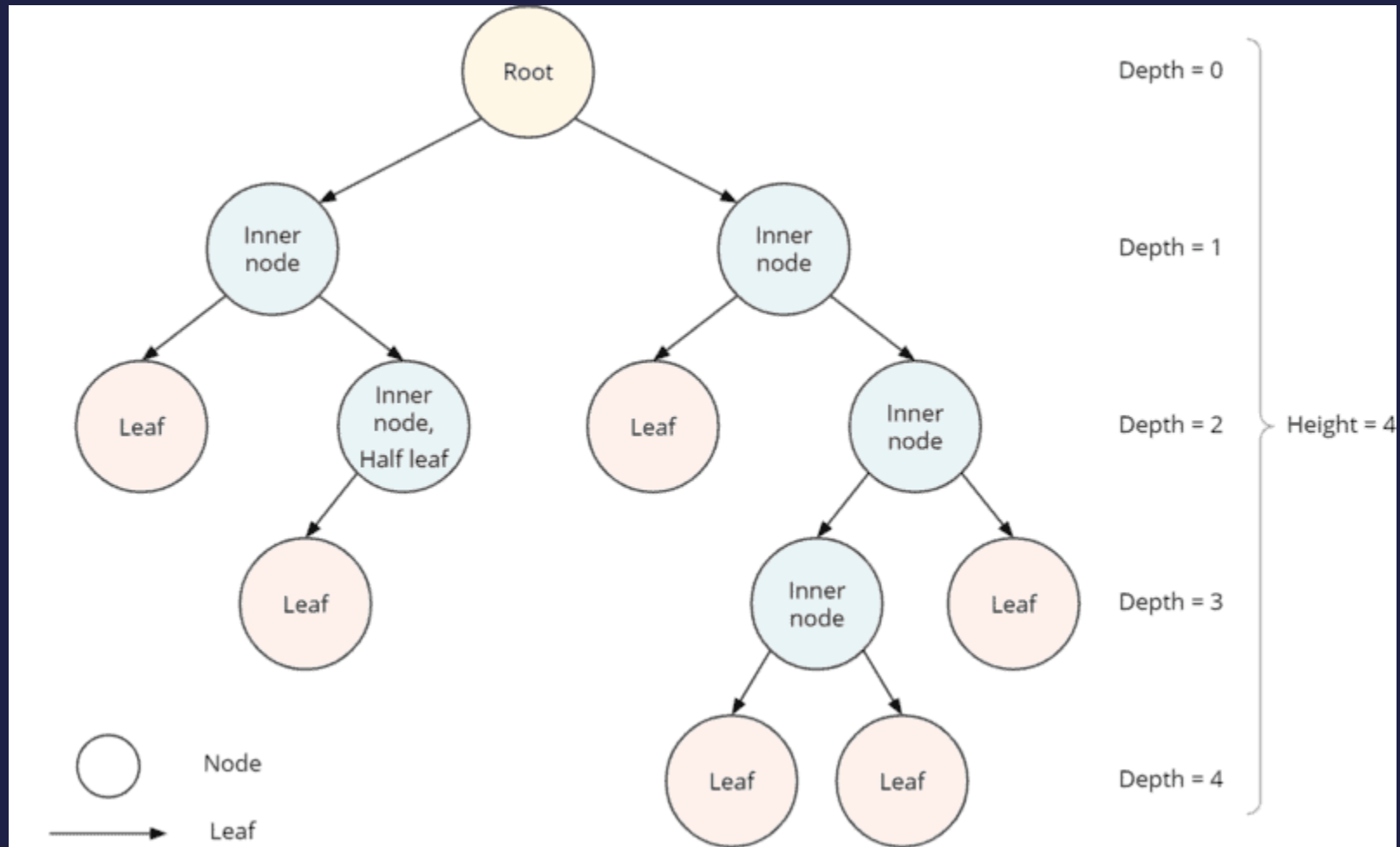
class Node {
    int valeur;
    Node left;
    Node right;
}

```



Un arbre binaire est une structure de données arborescente dans laquelle chaque nœud a au plus deux nœuds enfants. Les nœuds enfants sont appelés enfant gauche et enfant droit.





Un arbre binaire est une structure de données arborescente dans laquelle chaque nœud a au plus deux nœuds enfants. Les nœuds enfants sont appelés enfant gauche et enfant droit.

Pourquoi utiliser un arbre binaire ?

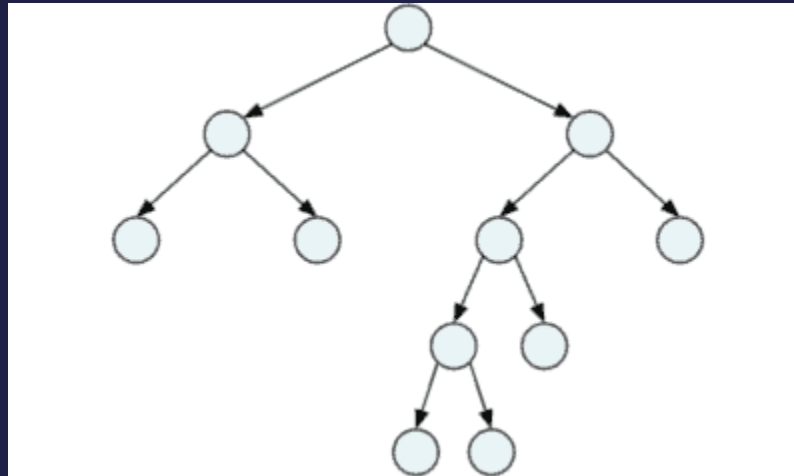


Comment fait-on ?

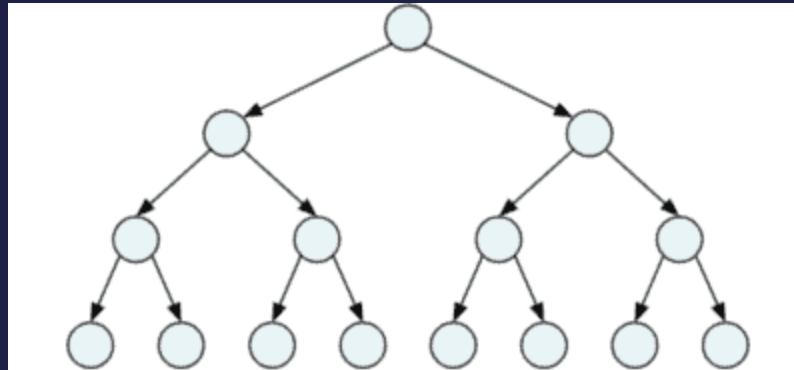
- Parcours en profondeur
- Parcours par niveau
- Recherche d'un élément par sa valeur
- Ajouter un élément



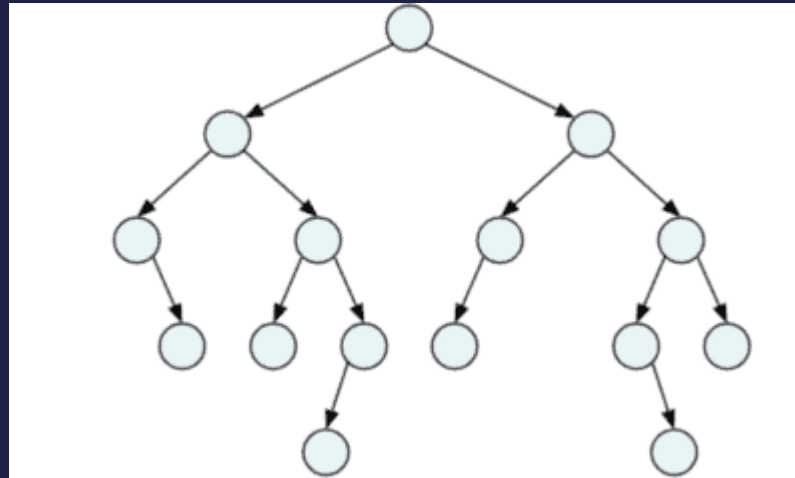
Arbre binaire complet : Dans un arbre binaire complet , tous les nœuds n'ont aucun enfant ou deux enfants.



Arbre binaire parfait : Un arbre binaire parfait est un arbre binaire complet dans lequel toutes les feuilles ont la même profondeur.

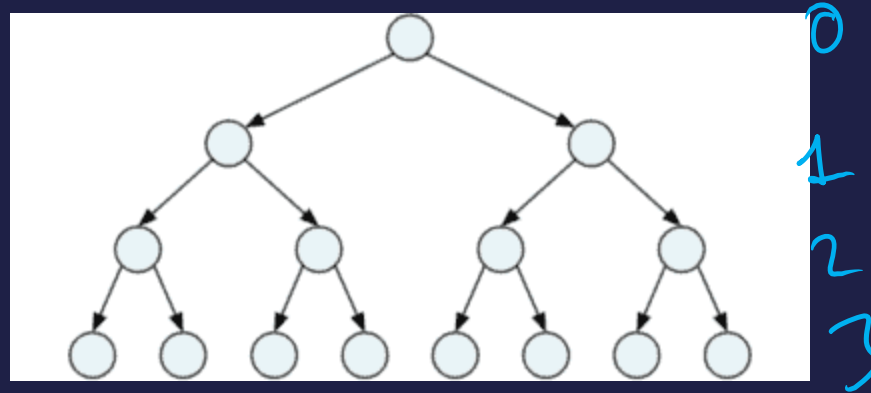


Arbre binaire équilibré : Dans un arbre binaire équilibré , les sous-arbres gauche et droit de chaque nœud diffèrent en hauteur d'au plus un.



Arbre binaire de recherche (BST) : Dans un arbre binaire trié (également appelé arbre binaire ordonné), le sous-arbre gauche d'un nœud ne contient que des valeurs inférieures (ou égales à) la valeur du nœud parent, et le sous-arbre droit ne contient que des valeurs supérieures (ou égales à) la valeur du nœud parent. Une telle structure de données est également appelée arbre de recherche binaire

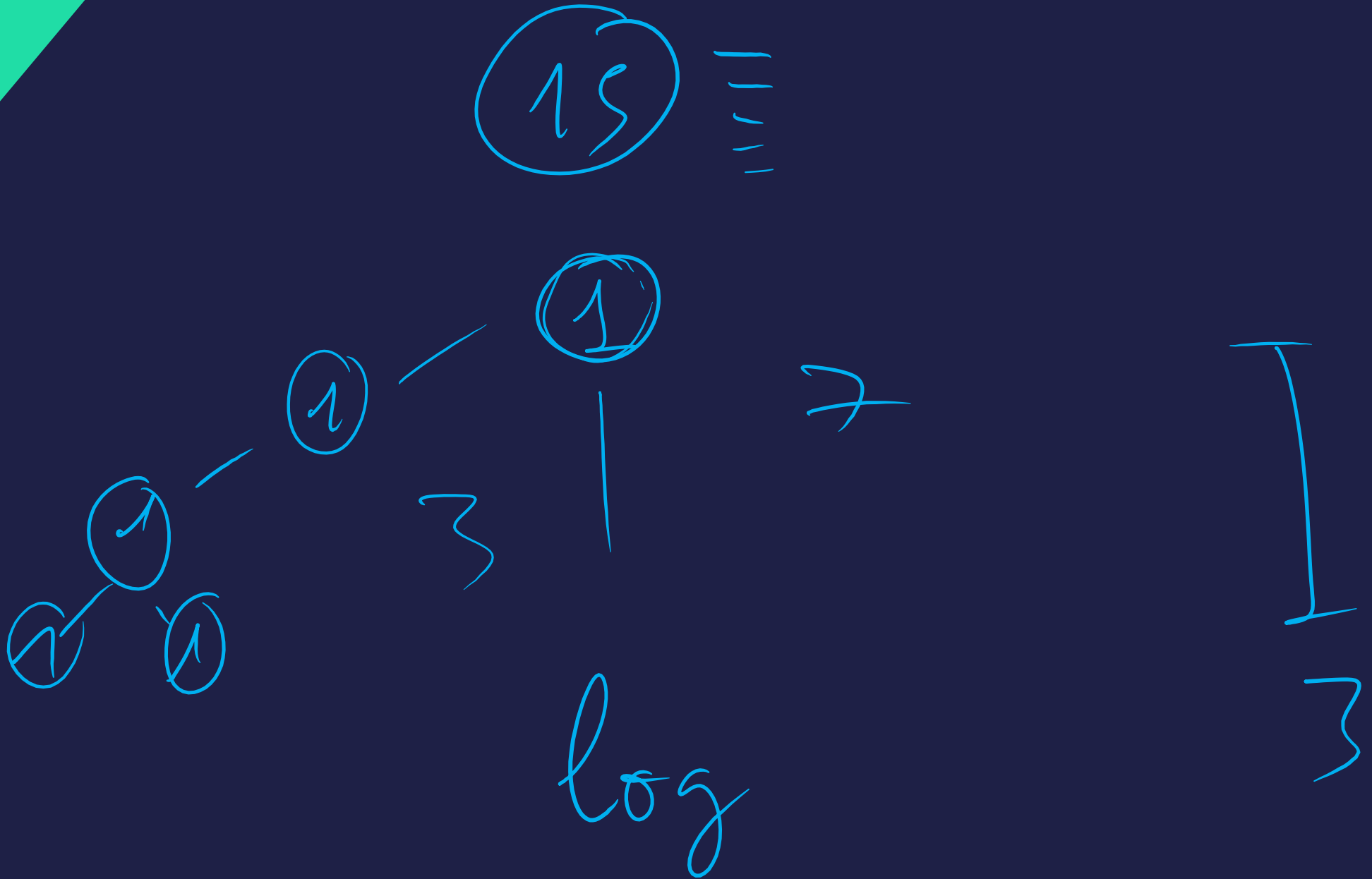
Arbre binaire parfait : Un arbre binaire parfait est un arbre binaire complet dans lequel toutes les feuilles ont la même profondeur.



$15 \rightarrow 3$ $n \rightarrow ?$

Quel est la hauteur d'un arbre
binaire ?





$$h = \log_2(n)$$

La hauteur d'un arbre binaire est considéré étant $\log(n)$, log étant le logarithme de base 2.

```
public class Node {  
    int data;  
    Node left;  
    Node right;  
}
```

```
public class BinaryTree {  
    Node root;  
}
```

Bilan

Pourquoi les utiliser ?



1. Un tableau fournit un accès direct aux éléments dans ce tableau, mais la taille est fixe
2. Une pile fournit un mécanisme de pile, mais pas d'adressage direct
3. Une file fournit un mécanisme de file, mais pas d'accès direct
4. Un arbre permet une organisation hiérarchique, mais les algorithmes pour traiter cette structure est plus compliqué




DS

Queue = LC + FIFO
 Stack = LC + LIFO

	Array n	Queue n	Stack n	Arbre binaire de recherche (BST) supposé équilibré n
Recherche d'un élément par sa position	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Recherche d'un élément par sa valeur	$O(n)$	$O(n)$	$O(n)$	$O(\log n)$
Ajout d'un nouvel élément	X	$O(1)$	$O(1)$	$O(\log n)$
Suppression d'un élément	X	$O(1)$	$O(1)$	$O(\log n)$

Desain

	Array 	Queue	Stack	Arbre binaire de recherche (BST) supposé équilibré
Recherche d'un élément par sa position	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Recherche d'un élément par sa valeur	$O(n)$	$O(n)$	$O(n)$	$O(\log(n))$
Ajout d'un nouvel élément	Impossible	$O(1)$ si à la fin	$O(1)$ si à la tête	$O(\log(n))$
Suppression d'un élément	Impossible	$O(1)$ si à la tête	$O(1)$ si à la tête	$O(\log(n))$



BONUS : Hashtable



V8 : La vérité ou je mens ?



