Data Structures

algorithms

Sommaire

- Definition
- Structure (enregistrement)
- Les arbres
- Les listes chainés
- Les tableaux
- Les graphes
- Les Piles et les files

Définition

Une structure de données peut être sélectionnée ou conçue pour stocker des données dans le but de les travailler avec divers algorithmes. Chaque structure de données contient des informations sur des valeurs, des données, des relations entre les données et les fonctions qui peuvent être appliquées aux données.

A Retenir

Avoir choisi les bons types de données permet d’avoir un programme

- plus lisible car auto documenté
- plus facile à maintenir
- souvent plus rapide, en tout cas plus facile à optimiser

Citation

“ I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important.

Bad programmers worry about the code.

Good programmers worry about data structures and their relationships. ” — **Linus Torvalds**

Structure (Enregistrement)

DEFINITION

Une structure est un regroupement de plusieurs donnée de manière logique et cohérente pour en faire une nouvelle type de donnée, on a tendance à le nommer *type défini* par l'utilisateur.

- Les données composants la structures sont des **champs**
- Elles sont repérées par des **identificateur de champs**
- Les champs sont souvent accéder par le **dot-notation**

Structure (Enregistrement)

EXEMPLE

modeleVoiture : chaine

Structure (Enregistrement)

EXEMPLE

modeleVoiture : chaine

marqueVoiture : chaine

Structure (Enregistrement)

EXEMPLE

modeleVoiture : chaine

marqueVoiture : chaine

anneeVoiture : chaine

Structure (Enregistrement)

EXEMPLE

modeleVoiture : chaine

marqueVoiture : chaine

anneeVoiture : chaine

Structure (Enregistrement)

EXEMPLE

modeleVoiture : chaine

marqueVoiture : chaine

anneeVoiture : chaine

```
Type Voiture = Structure
        modele : chaine
        marque : chaine
        annee  : Date
FinStructure
```

Structure (Enregistrement)

Quelques exemples d'implémentation

```
struct Voiture {
        String modele ;
        String marque ;
        Date annee
};
```

Langage C

Structure (Enregistrement)

Quelques exemples d'implémentation

```
struct Voiture {
        String modele ;
        String marque ;
        Date annee
};
```

Langage C

```
class Voiture:
        def __init__(self, modele, marque,  annee):
                self.modele = modele
                self.marque = marque
                self.annee = annee
```

Python

TABLEAUX

Un tableau est une structure de donnée linéaire et homogène composé de cellule dont chacune d'elle peut prendre une valeur. Tout tableau est caractérisé par :

son nom, son type de valeur, sa dimension, sa taille

- ***Vecteur***
- ***Matrice***

Vecteur

Un vecteur est un tableau à une ligne c'est-à-dire un tableau à une dimension composé de plusieurs colonnes et chacune d'elle pouvant contenir une valeur. Ces colonnes numérotées à partir de 1

Syntaxe

Const taille = valeur

Type nomvecteur = tableau [1...taille] type valeur

Var nomvariablevecteur = nomvecteur

Vecteur

Exemple : tableau d'entiers de 5 cellules

Const N = 5

Type tab = tableau [1...N] entier

Var T : tab

Matrice

Une matrice est un tableau à deux dimensions c'est-à-dire un tableau à plusieurs lignes et chacune de ces lignes se comporte comme un vecteur.

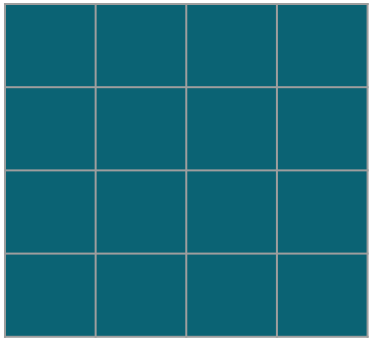
Syntaxe

```
Const Nbligne = valeur

Const Ncolonne = valeur

Type NomMatrice = tableau [1..Nbligne, 1..Ncolonne] type valeur

Var nomVariableMatrice : nomMatrice
```

Matrice

Exemple : matrice d'entier de 4 lignes et 4 colonnes

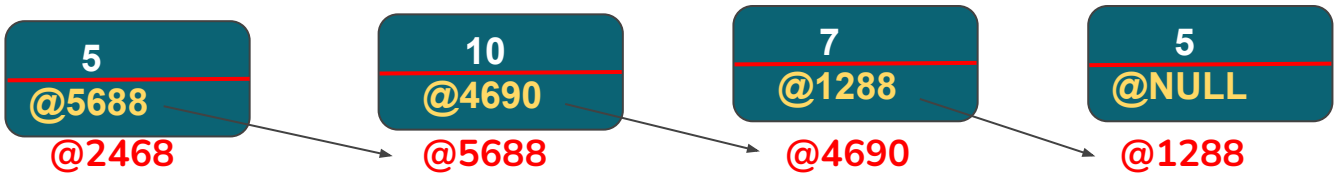
```
Const lignes = 4
Const cols = 4
Type matrice = tableau [1..lignes, 1..cols] entier
Var mat : matrice
```



Listes chaînées

Définition

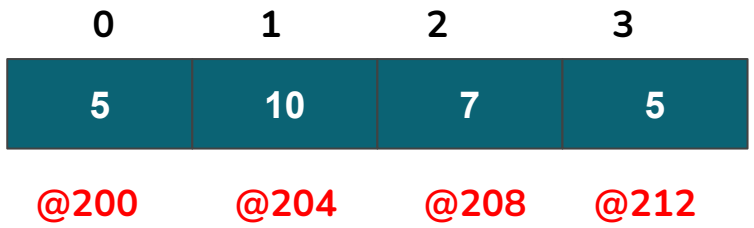
- Une liste chaînée est une collection d'élément de même type
- C'est une succession de maillons, dont le dernier pointe vers une adresse invalide (NULL)
- Le premier élément de la liste est appelé la **Tête** et le dernier élément, la **Queue** de la liste



Listes chaînées

Différences entre Listes chaînées et Tableaux

- Un tableau reste figé alors que ce n'est pas le cas pour une liste chaînée
- Les éléments d'un tableau sont contenus dans la mémoire de manière ordonnée alors que pour une liste chaînée ils y sont de manière dispersée



Listes chaînées

Syntaxe de déclaration d'une liste

Syntaxe:

Type

TypeElem = <type>

Cellule = **Enregistrement**

    Valeur : TypeElem

    Suiv : pointeur sur cellule

FinEnregistrement

Liste = pointeur sur cellule

Algorithme

```
typedef struct Element
Element;
struct Element
{
        int nombre;
        Element *suivant;
};
```

Langage C

Listes chaînées

Syntaxe d'initialisation d'une liste (Null)

```
Procédure Init(var L: Liste)

Debut
        L <- Null
Fin
```

Syntaxe de détermination de la valeur du premier élément

```
Fonction Premiere(L:Liste):TypeElem

Debut
        Premier <- L^.valeur
Fin
```

Arbres

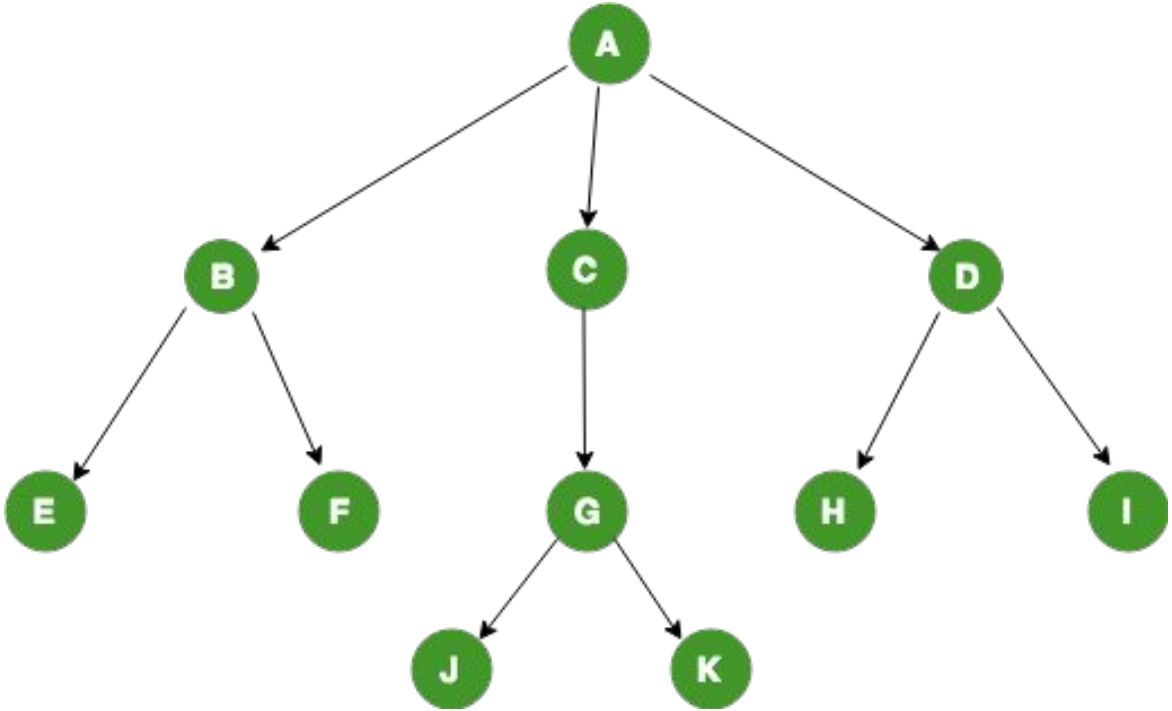
DEFINITION

Un arbre est une structure de données hiérarchique, non linéaire qui se compose de nœuds connectés par des arêtes pour simuler un ou imiter un arbre réel.

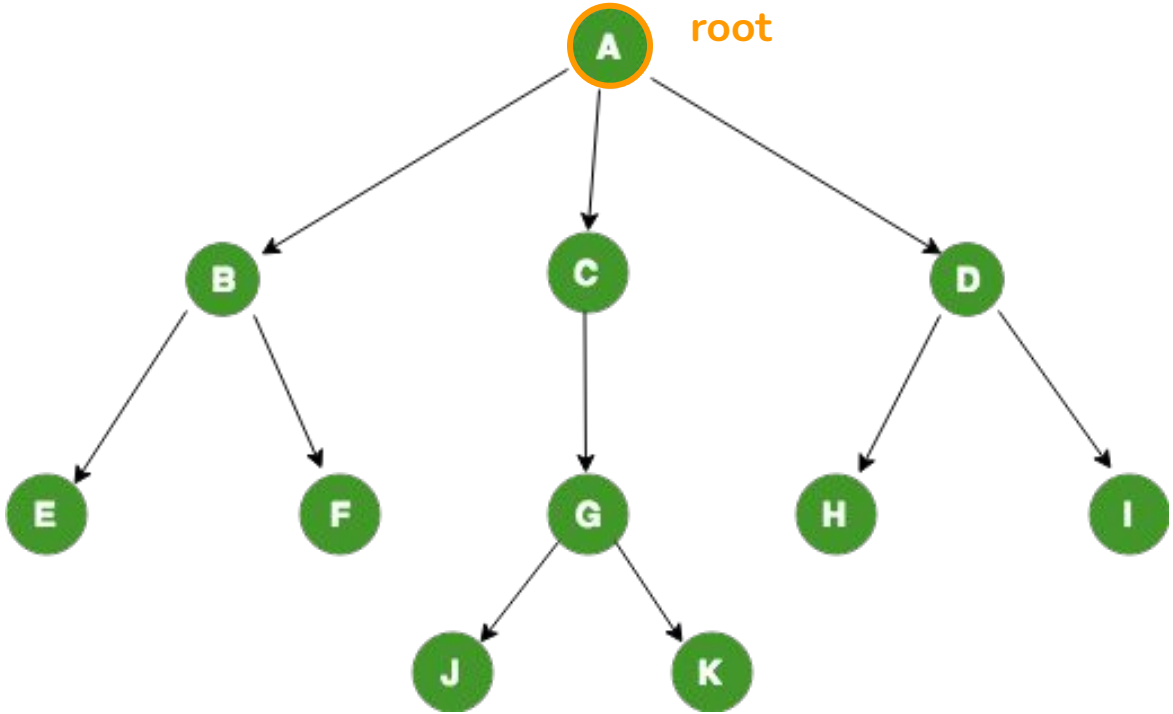
Les composants

- **Root** : le noeud racine
- **Parent** : un noeud qui
- **Child** : un noeud issu d'un autre noeud
- **Siblings** : deux fils de même niveau

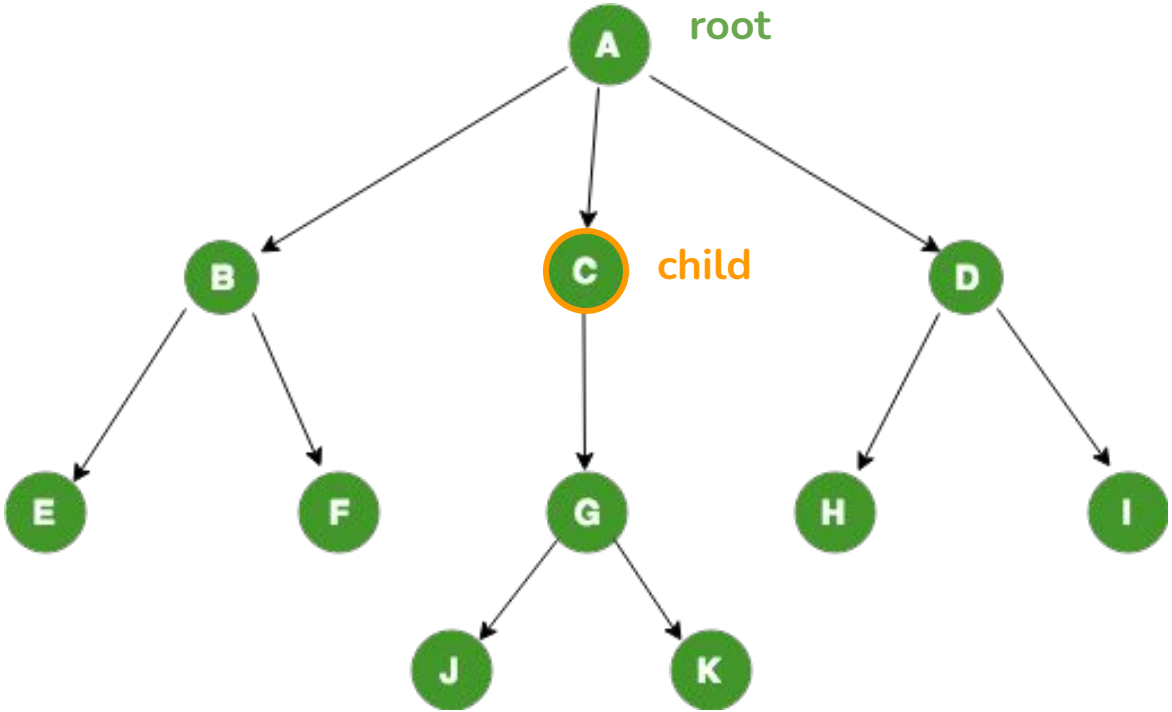
Arbres



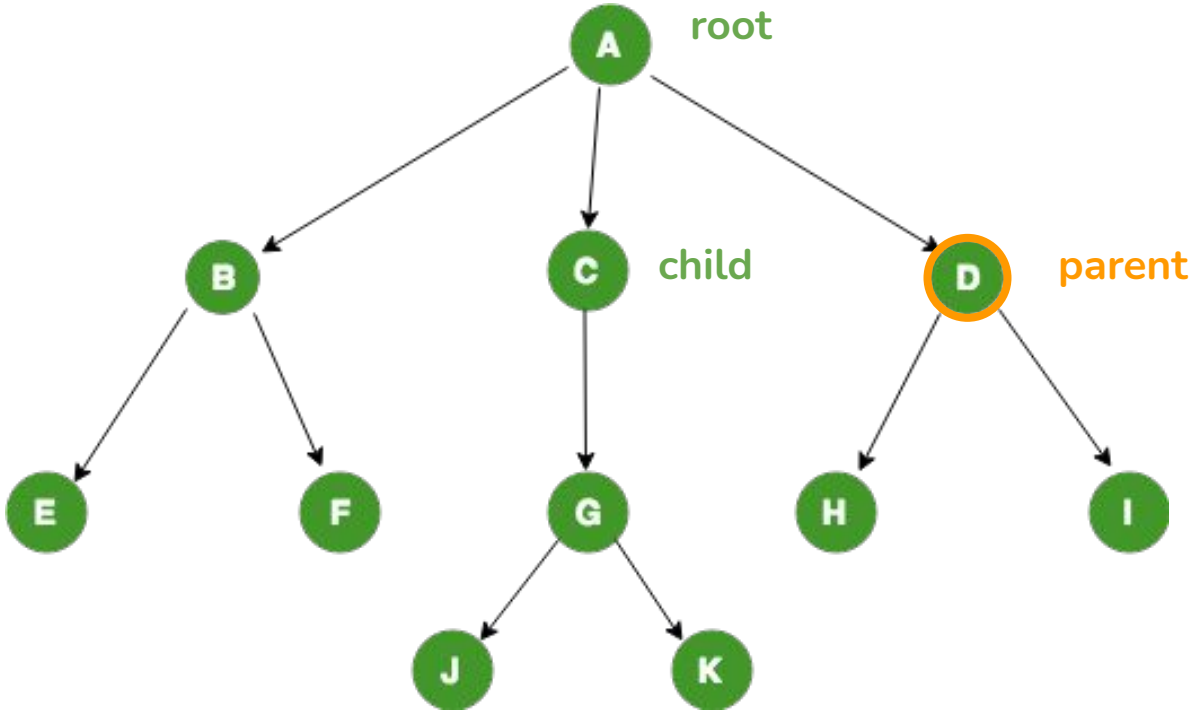
Arbres



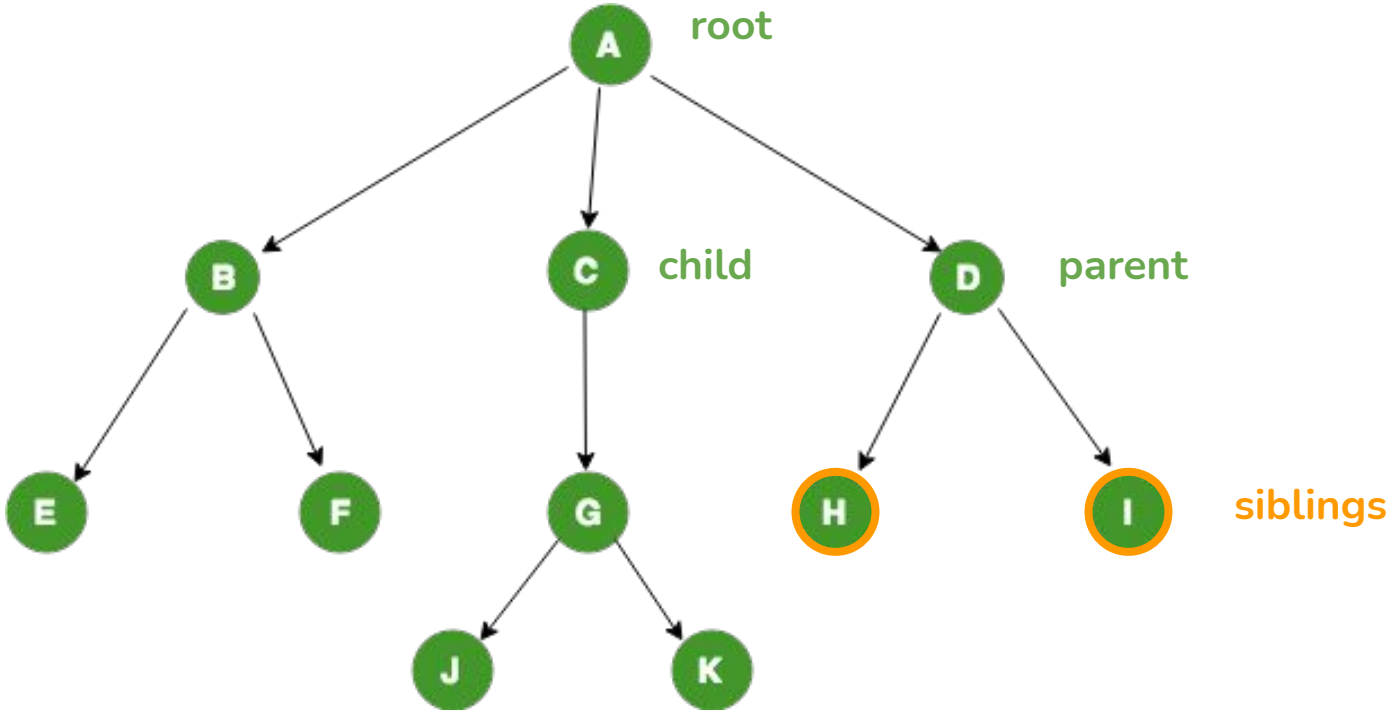
Arbres



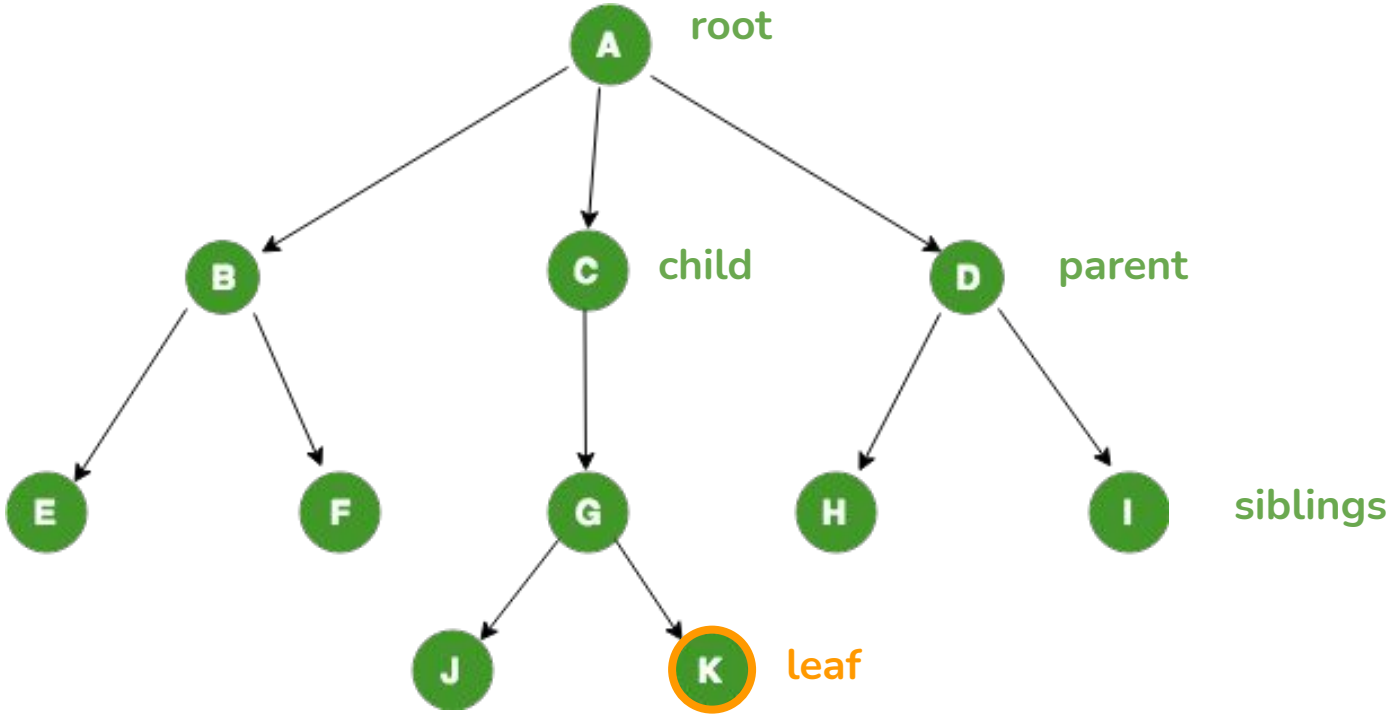
Arbres



Arbres



Arbres

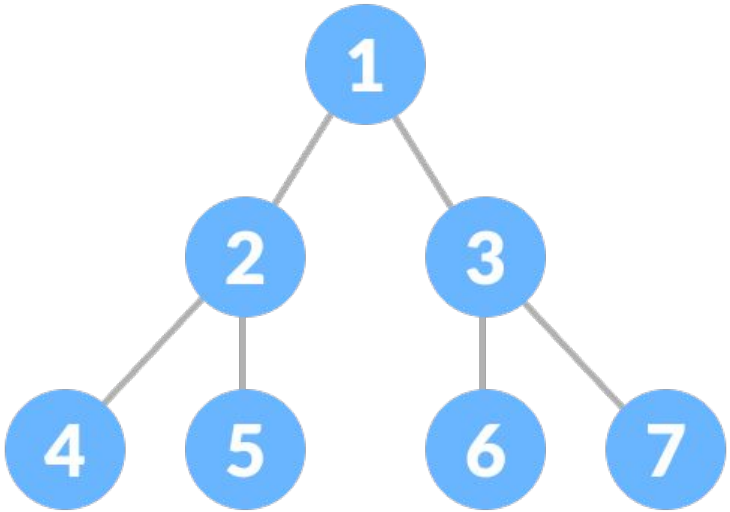


Arbre

Arbre binaire

Un arbre binaire est sous-type d’arbre, où chaque élément possède au plus deux éléments fils au niveau inférieur, habituellement appelés gauche et droit.

Arbre binaire



Arbre

Arbre binaire de recherche

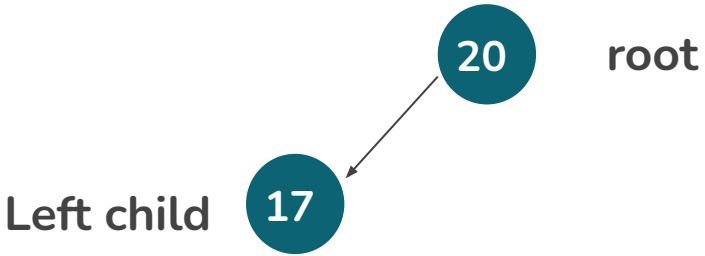
Un arbre binaire de recherche est un arbre binaire dans lequel chaque nœud possède une clé, telle que chaque nœud du sous-arbre gauche ait une clé inférieure à celle du nœud considéré, et que chaque nœud du sous-arbre droit possède une clé supérieure ou égale à celle-ci

Arbre

Arbre binaire de recherche

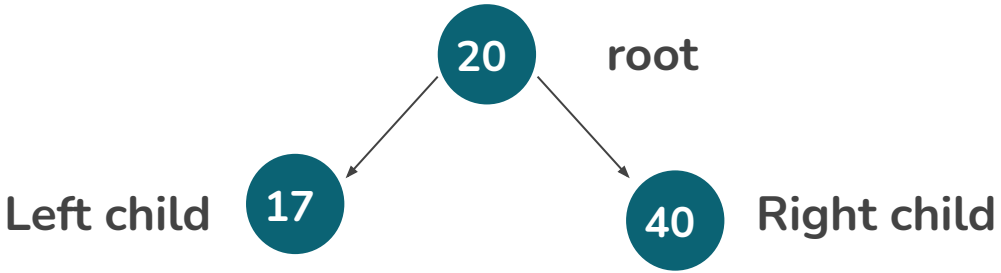
Arbre

Arbre binaire de recherche



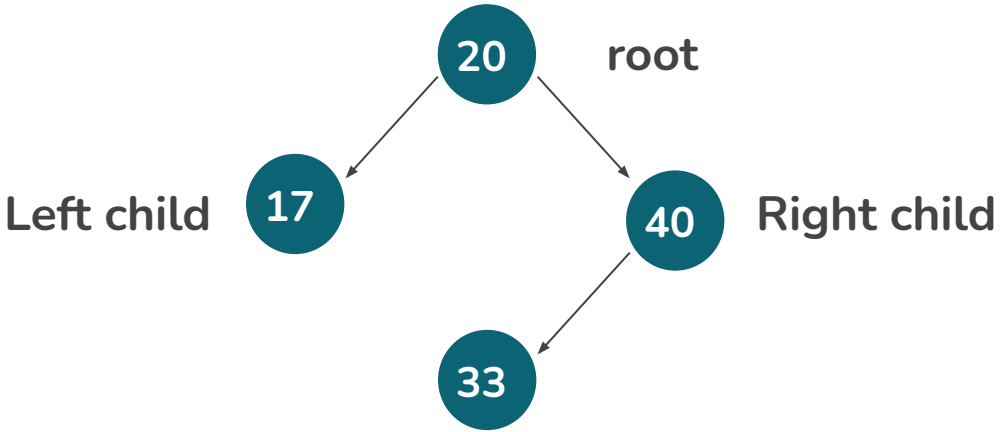
Arbre

Arbre binaire de recherche



Arbre

Arbre binaire de recherche



Arbre

Quelques exemples d'implémentation

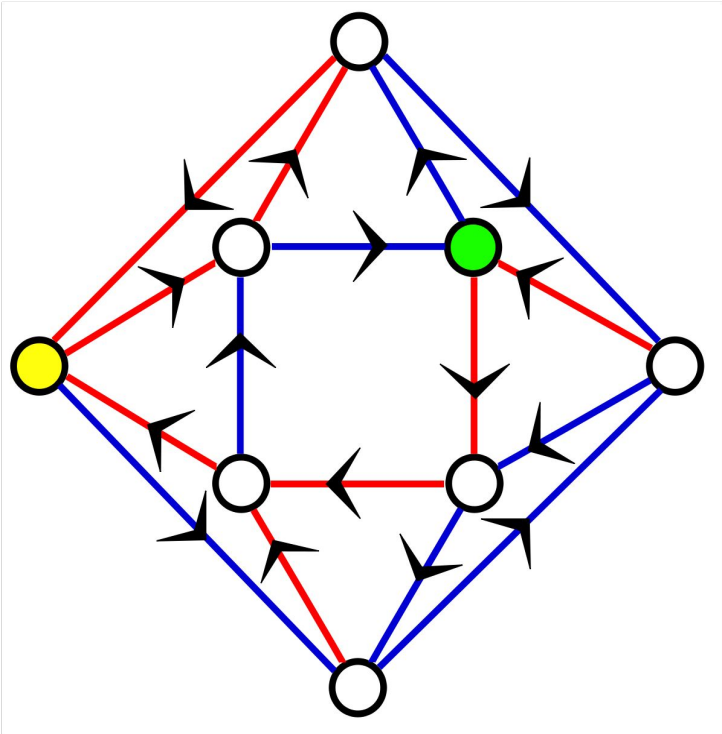
JAVA	TreeMap
C#	SortedDictionnary
C++	std::set

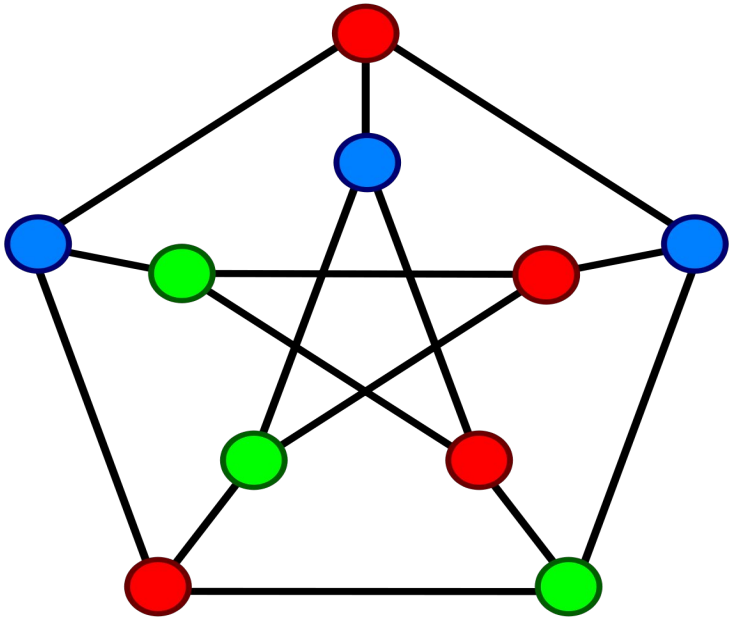
Les graphes

Définition

Un graphe est une façon de représenter des données ; formellement, il est défini comme un ensemble de sommets et d'arêtes reliant ces sommets. Celles-ci peuvent être orientées ou non et peuvent également avoir des poids différents (qui dépendent de la situation à modéliser). Selon que toutes ses arêtes sont orientées ou non, on dira que le graphe est orienté ou non orienté.

Exemples



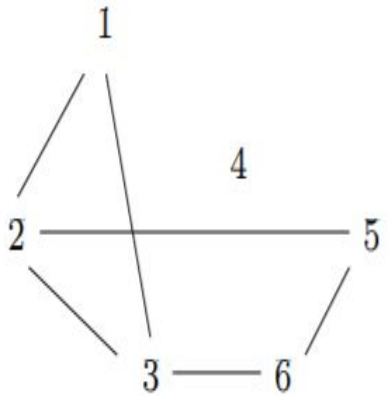


Les graphes non-orientés

Un graphe non-orienté G est un couple (S, A) formé d'un ensemble de sommets S et d'un ensemble A de paires {x, y} ∈ S2, les arêtes.

G = ({1, 2, 3, 4, 5, 6},

{{1, 2}, {3, 1}, {2, 3}, {6, 5}, {6, 3}, {2, 5}})



Soit une arˆete {x, y}, x et y sont les extrémités et sont

adjacents

G = ({1, 2, 3, 4, 5, 6},

{{1, 2}, {3, 1}, {2, 3}, {6, 5}, {6, 3}, {2, 5}})

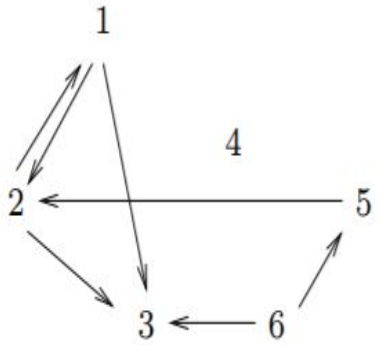
Soit une arˆete {x, y}, x et y sont les extrémités et sont adjacents

Les graphes orientés

Un graphe orienté est un couple (S, A) formé d'un ensemble de sommets S et d'un ensemble A de paires (x, y) ∈ S2, les arcs sont orientés.

G = ({1, 2, 3, 4, 5, 6}, {(1, 2),(2, 1),(1, 3),(2, 3),(6, 5),(6, 3),(5, 2)})

Soit un arc (x, y),



- x est l'extrémité initiale et y est l'extrémité finale,
- x est adjacent à y ;
- (x, y) est incident à x vers l'extérieur et incident

à y vers l'intérieur ;

- x est le prédécesseur de y et y est le successeur de x.

Exemple de déclaration d'un graphe en C

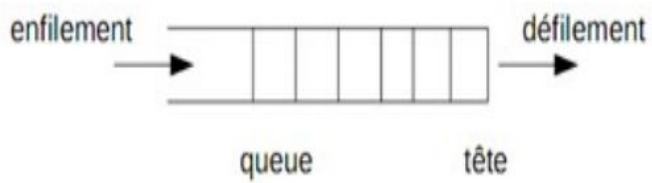
```c
/* Définition d'un Booléen */
typedef enum
{
    false,
    true
}Bool;

/* Définition d'une liste de Noeuds (sommet) */
typedef struct NodeListElement
{
    int value;
    struct NodeListElement *next;
}NodeListElement, *NodeList;
```

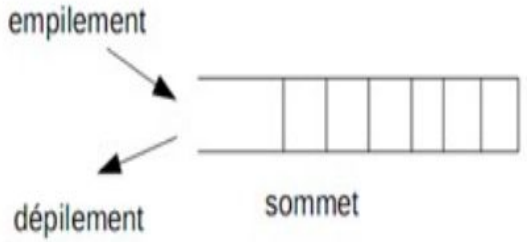
Exemple de déclaration d'un graphe en C

```c
/* Définition d'une liste d'ajdacence (tableau) */
typedef struct AdjencyListElement
{
    NodeListElement *begin;
}AdjencyListElement, *AdjencyList;

/* Définition d'un Graphe */
typedef struct GraphElement
{
    Bool is_oriented;
    int nb_vertices;
    AdjencyList tab_voisins;
}GraphElement, *Graph;
```

Piles et files d’attente

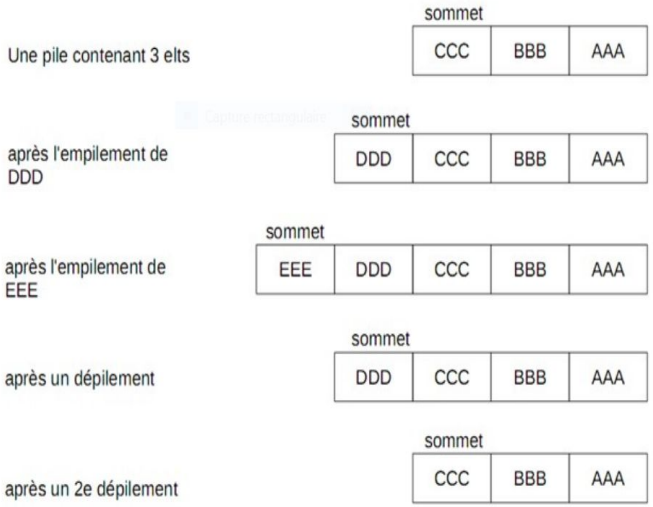
- **Pile** : collection (ensemble) d’elements gerees en LIFO (Last In First Out) - stack
- **File** : collection (ensemble) d’elements gerees en FIFO (First In First Out) - queue





exemples

Exemple de pile



Exemple de file