

Structures de données et algorithmes

Joëlle Cohen

20 mars 2018

Chapitre 1: [Structures linéaires](#)

Chapitre 2: [Structures arborescentes](#)

Dans ce document, la description des classes de l'API ne prétend aucunement être exhaustive. Reportez-vous à l'API en question pour connaître tous les détails de cette classe.

Introduction

Généralités

Les tableaux

Les piles

les files

Les listes

Chapitre I – Structures linéaires

- Introduction

- Généralités

- Les tableaux

- Les piles

- les files

- Les listes

Introduction

Généralités

Les tableaux

Les piles

les files

Les listes

Chapitre I – Structures linéaires

- Introduction

- Généralités

- Les tableaux

- Les piles

- les files

- Les listes

Introduction

Généralités

Les tableaux

Les piles

les files

Les listes

Chapitre I – Structures linéaires

- Introduction
- Généralités
- Les tableaux
- Les piles
- les files
- Les listes

Introduction

Généralités

Les tableaux

Les piles

les files

Les listes

Chapitre I – Structures linéaires

- Introduction

- Généralités

- Les tableaux

- Les piles

- les files

- Les listes

Introduction

Généralités

Les tableaux

Les piles

les files

Les listes

Chapitre I – Structures linéaires

- Introduction
- Généralités
- Les tableaux
- Les piles
- les files
- Les listes

Introduction

Généralités

Les tableaux

Les piles

les files

Les listes

Chapitre I – Structures linéaires

- Introduction
- Généralités
- Les tableaux
- Les piles
- les files
- Les listes

Algorithme : méthode de résolution d'un problème

Structure de données : méthodes pour stocker et manipuler des données.

Algorithme + Structure de données = programme.

Pour un problème donné, il peut y avoir plusieurs algorithmes qui peuvent utiliser des structures de données différentes d'un algorithme à l'autre.

Par exemple un tri peut utiliser un tableau (tri à bulle, quicksort, ...), une liste chaînée (tri fusion), un arbre binaire de recherche, un tas (tri par tas).

Introduction

Généralités

Les tableaux

Les piles

Les files

Les listes

Le but du cours est de savoir manipuler un ensemble de données selon un contexte imposé (par le problème ou par l'algorithme résolvant ce problème).

Pour un problème donné, on cherche à déterminer la meilleure façon de manipuler les données mises en jeu dans ce problème.

Pour cela on va

- *spécifier* \Rightarrow construire un modèle abstrait du réel et décrire ce modèle en vue d'assurer l'adéquation entre ce qui est produit et ce qu'on attend,

- *utiliser* une structure de données

structure de données \Rightarrow modèle qui décrit le comportement d'un ensemble d'informations muni de ses propriétés.

Une structure de données peut être vue de l'extérieur (modèle abstrait) et de l'intérieur (réalisation concrète).

Le but du cours est de savoir manipuler un ensemble de données selon un contexte imposé (par le problème ou par l'algorithme résolvant ce problème).

Pour un problème donné, on cherche à déterminer la meilleure façon de manipuler les données mises en jeu dans ce problème.

Pour cela on va

- *spécifier* = construire un modèle abstrait du réel et décrire ce modèle en vue d'assurer l'adéquation entre ce qui est produit et ce qu'on attend,

• *utiliser une structure de données*

structure de données = modèle qui décrit le comportement d'un ensemble d'informations muni de ses propriétés.

Une structure de données peut être vue de l'extérieur (modèle abstrait) et de l'intérieur (réalisation concrète).

Le but du cours est de savoir manipuler un ensemble de données selon un contexte imposé (par le problème ou par l'algorithme résolvant ce problème).

Pour un problème donné, on cherche à déterminer la meilleure façon de manipuler les données mises en jeu dans ce problème.

Pour cela on va

- *spécifier* = construire un modèle abstrait du réel et décrire ce modèle en vue d'assurer l'adéquation entre ce qui est produit et ce qu'on attend,

- *utiliser* une structure de données

structure de données = modèle qui décrit le comportement d'un ensemble d'informations muni de ses propriétés.

Une structure de données peut être vue de l'extérieur (modèle abstrait) et de l'intérieur (réalisation concrète).

Le but du cours est de savoir manipuler un ensemble de données selon un contexte imposé (par le problème ou par l'algorithme résolvant ce problème).

Pour un problème donné, on cherche à déterminer la meilleure façon de manipuler les données mises en jeu dans ce problème.

Pour cela on va

- *spécifier* = construire un modèle abstrait du réel et décrire ce modèle en vue d'assurer l'adéquation entre ce qui est produit et ce qu'on attend,

- *utiliser* une structure de données

structure de données = modèle qui décrit le comportement d'un ensemble d'informations muni de ses propriétés.

Une structure de données peut être vue de l'extérieur (modèle abstrait) et de l'intérieur (réalisation concrète).

étapes

- *spécification fonctionnelle* (ou *abstraite*) : on fait la liste des opérations (avec leurs définitions et restrictions) qui agiront sur les données. C'est le document de référence de l'utilisateur,
- *spécification opérationnelle* : elle consiste en deux étapes

Remarque : pour une même spécification fonctionnelle, il peut y avoir plusieurs spécifications opérationnelles dans le même langage.

étapes

- *spécification fonctionnelle* (ou *abstraite*) : on fait la liste des opérations (avec leurs définitions et restrictions) qui agiront sur les données. C'est le document de référence de l'utilisateur,
- *spécification opérationnelle* : elle consiste en deux étapes
 - description logique : on organise la structure de données en utilisant des structures informatiques classiques (par exemple des tableaux).
 - représentation physique : c'est l'implémentation complète faite dans un langage de programmation fixé.

Remarque : pour une même spécification fonctionnelle, il peut y avoir plusieurs spécifications opérationnelles dans le même langage.

étapes

- *spécification fonctionnelle* (ou *abstraite*) : on fait la liste des opérations (avec leurs définitions et restrictions) qui agiront sur les données. C'est le document de référence de l'utilisateur,
- *spécification opérationnelle* : elle consiste en deux étapes
 - description logique : on organise la structure de données en utilisant des structures informatiques classiques (par exemple des tableaux),
 - *représentation physique* : c'est l'implémentation complète faite dans un langage de programmation fixé.

Remarque : pour une même spécification fonctionnelle, il peut y avoir plusieurs plusieurs spécifications opérationnelles dans le même langage.

étapes

- *spécification fonctionnelle* (ou *abstraite*) : on fait la liste des opérations (avec leurs définitions et restrictions) qui agiront sur les données. C'est le document de référence de l'utilisateur,
- *spécification opérationnelle* : elle consiste en deux étapes
 - description logique : on organise la structure de données en utilisant des structures informatiques classiques (par exemple des tableaux),
 - représentation physique : c'est l'implémentation complète faite dans un langage de programmation fixé.

Remarque : pour une même spécification fonctionnelle, il peut y avoir plusieurs spécifications opérationnelles dans le même langage.

étapes

- *spécification fonctionnelle* (ou *abstraite*) : on fait la liste des opérations (avec leurs définitions et restrictions) qui agiront sur les données. C'est le document de référence de l'utilisateur,
- *spécification opérationnelle* : elle consiste en deux étapes
 - description logique : on organise la structure de données en utilisant des structures informatiques classiques (par exemple des tableaux),
 - représentation physique : c'est l'implémentation complète faite dans un langage de programmation fixé.

Remarque : pour une même spécification fonctionnelle, il peut y avoir plusieurs spécifications opérationnelles dans le même langage.

étapes

- *spécification fonctionnelle* (ou *abstraite*) : on fait la liste des opérations (avec leurs définitions et restrictions) qui agiront sur les données. C'est le document de référence de l'utilisateur,
- *spécification opérationnelle* : elle consiste en deux étapes
 - description logique : on organise la structure de données en utilisant des structures informatiques classiques (par exemple des tableaux),
 - représentation physique : c'est l'implémentation complète faite dans un langage de programmation fixé.

Remarque : pour une même spécification fonctionnelle, il peut y avoir plusieurs plusieurs spécifications opérationnelles dans le même langage.

pour la spécification fonctionnelle (ou abstraite) on va définir un *type de données abstrait* avec :

- ➊ TYPE : les noms des types définis
- ➋ UTILISE : les types abstraits déjà définis utilisés
- ➌ OPÉRATIONS : les opérations avec leur signature¹
- ➍ PRÉCONDITIONS : les restrictions éventuelles d'utilisation des opérations

1. la signature d'une opération est sa description syntactique

pour la spécification fonctionnelle (ou abstraite) on va définir un *type de données abstrait* avec :

- ❶ TYPE : les noms des types définis
- ❷ UTILISE : les types abstraits déjà définis utilisés
- ❸ OPÉRATIONS : les opérations avec leur signature¹
- ❹ PRÉCONDITIONS : les restrictions éventuelles d'utilisation des opérations

1. la signature d'une opération est sa description syntactique

pour la spécification fonctionnelle (ou abstraite) on va définir un *type de données abstrait* avec :

- ❶ TYPE : les noms des types définis
- ❷ UTILISE : les types abstraits déjà définis utilisés
- ❸ OPÉRATIONS : les opérations avec leur signature¹
- ❹ PRÉCONDITIONS : les restrictions éventuelles d'utilisation des opérations

1. la signature d'une opération est sa description syntactique

pour la spécification fonctionnelle (ou abstraite) on va définir un *type de données abstrait* avec :

- ❶ TYPE : les noms des types définis
- ❷ UTILISE : les types abstraits déjà définis utilisés
- ❸ OPÉRATIONS : les opérations avec leur signature¹
- ❹ PRÉCONDITIONS : les restrictions éventuelles d'utilisation des opérations

1. la signature d'une opération est sa description syntactique

Structures de données linéaires

Une structure de données est linéaire lorsque les données sont en quelque sorte les unes derrière les autres : chaque donnée a une donnée successeur et une donnée prédécesseur exception faite éventuellement des données qui sont aux extrémités.

Une fois les données rangées, la structure sera amenée à évoluer par l'*ajout* de nouvelles données ou la *suppression* de données appartenant à la structure. Selon la façon de procéder à ces modifications, on distinguera plusieurs types de structures.

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Tableaux

Un tableau est une structure de données linéaire dans laquelle on a un accès direct à chaque donnée par son indice dans le tableau.

Les problèmes généralement posés sur les tableaux sont

- la recherche d'une valeur
- la recherche de la valeur la plus grande/petite
- le tri des valeurs dans l'ordre croissant
- le calcul de la valeur moyenne
- ...

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Tableaux

Un tableau est une structure de données linéaire dans laquelle on a un accès direct à chaque donnée par son indice dans le tableau.

Les problèmes généralement posés sur les tableaux sont

- la recherche d'une valeur
- la recherche de la valeur la plus grande/petite

● le tri des valeurs dans l'ordre croissant

● le calcul de la valeur moyenne

● ...

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Tableaux

Un tableau est une structure de données linéaire dans laquelle on a un accès direct à chaque donnée par son indice dans le tableau.

Les problèmes généralement posés sur les tableaux sont

- la recherche d'une valeur
- la recherche de la valeur la plus grande/petite
- le tri des valeurs dans l'ordre croissant

● le calcul de la valeur moyenne

● ...

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Tableaux

Un tableau est une structure de données linéaire dans laquelle on a un accès direct à chaque donnée par son indice dans le tableau.

Les problèmes généralement posés sur les tableaux sont

- la recherche d'une valeur
- la recherche de la valeur la plus grande/petite
- le tri des valeurs dans l'ordre croissant
- le calcul de la valeur moyenne

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Tableaux

Un tableau est une structure de données linéaire dans laquelle on a un accès direct à chaque donnée par son indice dans le tableau.

Les problèmes généralement posés sur les tableaux sont

- la recherche d'une valeur
- la recherche de la valeur la plus grande/petite
- le tri des valeurs dans l'ordre croissant
- le calcul de la valeur moyenne
- ...

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

2-somme

Problème : étant données n valeurs entières, trouver tous les couples de valeurs dont la somme est nulle.

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

2-somme

Problème : étant données n valeurs entières, trouver tous les couples de valeurs dont la somme est nulle.

algorithme : tester toutes les sommes, afficher les sommes nulles

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

2-somme

Problème : étant données n valeurs entières, trouver tous les couples de valeurs dont la somme est nulle.

données : les n valeurs sont stockées dans un tableau d'entiers t .

algorithme : pour chaque indice k du tableau, calculer toutes les sommes $t(k)+t(i)$ pour i variant dans les indices du tableau, les afficher si elles sont nulles

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

2-somme

Problème : étant données n valeurs entières, trouver tous les couples de valeurs dont la somme est nulle.

données : les n valeurs sont stockées dans un tableau d'entiers t .

algorithme : pour chaque indice k du tableau, pour chaque indice i du tableau supérieur à k , calculer $t(k)+t(i)$, l'afficher si nulle

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

programme

```
public class DeuxSomme{
public static void afficherDeuxSomme(int[] t) {
    for (int k = 0; k < t.length; k++) {
        for (int i = k; i < t.length; i++) {
            if (t[k] + t[i] == 0) {
                System.out.println(t[k]+ " " + t[i]);
            }
        }
    }
}
public static void main(String[] args) {
    . . .
}
```

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

temps d'exécution

Cette procédure exécute $\frac{n(n-1)}{2}$ itérations donc la complexité est en $O(n^2)$.

Peut-on tester le temps d'exécution ? oui

Peut-on faire mieux ? oui

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

temps d'exécution

Cette procédure exécute $\frac{n(n-1)}{2}$ itérations donc la complexité est en $O(n^2)$.

Peut-on tester le temps d'exécution ? oui

Peut-on faire mieux ? oui

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

temps d'exécution

Cette procédure exécute $\frac{n(n-1)}{2}$ itérations donc la complexité est en $O(n^2)$.

Peut-on tester le temps d'exécution ? oui

Peut-on faire mieux ? oui

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

temps d'exécution

Cette procédure exécute $\frac{n(n-1)}{2}$ itérations donc la complexité est en $O(n^2)$.

Peut-on tester le temps d'exécution ? oui

Peut-on faire mieux ? oui

mesurer le temps d'exécution

```
public class Chronometre {  
    private final long debut = System.currentTimeMillis();  
    public double tempsEcoule() {  
        long maintenant = System.currentTimeMillis();  
        return (maintenant - debut) / 1000.0;}}}
```

La méthode `currentTimeMillis()` de la classe `System` donne en milliseconde le temps écoulé depuis le 1er Janvier 1970 minuit jusqu'à l'invocation de la méthode.

Une instance de la classe `Chronometre` a donc l'attribut `debut` initialisé à cette durée au moment de l'appel au constructeur de la classe.

La méthode `tempsEcoule` mesure en seconde le temps écoulé entre l'appel au constructeur de la classe et l'invocation de la méthode.

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

mesurer le temps d'exécution

```
public static void main(String[] args) {  
    int[] t ;  
    // initialiser et remplir t  
    // utiliser la classe Random du paquetage java.util  
    Chronometre c = new Chronometre();  
    afficherDeuxSomme(t);  
    System.out.println  
    ("temps d'exécution " + c.tempsEcoule()+ " secondes");  
}
```

exercice 1 : : faire des tests sur des tableaux de dimension 10, 100, 1000, 2000, 4000, 8000.

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

Supposons le tableau classé par ordre croissant.

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

Supposons le tableau classé par ordre croissant.

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

-7 et 7

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

-7 et 7

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

-7 et 7

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

-7 et 7

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

-7 et 7

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

-7 et 7

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

-7 et 7

-2 et 2

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Faire mieux ?

0	1	2	3	4	5	6	7	8	9	10	11
-7	-5	-4	-2	-1	2	3	6	7	8	11	15

-7 et 7

-2 et 2

arrêt car même signe

exercice 2 : implémenter cette méthode

afficherDeuxSommeRapide

Tri par insertion

C'est le principe du joueur de carte :

on trie au fur et à mesure que l'on découvre une nouvelle valeur en l'insérant parmi celles déjà triées.

Par exemple : on a déjà

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	34	56	74				

Tri par insertion

C'est le principe du joueur de carte :

on trie au fur et à mesure que l'on découvre une nouvelle valeur en l'insérant parmi celles déjà triées.

Par exemple : on a déjà

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	34	56	74				

et on donne la valeur 25 à insérer

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	34	56	74				

Tri par insertion

C'est le principe du joueur de carte :

on trie au fur et à mesure que l'on découvre une nouvelle valeur en l'insérant parmi celles déjà triées.

Par exemple : on a déjà

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	34	56	74				

On décale les trois valeurs supérieures à 25 d'un rang vers la droite puis on place 25

i	1	2	3	4	5	6	7	8	9	10
$t(i)$	-75	-54	15	25	34	56	74			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

En pratique, le tableau est déjà rempli. Donc on commence par insérer la deuxième valeur dans la partie gauche du tableau réduite à la première valeur, puis la troisième valeur dans la partie gauche du tableau composée des deux premières valeurs déjà triées et ainsi de suite jusqu'à insérer la dernière valeur.

L'insertion peut se réaliser de façons

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

En pratique, le tableau est déjà rempli. Donc on commence par insérer la deuxième valeur dans la partie gauche du tableau réduite à la première valeur, puis la troisième valeur dans la partie gauche du tableau composée des deux premières valeurs déjà triées et ainsi de suite jusqu'à insérer la dernière valeur.

L'insertion peut se réaliser de façons

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

En pratique, le tableau est déjà rempli. Donc on commence par insérer la deuxième valeur dans la partie gauche du tableau réduite à la première valeur, puis la troisième valeur dans la partie gauche du tableau composée des deux premières valeurs déjà triées et ainsi de suite jusqu'à insérer la dernière valeur.

L'insertion peut se réaliser de façons

- on décale d'un rang vers la droite toutes les valeurs de rang inférieur qui sont supérieures à la valeur à insérer
- on recule la valeur à insérer par échange de la valeur avec sa voisine de gauche jusqu'à ce sa voisine lui soit inférieure

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

En pratique, le tableau est déjà rempli. Donc on commence par insérer la deuxième valeur dans la partie gauche du tableau réduite à la première valeur, puis la troisième valeur dans la partie gauche du tableau composée des deux premières valeurs déjà triées et ainsi de suite jusqu'à insérer la dernière valeur.

L'insertion peut se réaliser de façons

- on décale d'un rang vers la droite toutes les valeurs de rang inférieur qui sont supérieures à la valeur à insérer
- on recule la valeur à insérer par échange de la valeur avec sa voisine de gauche jusqu'à ce sa voisine lui soit inférieure

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Complexité des tris de tableau

Les tris simples (insertion, à bulle, par sélection) ont une complexité en $O(n^2)$.

Donc la méthode DeuxSommeRapide n'apporte rien si on ne peut pas améliorer les tris.

Les tris rapide, fusion (voir *listes*) ou par tas (voir *file de priorité*) ont une complexité en moyenne de $O(n \ln(n))$.

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Tri rapide

Le tri rapide (Quicksort) est basé sur une méthode `pivoter` qui partage le tableau en deux parties (non nécessairement équilibrées).

A la frontière des deux parties se trouve une valeur de référence, le *pivot*, qui aura servi à faire ce partage.

La méthode `pivoter` est alors rappelée sur les deux parties de façon récursive (avec un pivot propre à chaque partie).

exercice 3 : Ecrire les méthodes `pivoter` puis `triRapide`

exercice 4 : tester le temps d'exécution de `tri rapide` + `afficherDeuxSommeRapide`.

Autres tris

Tous les tris vus précédemment sont basés sur des comparaisons des valeurs du tableau deux à deux. Il existe d'autres méthodes de tris basées sur d'autres principes :

- tri comptage (counting sort) : cette méthode nécessite que l'on connaisse les bornes des valeurs à trier, et que l'intervalle de ces valeurs ne soit pas trop « grand »
- tri radix (radix sort) : cette méthode est basée sur l'écriture en base 2 (voire 10) des nombres
- tri par paquets (bucket sort) : cette méthode nécessite que les valeurs à trier soient réparties uniformément sur un intervalle connu

Tri comptage

On suppose que les valeurs à trier appartiennent à l'intervalle $[0, N]$.
On utilise alors un tableau `tIndex` indexé de 0 à N .

Principe :

- on parcourt le tableau à trier `t` ; si `t[i]` vaut k alors on incrémente `tIndex[k]`.
- on parcourt le tableau `tIndex` ; on place la valeur k `tIndex[k]` fois successivement dans `t`

Exemple :

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	4	7	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	0	0	0	0	0	0	0			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	4	7	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	0	0	0	1	0	0	0			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	4	7	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	0	0	0	1	0	0	1			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	4	7	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	0	0	0	2	0	0	1			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	4	7	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	0	0	0	2	1	0	1			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	4	7	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	0	0	0	3	1	0	1			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	4	7	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	0	0	0	3	2	0	1			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	4	7	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	0	0	0	3	2	0	2			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	4	7	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	1	0	0	3	2	0	2			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	4	7	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	1	1	0	3	2	0	2			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	4	7	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	2	1	0	3	2	0	2			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	4	7	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	2	1	0	3	2	0	3			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	1	1	4	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	2	1	0	3	2	0	3			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	1	1	2	5	4	5	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	2	1	0	3	2	0	3			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	1	1	2	4	4	4	7	1	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	2	1	0	3	2	0	3			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	1	1	2	4	4	4	5	5	2	1	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	2	1	0	3	2	0	3			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	1	1	2	4	4	4	5	5	7	7	7
k	0	1	2	3	4	5	6	7			
$tIndex(k)$	0	2	1	0	3	2	0	3			

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

Tri radix

Le principe présenté s'adapte à d'autres bases ou même à l'alphabet. Principe : on classe les nombres par ordre croissant selon les chiffres qui le composent depuis le moins significatif (unité) jusqu'au plus significatif tout en conservant l'ordre obtenu à l'itération précédente pour les nombres ayant le même chiffre significatif considéré.

Exemple :

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	473	70	24	651	42	55	709	19	2	192	97

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	70	651	42	2	192	473	24	55	97	19	709

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	02	709	19	24	42	651	55	70	473	192	97

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

i	0	1	2	3	4	5	6	7	8	9	10
$t(i)$	002	019	024	042	055	070	097	192	473	651	709

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

En pratique ce tri nécessite 10 tableaux de même longueur que le tableau d'origine t , chacun est prévu pour recevoir les valeurs dont le chiffre significatif est respectivement 0, ... 9. A chaque itération, le tableau t est lu par ordre croissant d'indice et on remplit ces 10 tableaux, puis on remet les valeurs dans le tableau t depuis les tableaux intermédiaires en partant du tableau t_0 jusqu'au tableau t_9 . La complexité de ce tri est en $O(n)$ mais nécessite de la mémoire auxiliaire.

Tri par paquets

Ce tri fonctionne pour des valeurs réelles contenues dans un intervalle connu $[a, b[$. Soit N le nombre de ces valeurs.

On partitionne l'intervalle en N sous-intervalles de la forme suivante

$$I_j = [a + j \frac{b-a}{N}, a + (j+1) \frac{b-a}{N}[\text{ pour } 0 \leq j < N.$$

On crée N listes L_j pour $0 \leq j < N$.

Principe :

- Pour chaque valeur de t
 - calculer l'intervalle I_j contenant cette valeur
 - placer cette valeur dans la liste L_j
- Trier les N listes L_0, \dots, L_{N-1} .
- Copier dans t depuis l'indice 0 L_0 puis $L_1 \dots$ puis L_{N-1} .

Ce tri est performant en $O(n)$ si les valeurs sont uniformément réparties; en effet puisqu'il y a autant d'intervalles que de valeurs, une distribution uniforme garantit des listes de longueur 1 qui sont dès lors déjà triées.

Le pire des cas est une distribution qui placerait toutes les valeurs dans le même intervalle de la partition.

Autres structures linéaires : exemple

Un robot piloté à distance reçoit différents types d'instructions pour se déplacer : en avant, en arrière (demi-tour), à droite, à gauche. Il est de plus muni d'une mémoire « linéaire » qui enregistre la suite d'instructions et ne démarre qu'une fois toutes les instructions reçues. Par exemple, en avant, en avant, à droite, en avant, à gauche, à gauche, en arrière.

Le choix du déplacement dépendra de la façon dont est gérée la mémoire : il choisit de lire tout d'abord soit la première instruction entrée soit la dernière.

Dans le premier cas sa mémoire fonctionne comme une file.

Dans ce dernier cas sa mémoire fonctionne comme une pile.

Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

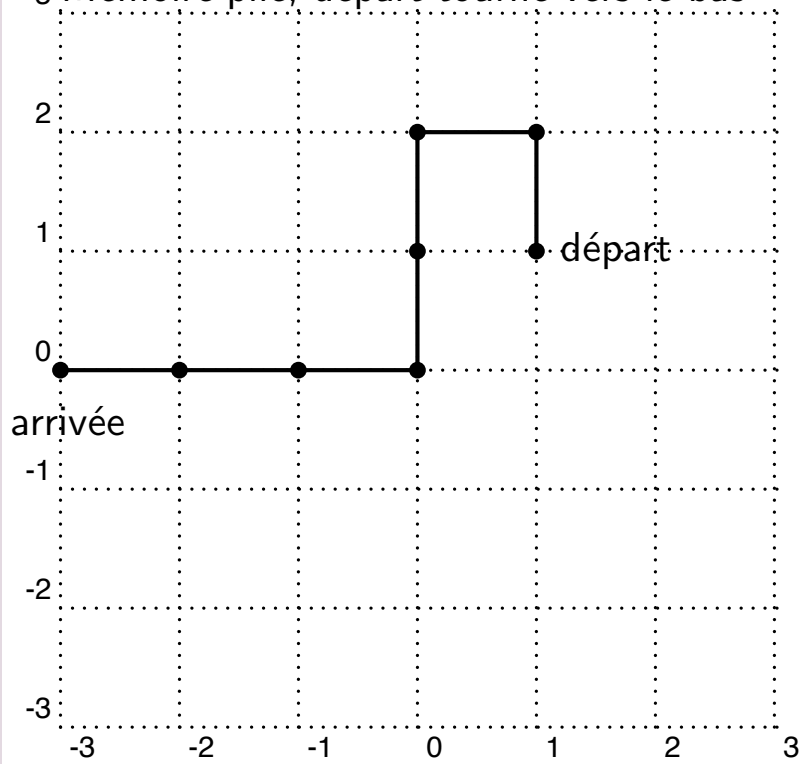
Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

3 Mémoire pile, départ tourné vers le bas



Introduction

Généralités

Les tableaux

Tableaux

Trier un tableau

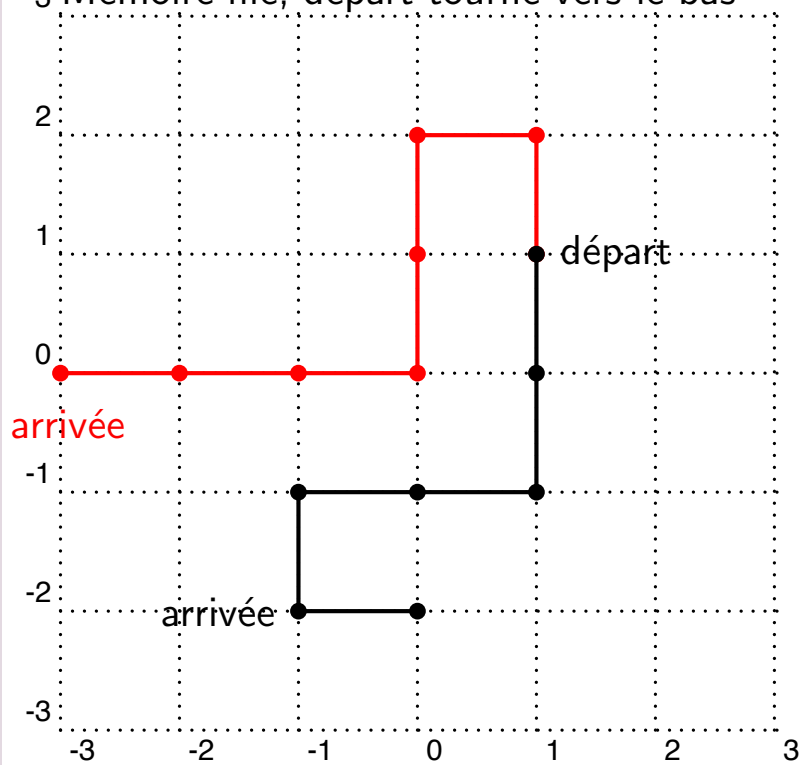
Autres structures
linéaires : un
exemple

Les piles

les files

Les listes

3 Mémoire file, départ tourné vers le bas



Les piles

Une pile est donc une structure de données linéaire dans laquelle ajout et suppression se font au même bout de la structure : son *sommet*.

On dit qu'on *empile* lorsque l'on ajoute un nouvel élément à la pile et cet ajout se fait au sommet de la pile.

On dit qu'on *dépile* lorsque l'on supprime un élément de la pile et puisque la suppression se fait aussi au sommet de la pile, l'élément supprimé est nécessairement le dernier élément à avoir été empilé.

Une pile est aussi désignée par l'acronyme anglais LIFO (**L**ast **I**n **F**irst **O**ut).

Une pile permet par exemple à un navigateur (Firefox, IE ...) de gérer le bouton « reculer d'une page » ou pour un logiciel de gérer la fonctionnalité « undo ».

spécification fonctionnelle

Elle est donc définie par le type de données abstrait PILE suivant :

type PILE

utilise type E, **type** BOOLÉEN

signature

- *initilialiser* : $\rightarrow \text{PILE}$
- *empiler* : $(\text{PILE}, \text{E}) \rightarrow \text{PILE}$
- *dépiler* : $\text{PILE} \rightarrow \text{PILE}$
- *sommet* : $\text{PILE} \rightarrow \text{E}$
- *vide* : $\text{PILE} \rightarrow \text{BOOLÉEN}$

préconditions

- $dépiler(p)$ est défini *si et seulement si* $vide(p) = \text{Faux}$
- $sommet(p)$ est défini *si et seulement si* $vide(p) = \text{Faux}$

axiomes

- $vide(\text{initilialiser}) = \text{Vrai}$
- $vide(\text{empiler}(p, E)) = \text{Faux}$
- $dépiler(\text{empiler}(p, E)) = p$
- $sommet(\text{empiler}(p, E)) = E$

remarques :

- les préconditions précisent la restriction du domaine de définition des deux opérations *dépiler* et *sommet*
- les axiomes assurent le fonctionnement LIFO.

Introduction

Généralités

Les tableaux

Les piles

Interface

Implémentation

Implémentation

Exercices

les files

Les listes

Interface Pile

Pour traduire en partie les spécifications fonctionnelles on va utiliser la notion d'interface Java.

```
/**
 * Interface pour le type pile
 * @author moi
 * @version 0.1
 */
public Interface Pile{
    boolean vide();
    void empiler(int n);
    void depiler();
    int sommet();
}
```

remarque ? : on a créé une interface spéciale pour les piles d'entiers alors que des piles de flottants ou autre objets se comportent de la même façon.

Solutions ??

Introduction

Généralités

Les tableaux

Les piles

Interface

Implémentation

Implémentation

Exercices

les files

Les listes

Interface Pile

Pour traduire en partie les spécifications fonctionnelles on va utiliser la notion d'interface Java.

```
/**
 * Interface pour le type pile
 * @author moi
 * @version 0.1
 */
public Interface Pile{
    boolean vide();
    void empiler(int n);
    void depiler();
    int sommet();
}
```

remarque ? : on a créé une interface spéciale pour les piles d'entiers alors que des piles de flottants ou autre objets se comportent de la même façon.

Solutions ? ?

Interface Pile

Pour traduire en partie les spécifications fonctionnelles on va utiliser la notion d'interface Java.

```
/**
 * Interface pour le type pile
 * @author moi
 * @version 0.1
 */
public Interface Pile{
    boolean vide();
    void empiler(int n);
    void depiler();
    int sommet();
}
```

remarque ? : on a créé une interface spéciale pour les piles d'entiers alors que des piles de flottants ou autre objets se comportent de la même façon.

Solutions ??

Introduction

Généralités

Les tableaux

Les piles

Interface

Implémentation

Implémentation

Exercices

les files

Les listes

Interface Pile

Avec le type Object

```
/**
 * Interface pour le type pile
 * @author moi
 * @version 0.2
 */
public interface Pile{
    boolean vide();
    void empiler(Object o);
    void depiler();
    Object sommet();
}
```

inconvénient : utilisation importante du transtypage.

avantage : on peut empiler des objets instances de classes différentes

mais après on ne saura pas les manipuler finement.

Introduction

Généralités

Les tableaux

Les piles

Interface

Implémentation

Implémentation

Exercices

les files

Les listes

Interface Pile

Avec le type Object

```
/**
 * Interface pour le type pile
 * @author moi
 * @version 0.2
 */
public interface Pile{
    boolean vide();
    void empiler(Object o);
    void depiler();
    Object sommet();
}
```

inconvénient : utilisation importante du transtypage.

avantage : on peut empiler des objets instances de classes différentes
mais après on ne saura pas les manipuler finement.

Introduction

Généralités

Les tableaux

Les piles

Interface

Implémentation

Implémentation

Exercices

les files

Les listes

Interface Pile générique

La généricité est une notion de *polymorphisme paramétrique* qui permet de définir ici des piles contenant des objets de type (uniforme) mais arbitraire.

```
/**
 * Interface pour le type pile générique
 * @author moi
 * @version 1.0
 */
public Interface Pile<T>{
    /** T est un paramètre de type de cette interface */
    /** T représente le type des éléments de la pile */
    boolean vide();
    void empiler(T o);
    void depiler();
    T sommet();
}
```

et maintenant l'implémentation, des idées ???

Introduction

Généralités

Les tableaux

Les piles

Interface

Implémentation

Implémentation

Exercices

les files

Les listes

Interface Pile générique

La généricité est une notion de *polymorphisme paramétrique* qui permet de définir ici des piles contenant des objets de type (uniforme) mais arbitraire.

```
/**
 * Interface pour le type pile générique
 * @author moi
 * @version 1.0
 */
public Interface Pile<T>{
 /** T est un paramètre de type de cette interface */
 /** T représente le type des éléments de la pile */
 boolean vide();
 void empiler(T o);
 void depiler();
 T sommet();
}
```

et maintenant l'implémentation, des idées ???

Introduction

Généralités

Les tableaux

Les piles

Interface

Implémentation

Implémentation

Exercices

les files

Les listes

avec un tableau dynamique

```
import java.util.ArrayList;
public class PileArrayList<T> implements Pile<T>{
    private int top=-1;
    private ArrayList<T> tabPile;
    // constructeur
    public PileArrayList(){ tabPile= new ArrayList<T>();}
    // implémentation de l'interface
    boolean vide(){ return this.top== -1;}
    void empiler(T o){ tabPile.add(o); top ++;}
    void depiler(){ top --;}
    T sommet(){ return tabPile.get(top); }
}
```

Introduction

Généralités

Les tableaux

Les piles

Interface

Implémentation

Implémentation

Exercices

les files

Les listes

utilisation

```
public class EssaiPileArrayList {
    public static void main(String[] args) {
        Pile<Integer> p = new PileArrayList<Integer>();
        for (int i=1; i <=10; i++) p.empiler(i);
        p.depiler();
        p.depiler();
        System.out.println
            ("le sommet de la pile p est " + p.sommet());

        Pile<String> p2 = new PileArrayList<String>();
        for (int i=1; i <=10; i++)
            p2.empiler("toto " + i);
        p2.depiler();
        System.out.println
            ("le sommet de la pile p2 est " + p2.sommet());
    }
}
```

le sommet de la pile p est 8

le sommet de la pile p2 est toto 9

Introduction

Généralités

Les tableaux

Les piles

Interface

Implémentation

Implémentation

Exercices

les files

Les listes

utilisation

```
public class EssaiPileArrayList {  
    public static void main(String[] args) {  
        Pile<Integer> p = new PileArrayList<Integer>();  
        for (int i=1; i<=10;i++) p.empiler(i);  
        p.depiler();  
        p.depiler();  
        System.out.println  
            ("le sommet de la pile p est " + p.sommet());  
  
        Pile<String> p2 = new PileArrayList<String>();  
        for (int i=1; i<=10;i++)  
            p2.empiler("toto " + i);  
        p2.depiler();  
        System.out.println  
            ("le sommet de la pile p2 est " + p2.sommet());  
    }  
}
```

le sommet de la pile p est 8

le sommet de la pile p2 est toto 9

Introduction

Généralités

Les tableaux

Les piles

Interface

Implémentation

Implémentation

Exercices

les files

Les listes

autre implémentation

```
public class PileAuto<T> implements Pile<T>{
//classe interne privée
    private class Cell{
        private T val;
        private Cell suiv;
        Cell(T v, Cell c){val=v; suiv=c;}
        T getVal(){return val;}
        Cell getSuiv(){return suiv;}
    }
//1 unique attribut
    private Cell top;
// les méthodes de l'interface Pile
    public boolean vide(){
        return top == null;
    }
    public void empiler(T o) {
        top=new Cell(o,top);
    }
    public void depiler(){
        top=top.getSuiv();
    }
    public T sommet(){
        return this.top.getVal();
    }
}
```

Introduction

Généralités

Les tableaux

Les piles

Interface

Implémentation

Implémentation

Exercices

les files

Les listes

```
public class EssaiPileArrayListEtPileAuto {
    public static void main(String[] args) {
        Pile<Integer> p = new PileArrayList<Integer>();
        for (int i=1; i <=10; i++) p.empiler(i);
        p.depiler();
        p.depiler();
        System.out.println
            ("le sommet de la pile p est " + p.sommet());

        Pile<Float> p3= new PileAuto<Float>();
        for (int i=10; i >=1; i--) p3.empiler(i+20.0f);
        p3.depiler();
        System.out.println
            ("le sommet de la pile p3 est " + p3.sommet());
    }
}
```

le sommet de la pile p est 8

le sommet de la pile p3 est 22.0

Introduction

Généralités

Les tableaux

Les piles

Interface

Implémentation

Implémentation

Exercices

les files

Les listes

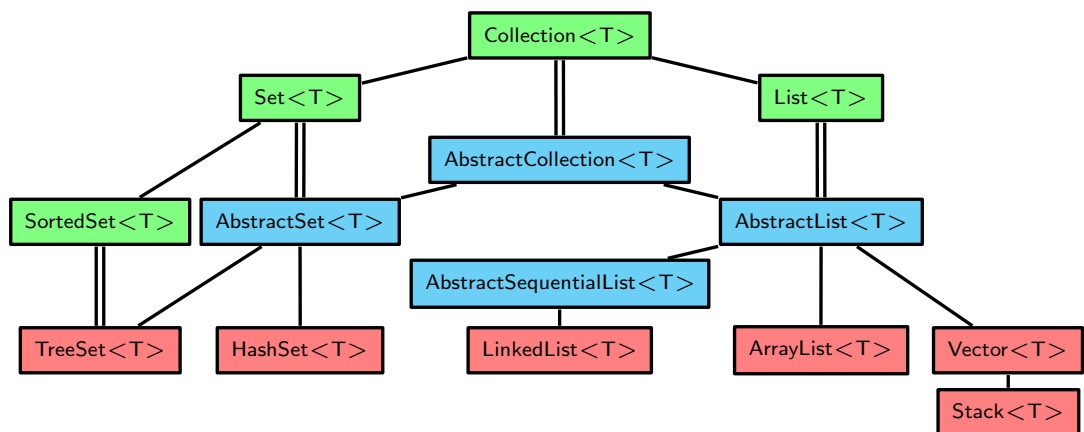
```
public class EssaiPileArrayListEtPileAuto {  
    public static void main(String[] args) {  
        Pile<Integer> p = new PileArrayList<Integer>();  
        for (int i=1; i<=10;i++) p.empiler(i);  
        p.depiler();  
        p.depiler();  
        System.out.println  
            ("le sommet de la pile p est " + p.sommet());  
  
        Pile<Float> p3= new PileAuto<Float>();  
        for (int i=10; i>=1;i--) p3.empiler(i+20.0f);  
        p3.depiler();  
        System.out.println  
            ("le sommet de la pile p3 est " + p3.sommet());  
    }  
}
```

le sommet de la pile p est 8

le sommet de la pile p3 est 22.0

une classe toute prête

Dans le paquetage `java.util`



la classe Stack

Il existe aussi la classe `Stack` : c'est une sous-classe de la classe `Vector` qui implémente entre autres les interfaces `List<T>` et `Collection<T>`. Elle étend la classe `Vector` avec 5 opérations qui permettent de traiter un objet `Vector` comme une pile.

- `public boolean empty()` \Leftrightarrow `public boolean vide()`
- `public T push(T item)` : empile `item` et renvoie `item`
- `public T pop()` : dépile le sommet et renvoie le sommet
- `public T peek()` \Leftrightarrow `public T sommet()`
- `public int search(Object o)` : renvoie la position de l'objet `o` dans la pile avec la convention suivante
 - si `o` n'est pas dans la pile renvoie -1
 - sinon renvoie le numéro d'ordre depuis le sommet jusqu'à la première occurrence de `o` en "descendant" (le sommet ayant le numéro 1)

Exercices

exercice 5 : Ecrire l'algorithme et une méthode qui inverse une pile en utilisant uniquement les fonctionnalités des piles

exercice 6 : Ecrire l'algorithme et une méthode d'instance qui renvoie une pile d'entiers dans laquelle les valeurs positives de l'objet sont au-dessus des valeurs négatives de l'objet, dans le même ordre relatif.

exercice 7 : Réécrire la méthode `equals()` pour une égalité en profondeur.

exercice 8 : Réécrire la méthode `clone()` pour un clonage en profondeur.

exercice 9 : Utiliser les piles pour résoudre le problème des tours Hanoï.

Introduction

Généralités

Les tableaux

Les piles

les files

Interface

Implémentation

La classe

LinkedList

Exercices

Les listes

Les files

Une file est donc une structure de données linéaire dans laquelle l'ajout se fait à une extrémité - la queue - et la suppression se fait à l'autre extrémité de la structure - la tête.

On dit qu'on *enfile* lorsque l'on ajoute un nouvel élément à la file.

On dit qu'on *défile* lorsque l'on supprime un élément de la file et puisque la suppression se fait en tête de la file, l'élément supprimé est nécessairement le premier élément à avoir été enfilé. Une file est aussi désignée par l'acronyme anglais FIFO (**F**irst **I**n **F**irst **O**ut).

Une file permet par exemple à un serveur de gérer les requêtes qu'il reçoit.

spécification fonctionnelle

Elle est donc définie par le type de données abstrait `FILE` suivant :

type `FILE`

utilise `TYPE E`, `TYPE BOOLÉEN`

signature

- *initilialiser* : $\rightarrow \text{FILE}$
- *enfiler* : $(\text{FILE}, \text{E}) \rightarrow \text{FILE}$
- *défiler* : $\text{FILE} \rightarrow \text{FILE}$
- *premier* : $\text{FILE} \rightarrow \text{E}$
- *vide* : $\text{FILE} \rightarrow \text{BOOLÉEN}$

préconditions

- *défiler*(f) est défini *si et seulement si* *vide*(f) = Faux
- *premier*(f) est défini *si et seulement si* *vide*(f) = Faux

Introduction

Généralités

Les tableaux

Les piles

les files

Interface

Implémentation

La classe

LinkedList

Exercices

Les listes

axiomes

- $vide(initialiser) = \text{Vrai}$
- $vide(enfiler(f, E)) = \text{Faux}$
- $premier(enfiler(f, E)) = \begin{cases} E & \text{si } vide(f) \\ premier(f) & \text{sinon} \end{cases}$
- $défiler(enfiler(f, E)) = \begin{cases} initialiser(f) & \text{si } vide(f) \\ enfiler(défiler(f), E) & \text{sinon} \end{cases}$

Introduction

Généralités

Les tableaux

Les piles

les files

Interface

Implémentation

La classe

LinkedList

Exercices

Les listes

Interface File

Pour traduire en partie les spécifications fonctionnelles on va utiliser la notion d'interface Java.

```
/**
 * Interface pour le type file générique
 * @author moi
 * @version 1.0
 */
public interface File<T>{
    /** T est un paramètre de type de cette interface */
    /** T représente le type des éléments de la file */
    boolean vide();
    void enfiler(T o);
    void defiler();
    T premier();
}
```

Introduction

Généralités

Les tableaux

Les piles

Les files

Interface

Implémentation

La classe

LinkedList

Exercices

Les listes

implémentation

```
public class FileAuto<T> implements File<T>{
//classe interne privée
    private class Cell{
        private T val;
        private Cell suiv;
        Cell(T v, Cell c){val=v; suiv=c;}
        T getVal(){return val;}
        Cell getSuiv(){return suiv;}
        void setSuiv(Cell c){suiv =c;}
    }
//1 unique attribut
    private Cell last;
// les méthodes de l'interface File
    public boolean vide(){
        return last == null;
    }
}
```


Introduction

Généralités

Les tableaux

Les piles

les files

Interface

Implémentation

La classe

LinkedList

Exercices

Les listes

suite

```
public void enfiler(T o) {
    if (first == null) {
        last=new Cell(o,first);
        last.setSuiv(first);}
    else {
        Cell c=new Cell(o,last.getSuiv());
        last.setSuiv(c);
        last=c;
    }
    public void defiler(){
        if(last.getSuiv()==last) last=null;
        else last.setSuiv(last.getSuiv().getSuiv());
    }
    public T premier(){
        return this.last.getSuiv().getVal();
    }}
}}
```

Introduction

Généralités

Les tableaux

Les piles

les files

Interface

Implémentation

La classe

LinkedList

Exercices

Les listes

```
public class EssaiFileAuto {
    public static void main(String[] args) {
        File<Integer> f = new FileAuto<Integer>();
        for (int i=1; i <=25; i++) f.enfiler(i);
        f.defiler();
        f.defiler();
        System.out.println
            ("la tête de la file f est " + f.premier());

        File<Float> f2= new FileAuto<Float>();
        for (int i=25; i >=1; i--) f2.enfiler(i+20.0f);
        f2.defiler();
        System.out.println
            ("la tête de la file f2 est " + f2.premier());
    }
}
```

la tête de la file f est 3

la tête de la file f2 est 44.0

Introduction

Généralités

Les tableaux

Les piles

les files

Interface

Implémentation

La classe

LinkedList

Exercices

Les listes

```
public class EssaiFileAuto {
    public static void main(String[] args) {
        File<Integer> f = new FileAuto<Integer>();
        for (int i=1; i <=25; i++) f.enfiler(i);
        f.defiler();
        f.defiler();
        System.out.println
            ("la tête de la file f est " + f.premier());

        File<Float> f2= new FileAuto<Float>();
        for (int i=25; i >=1; i--) f2.enfiler(i+20.0f);
        f2.defiler();
        System.out.println
            ("la tête de la file f2 est " + f2.premier());
    }
}
```

la tête de la file f est 3

la tête de la file f2 est 44.0

La classe LinkedList

La classe `LinkedList<T>` du package `java.util` implémente l'interface `Collection<T>` et possède notamment les méthodes suivantes :

- `T getFirst()` : retourne le premier élément de la liste
- `boolean add(T o)` : ajoute l'objet `o` à la fin de la liste
- `T removeFirst()` : supprime et retourne le premier élément de la liste

Le classe `FileLL<T>` utilise alors un attribut de type `LinkedList<T>` pour implémenter une file

```
import java.util.*;
public class FileLL<T> implements File<T> {
    private List lg=0;
    private LinkedList<T> list;
    public FileLL(){list = new LinkedList<T>();}
    public boolean vide(){return lg==0;}
    public void enfiler(T o){list.add(o); lg++;}
    public T premier(){return list.getFirst();}
    public void defiler(){list.removeFirst();lg--;}
}
```

La classe LinkedList

La classe `LinkedList<T>` du package `java.util` implémente l'interface `Collection<T>` et possède notamment les méthodes suivantes :

- `T getFirst()` : retourne le premier élément de la liste
- `boolean add(T o)` : ajoute l'objet `o` à la fin de la liste
- `T removeFirst()` : supprime et retourne le premier élément de la liste

Le classe `FileLL<T>` utilise alors un attribut de type `LinkedList<T>` pour implémenter une file

```
import java.util.*;
public class FileLL<T> implements File<T> {
    private List lg = new LinkedList<T>();
    public FileLL() { lg = new LinkedList<T>(); }
    public boolean vide() { return lg.isEmpty(); }
    public void enfiler(T o) { lg.add(o); }
    public T premier() { return lg.getFirst(); }
    public void defiler() { lg.removeFirst(); }
}
```

La classe LinkedList

La classe `LinkedList<T>` du package `java.util` implémente l'interface `Collection<T>` et possède notamment les méthodes suivantes :

- `T getFirst()` : retourne le premier élément de la liste
- `boolean add(T o)` : ajoute l'objet `o` à la fin de la liste
- `T removeFirst()` : supprime et retourne le premier élément de la liste

Le classe `FileLL<T>` utilise alors un attribut de type `LinkedList<T>` pour implémenter une file

```
import java.util.LinkedList;
public class FileLL<T> implements File<T> {
    private T[] lg=0;
    private LinkedList<T> list;
    public FileLL(){list = new LinkedList<T>();}
    public boolean vide(){return lg==0;}
    public void enfiler(T o){list.add(o); lg++;}
    public T premier(){return list.getFirst();}
    public void defiler(){list.removeFirst();lg--;}
}
```

La classe LinkedList

La classe `LinkedList<T>` du package `java.util` implémente l'interface `Collection<T>` et possède notamment les méthodes suivantes :

- `T getFirst()` : retourne le premier élément de la liste
- `boolean add(T o)` : ajoute l'objet `o` à la fin de la liste
- `T removeFirst()` : supprime et retourne le premier élément de la liste

Le classe `FileLL<T>` utilise alors un attribut de type `LinkedList<T>` pour implémenter une file

```
import java.util.LinkedList;
public class FileLL<T> implements File<T> {
    private T[] lg=0;
    private LinkedList<T> list;
    public FileLL(){list = new LinkedList<T>();}
    public boolean vide(){return lg==0;}
    public void enfiler(T o){list.add(o); lg++;}
    public T premier(){return list.getFirst();}
    public void defiler(){list.removeFirst();lg--;}
}
```

La classe LinkedList

La classe `LinkedList<T>` du package `java.util` implémente l'interface `Collection<T>` et possède notamment les méthodes suivantes :

- `T getFirst()` : retourne le premier élément de la liste
- `boolean add(T o)` : ajoute l'objet `o` à la fin de la liste
- `T removeFirst()` : supprime et retourne le premier élément de la liste

Le classe `FileLL<T>` utilise alors un attribut de type `LinkedList<T>` pour implémenter une file

```
import java.util.LinkedList;
public class FileLL<T> implements File<T> {
    private int lg=0;
    private LinkedList<T> list;
    public FileLL(){ list = new LinkedList<T>();}
    public boolean vide(){ return lg==0;}
    public void enfiler(T o){ list.add(o); lg++;}
    public T premier(){ return list.getFirst();}
    public void defiler(){ list.removeFirst();lg--; }
}
```


Introduction

Généralités

Les tableaux

Les piles

Les files

Interface

Implémentation

La classe

LinkedList

Exercices

Les listes

Exercices

exercice 10 : Ecrire l'algorithme et une méthode qui inverse une file en utilisant uniquement une pile et réécrire l'inversion d'une pile en utilisant une file.

exercice 11 : Réécrire la méthode `equals()` pour une égalité en profondeur.

exercice 12 : Réécrire la méthode `clone()` pour un clonage en profondeur.

Introduction

Généralités

Les tableaux

Les piles

Les files

Les listes

Interface

Implémentation

Le tri fusion

Exercices

Interface Liste<T>

On va fabriquer notre version des listes pour les manipuler « à la LISP » c'est-à-dire en privilégiant les fonctions récursives.

```
public interface Liste<T extends Comparable> {  
    boolean vide();  
    Liste<T> ajout(T o);  
    Liste<T> queue();  
    T tete();  
    Liste<T> fusion(Liste<T> l);  
    public Liste<T> triFusion();  
    int longueur();  
}
```

Introduction

Généralités

Les tableaux

Les piles

les files

Les listes

Interface

Implémentation

Le tri fusion

Exercices

```
public class ListeAuto<T extends Comparable<T>> implements Liste<T>{
    private class Cell{
        private T val;
        private Cell suiv;
        Cell(){ };
        Cell(T v, Cell c){ val=v; suiv=c;}
        void setVal(T v){ val=v;}
        void setSuiv(Cell c){ suiv=c;}
        T getVal(){ return val;}
        Cell getSuiv(){ return suiv;}
    }
    private Cell top=null;
    public ListeAuto(){ }
    private ListeAuto(Cell c){
        top=c;
    }

    public boolean vide(){ return top==null;}
}
```

Introduction

Généralités

Les tableaux

Les piles

les files

Les listes

Interface

Implémentation

Le tri fusion

Exercices

```
public Liste<T> ajout(T o){
    Cell c= new Cell(o, this.top);
    ListeAuto<T> l=new ListeAuto<T>(c);
    return l;}

public Liste<T> queue(){
    Cell c= top.getSuiv();
    ListeAuto<T> l=new ListeAuto<T>(c);
    return l;}

public T tete(){
    return top.getVal();}

public int longueur(){
    Liste<T> l=this;
    if (l.vide()) return 0;
    else {
        l=l.queue();
        return(1+l.longueur());
    }
}
```

Introduction

Généralités

Les tableaux

Les piles

Les files

Les listes

Interface

Implémentation

Le tri fusion

Exercices

Tri fusion

Le tri fusion est un algorithme efficace permettant de trier une liste de nombres x_1, \dots, x_n par comparaison et échange : il repose sur un post-traitement de la liste de nombres - que l'on appellera fusion - qui consiste à prendre deux listes déjà classées pour en faire une unique liste classée.

On partage donc en deux listes l_1 et l_2 la liste initiale. On trie l_1 et l_2 puis on les fusionne.

Pour trier l_1 et l_2 , on va appliquer le même principe : c'est donc un algorithme récursif.

La récursivité s'arrête lorsque la liste est vide ou de longueur 1.

Tri fusion

Le tri fusion est un algorithme efficace permettant de trier une liste de nombres x_1, \dots, x_n par comparaison et échange : il repose sur un post-traitement de la liste de nombres - que l'on appellera fusion - qui consiste à prendre deux listes déjà classées pour en faire une unique liste classée.

On partage donc en deux listes l_1 et l_2 la liste initiale. On trie l_1 et l_2 puis on les fusionne.

Pour trier l_1 et l_2 , on va appliquer le même principe : c'est donc un algorithme récursif.

La récursivité s'arrête lorsque la liste est vide ou de longueur 1.

Tri fusion

Le tri fusion est un algorithme efficace permettant de trier une liste de nombres x_1, \dots, x_n par comparaison et échange : il repose sur un post-traitement de la liste de nombres - que l'on appellera fusion - qui consiste à prendre deux listes déjà classées pour en faire une unique liste classée.

On partage donc en deux listes l_1 et l_2 la liste initiale. On trie l_1 et l_2 puis on les fusionne.

Pour trier l_1 et l_2 , on va appliquer le même principe : c'est donc un algorithme récursif.

La récursivité s'arrête lorsque la liste est vide ou de longueur 1.

Exercices

exercice 13 : Ecrire les fonctions fusion puis triFusion

exercice 14 : Ecrire l'algorithme récursif et la fonction qui renvoie le miroir d'une liste.

exercice 15 : Ecrire l'algorithme récursif et la fonction qui renvoie la concaténation de deux listes.

exercice 16 : Ecrire l'algorithme et la fonction qui teste si une liste donnée est le début d'une liste donnée en paramètre.

exercice 17 : Ecrire l'algorithme et la fonction qui teste si une liste donnée est contenue dans une liste donnée en paramètre.

Structures
arborescentes

Arbres binaires de
recherche

Les files de
priorité

Chapitre II – Structures arborescentes

- Structures arborescentes
 - Parcours en profondeur
 - Parcours en largeur
- Arbres binaires de recherche
- Les files de priorité

Chapitre II – Structures arborescentes

- Structures arborescentes
 - Parcours en profondeur
 - Parcours en largeur
- Arbres binaires de recherche
- Les files de priorité

Chapitre II – Structures arborescentes

- Structures arborescentes
 - Parcours en profondeur
 - Parcours en largeur
- Arbres binaires de recherche
- Les files de priorité

Structures arborescentes

- les arbres binaires
- Interface
- Implémentation
- Exercices
- Parcours des arbres binaires

Arbres binaires de recherche

- Les files de priorité

Structures de données arborescentes

De façon informelle, on parlera de structure de données arborescente lorsqu'une donnée a plusieurs successeurs mais un seul prédécesseur, sauf pour exactement une donnée qui est sans prédécesseur et appelée *racine*.

On s'intéressera tout d'abord aux arbres binaires dans lesquels chaque donnée a au plus deux successeurs : un fils droit et un fils gauche.

Spécification fonctionnelle

On définit le type de données abstrait ARBRE BINAIRE suivant :

type ARBRE BINAIRE

utilise type E, **type** BOOLÉEN

signature

- *initilialiser* : \rightarrow ARBRE BINAIRE
- *faire_arbre* : $(E, \text{ARBRE BINAIRE}, \text{ARBRE BINAIRE}) \rightarrow$ ARBRE BINAIRE
- *fils_gauche* : $\text{ARBRE BINAIRE} \rightarrow \text{ARBRE BINAIRE}$
- *fils_droit* : $\text{ARBRE BINAIRE} \rightarrow \text{ARBRE BINAIRE}$
- *valracine* : $\text{ARBRE BINAIRE} \rightarrow E$
- *feuille* : $\text{ARBRE BINAIRE} \rightarrow \text{BOOLÉEN}$
- *vide* : $\text{ARBRE BINAIRE} \rightarrow \text{BOOLÉEN}$

préconditions

- $\text{valracine}(a)$ est défini *si et seulement si* $\text{sivide}(a) = \text{Faux}$
- $\text{fils_gauche}(a)$ est défini *si et seulement si* $\text{sivide}(a) = \text{Faux}$
- $\text{fils_droit}(a)$ est défini *si et seulement si* $\text{sivide}(a) = \text{Faux}$

axiomes

- $\text{vide}(\text{initilialiser}) = \text{Vrai}$
- $\text{vide}(\text{faire_arbre}(E, ag, ad)) = \text{Faux}$
- $\text{valracine}(\text{faire_arbre}(E, ag, ad)) = E$
- $\text{fils_gauche}(\text{faire_arbre}(E, ag, ad)) = ag$
- $\text{fils_droit}(\text{faire_arbre}(E, ag, ad)) = ad$
- $\text{feuille}(\text{faire_arbre}(E, \text{vide}, \text{vide})) = \text{Vrai}$
- $\text{feuille}(\text{vide}) = \text{Faux}$

Structures
arborescentes

les arbres
binaires

Interface

Implémentation

Exercices

Parcours des
arbres binaires

Arbres binaires de
recherche

Les files de
priorité

Interface ArbreBinaire

```
/**
 * Interface pour le type arbre binaire générique
 * @author moi
 * @version 1.0
 */
public Interface ArbreBinaire<T>{
    /** T est un paramètre de type de cette interface */
    /** T représente le type des éléments de l' ArbreBinaire */
    boolean vide();
    boolean feuille();
    T valRacine();
    ArbreBinaire<T> filsGauche();
    ArbreBinaire<T> filsDroit();
}
```

Structures arborescentes

les arbres binaires

Interface

Implémentation

Exercices

Parcours des arbres binaires

Arbres binaires de recherche

Les files de priorité

```
public class AB<T> implements ArbreBinaire<T>{
//classe interne privée
    private class Noeud{
        private T val;
        private Noeud fg,fd;
        Noeud(T v, Noeud cg, Noeud cd){val=v; fg=cg; fd=cd;}
        T getVal(){return val;}
        Noeud getFilsGauche(){return fg;}
        Noeud getFilsDroit(){return fd;}
    }
//1 unique attribut
    private Noeud racine;
//Constructeurs
    public AB(){racine=null;}
    public AB(T item, AB<T> g, AB<T> d)
    {Noeud c=new Noeud(item, g.racine, d.racine);
     this.racine = c;}
//les méthodes de l'interface ArbreBinaire
    public boolean vide(){
        return racine == null;
    }
    public T valRacine(){
        return this.racine.getVal();
    }
}
```


Structures arborescentes

les arbres
binaires

Interface

Implémentation

Exercices

Parcours des
arbres binaires

Arbres binaires de
recherche

Les files de
priorité

```
public AB<T> filsGauche(){
    AB<T> a= new AB<T>();
    a.racine = this.racine.fg;
    return a;}
public AB<T> filsDroit(){
    AB<T> a= new AB<T>();
    a.racine = this.racine.fd;
    return a;
}
public boolean feuille() {
    if (this.vide()) return false;
    if(this.racine.fg== null && this.racine.fg==null)
        return true;
    else return false;
}
}
```

Exercices

exercice 18 : Ecrire la méthode qui renvoie la hauteur d'un arbre binaire

exercice 19 : Ecrire la méthode qui renvoie la taille d'un arbre binaire

exercice 20 : Ecrire la méthode qui renvoie le nombre de feuilles d'un arbre binaire

exercice 21 : Réécrire la méthode `equals` (mêmes valeurs aux mêmes noeuds)

Parcours des arbres binaires

On va étudier deux méthodes de visites des arbres binaires :

- en profondeur : à partir de la racine de l'arbre on va toujours le plus à gauche possible. Lorsque le sous-arbre gauche d'un noeud est vide :
 - si le sous-arbre droit n'est pas vide on va sur sa racine puis on reprend le chemin le plus à gauche possible.
 - si le sous-arbre droit est vide le noeud est une feuille et on va sur le père de ce noeud, puis à la racine de son sous-arbre droit (frère du noeud feuille) pour reprendre à partir de là le chemin le plus à gauche possible.
- en largeur : niveau par niveau, de gauche à droite en commençant par la racine ; on parle aussi de parcours hiérarchique.

Parcours en profondeur

Dans ce parcours, chaque noeud est vu trois fois, dans l'ordre suivant :

- ❶ en descente : préfixe
- ❷ en montée depuis la gauche : infixe
- ❸ en montée depuis la droite : suffixe

Ce principe de parcours en profondeur est **récuratif**.

exercice 22 : Ecrire trois méthodes d'affichage préfixe, infixe et suffixe d'un arbre binaire.