Exercices de Programmation & Algorithmique II

Série 9 – Listes chaînées : Piles et Files

(30 avril 2021)

Département d'Informatique – Faculté des Sciences – UMONS

1 Contexte : Listes chaînées

Cette séance a pour but de manipuler des listes chaînées contenant des entiers.

Considérons la classe `Node` dont le code est donné ci-dessus :

```java
public class Node{
    protected int data;
    protected Node next;

    public Node(int data){
        this(data,null);
    }

    public Node(int data, Node next){
        this.data = data;
        this.next = next;
    }

    public void setNext(Node next){
        this.next = next;
    }

    public int getData(){
        return this.data;
    }
}
```

Chaque instance de la classe `Node` possède deux attributs :
— `data` qui peut contenir une donnée (ici un entier) ;
— `next` qui fait référence au noeud suivant s'il existe et `None` sinon.

Considérons la classe `LinkedList` (i.e. liste chaînée). Une instance de la classe `LinkedList` possède trois attributs :
— `head` qui référencie le premier noeud de la liste ou `null` si la liste est vide ;
— `tail` qui référencie le dernier noeud de la liste ou `null` si la liste est vide ;
— `size` dans lequel est stocké le nombre de noeuds de la liste chaînée.

2 Le contrat

2.1 Listes chaînées

En suivant le modèle présenté en séance théorique et en vous servant des éléments présentés plus tôt, réalisez une classe `LinkedList` qui implémente les opérations demandées en essayant de minimiser la complexité et en utilisant à bon escient les exceptions `IndexOutOfBoundsException` et `IllegalStateException` (par exemple, lorsqu'une opération inappropriée est appliquée sur une liste vide).

Pour chacun des exercices demandés, nous vous conseillons vivement de faire un schéma qui représente les différentes étapes de l'opération à effectuer, et de créer quelques tests.

1 `public void` ***insertBeginning*** `(int n)`
Entrée : Un entier `n`
Sortie : Néant.
Cette méthode insère l'entier `n` au début de la liste chaînée sur laquelle la méthode est invoquée.

2 `public void` ***insertEnd*** `(int n)`
Entrée : Un entier `n`
Sortie : Néant.
Cette méthode insère l'entier `n` à la fin de la liste chaînée sur laquelle la méthode est invoquée.

3 `public void` ***insert*** `(int i, int n)`
Entrée : Deux entiers : `i` et `n`
Sortie : Néant.
Cette méthode insère l'entier `n` en `i`ème position de la liste chaînée sur laquelle la méthode est invoquée. S'il est impossible pour la méthode de réaliser sa tâche, une exception doit être générée.

4 `public int` ***get*** `(int i)`
Entrée : Un entier `i`
Sortie : L'entier présent à la `i`ème position de la liste chaînée sur laquelle la méthode est invoquée.
Cette méthode ne modifie pas la liste chaînée sur laquelle elle est invoquée. S'il est impossible pour la méthode de réaliser sa tâche, une exception doit être générée.

5 `public int` ***removeBeginning*** `()`
Entrée : Néant
Sortie : L'entier présent au début de la liste chaînée sur laquelle la méthode est invoquée.
En plus de retourner l'entier présent au début de la liste chaînée, cette méthode le supprime de la liste. S'il est impossible pour la méthode de réaliser sa tâche, une exception doit être générée.

6 `public int` ***removeEnd*** `()`
Entrée : Néant
Sortie : L'entier présent à la fin de la liste chaînée sur laquelle la méthode est invoquée.
En plus de retourner l'entier présent à la fin de la liste chaînée, cette méthode le supprime de la liste. S'il est impossible pour la méthode de réaliser sa tâche, une exception doit être générée.

7 `public int` ***remove*** `(int i)`
Entrée : Un entier `i`
Sortie : L'entier présent à la `i`ème position de la liste chaînée sur laquelle la méthode est invoquée.
En plus de retourner l'entier présent à la position demandée, cette méthode le supprime de la liste. S'il est impossible pour la méthode de réaliser sa tâche, une exception doit être générée.

8 `public String` ***toString*** `()`
Entrée : Néant
Sortie : une représentation de la liste chainée sous la forme d'une chaine de caractères. Cette chaine se compose comme suit : elle débute par un crochet ouvert, se termine par un crochet fermé et les éléments de la liste chainée sont séparés par une virgule suivie d'un espace. Par exemple, si la liste contient les éléments `1, 2, 3`, un appel à cette méthode retourne la chaine de caractères `'[1, 2, 3]'`.

9 `public LinkedList` ***clone*** `()`
Entrée : Néant
Sortie : Une copie de la liste chaînée. Chaque noeud de la liste chaînée doit également être "copié" afin qu'il n'y ait pas d'effet de bord lors de futures opérations appliquées sur cette copie.

2.2 Pile – LIFO (Last In, First Out)

Une Pile est une structure de données dont l'ajout et la suppression de données sont régis par la contrainte suivante : « Dernier entré, premier sorti ».

Veuillez créer une classe `Lifo` qui contiendra les méthodes d'interactions avec une Pile qui sont énumérées ci-après.

Une Pile peut être implémentée en utilisant une liste chaînée comme structure de données sous-jacente. Veillez à utiliser la classe `LinkedList` et les méthodes implémentées précédemment. Exploitez les propriétés des listes chaînées pour obtenir la meilleure complexité possible.

10 `public void` ***push*** `(int n)`
Entrée : Un entier `n`
Sortie : Néant.
Cette méthode ajoute l'entier `n` à la pile sur laquelle la méthode est invoquée.

11 `public int` ***top*** `()`
Entrée : Néant
Sortie : L'entier que l'on peut obtenir de la pile sur laquelle la méthode est invoquée.
Cette méthode ne supprime aucun élément. S'il est impossible pour la méthode de réaliser sa tâche, une exception doit être générée.

12 `public int` ***pop*** `()`
Entrée : Néant
Sortie : L'entier que l'on peut obtenir de la pile sur laquelle la méthode est invoquée.
Cette méthode supprime l'entier qui est retourné. S'il est impossible pour la méthode de réaliser sa tâche, une exception doit être générée.

2.3 File – FIFO (First In, First Out)

Une File est une structure de données dont l'ajout et la suppression de données sont régis par la contrainte suivante : « Premier entré, premier sorti ».

Veuillez créer une classe `Fifo` qui contiendra les méthodes d'interactions avec une File qui sont énumérées ci-après.

Une File peut être implémentée en utilisant une liste chaînée comme structure de données sous-jacente. Veillez à utiliser la classe `LinkedList` et les méthodes implémentées précédemment. Exploitez les propriétés des listes chaînées pour obtenir la meilleure complexité possible.

13 `public void` ***enqueue*** `(int n)`
Entrée : Un entier `n`
Sortie : Néant.
Cette méthode ajoute l'entier `n` à la file sur laquelle la méthode est invoquée.

14 `public int` ***first*** `()`
Entrée : Néant
Sortie : L'entier que l'on peut obtenir de la file sur laquelle la méthode est invoquée.
Cette méthode ne supprime aucun élément. S'il est impossible pour la méthode de réaliser sa tâche, une exception doit être générée.

15 `public int` ***dequeue*** `()`
Entrée : Néant
Sortie : L'entier que l'on peut obtenir de la file sur laquelle la méthode est invoquée.
Cette méthode supprime l'entier qui est retourné. S'il est impossible pour la méthode de réaliser sa tâche, une exception doit être générée.

3 Exercices complémentaires

Veuillez implémenter les opérations suivantes sur la classe `LinkedList`.

★☆☆ 16 `is_empty()` : retourne vrai si la liste est vide, faux sinon ;

★★☆ 17 `count()` : retourne le nombre d'éléments présents dans la liste ;

★★☆ 18 `display()` : affiche le contenu de la liste ;

★☆☆ 19 `clear()` : vide la liste ;

★★☆ 20 `contains(int n)` : retourne vrai si la liste contient l'élément `n` ;

★★☆ 21 `mutiply(int n)` : retourne une nouvelle liste où chaque élément de la liste actuelle a été multipliée par `n` ;