Numérique et Sciences Informatiques
Chapitre II - Structures de données linéaires

Classe de terminale

I. Structure de données, interface et implémentation

I.1. Généralités

Un **type abstrait de données** est la spécification d'un ensemble de données et de l'ensemble des opérations qu'on peut effectuer sur elles. Il s'agit donc d'un cahier des charges qu'une structure de données doit mettre en oeuvre. Le type abstrait de données représente un concept et la structure de données sa réalisation.

Quelques exemples :

- le concept du transport :
  - une opération possible est :
    - le déplacement.
  - La réalisation du concept se fait par :
    - un vélo
    - à pieds
    - en train
    - etc.
- le concept des entiers :
  - les opérations possibles sont :
    - l'addition
    - la soustraction
    - la multiplication
    - etc.
  - La réalisation peut être :
    - la représentation avec des allumettes
    - la représentation binaire
    - la représentation décimale
    - etc.
- Le concept des listes : une liste est une collection d'éléments. Elle peut être vide ou composée d'un élément suivi d'une suite d'éléments.
  - Les opérations possibles sont :
    - l'ajout en tête
    - récupérer la tête
    - récupérer le reste
    - la **vacuité** (vérifier si la liste est vide).
  - Les réalisation possibles sont :
    - des couples : tête, reste
    - des tableaux (séquence finie de données. celle-ci sont en général placées à côté dans la mémoire)

Une **structure de données** est une manière d'organiser et de stocker les données auxquelles on veut accéder plus tard.
Elle possède une **interface** qui consiste en un ensemble de **primitives** , c'est-à-dire des opérations (ou fonctions) que l'on peut appliquer sur ces données.

Exemple

Un exemple de structure de données que nous avons vu en première est le tableau.
Son interface possède différentes primitives dont l'ajout d'une donnée (avec la méthode `.append()` par exemple), la suppression d'une donnée (avec la méthode `.pop()` par exemple), la réorganisation des données (le tri), etc.

Suivant la structure de données choisie, on ne pourra pas faire les mêmes opérations. Et une même opération faite sur des structures de données différentes n'aura pas le même coût (complexité).

C'est pourquoi, suivant les opérations que l'on souhaite utiliser dans un programme, on choisira la structure de données la plus adaptée afin que les coûts soient minimes.

La complexité des primitives dépend aussi de leur **implémentation** , c'est-à-dire de la façon dont elles ont été codées dans le langage de programmation choisi.

I.2. Structures de données linéaires

Une structure de données linéaire est une collection finie d'éléments qui sont stockés en mémoire de manière séquentielle.
Les structures de données linéaires que nous allons étudier sont :

- les Listes
- les Piles
- les Files
- les dictionnaires

On peut implémenter une même structure de données de plusieurs façons : en utilisant la programmation impérative ou orientée objet par exemple.

II. Les Listes

Définition

Une Liste d'éléments d'un ensemble E, est :

- soit la Liste vide
- soit un couple (x, ℓ) où $x \in E$ et ℓ est une Liste d'éléments de E. x est appelé la **tête** et ℓ le **reste**.

Exemple

Soit ℓ_1 la Liste composée des nombres $3, 1, 4, 1$ et 5.
$\ell_1 = (3, \ell_2)$ avec ℓ_2 la Liste composée des nombres $1, 4, 1$ et 5.
$\ell_1 = (3, (1, \ell_3))$ avec ℓ_3 la Liste composée des nombres $4, 1$ et 5.
etc.
$\ell_1 = (3, (1, (4, (1, (5, ()))))))$

Interface des Listes

Plusieurs opérations principales (appelées aussi primitives) se dégagent de la définition précédente :

- Le **constructeur** doit permettre de produire soit une Liste vide, soit une Liste à partir de deux arguments : un élément et une autre Liste (éventuellement vide).
- un **sélecteur** qui permet d'accéder à la tête de la Liste .
- un sélecteur qui permet d'accéder au reste de la Liste .
- un **prédicat** qui teste la vacuité d'une Liste (le fait qu'elle soit vide) est souvent utile.

Remarque

Il est possible d'implémenter des Listes avec des listes chaînées ou doublement chaînées par exemple.
Une Liste chaînée est composée de maillons. Chaque maillon est :

- soit un maillon vide
- soit un maillon composé d'une donnée et d'un pointeur vers le maillon suivant.

Une Liste doublement chaînée est composée de maillons. Chaque maillon est :

- soit un maillon vide
- soit un maillon composé d'une donnée et de deux pointeurs : un vers le maillon suivant et un autre vers le maillon précédent.

Implémentation des Listes

Voir TD

III. Les Piles

Définition

Une **Pile** (Stack en anglais) est une structure de données linéaire agissant comme une pile d'assiette : on peut ajouter et retirer des éléments de la structure, mais pour retirer un élément, il faut d'abord enlever tous les élément empilés sur lui. Elle est aussi appelée LIFO (Last In, First Out).

Interface de la pile

La Pile possède 4 opérations primitives :

- un constructeur de Pile vide
- l'empilement d'un élément sur une Pile
- le dépilement du sommet de la Pile
- un prédicat qui teste la vacuité de la Pile

On peut ajouter un sélecteur qui permet lire l'élément du haut de la Pile

Implémentation des Piles

Voir TD

IV. Les Files

Définition

Une **File** (Queue en anglais) est une structure de données linéaire agissant comme une file d'attente : on peut ajouter et retirer des éléments de la structure, mais les premiers éléments ajoutés seront toujours les premiers à sortir. Elle est aussi appelée FIFO (First In, First Out)

Interface de la file

La File possède 4 opérations primitives :

- un constructeur de File vide
- l'ajout d'un élément à File
- la suppression du premier élément de la File
- un prédicat qui teste la vacuité de la File

On peut ajouter un sélecteur qui permet de lire l'élément de début de File

Implémentation des Files

Voir TD

V. Les Dictionnaires

Définition

Un **Dictionnaire** est une collection qui utilise des clés plutôt que des index (comme pour les listes). On dit qu'un Dictionnaire contient des couples clé:valeur.

Exemple

```
>>> mon_dico={'nom': 'Wayne', 3: 'bidule', 'tel': 123, 32: 54}
>>> mon_dico[3]
 'bidule'
>>> mon_dico[32]
 54
>>> mon_dico['nom']
 'Wayne'
>>> mon_dico['tel']
 123
```

Interface du Dictionnaire

Le Dictionnaire possède 5 opérations primitives :

- création d'un Dictionnaire vide
- ajout d'un élément au dictionnaire (une clé et la valeur associée)
- Modification d'un élément : la valeur d'une clé donnée
- suppression d'un élément du Dictionnaire (la clé et la valeur associée)
- récupérer la valeur d'un élément de clé connue

Comparaison des structures de données Dictionnaire et Liste

Voir TD