

DIU-EIL BLOC 4

STRUCTURES DE DONNÉES

Romuald THION, Nicolas PRONOST

`https://forge.univ-lyon1.fr/diu-eil/bloc4`

27 mai 2020

Plan

1 Présentation générale

Positionnement programme NSI

Contenus	Capacités attendues	Commentaires
Structures de données, Interface et Implémentation.	Spécifier une structure de données par son Interface. Distinguer Interface et Implémentation. Écrire plusieurs Implémentations d'une même structure de données.	L'abstraction des structures de données est Introduite après plusieurs implémentations d'une structure simple comme la file (avec un tableau ou avec deux piles).
Vocabulaire de la programmation objet : classes, attributs, méthodes, objets.	Écrire la définition d'une classe. Accéder aux attributs et méthodes d'une classe.	On n'aborde pas ici tous les aspects de la programmation objet comme le polymorphisme et l'héritage.

Positionnement programme NSI

Listes, piles, files : structures linéaires. Dictionnaires, index et clé.	Distinguer des structures par le jeu des méthodes qui les caractérisent. Choisir une structure de données adaptée à la situation à modéliser. Distinguer la recherche d'une valeur dans une liste et dans un dictionnaire.	On distingue les modes FIFO (<i>first in first out</i>) et LIFO (<i>last in first out</i>) des piles et des files.
--	--	--

Les autres structures (arbres, graphes) étant dans le bloc 5

Définition générale

Extrait de Wikipedia

*In computer science, a data structure is a data organization, management, and storage format that enables **efficient access and modification**. [...] Data structures serve as the basis for abstract data types (ADT). The ADT defines the **logical form** of the data type. The data structure **implements the physical form** of the data type.*

Type abstrait VERSUS structure de données

On peut différencier

- Le **type abstrait** : les opérations et les propriétés qui les lient
 - notion d'interface (en POO), de spécification, d'API
 - *e.g., `p.push(x)` ; `p.pop()` laissent une pile `p` inchangée*
- La **structure de données** : une implantation physique qui réalise cette spécification
 - ce qui va compter c'est la performance (évaluée en moyenne ou au pire cas) des opérations
 - *e.g., un tableau dynamique, une liste doublement chaînée, etc.*

Interface de liste (simplement chaînée)

```
class Liste :  
    """ Liste chaînée non circulaire """  
    def __init__( self ) :  
        pass  
    def est_vide( self ) :  
        pass  
    def vider( self ) :  
        pass  
    def taille( self ) :  
        pass  
    def ajouter_element( self , contenu ) :  
        pass  
    def acceder_element( self , position ) :  
        pass  
    # etc.
```

./Liste.py

Type abstrait VERSUS structure de données

Attention : *pour un type abstrait, on peut avoir plusieurs implantation avec des structures de données différentes !*

Attention : pour un type abstrait, on « s'attend » souvent à avoir les performances d'une certaine structure, ce qui peut-être contre-intuitif
Exemple typique : le type `list` de Python

Type abstrait VERSUS structure de données

Attention : *pour un type abstrait, on peut avoir plusieurs implantation avec des structures de données différentes !*

Attention : *pour un type abstrait, on « s'attend » souvent à avoir les performances d'une certaine structure, ce qui peut-être contre-intuitif*
Exemple typique : le type `list` de Python

Tableau de synthèse

Réaliser un logiciel consiste à, notamment, choisir **les structures de données adaptées** pour résoudre le problème posé.

Data Structure	Time Complexity							
	Average				Worst			
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion
Array	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
Stack	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
Queue	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
Singly-Linked List	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
Doubly-Linked List	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
Skip List	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
Hash Table	N/A	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	N/A	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
Binary Search Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$

FIGURE – <https://www.bigocheatsheet.com/>

Les contenus sur cette partie du programme

`https://forge.univ-lyon1.fr/diu-eil/bloc4/-/tree/master/2_structures_de_donnees`

En TD/TP : vous allez réaliser les principales structures de données en programmation orientée objet **sans utiliser les bibliothèques** : on fait tout *from scratch*