DIU-EIL Bloc 4
Structures de données

Romuald THION, Nicolas PRONOST

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

27 mai 2020

Plan

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 *(first in first out)* et LIFO *(last in first out)* 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 chainée, etc.*

Interface de liste (simplement chainée)

```
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*

Tableaux :

. Si on fait beaucoup d'insertion => à un moment il faut étendre

Tables de hash : (dictionnaires)

. Très bien pour l'accès direct mais coût mémoire

Stack : Bien pour désérécursiver les algorithmes

Queue : Bien pour les messageries, les files d'attente

Binary search tree.

Par exemple on a une structure de données les B-arbres qui combinent arbre binaire de recherche et liste doublement chaînée => permet

de résoudre les pbs de recherche de données

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	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(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*