Structures de données abstraites

I Structure de données

Définition 1 : Structure de donnée

Une structure de données est un type muni d'opérations.

Exemples 1 : Les listes chaînées et les tableaux constituent deux exemples de structures de données.

Elles jouent un rôle à part car ce sont des structures de données de très bas niveau.

Voyons néanmoins comment on pourrait les expliciter.

— Pour la structure "liste chaînée", les opérations sont :

1. `nil` : créer une liste vide ;
2. `cons` : créer une liste à partir d'une tête et d'une queue ;
3. `tete` : obtenir la tête d'une liste ;
4. `queue` : obtenir la queue d'une liste.

— Pour la structure "tableau", les opérations sont :

1. `initialiser` : créer un tableau de longueur donnée ;
2. `longueur` : obtenir la longueur du tableau ;
3. `acceder` : obtenir l'élément d'indice donné du tableau ;
4. `modifier` : modifier l'élément d'indice donné du tableau.

Définition 2

On distingue trois familles d'opérations structurelles : les constructeurs qui permettent l'initialisation d'une structure, les accesseurs pour récupérer une valeur et les transformateurs pour modifier l'état de la structure ou obtenir une nouvelle structure.

Exemple 2 : Sur les deux structures précédentes :

Remarque 1 : Il n'y a pas unicité de l'implémentation d'une structure de données.

Définition 3

Une structure de données est dite mutable (on peut aussi dire : *modifiable*, ou *impérative*) lorsque ses objets peuvent être modifiés en place par les opérations de la structure.

Une structure de données est dite immuable (ou *persistante*) lorsque ses opérations ne la modifient pas en place (mais peuvent renvoyer nouvelles instances de la structure).

Exemple 3 : Pour les les deux structures précédentes :

A priori, **toutes les structures de données se déclinent en deux variantes**, une modifiable et une persistante.

II D'autres structures de données au programme

II.1 Piles

C'est le type des piles d'assiettes qui s'accumulent à côté de l'évier en attendant d'être lavées. C'est une structure LIFO (last in first out).

Exemple d'utilisation de pile en informatique :

Les opérations sont :

1. `creer_pile_vide` : créer une pile vide ;
2. `est_vide` : teste si une pile est vide ;
3. `empiler` : ajoute un élément sur le sommet de la pile ;
4. `dépiler` : enlève l'élément du sommet de la pile et le retourne.

Deux variantes pour déplier : on peut avoir une structure modifiable et dépiler modifie la pile, ou bien une structure persistante et dépiler retourne le couple (élément au sommet, pile restante) ou bien se décline en deux opérations.
Cela donne les signatures suivantes :

Pour la version immuable :

Pour la version impérative :

Remarque 2 : Les piles immuables ne sont autre que
Caml propose déjà une implémentation des piles modifiables (type `Stack`). On l'explorera dans le TD05.

II.2 Files

C'est le type des files d'attente. C'est une structure FIFO (first in first out).

Exemples d'utilisation de file en informatique :

Les opérations sont :

1. `creer_file_vide` : créer une file vide ;
2. `est_vide` : teste si une file est vide ;
3. `enfiler` : ajoute un élément à la fin de la file ;
4. `défiler` : enlève l'élément du début de la file et le retourne.

Deux variantes pour défiler : on peut avoir une structure modifiable et défiler modifie la file, ou bien une structure persistante et défiler retourne le couple (ou se décline en deux opérations).

Exercice 1. Écrire les types précis de toutes les opérations pour chaque version.

II.3 Dictionnaires (ou tableaux associatifs)

C'est la structure vue en Python. À chaque clé est associée une valeur.

Les opérations sont :

1. `creer_dico_vide` : créer un dictionnaire vide ;
2. `est_vide` : teste si un dictionnaire est vide ;
3. `inserer` : ajoute un couple (clé,valeur) au dictionnaire ;
4. `supprimer` : retire un couple (clé,valeur) au dictionnaire ;
5. `modifier` : modifie la valeur d'une clé.

Exercice 2. Écrire les types précis de toutes les opérations pour les versions impérative et modifiable.

II.4 Files de priorité

C'est le type des choses-à-faire. Chaque chose à faire vient avec une date à laquelle elle doit être faite et on traite prioritairement les choses à faire les plus urgentes.

Les opérations sont :

1. `creer_fp_vide` : créer une file de priorité vide ;
2. `est_vide` : teste si une file de priorité est vide ;
3. `inserer` : ajoute un élément (avec sa priorité) à la file de priorité ;
4. `retire_prioritaire` : retire un élément avec la plus grande priorité et le retourne.

Là encore, deux variantes pour la dernière opération.

Exercice 3. Écrire les types précis de toutes les opérations pour chaque version.

III Manipulation d'une structure de données abstraite

L'idée ici est de manipuler une structure de données

— *y compris sans qu'on l'ait implémentée, et même,*

— *y compris sans qu'on ait la moindre idée de comment l'implémenter.*

Un exemple est dans le sujet des Mines de 2016 où on nous fait manipuler plusieurs structures de données abstraites, en imaginant qu'elles ont été implémentées, et sans information sur cette implémentation :

Épreuve d'informatique 2016

On va utiliser dans la suite de l'exercice un type de données `dictionnaire` qui permet de stocker des couples formés d'une chaîne de caractères (une *clef*) et d'un entier (une *valeur*). On dit que le dictionnaire *associe* la valeur à la clef. À chaque clef présente dans le dictionnaire est associée une seule valeur. Les fonctions suivantes sont supposées être prédéfinies :

— `dictionnaire_vide : unit -> dictionnaire.`
L'appel `dictionnaire_vide ()` crée un nouveau dictionnaire vide.

— `ajoute : string -> int -> dictionnaire -> dictionnaire.`
L'appel `ajoute clef valeur dict` renvoie un nouveau dictionnaire identique au dictionnaire *dict*, sauf qu'un couple (*clef*, *valeur*) y a été ajouté. Cette fonction s'exécute en temps $O(\log n)$ où n est le nombre d'entrées du dictionnaire.

— `contient : string -> dictionnaire -> bool.`
L'appel `contient clef dict` renvoie un booléen indiquant s'il y a un couple dont la clef est *clef* dans le dictionnaire *dict*. Cette fonction s'exécute en temps $O(\log n)$ où n est le nombre d'entrées du dictionnaire.

— `valeur : string -> dictionnaire -> int.`
L'appel `valeur clef dict` renvoie la valeur associée à la clef *clef* dans le dictionnaire *dict*. Cette fonction s'exécute en temps $O(\log n)$ où n est le nombre d'entrées du dictionnaire. Cette fonction ne peut être appelée que si la clef *clef* est présente dans le dictionnaire.

On suppose pour la suite de l'exercice que le type de données `dictionnaire` est prédéfini ; on ne demande pas de l'implémenter.

Faute d'information, lorsqu'on manipule une structure de données sans savoir comment elle a été implémentée, le modèle choisi pour les calculs de complexité est de considér chaque d'opération structurelle comme une opération élémentaire.

Exemple 4 : Prenons un autre exemple : le **parcours en largeur d'un arbre**.

Rappel de l'algorithme :

- on considère une file d'arbres qui initialement a un seul élément, l'arbre que l'on souhaite parcourir, et une liste d'éléments initialement vide, correspondant à la liste du parcours ;
- lorsqu'on rencontre un arbre non vide dans la file, on ajoute l'étiquette de son nœud dans la liste des éléments, puis on rajoute **à la fin** de la file d'arbres les fils gauche et droit de l'arbre ;
- lorsque la file d'arbres est vide, on retourne (le renversement de) la liste d'éléments.

Caml propose déjà une implémentation des files (type `Queue`) : on continuera d'explorer ce type dans le TD05, mais commençons tout de suite en implémentant le parcours en largeur. Identifions tout d'abord les opérations structurelles.

IV Implémentation d'une SDD

L'idée ici est de coder une structure de données ; soit en utilisant d'autres strutures de données déjà codées, soit en utilisant les mécanismes fondamentaux de création de type (ou les deux).

Exemple 5 : La structure de données "ensemble de mots" peut s'implémenter par des listes ou des tableaux de chaînes de caractères (c'est très inefficace) ou par des *tries* (c'est mieux, et c'est pour ça qu'on a choisi cette implémentation).

L'idée est de faire en sorte que les opérations structurelles **soient les plus rapides possibles**. En effet, lorsqu'on manipule une structure de données sans savoir comment elle est implémentée, le modèle choisi pour les calculs de complexité est de compter le nombre d'opérations structurelles effectuées.

IV.1 Piles et files

Pour les files, cf TP03 :

- Les files immuables peuvent être implémentées par un couple de listes. La complexité *amortie* des opérations est en $O(1)$, ce qui signifie qu'une opération qui coûte $O(n)$ sera suivie par au moins n opérations en $O(1)$, conduisant à une complexité moyenne en $O(1)$ sur cette sucession d'opérations.
- Les files modifiables peuvent être implémentées par un tableau circulaire. La complexité des opérations est systématiquement en $O(1)$, mais la file aura une taille maximale qui est celle du tableau.

Pour les piles :

- On l'a déjà vu, les piles immuables ne sont autres que les listes !
- Les piles modifiables peuvent être implémentées par un tableau circulaire sur le même modèle que les files (il suffit de tourner dans l'autre sens).

IV.2 Dictionnaires et files de priorité

Pour les dictionnaires, cf TP04 et TP05 :

- La version modifiable des dictionnaires peut être implémentée par une table de hashage (TP04).
- La version immuable des dictionnaires peut être implémentée par des arbres binaires de recherche (TP05).

Pour les files de priorité : pour la version immuable comme pour la version mutable, on peut utiliser des tas (implémentés par des tableaux pour la version mutable). Mais pour cela, il faudra attendre le programme de seconde année...