Structures de données abstraites

I Structure de données

Définition (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 cependant 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.

Exemple 2
Vous avez essentiellement implémenté la structure de données "polynôme" en TP d'informatique de tronc commun.
Cette implémentation repose sur la représentation dense des polynômes.

On retient déjà :

Remarque 1

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

Définition .

Une structure de données est dite modifiable (ou impérative) lorsque ses objets peuvent être modifiés en place par les opérations de la structure.

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

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

II Les structures de données du programme

II.1 Arbres

On les a déjà vus. Les opérations sont :

1. `creer_feuille` : crée un arbre composée d'une seule feuille ;
2. `creer_nœud` : crée un arbre composée d'un nœud, 'un fils gauche et d'un fils droit ;
3. `fils_gauche` : retourne le fils gauche de l'arbre ;
4. `fils_droit` : retourne le fils droit de l'arbre ;
5. `etiquette` : retourne l'étiquette du nœud de l'arbre, ou de sa feuille si c'est une feuille.

II.2 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).

Exemples d'utilisation de pile en informatique :

— Pile d'appels d'un runtime

— Analyse syntaxique (parsers)

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 lélément 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 :

```
creer_liste_vide : unit -> 'a pile
est_vide : 'a pile -> bool
empiler : 'a -> 'a pile -> 'a pile
dépiler : 'a pile -> 'a * 'a pile
```

Pour la version impérative :

```
creer_liste_vide : unit -> 'a pile
est_vide : 'a pile -> bool
empiler : 'a -> 'a pile -> unit
dépiler : 'a pile -> 'a
```

Remarque 2

Les piles immuables ne sont autre que .....

II.3 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 :

.....

- file d'attente d'impression pour une imprimante partagée (ou pas) ;
- implémentation efficace d'un parcours en largeur ;
- ...

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 lélément du début de la file et le retourne.

Deux variantes pour déflier : 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.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 2. Écrire les types précis de toutes les opérations pour chaque version.

II.5 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 une file de priorité vide ;
2. `est_vide` : teste si une file de priorité 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. ;

6. `accéder` : accéder à une valeur à partir d'une clé. ;
7. `a_clé` : tester si une clé est dans le dictionnaire ;

Exercice 3. Écrire les types précis de toutes les opérations pour les versions impérative et modifiable.

III 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 vus avant les vacances (ou les deux).

Exemple 3 On a déjà implémenté une structure de données du programme : les files de priorité. Il suffit d'utiliser des tas (codable par des listes ou des tableaux)

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

Sur l'exemple des files de priorité :

1. Une première idée est de les implémenter par des listes. Pb : le retrait de l'élément de priorité maximale est en $O(n)$
2. Avec des listes triées : l'insertion coûte cher.
3. Pour les tas, on peut faire ces deux ops en $O(\log n)$

Pour les autres exemples du programme :

1. Piles : déjà dit pour la version immuable, ce sont simplement des .....·
2. Files : on verra en TP deux méthodes ; en utilisant un tableau circulaire (pour la version persistante), ou en utilisant deux listes chaînées (pour la version immuable).
3. Dictionnaires : on verra deux méthodes ; en utilisant des tables de hachage (pour la version persistante), ou en utilisant des arbres binaires de recherche (pour la version immuable). Cela, la seconde est seulement au programme en seconde année, et pour la première, ce sera sans doute le tout premier TP de spé.

IV Manipulation d'une structure de donnée 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.

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 largueur d'un arbre**.

Rappel de l'algorithme :

- on considère une liste 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 liste, on ajoute l'étiquette de son nœud dans la liste des éléments, puis on rajoute **à la fin** de la liste d'arbres les fils gauche et droit de l'arbre ;

- lorsque la liste d'arbres est vide, on retourne la liste d'éléments.

Faisons, puis faisons mieux :