Algorithmique

Types de données abstraits

Florent Hivert

Mél : Florent.Hivert@lri.fr
Page personnelle : http://www.lri.fr/~hivert

Spécification des structures de données

Dans un programme de bonne taille, il faut identifier et spécifier les structure de données très tôt dans le processus de développement.

Retenir (Spécification)

Une spécification est une définition formelle du comportement d'une certaine structure de donnée.

- contrat entre l'utilisateur et l'implanteur de la structure de données ;
- dit ce que doit faire la structure de données ;
- ne dit pas comment faire (choix de l'implantation) ;
- précis et rigoureux ;
- doit éviter de poser des contraintes d'implantations.

Spécification d'un type de données abstrait

Un type de données abstrait (TDA, abstract data type ou ADT en anglais) est un ensemble d'éléments muni d'opérations agissant sur ses éléments.

Définition

*Une **spécification algébrique** est composée de :*

- *la définition d'un certain nombre d'opérations avec leurs signatures (ensemble de départ et d'arrivée)*
- *des axiomes qui définissent le comportement*
- *des préconditions si les opérations sont partiellement définies.*

Implantation des TDA

Lors du passage à la programmation :

- Les TDA sont implantés par des types de données (classes si programmation orientée objets) ;
- Les opérations sont implantés par des sous-programmes (méthode si POO).
- La spécification algébrique peut être implantés par plusieurs types ;
- Mode mutatif ou constructifs.

Exemple : les tableaux

égalité de T notée $=_T$, n est un entier positif

Définition (**Type abstrait** Tab(n, T))

Opérations :

- Create : $(t_1, \ldots, t_n) \to \text{Tab}(T, n)$
- Get : $\text{Tab}(T, n) \times \text{Entier} \to T$
- Set : $\text{Tab}(T, n) \times \text{Entier} \times T \to \text{Tab}(n, T)$

Préconditions :

- $\text{Get}(t, i)$ *défini seulement si* $0 \leq i < n$ *avec* $t \in \text{Tab}(n, T)$.

Axiomes : *pour tout* $x, y \in T$, $i, j \in \text{Entier}$, $t \in \text{Tab}(n, T)$

- *si* $i \neq j$ *alors* $\text{Set}(\text{Set}(t, i, x), j, y) = \text{Set}(\text{Set}(t, j, y), i, x)$;
- *si* $i = j$ *alors* $\text{Set}(\text{Set}(t, i, x), i, y) = \text{Set}(t, i, y)$;
- $\text{Get}(\text{Create}(t_1, \ldots, t_n), i) = t_i$;
- $\text{Get}(\text{Set}(t, i, x), i) =_T x$.

Les tables d'associations

Définition du type $\mathrm{Tab}(U, T)$ (U et T sont deux types).

- égalité de T notée $=_T$, égalité de U notée $=_U$,

Opérations :

- $\mathrm{Create} : \{\} \to \mathrm{Tab}(U, T)$
- $\mathrm{Get} : \mathrm{Tab}(T, n) \times U \to T \cup \{\mathrm{Erreur}\}$
- $\mathrm{Set} : \mathrm{Tab}(T, n) \times U \times T \to \mathrm{Tab}(T, n)$

Préconditions : aucune

Axiomes : pour tout $x, y \in T, \quad i, j \in U$,

- si $i \neq_U j$ alors $\mathrm{Set}(\mathrm{Set}(t, i, x), j, y) = \mathrm{Set}(\mathrm{Set}(t, j, y), i, x)$;
- si $i =_U j$ alors $\mathrm{Set}(\mathrm{Set}(t, i, x), i, y) = \mathrm{Set}(t, i, y)$;
- $\mathrm{Get}(\mathrm{Create}(n), i) = \mathrm{Erreur}$;
- $\mathrm{Get}(\mathrm{Set}(t, i, x), i) =_T x$.

Type Abstrait = contrat (1)

Retenir

- *En phase de conception,*
  - *ne pas être encombré de détails techniques ;*
  - *ne pas être influencé par les contraintes secondaires*
  - *repousser les décisions de représentation physique au étapes ultérieures.*
- *Type de données abstrait :*
  - *quelles sont les opérations*
  - *quelles sont les propriétés intrinsèques des opérations*

Type Abstrait = contrat (2)

Retenir

- *Pour l'implanteur du type*
  - *vérifier que la réalisation respecte les propriétés du contrat*
  - *guide à la mise au point (certifications, tests)*
- *Pour l'utilisateur du type*
  - *connaître a priori le comportement sans savoir les détails de l'implantation*
  - *vérifier la validité de son utilisation*

Implantation d'un TDA

Retenir

- *Transformer la notation fonctionnelle :*
  - *opérations de construction : T est résultat sans être argument.*
  - *opérations d'accès : T est argument sans être résultat.*
  - *opérations de transformation : T est argument et résultat. On modifie l'argument par «effet de bord» (mode mutatif).*
- *Représentation en mémoire :*
  - *Cachée pour pouvoir la modifier et l'optimiser ;*
  - *structures séquentielles (tableaux) ou linéaires (listes chaînées).*

Types abstraits et bibliothèque standard

La plupart des langages modernes fournissent les un certain nombre de TDA de base, soit dans le langage lui même, soit dans des bibliothèques.

Exemples de TDA courant :

- tableau
- table d'association, dictionnaire, table de hachage
- liste récursive
- file d'attente (FIFO)
- pile (LIFO)
- ensemble

Exemple de TDA : les piles (1)

Retenir (Spécification informelle :)

Une pile est une liste linéaire d'objets où les consultations les insertions et les suppressions se font du même coté. Dernier arrivé, premier servi.

Vocabulaire basé sur une représentation verticale : empiler, dépiler, sommet, pile vide (en anglais : Last-in-first-out (LIFO), stack, push, pop, top, empty).

Applications : compilateur, gestion des appels, évaluation d'expressions, parcours en profondeur.

Exemple de TDA : les piles (1)

Retenir (Spécification informelle :)

Une pile est une liste linéaire d'objets où les consultations les insertions et les suppressions se font du même coté. Dernier arrivé, premier servi.

Vocabulaire basé sur une représentation verticale : empiler, dépiler, sommet, pile vide (en anglais : Last-in-first-out (LIFO), stack, push, pop, top, empty).

Applications : compilateur, gestion des appels, évaluation d'expressions, parcours en profondeur.

Exemple de TDA : les piles (1)

Retenir (Spécification informelle :)

Une pile est une liste linéaire d'objets où les consultations les insertions et les suppressions se font du même coté. Dernier arrivé, premier servi.

Vocabulaire basé sur une représentation verticale : empiler, dépiler, sommet, pile vide (en anglais : Last-in-first-out (LIFO), stack, push, pop, top, empty).

Applications : compilateur, gestion des appels, évaluation d'expressions, parcours en profondeur.

Exemple de TDA : les piles (2)

Définition (**Type abstrait** Pile(T))

Opérations :

- PileVide : $\{\} \rightarrow \text{Pile}(T)$
- EstVide : $\text{Pile}(T) \rightarrow \text{Booleen}$
- Empiler : $T \times \text{Pile}(T) \rightarrow \text{Pile}(T)$
- Depiler : $\text{Pile}(T) \rightarrow T \times \text{Pile}(T)$

Préconditions :

- Depiler(p) *valide seulement si non* EstVide(p)

Axiomes :

- $\text{EstVide}(\text{PileVide}()) = \text{VRAI}$
- $\text{EstVide}(\text{Empiler}(x, p)) = \text{FAUX}$
- $\text{Depiler}(\text{Empiler}(x, p)) = (x, p)$

Théorie des Piles

En utilisant les axiomes, on peut montrer que

Proposition (Formes canoniques)

Soit $p \in \text{Pile}(T)$ *une pile. Il existe un unique* n *et un unique* n*-uplet* $(a_1, \ldots, a_n) \in T^n$ *tel que*

$$p = \text{Empiler}(a_n, \text{Empiler}(a_{n-1}, \ldots (\text{Empiler}(a_1, \text{PileVide}()) \ldots)) .$$

Implantations possibles des piles :

- longueur + tableaux d'éléments

```
#define MAXPILE ...
struct pile_s {
    int taille;
    elem valeurs[MAXPILE];
};
```

- liste chaînée

Note : en mutatif, on aura également besoin des opérations de copie et de destruction.

Implantations possibles des piles :

- longueur + tableaux d'éléments

```
#define MAXPILE ...
struct pile_s {
    int taille;
    elem valeurs[MAXPILE];
};
```

- liste chaînée

Note : en mutatif, on aura également besoin des opérations de copie et de destruction.

Exemple de TDA : les files (1)

Retenir (Spécification informelle :)

Une pile est une liste linéaire d'objets où les consultations et les suppressions se font du même coté, les insertions se font de l'autre.
Premier arrivé, premier servi.

Vocabulaire basé sur une représentation horizontale : enfiler, défiler, tête, queue, file vide (en anglais : first-in-first-out (FIFO), queue, enque, dequeue, head, tail, empty).

Applications : tampons, parcours en largeur.

Exemple de TDA : les files (1)

Retenir (Spécification informelle :)

Une pile est une liste linéaire d'objets où les consultations et les suppressions se font du même coté, les insertions se font de l'autre.
Premier arrivé, premier servi.

Vocabulaire basé sur une représentation horizontale : enfiler, défiler, tête, queue, file vide (en anglais : first-in-first-out (FIFO), queue, enque, dequeue, head, tail, empty).

Applications : tampons, parcours en largeur.

Exemple de TDA : les files (1)

Retenir (Spécification informelle :)

Une pile est une liste linéaire d'objets où les consultations et les suppressions se font du même coté, les insertions se font de l'autre. Premier arrivé, premier servi.

Vocabulaire basé sur une représentation horizontale : enfiler, défiler, tête, queue, file vide (en anglais : first-in-first-out (FIFO), queue, enque, dequeue, head, tail, empty).

Applications : tampons, parcours en largeur.

Exemple de TDA : les files (2)

Définition (**Type abstrait** Pile(T))

Opérations :

- FileVide : $\{\} \rightarrow$ File(T)
- EstVide : File(T) $\rightarrow$ Booleen
- Enfiler : $T \times$ File(T) $\rightarrow$ File(T)
- Defiler : File(T) $\rightarrow T \times$ File(T)

Préconditions : Defiler(f) *valide seulement si non* EstVide(f)

Axiomes :

- EstVide(FileVide()) = VRAI EstVide(Enfiler(x, f)) = FAUX
- Defiler(Enfiler(x, FileVide())) = (x, FileVide())
- *si non* EstVide(f) *alors*
  Defiler(Enfiler(x, f)) = (y, Enfiler(x, g)) *où* (y, g) = Defiler(f)

Théorie des Files

En utilisant les axiomes, on peut montrer que

Proposition (Formes canoniques)

Soit $f \in \text{File}(T)$ *une file. Il existe un unique* n *et un unique* n*-uplet* $(a_1, \ldots, a_n) \in T^n$ *tel que*

$$f = \text{Enfiler}(a_n, \text{Enfiler}(a_{n-1}, \ldots (\text{Enfiler}(a_1, \text{FileVide}()) \ldots)) .$$

Exemple de TDA : les ensembles (1)

Retenir (Spécification informelle :)

On décrit une collection d'objets de type arbitraire, avec les opérations d'ajout d'un élément, de suppression, d'appartenance et de cardinalité (nombre d'éléments).

Note : un élément, s'il appartient à un ensemble n'y apparaît qu'une fois et une seule.

Exemple de TDA : les ensembles (1)

Retenir (Spécification informelle :)

On décrit une collection d'objets de type arbitraire, avec les opérations d'ajout d'un élément, de suppression, d'appartenance et de cardinalité (nombre d'éléments).

Note : un élément, s'il appartient à un ensemble n'y apparaît qu'une fois et une seule.

Exemple de TDA : les ensembles (2)

Définition (**Type abstrait** Ens(T))

Opérations :

- Vide : $\{\} \rightarrow \text{Ens}(T)$
- EstVide : $\text{Ens}(T) \rightarrow \text{Booleen}$
- Ajoute : $T \times \text{Ens}(T) \rightarrow \text{Ens}(T)$
- Supprime : $T \times \text{Ens}(T) \rightarrow \text{Ens}(T)$
- Appartient : $T \times \text{Ens}(T) \rightarrow \text{Booleen}$

Axiomes :

- $\text{EstVide}(\text{Vide}()) = \text{VRAI} \qquad \text{EstVide}(\text{Ajoute}(x, e)) = \text{FAUX}$
- $\text{Appartient}(x, \text{Vide}()) = \text{FAUX}$
- $\text{Appartient}(x, \text{Ajoute}(y, e)) = (x =_T y \ ou \ \in (x, e))$
- ...

Exemple de TDA : les ensembles (2)

Définition (**Type abstrait** Ens(T) **(suite)**)

- ...
- *si* $x =_T y$ *alors*

$$\text{Ajoute}(x, \text{Ajoute}(y, e)) = \text{Ajoute}(y, e)$$

$$\text{Supprime}(x, \text{Ajoute}(y, s)) = s$$

- *si* $x \neq_T y$ *alors*

$$\text{Ajoute}(x, \text{Ajoute}(y, e)) = \text{Ajoute}(y, \text{Ajoute}(x, e))$$

$$\text{Supprime}(x, \text{Ajoute}(y, e)) = \text{Ajoute}(y, \text{Supprime}(x, e))$$

Implantations possibles des ensembles :

- tableaux de booléens
- tableaux ou liste d'élements
- tableaux triés d'élements
- arbres de recherches
- tables de hachage