UFR IM²AG Université Grenoble Alpes

L3 Informatique
Année 2021 – 2022

UE ALGO5 — TD2 — Séance 5 : Tableaux et collections

Objectifs

À la fin de cette séance, vous devriez être capable de :
- manipuler et concevoir des tableaux comme des structures abstraites ;
- réfléchir aux propriétés de collections variées ;
- proposer des implémentations de gestion de collections cohérentes avec les spécifications choisies.

Exercice 1. Type tableau

Q 1. Rappeler brièvement les principales caractéristiques d'une structure de données de type «tableau». Donner une spécification du type «tableau».

Corrigé —

Caractéristiques d'un tableau

—Tous les éléments d'un tableau sont de même type T ;
—à chaque élément est associé un *indice* qui en permet un *accès direct* (en coût constant) ;
—l'ensemble des indices I est un intervalle d'un *type discret* (plus ou moins contraint selon les langages de programmation : nécessairement entre 0 et $N-1$ en C/C++/Java, plus flexible en Ada...)

Spécification

Deux opérations à identifier :
—*get* (accès à un élément du tableau)
—*set* (définition d'un élément du tableau)

Nom Tableau(Élément)

Utilise Élément, entier

Opérations $\mathsf{get} : \mathsf{Tableau} \times \mathsf{entier} \to \mathsf{Élément}$
$\mathsf{set} : \mathsf{Tableau} \times \mathsf{entier} \times \mathsf{Élément} \to \mathsf{Tableau}$

Axiomes $\mathsf{get}(\mathsf{set}(T, i, e), i) = e$
$\mathsf{get}(\mathsf{set}(T, i, e), j) = get(T, j)$ si $i \neq j$

Remarques importantes concernant cette spécification :
—il n'y a pas de primitive pour «initialiser» un tableau, on ne peut pas supposer qu'un tableau est initialisé avec des valeurs qui nous arrangent...
—Il n'y a pas de primitive pour «tester la présence d'une valeur dans une case d'un tableau», la notion de «case vide» n'a pas de sens. Celle de «supprimer/ajouter une case» non plus.
—On ne peut faire de «get» que sur une case déjà définie !
Enfin on peut aussi remarquer que la taille d'un tableau est (en général) fixe :
—soit déterminée au moment de la compilation (tableau statique, comme en C) ;
—soit déterminée au moment de l'exécution (tableau dynamique, comme ça peut être le cas en Ada).
On considérera donc dans la suite que les tableaux ne sont pas extensibles en cours d'exécution (même si cette facilité existe dans certains langages...).

On pourra donc avoir besoin de distinguer la longueur maximale du tableau (N), du nombre d'éléments effectivement présents (ou l'indice du dernier élément si les éléments sont contigus).

Une notation algo (générale) :

```
I : un type intervalle sur un type discret
T : un type
Tab : le type tableau sur I de T
```

En C, les tableaux étant de taille fixe et indicés à partir de 0, on considérera les types suivants :

```
TAILLE_MAX : une constante ≥ 0
T : un type
Tab : le type tableau sur 0..TAILLE_MAX−1 de T
```

Soit en C :

```
#define TAILLE_MAX 666
typedef T[TAILLE_MAX] Tab;
```

Tab est un type qui permet de représenter des tableaux de tailles N, avec $0 <= N <$ TAILLE_MAX, dont les éléments sont indices de 0 à N-1. Un tableau est alors un couple (T : un Tab, N : un entier).

Exercice 2. Implémentation d'une collection dans un tableau

Q 2. On considère une collection de N éléments, implémentée dans un tableau T de NMAX entiers (NMAX $\geq$ N > 0). Discutez des différentes implémentations possibles de cette collection.

Corrigé —

Différents critères de représentation :

Organisation des éléments dans le tableau :
—représentation contiguë ou non ?
—l'ordre des éléments a-t-il une importance ?
—l'indice des éléments a-t-il une importance ?
Pour définir la taille de la collection (on suppose que toutes les cases du tableau ne sont pas occupées par un élément de la collection) :
—tableau + entier N (représentation contiguë)
—tableau + tableau de booléens (représentation non contiguë)
—tableau avec valeurs spéciales pour marquer :
  —la fin des éléments de la collection (représentation contiguë) ;
  —les cases non occupées par un élément de la collection (c'est l'occasion de demander aux étudiants ce qu'ils ont contre les entiers 0 ou -1, qu'ils ne souhaitent visiblement pas voir apparaître comme éléments d'une collection).
Il n'y a pas de bon ou de mauvais choix : tout dépend de l'utilisation qui sera faite de la structure de donnée ! Cf exercice suivant...

Exercice 3. Insertion et suppression dans une collection

Q 3. On s'intéresse aux deux primitives suivantes (i est donné) :
— ajouter un nouvel élément à l'indice i du tableau
— supprimer l'élément d'indice i du tableau
Proposez une réalisation possible de ces deux primitives en discutant de leur «coût» en nombre d'accès aux éléments du tableau en fonction de N. On pourra imaginer plusieurs solutions en fonction des propriétés que l'on souhaite préserver sur le tableau initial...

Corrigé —

Ces spécifications sont (volontairement) incomplètes, des questions vont se poser, il faut y répondre avant de se lancer dans l'écriture d'un algo...

—Que faire du «trou» quand on supprime un élément ?
  Plein de solutions a priori :
  —laisser le «trou» :
    —soit en utilisant une valeur particulière de T pour le représenter
    —soit en utilisant un 2e tableau S de booléens qui indique les trous ($S[i]$ = vrai ssi $T[i]$ a été supprime).
  —décaler les suivants (a partir de $T[i+1]$) vers la gauche
  —le remplacer par le dernier élément de la collection ($T[i] \leftarrow T[N-1]$)
  —autre ?
  En fait tout dépend des contraintes que l'on veut préserver ou non sur les éléments du tableau (ordre, contiguïté, indexation, ...). Il n'y a pas de solution générale, il faut adapter au problème donné.
—On retrouve les mêmes questions pour insérer un élément ...
  —on peut décaler tous les suivants à droite
  —ou mettre l'élément qui était à l'indice i en position N
—Enfin, une autre question qui va se poser : que fait-on dans les cas ou l'opération n'est pas possible (le tableau déborde, l'élément d'indice i n'existe pas, etc.) ?
  Il est suggéré de mettre d'abord ces problèmes de côté, et d'étendre ensuite l'algo pour les traiter (en affichant un message d'erreur, ou renvoyant un code d'erreur, etc.).

Une fois toute ces questions discutées, on peut écrire ces algos d'insertion et de suppression...

En terme de coûts en nombre d'accès au tableau, on voit rapidement que :

—pour l'insertion :
  —si on déplace uniquement l'élément d'indice i en position $N+1$, le coût est constant (ne dépend pas de N, ni de i) (mais on ne préserve pas l'ordre initial)
  —si on décale tous les éléments de $T[i]$ à $T[N]$ vers la droite, le coût est de l'ordre de $N-i$ accès (donc de N pour $i=0$) (mais on préserve l'ordre initial !)
—pour la suppression c'est pareil :
  —si on laisse un «trou» le coût est constant (on préserve l'ordre, mais on perd la contiguïté)
  —si on permute avec le dernier le coût est constant (on préserve la contiguïté, mais on perd l'ordre)
  —si on décale vers la gauche le coût est en $N-i$ (on préserve l'ordre et la contiguïté...)
