STRUCTURES DE DONNÉES ET ALGORITHMES · CHAPITRE 3

Accéder, insérer et déplacer dans un tableau

Distinguez tableau contigu, liste Python et vecteur MATLAB. Tracez les déplacements nécessaires à une insertion et séparez longueur et capacité.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Distinguer accès indicé, insertion et capacité d'un tableau.
- Un accès par indice ne parcourt pas les éléments précédents dans un tableau.
- Préserver l'ordre lors d'une insertion peut demander de déplacer tout un suffixe.

Prérequis. Avoir étudié les chapitres DSA précédents ; savoir accéder à une collection et suivre des indices.

Partir d'un besoin

Une collection doit permettre de lire rapidement son centième élément, mais aussi d'insérer une valeur au début. Pourquoi ces opérations n'ont-elles pas le même coût ?

Définir la structure et les conventions

Un **tableau** associe des positions à des emplacements contigus de même taille. L'adresse d'un élément se calcule à partir de la position : aucun parcours des éléments précédents n'est nécessaire. La **longueur** compte les éléments utilisés ; la **capacité** compte les emplacements réservés.

Une **liste dynamique** peut agrandir son stockage. La liste Python est un tableau dynamique de références vers des objets : ses valeurs ne doivent pas toutes avoir le même type. Le vecteur numérique MATLAB utilisé ici stocke des valeurs d'un même type. Ces deux représentations

offrent un accès indicé, mais leurs règles de copie et d'agrandissement diffèrent.

Dans [4, 7, 9], insérer 5 avant 7 exige de laisser une place. Les valeurs 7 et 9 doivent être déplacées ou recopiées. En Python, `insert(1, 5)` modifie la liste. MATLAB construit ici un nouveau vecteur par concaténation, c'est-à-dire en assemblant des segments. `1:1` et `2:end` désignent le premier élément puis la fin ; end représente le dernier indice dans ce contexte.

Suivre les états

Insertion avant le deuxième élément

Étape	Suite
Initiale	[4, 7, 9]
Créer une place	[4, _, 7, 9]
Écrire 5	[4, 5, 7, 9]

Comparer les réalisations

La deuxième valeur devient 5 et la longueur passe de 3 à 4. Le pseudo-code décrit l'opération sans supposer une primitive universelle d'allocation. Les fonctions de bibliothèque réalisent les mouvements dans les exemples exécutables.

Pseudo-code

```
// example.pseudo
// insert_array
// Insert 5 before position 1; indices start at 0.
// Shift the suffix one position right, then write 5.
// [4, 7, 9] becomes [4, 5, 7, 9].
```

Python

```python
# insert_array.py
values = [4, 7, 9]
values.insert(1, 5)
second = values[1]
print(values, second)
```

MATLAB

```matlab
% insert_array.m
values = [4, 7, 9];
values = [values(1:1), 5, values(2:end)];
second = values(2);
disp(values);
disp(second);
```

Justifier et compter

L'accès indicé est Θ(1) dans le modèle du tableau. Insérer ou retirer près du début est Θ(n) au pire. Pour une liste Python, ajouter en fin avec append est O(1) amorti ; une réallocation ponctuelle peut coûter Θ(n). La concaténation MATLAB montrée ici construit un vecteur de taille n + 1 : temps et stockage du résultat sont Θ(n). Elle n'hérite pas automatiquement de la garantie amortie de append.

Prévoir puis expérimenter

Prédisez combien d'éléments sont après une insertion au début, au milieu et à la fin. Pour la version MATLAB, observez que la construction d'un nouveau vecteur peut recopier même les valeurs situées avant l'insertion.

Exercice corrigé

▼ Pourquoi un indice valide ne garantit-il pas qu'une insertion soit constante ?

Trouver une position par son indice ne déplace aucune valeur. Insérer à cet endroit doit préserver l'ordre et peut exiger le déplacement de tout le suffixe.

À retenir

- Un accès par indice ne parcourt pas les éléments précédents dans un tableau.
- Préserver l'ordre lors d'une insertion peut demander de déplacer tout un suffixe.
- L'ajout amorti d'une liste Python et la concaténation MATLAB ne possèdent pas la même garantie.

Version web : https://www.clearcode.fr/courses/programming/dsa/linear/arrays