COURS

Structures de données — Listes, piles, files

1re

Programme de première générale, enseignement de spécialité — BO spécial n°1 du 22 janvier 2019

Une pile d'assiettes et une file d'attente contiennent les mêmes objets et n'en rendent pas le même en premier. On choisit une structure d'après l'ordre dans lequel les choses doivent ressortir.

1 Deux structures aux ordres opposés

RÈGLE

Une **pile** fonctionne en **LIFO** — *last in, first out* : le **dernier entré** est le **premier sorti**. Ses deux opérations sont *empiler* et *dépiler*, toutes deux sur le **sommet**. Une **file** fonctionne en **FIFO** — *first in, first out* : le **premier entré** est le premier sorti. On **enfile** à une extrémité et on **défile** à l'autre. Le choix se fait donc d'après l'**ordre de sortie** attendu, jamais d'après le contenu.

$$pile : LIFO \cdot file : FIFO$$

PROPRIÉTÉ

Chacune correspond à des usages précis. La **pile** sert partout où il faut **revenir en arrière** dans l'ordre inverse : annulation d'actions, historique de navigation, évaluation d'expressions parenthésées, et la **pile d'appels** d'un programme. La **file** sert partout où il faut traiter **dans l'ordre d'arrivée** : file d'impression, gestion de requêtes, parcours en largeur d'un graphe.

2 Tableau ou liste chaînée

DÉFINITION

Une **liste chaînée** est une suite de cellules dont chacune contient une valeur et une **référence** vers la suivante. À la différence d'un **tableau**, ses éléments ne sont **pas contigus** en mémoire : on ne peut donc pas sauter directement au i-ième, il faut **parcourir** depuis le début. En contrepartie, insérer ou supprimer au milieu ne demande que de modifier deux références, sans décaler quoi que ce soit.

LE PIÈGE QUI COÛTE LE PLUS DE POINTS

Supposer qu'accéder au i-ième élément d'une liste chaînée coûte autant que dans un tableau. — Dans un tableau, l'adresse du i-ième élément se **calcule** : l'accès est en O(1). Dans une **liste chaînée**, il faut **suivre les références** une par une : l'accès est en O(n). Écrire une boucle qui accède à l'indice i à chaque tour transforme donc un parcours en O(n) en un parcours en $O(n^2)$, sans que rien dans le code ne le signale.

Le réflexe : Pour parcourir une **liste chaînée**, suivre les **références**, jamais les indices.

Le compromis est symétrique et c'est ce qui rend le choix intéressant. Le **tableau** donne un accès direct et une insertion coûteuse — il faut décaler. La **liste chaînée** donne une insertion locale bon marché et un accès coûteux. Aucune n'est meilleure : la question est de savoir quelle **opération** sera la plus fréquente dans le programme.

3 Dictionnaires et hachage

DÉFINITION

Un **dictionnaire** associe des **clés** à des **valeurs**. Il repose sur une table de **hachage** : une fonction transforme la clé en un indice dans un tableau interne, ce qui permet d'accéder à la valeur **sans parcourir** la structure. L'accès est donc en **O(1) en moyenne** — et non dans tous les cas : si plusieurs clés produisent le même indice, il y a **collision**, et la structure doit les départager.

Deux conséquences pratiques. Une clé doit être **immuable** : si l'objet servant de clé changeait, son **hachage** changerait et la valeur deviendrait introuvable — c'est pourquoi une liste ne peut pas servir de clé. Et la performance annoncée est une **moyenne** : une fonction de hachage mal choisie concentre les collisions et fait retomber l'accès vers O(n).

4 Les arbres

DÉFINITION

Un **arbre binaire** est une structure hiérarchique où chaque **nœud** possède au plus **deux** enfants. Le nœud sans parent est la **racine** ; les nœuds sans enfant sont les **feuilles**. La **profondeur** d'un nœud est sa distance à la racine ; la **hauteur** de l'arbre est la profondeur maximale. Un arbre **équilibré** de n nœuds a une hauteur de l'ordre de log n — ce qui explique l'efficacité des recherches qu'on y mène.

Un **arbre binaire** dégénéré, où chaque nœud n'a qu'un seul enfant, n'est rien d'autre qu'une **liste chaînée** déguisée : sa hauteur vaut n, et toute recherche y redevient linéaire. C'est pourquoi les structures d'arbres utilisées en pratique **maintiennent** leur équilibre lors des

insertions : l'efficacité annoncée dépend entièrement de cette propriété.

CHOISIR LA STRUCTURE ADAPTÉE À UN PROBLÈME

1. Déterminer l'**ordre de sortie** attendu : le dernier arrivé, le premier arrivé, ou un accès quelconque ?
2. Recenser les **opérations fréquentes** : accès par indice, par clé, insertion, suppression, parcours.
3. Confronter chaque opération au **coût** de la structure envisagée.
   Accès par indice : O(1) en tableau, O(n) en liste chaînée.
4. Vérifier les **contraintes** : mémoire disponible, taille connue ou variable, clés immuables.
5. Conclure en nommant la structure **et** l'opération qui a décidé du choix.

À VÉRIFIER

- Ai-je choisi d'après l'**ordre de sortie** et non d'après le contenu ?
- **LIFO** pour une **pile**, **FIFO** pour une **file** : ai-je vérifié ?
- Ai-je évité d'accéder par **indice** dans une **liste chaînée** ?
- Ai-je dit que l'accès d'un **dictionnaire** est O(1) **en moyenne** ?
- Mes clés de **dictionnaire** sont-elles **immuables** ?
- Ai-je vérifié qu'un **arbre binaire** est **équilibré** avant d'annoncer log n ?

À RETENIR

- On choisit une structure d'après l'**ordre de sortie** et le **coût** des opérations.
- **Pile** : **LIFO**, dernier entré premier sorti. **File** : **FIFO**.
- Pile : retour en arrière, historique, **pile d'appels**. File : ordre d'arrivée.
- **Tableau** : accès O(1), insertion coûteuse. **Liste chaînée** : l'inverse.
- Accéder par **indice** dans une liste chaînée coûte **O(n)** : parcourir par références.
- **Dictionnaire** : table de **hachage**, accès **O(1) en moyenne**.
- Une clé doit être **immuable**, sinon la valeur devient introuvable.
- Les **collisions** peuvent faire retomber l'accès vers O(n).
- **Arbre binaire** : **racine**, **feuilles**, **profondeur**, **hauteur**.
- Un arbre **équilibré** de n nœuds a une hauteur d'ordre **log n**.
- Un arbre dégénéré est une **liste chaînée** : la performance dépend de l'équilibre.