CORRIGÉ

1re

Structures de données — Listes, piles, files

Corrigé détaillé
4 exercices

> Ce qui est noté au contrôle, ce n'est pas le résultat seul : c'est le raisonnement posé, puis le calcul mené jusqu'au bout avec l'unité.

1 Vocabulaire. Répondre en une phrase précise.

1. a. **LIFO** : dernier entré, premier sorti — la **pile**. **FIFO** : premier entré, premier sorti — la **file**.
2. b. Une suite de cellules contenant chacune une valeur et une **référence** vers la suivante. Ses éléments ne sont **pas contigus** en mémoire, contrairement au **tableau**.
3. c. Sur une table de **hachage**. L'accès est en **O(1) en moyenne**, pas dans tous les cas.
4. d. La **racine** est le nœud sans parent ; une **feuille** n'a pas d'enfant ; la **profondeur** est la distance d'un nœud à la racine ; la **hauteur** est la profondeur maximale.
5. e. Le cas où **deux clés différentes** produisent le **même indice** dans le tableau interne.

2 On empile successivement 3, 7, 2, 9 dans une pile, puis on dépile deux fois. On enfile les mêmes valeurs dans une file, puis on défile deux fois.

1. a. **9** puis **2** : la **pile** rend le **dernier entré** en premier.
2. b. **3** et **7**, le sommet étant désormais le 7.
3. c. **3** puis **7** : la **file** rend le **premier entré** en premier.
4. d. **2** et **9**, dans cet ordre.
5. e. Une **pile** pour l'historique de navigation — revenir en arrière dans l'ordre inverse. Une **file** pour l'impression — traiter dans l'ordre d'arrivée.

3 Un développeur parcourt une liste chaînée de n éléments avec une boucle qui accède à l'indice i à chaque tour. Analyser.

1. a. **O(n)** : il faut **suivre les références** depuis le début, aucun calcul d'adresse n'est possible.
2. b. **O(n²)** : n tours de boucle, chacun coûtant O(n).
3. c. **O(n)** : en conservant la référence courante et en passant à la suivante, chaque élément n'est atteint **qu'une fois**.
4. d. Parce que le code est **lisible et correct** : il produit le bon résultat. Seule la **performance** diffère, et elle ne se voit que sur de grandes entrées. Rien dans la syntaxe ne signale que l'accès par indice est coûteux ici.
5. e. **Non**. Dans un **tableau**, l'adresse du i-ième élément se **calcule** : l'accès est en O(1), et le parcours reste en O(n). C'est le même code avec deux coûts différents — d'où l'importance de savoir sur quelle structure on travaille.

4 Dictionnaires et arbres. Raisonner.

1. a. Parce que son **hachage** détermine l'emplacement de la valeur. Si la clé changeait, son hachage changerait et la valeur deviendrait **introuvable** — c'est pourquoi une liste ne peut pas servir de clé.
2. b. À cause des **collisions** : si plusieurs clés produisent le même indice, la structure doit les départager. Une fonction de **hachage** mal choisie concentre les collisions et fait retomber l'accès vers **O(n)**.
3. c. De l'ordre de **log n** — c'est ce qui rend les recherches efficaces.
4. d. Une **liste chaînée** déguisée : sa **hauteur** vaut **n**, et toute recherche y redevient **linéaire**.
5. e. Qu'une performance annoncée repose toujours sur une **hypothèse** — hachage bien réparti, arbre **équilibré** — et qu'elle s'effondre quand l'hypothèse tombe. Retenir « un dictionnaire, c'est O(1) » sans retenir la condition revient à retenir la moitié de l'énoncé.

Corrige-toi toi-même. Compte tes points exercice par exercice : c'est ainsi qu'on voit où l'on perd, et ce n'est presque jamais là où on le croit.

Ce qui est évalué	Points	Ce qui fait perdre le point
Distinguer LIFO et FIFO et leurs usages	4 pts	Choisir une structure d'après son contenu
Comparer tableau et liste chaînée par les coûts	5 pts	Accéder par indice dans une liste chaînée
Expliquer le hachage et la réserve « en moyenne »	4 pts	Annoncer O(1) sans condition

Justifier l'immuabilité des clés	3 pts	Employer une liste comme clé
Relier hauteur d'arbre et efficacité	4 pts	Annoncer log n pour un arbre quelconque