INF2010 - Structures de données et algorithmes

Travail Pratique 2

Tables de hachage

Département de génie informatique et logiciel

École Polytechnique de Montréal

Automne 2019

Objectifs

- Apprendre le fonctionnement d'une table de hachage
- Comprendre la complexité en temps d'une table de hachage
- Utiliser une table de hachage dans un problème complexe

Pour ce laboratoire, il est recommandé d'utiliser l'IDE IntelliJ offert par JetBrains. Vous avez accès à la version complète (Ultimate) en tant qu'étudiant à Polytechnique Montréal. Il suffit de vous créer un compte étudiant en remplissant le formulaire au lien suivant:

https://www.jetbrains.com/shop/eform/students

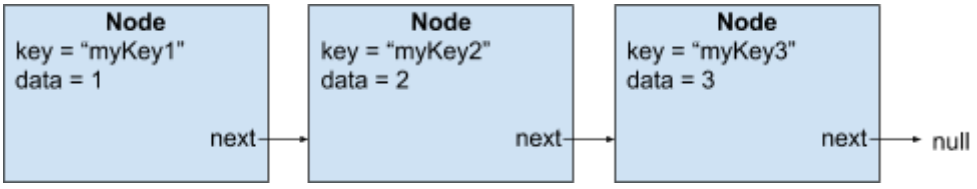
La correction du travail pratique sera partiellement réalisée par les tests unitaires implémentés dans les fichiers sources fournis. La qualité de votre code ainsi que la performance de celui-ci (complexité en temps) seront toutes deux évaluées par le correcteur. Un barème de correction est fourni à la fin de ce *.pdf.

ATTENTION! ATTENTION! ATTENTION!
Pour ceux qui voudraient déposer leur laboratoire sur **GitHub**, assurez-vous que vos répertoires soient en mode **privée** afin d'éviter la copie et l'utilisation non autorisée de vos travaux. **Un répertoire public peut mener à une sanction de plagiat**.

Partie 1 : Implémentation d'une table de hachage

Une table de hachage est une structure de données qui utilise une fonction de dispersement pour donner une valeur numérique à une clé qui peut être d'un type quelconque (string, int, MyCustomClass, ..). Cette valeur numérique retournée par la fonction de dispersement est utilisée comme index dans un tableau, ce qui nous donne un opération d'accès en O(1).

Il arrive que la fonction de dispersement retourne la même valeur numérique pour deux clés différentes. Ce phénomène nommé «collision» est un problème connu des tables de hachage et plusieurs techniques existent pour pallier à ce problème. Dans le cadre de ce laboratoire, l'utilisation de listes chaînées nous permettra de gérer les collisions. La classe «Node» contenue dans LinkedHashMap.java vous permettra de créer des listes chaînées de la manière suivante :



Pour bien implémenter la-dite table de hachage, suivez les tests contenus dans LinkedHashMapTester.java. Aussi, n'oubliez pas d'utiliser *getIndex(KeyType key)* comme fonction de dispersement.

Une note de 0 sera attribuée à cette partie si l'étudiant utilise une table de hachage déjà implémentée provenant d'une librairie quelconque.

Partie 2 : Problème typique d'entrevue

Le problème que vous aurez à résoudre s'intitule «Matching Pairs».

Entrées

- Collection de nombres entiers (*values*)
- Entier (*targetSum*)

Sortie

Collection de toutes les paires possibles dans *values* dont la somme donne *targetSum*

Contraintes

- La collection en sortie doit contenir aucune permutation
- La collection en sortie doit contenir toutes les combinaisons
- *values* peut contenir des duplications
- *values* peut contenir des entiers négatifs

Pour bien implémenter l'algorithme, suivez les tests contenus dans InterviewTester.java .

Barème de correction

Partie 1	Réussite des tests	/11.5
Partie 2	Réussite des tests	/4
	Complexité en temps	/3.5
Qualité du code		/1
		/20

Un chargé s'assurera que votre code ne contourne pas les tests avant de vous attribuer vos points dans la catégorie «Réussite des tests».

Pour avoir tous les points dans la partie 2 dans la catégorie «Complexité en temps», vous devez réaliser un algorithme en O(n) pour le meilleur cas.

Qu'est-ce que du code de qualité ?

- Absence de code dédoublé
- Absence de *warnings* à la compilation
- Absence de code mort
- Respecte les mêmes conventions de codage dans tout le projet
- Variables, fonctions et classes avec des noms qui expliquent leur intention et non leur comportement

Instructions pour la remise

Veuillez envoyer les fichiers .java contenant le code source utile à la résolution des parties 1 et 2. Minimalement, les fichiers suivants devraient faire partie de votre remise :

- LinkedHashMap.java
- Interview.java

Vos fichiers devront être compressés dans une archive *.zip. Le nom de votre fichier compressé devra respecter la formule suivante où MatriculeX < MatriculeY :

inf2010_lab2_MatriculeX_MatriculeY

Chaque jour de retard créera une pénalité additionnelle de 20%. Aucun travail ne sera accepté après 4 jours de retard.