TP2. structures de données

1 Table de hachage

Dans cette section, il n'y a pas de code fourni, à vous de faire l'implémentation.

L'objectif de cette section est d'implémenter une version simplifiée de la table de hachage. [1].

Rappelons le principe. On veut placer des valeurs numériques [2] en les rangeant dans un tableau de dix blocs (listes chaînées) selon la valeur d'une clé.

Pour définir une clé entre 0 et 9 à partir d'un nombre, nous calculerons la somme des chiffres du nombre, dont nous ne garderons que le chiffre des unités. Ainsi, le nombre 2847, dont les chiffres ont pour somme 21, sera associé à la clé 1.

La structure de donnée est donc un tableau de 10 listes chaînées, numérotées de 0 à 9. La liste chaînée numéro x contiendra tous les nombres dont la clé vaut x.

1. Écrivez une classe `Table`, avec pour commencer une simple méthode de classe `int hashcode(int value)` qui calcule la clé d'une valeur en entrée. (Pourquoi ne pas en profiter pour l'écrire sous forme récursive ?)
2. Ajoutez un attribut correspondant à un tableau de dix listes chaînées et un constructeur qui initialise chaque case du tableau avec une liste vide.
3. Pour ranger une valeur dans le tableau, on calcule sa clé, et on insère la valeur dans la liste correspondant à la clé (à la suite des autres valeurs présentes). Implémentez la méthode `add` dans la classe `Table` qui effectue cette insertion.
4. Pour vérifier si la table contient un élément ou pour le retirer, on calcule sa clé (à nouveau), et on le cherche dans la liste associée. Implémentez les méthodes `contains` et `remove` qui cherchent si la structure contient une valeur et suppriment une valeur.
5. Testez en insérant 5000 valeurs aléatoires (entre 0 et 100000) dans le tableau. Vous pouvez faire un `toString` pour afficher l'objet rempli.
6. Comparez le temps nécessaire pour faire des requêtes *contains* sur tous les éléments de 1 à 100 000 dans la table par rapport au temps pour chercher les valeurs dans une ArrayList usuelle. [3]
7. Améliorez encore les performances en changeant la base de la clé. Testez avec une clé sur deux chiffres (en remplaçant tous vos 10 par 100) ou sur 3 chiffres (par 1000). [4]

Félicitations, vous avez implémenté votre propre table de hachage !

2 Arbres binaires

Une classe `Node` vous est fournie, avec de quoi mettre en place un arbre binaire de recherche. Les méthodes structurelles `contains`, `insert` et `remove` sont volontairement laissées vides.

8. complétez la méthode `toSortedString` qui retourne la chaîne de caractères comportant l'ensemble des valeurs contenues dans l'arbre dans l'ordre croissant, séparées par des espaces.
9. Complétez la méthode `contains` qui indique si l'arbre contient une valeur donnée. Pour simplifier la gestion d'un arbre vide, elle est définie de façon statique.
10. Complétez la méthode `insert` qui insère une valeur dans l'arbre.
11. Complétez la méthode `remove` qui retire une valeur de l'arbre.

1. Comme vu en cours avec le modèle de la bataille navale de CS-unplugged
2. Rappelez-vous, ces valeurs étaient associées à des bateaux dans l'exemple du cours.
3. Vous devriez obtenir un résultat en moyenne 2 fois plus rapide avec la table de hachage.
4. On peut obtenir des ratios 15 fois et plus de 50 fois plus rapide.