Tables de hachage

POGL, TD2. Conception, diagrammes de séquence, spécification des opérations

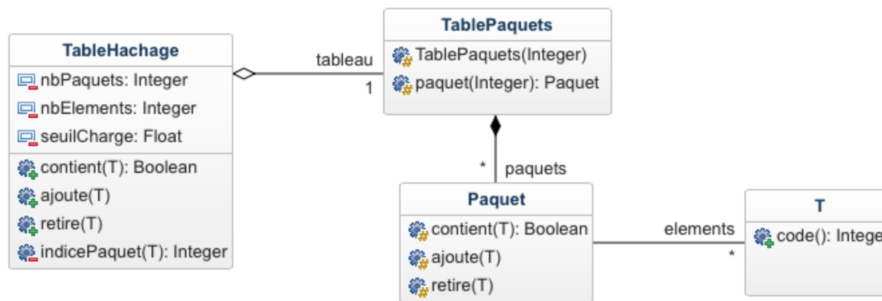
Dans ce TD, nous nous intéressons à la conception d'une structure de table de hachage, et en particulier à la manière dont les fonctionnalités principales se décomposent en enchaînements d'opérations des différentes classes composant la structure.

Tables de hachage

Une table de hachage permet de stocker un ensemble d'éléments dans un tableau, de telle sorte que l'ajout et le test d'appartenance d'éléments puissent se faire en temps constant en moyenne.

La structure repose sur une fonction de codage, qui à chaque élément `elt` associe un entier `c` dont on déduit l'indice du tableau auquel l'élément doit être placé. Notez qu'il est possible que deux éléments distincts soit associés au même code (on parle de *collision*), ou qu'un code ne soit associé à aucun élément. Chaque case du tableau ne contient donc pas un unique élément, mais un paquet de zéro, un ou plus d'éléments. La manipulation de cette structure est efficace dans la mesure où les paquets restent en général petits.

Voici un diagramme de classes représentant les différentes entités entrant dans la conception de notre table de hachage.



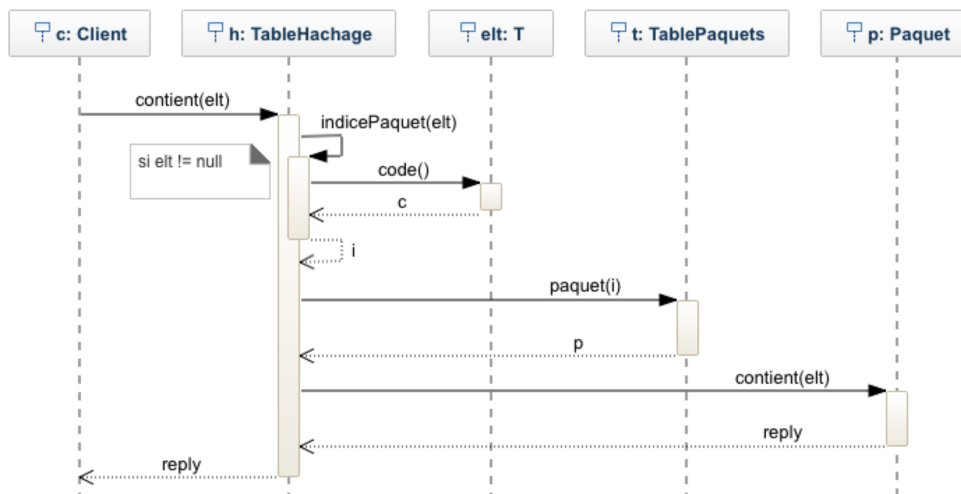
Dans la classe `TablePaquets`, les `paquets` sont stockés dans un tableau auquel on peut accéder par des indices. Dans la classe `Paquet`, les `elements` sont stockés dans une liste chaînée. Les invariants associés à ces classes sont :

— l'attribut `nbPaquets` est égal au nombre de paquets associés au `tableau`,
— l'attribut `nbElements` est égal au total des nombres d'éléments présents dans chaque paquet,
— le `tableau` ne contient aucun paquet égal à `null`, et aucun paquet ne contient l'élément `null`,
— tout élément `elt` de la table est dans le paquet d'indice `elt.code() % nbPaquets`.

Q1. Considérons la situation suivante : une table de hachage à 8 paquets, contenant les entiers -2, -3, 5, 7, 11, 13 et 17, l'opération `code` renvoyant la multiplication par 3 du nombre à coder. Donner un diagramme d'instance correspondant à cette situation.

Test d'appartenance

L'une des fonctionnalités de notre table de hachage est le test d'appartenance d'un élément à l'ensemble représenté par la table, réalisé par l'opération `TableHachage::contient(T elt)` qui déclenche l'enchaînement d'opérations suivant dans le cas où le paramètre `elt` n'est pas `null` :



Les spécifications des méthodes utilisées sont :

- `TableHachage::contient(T elt)`
  - Précondition : néant
  - Résultat : renvoie `true` si `elt` est dans la table, et `false` sinon ou si `elt` est `null`
  - Postcondition : néant
- `TableHachage::indicePaquet(T elt)`
  - Précondition : `elt` n'est pas `null`
  - Résultat : renvoie l'indice du paquet susceptible de contenir `elt`
  - Postcondition : le résultat est compris entre `0` inclus et `nbPaquets` exclu
- `TablePaquets::paquet(int i)`
  - Précondition : `i` est compris entre `0` inclus et `nbPaquets` exclu
  - Résultat : renvoie le paquet d'indice `i`
  - Postcondition : le résultat n'est pas `null`
- `Paquet::contient(T elt)`
  - Précondition : néant
  - Résultat : renvoie `true` si `elt` est dans le paquet, et `false` sinon
  - Postcondition : néant
- `T::code()`
  - Précondition : néant
  - Résultat : renvoie le code de hachage de l'élément
  - Postcondition : néant

Q2. Décrivez la séquence des opérations effectuées. Détaillez en particulier les informations circulant d'un objet à l'autre par l'intermédiaire des paramètres et résultats des appels de méthodes.

Q3. En quoi les préconditions choisies permettent-elles de simplifier le code des méthodes ? Pour chacune de ces méthodes, expliquer ce qui assure que les préconditions sont satisfaites, ou ce qu'il faut faire pour s'en assurer.

Q4. Pour simplifier plus encore, quelqu'un propose de donner comme précondition à l'opération `TableHachage::contient(T elt)` que `elt` ne soit pas `null`. Discuter.

Ajouter et retirer des éléments

On fixe les spécifications suivantes pour les opérations de la classe `Paquet` :

- `Paquet::ajoute(T elt)`
  - Précondition : `elt` n'est pas `null`
  - Résultat : néant
  - Postcondition : une occurrence de `elt` est ajoutée au paquet
- `Paquet::retire(T elt)`
  - Précondition : néant
  - Résultat : néant
  - Postcondition : une occurrence de `elt` est retirée du paquet s'il y en avait au moins une, aucun effet sinon

Q5. On veut donner à la méthode `TableHachage::retire(T elt)` la spécification suivante :
- Précondition : néant
- Résultat : néant
- Postcondition : toutes les occurrences de `elt` sont retirées du paquet et `nbElements` est décrémenté de 1 si l'élément était présent, aucun effet si `elt` est `null`

Que faudrait-il faire pour réaliser une telle opération en ne supposant rien de plus que les invariants et spécifications actuelles ? Quel invariant sur la table de hachage permettrait de faire plus simple ? Proposer une spécification pour `TableHachage::ajoute(T elt)` qui assure le maintient de cet invariant, et donner un diagramme de séquence détaillant cette opération.

Q6. Pour limiter la taille moyenne des paquets, on veut appliquer la stratégie habituelle suivante : lorsque le rapport du nombre d'éléments sur le nombre de paquets dépasse le seuil de charge (dont une valeur typique est 0.75), doubler le nombre de paquets et réinsérer chaque élément dans le paquet qui convient. Quelle opération modifier pour introduire ce nouveau comportement ? Donner des versions mises à jour de sa spécification et de son diagramme de séquence.