INF2010 - Structures de données et algorithmes
Hiver 2018
Travail Pratique 3
Fonctions de hachage et tables de dispersement

PROBLÈME : Hachage parfait

Les tables de dispersement tentent de réaliser l'idée simple que des données peuvent être accédées en temps $O(1)$ si une fonction de hachage est utilisée pour retrouver la position en mémoire des éléments. Le problème principal associé à cette approche est que deux objets différents peuvent produire par hachage la même position en mémoire, ce que l'on appelle une collision. Différentes techniques existent pour gérer les collisions (listes-chaînées, sondage linéaire ou quadratique etc.). Le hachage parfait est une technique permettant l'implémentation de tables de dispersement sans collision qui repose sur le fait que dans certains cas, les données sont connues à avance et elles ne changent pas (par exemple dans le cas d'un éditeur de texte qui utilise la coloration syntaxique d'un nombre fini de mot clés (for, if, while, etc.).

Objectifs :

- Implanter une table de dispersement parfaite (sans collision).
- Minimiser l'espace occupée par une telle table.

Exercice 1 : Hachage parfait à occupation d'espace quadratique (1 point)

Supposons que nous disposions d'un ensemble de n objets statiques (qui ne changent pas après leur création) et qui ont une valeur de `hashCode()` inférieure à un nombre premier p donné. Il existe alors un hachage permettant de stocker ces n objets dans un espace mémoire de taille $m = n^2$ sans causer une seule collision. La position de chaque object x est donnée par : $((a \cdot x.\texttt{hashCode()} + b) \bmod p) \bmod m$, où $m < p$, $0 < a < p$ et $0 \leq b <$ p, et où $x.\texttt{hashCode()}$ est supposé renvoyer une valeur inférieur à p !

Sachant qu'il y a moins de 50% de chances qu'une collision survienne si a et b sont choisis au hasard, proposez une implémentation correcte de cette méthode en modifiant la classe `QuadratiqueSpacePerfectHashing` qui vous est fournie. Vous n'avez pas le droit de changer la signature des méthodes. Vous avez le droit de rajouter de nouvelles méthodes au besoin, bien que cela ne soit pas nécessaire. On vous impose $p = 46\,337$.

Exercice 2 : Minimiser l'espace requis (2 points)

Dans cette première approche, le hachage parfait était possible au prix d'une occupation de mémoire proportionnelle au carré du nombre de données n. Il est possible d'atteindre une occupation d'espace qui soit linéairement proportionnelle à la quantité de données en utilisant une approche qui réunit les concepts de listes chaînées et de hachage double.

Dans un premier temps, on utilise un premier tableau de taille $m=n$. Chaque objet x vise l'alvéole (case) $j = ((a \cdot x.\texttt{hashCode()} + b) \bmod p) \bmod m$, où $m = n < p$ $0 < a < p$, $0 \leq b <$ p, et où $x.\texttt{hashCode()}$ est supposé renvoyer une valeur inférieur à p !. On peut choisir a et b aléatoirement pour ce premier niveau. Pour chaque alvéole j, on a n_j objets en collusion. On utilise alors un hachage parfait à occupation quadratique pour stocker les n_j objets dans un espace $m_j = n_j^2$ sans collision Proposez une implémentation de cette approche en modifiant la classe `LinearSpacePerfectHashing` qui est fournie.

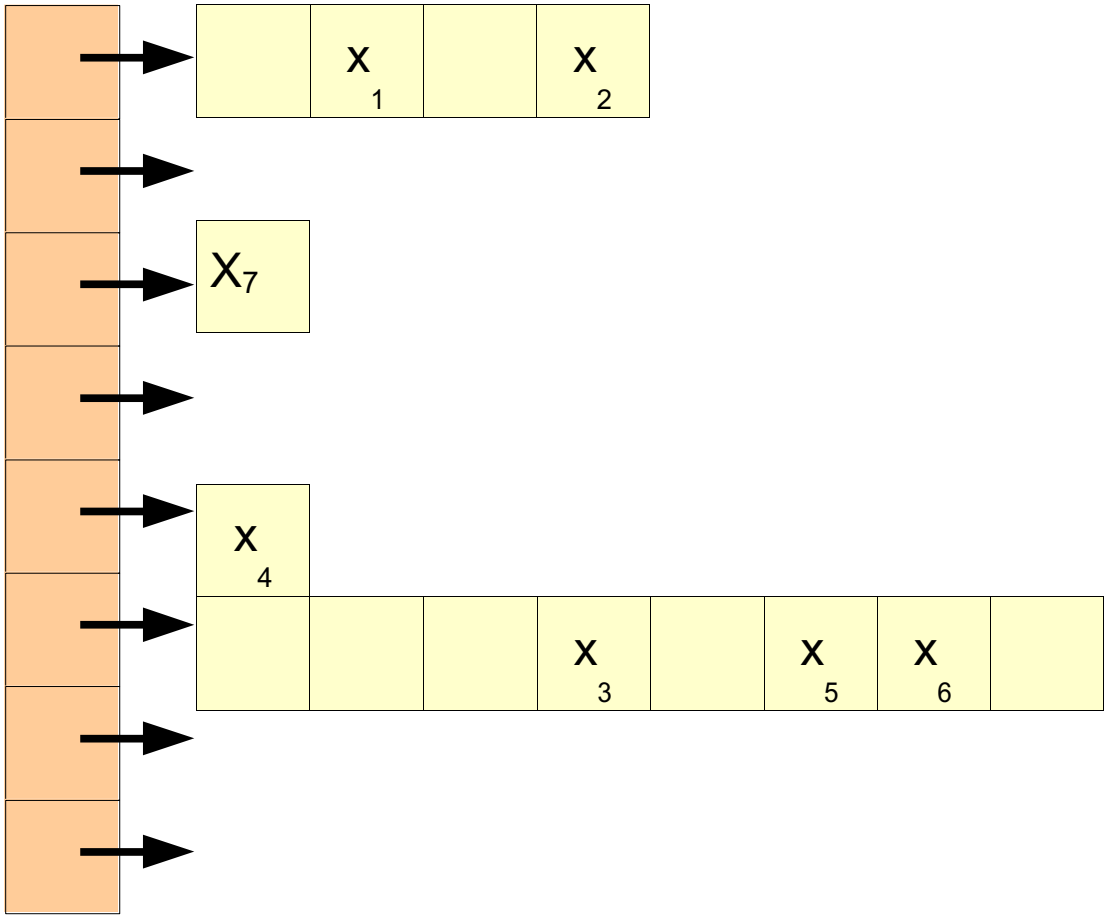

Figure 1 Illustration d'un LinearPerfecthashing contenant 7 éléments.

Question 1 : Linéaire ou pas ? (2 points)

La preuve que l'approche implémentée dans `LinearSpacePerfectHashing` occupe un espace linéairement proportionnel à la quantité de données dépasse le cadre de notre cours.

- On vous demande de vous en convaincre en effectuant des tests aléatoires. Implémentez la fonction `randomIntegers(int length)`, permettant de créer un `ArrayList` de taille `length` comportant des `Integer` dont la valeur est majorée par p. La liste obtenue ne doit pas inclure de doublons ! Rapportez sur un graphique les points obtenus en utilisant un tableur ou un logiciel mathématique tel que Matlab ou Octave.

- Expliquez pourquoi on a choisi p = 46 337 pour les classes `LinearSpacePerfectHashing` et `QuadratiqueSpacePerfectHashing`. Quelle limite sur la taille des données cela impose-t-il ? Votre code reflète-t-il cette limite ?

Instructions pour la remise :

Le travail doit être remis via Moodle :

- 27 Février avant 23h59 pour le groupe (B2).
- 6 Mars avant 23h59 pour le groupe (B1).

Veuillez envoyer dans une archive de type *.zip qui portera le nom inf2010_lab3_MatriculeX_ MatriculeY (de sorte que MatriculeX < MatriculeY) :

1) Vos fichiers .java
2) Un document .pdf qui contient vos réponses pour la question 1

Les travaux en retard seront pénalisés de 20 % par jour de retard. Aucun travail ne sera accepté après 4 jours de retard.