Réponses aux questions de l’énonce

**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? :**

Dans ce laboratoire, nous utilisons la méthode ‘hasCode()’ de Java pour générer un ‘hashcode’ à chacune de nos valeurs. Ce ‘hashcode’ est par la suite utilisé pour générer une clé, qui est placé à l’intérieur d’un int de 16 bits, pour placer nos données à l’intérieur des tables. Cette méthode de Java renvoie par contre un entier de 32 bits. En utilisant p = 46 337, nous limitons la valeur de ce ‘hashcode’ à 16 bits. Ce qui nous permet par la même occasion de réduire la taille de nos données à 16 bits également. Nous avons donc une économie d’espace mémoire.

**Votre code reflète-t-il cette limite?**

En effet, nous sommes en mesure d’affirmer que notre code reflète bel et bien cette limite. Chacune des valeurs générées à partir de la fonction globale ‘randomIntegers(int length)’ est un entier qui ne dépasse pas les 16 bits. Bref, l’exemple ci-dessous est fourni en appui. La taille (ou length) passée en paramètre dans ce cas-ci est 20.

|  |
| --- |
| 8740 |
| 27064 |
| 37997 |
| 3011 |
| 20643 |
| 29402 |
| 3183 |
| 38941 |
| 42751 |
| 27689 |
| 37922 |
| 43450 |
| 24558 |
| 5076 |
| 29445 |
| 28701 |
| 26812 |
| 30379 |
| 25832 |
| 859 |

**Output console :**

Number of elements : 20 Size : 400