Projet LSINF1121 - Algorithmique et structures de données

-

Bilan Mission 3

Groupe 26

Laurian Detiffe (6380-12-00)

Sundeep Dhillon (6401-11-00)

Alexis Macq (5910-12-00)

Xavier Pérignon (8025-11-00)

Thibaut Piquard (4634-13-00)

Thomas Wyckmans (3601-12-00)

Année académique 2015-2016

Questions et réponses

1. Voici la formule utilisée par Java pour calculer une fonction de hachage sur les doubles (bits est un tableau de 64 bit représenté sous forme de long) :

   $$((int)bits \wedge (bits >>> 32))$$

   Le livre Algorithms-4 suggère qu'une bonne fonction de hachage doit utiliser tous les bits pour son calcul. Est-ce le cas pour la fonction de hachage d'un Double en Java ? Pour rappel un double en Java est représenté en 64 bits sous la forme $(-1)^s$ x m x $2^{(e-1023)}$. Le premier bit s est le signe, les 11 bits suivants représentent l'exposant sous forme binaire et les 52 derniers bits représentent la mantisse sous forme binaire. Est-ce qu'un nombre décimal positif et son opposé obtiennent des fonctions de hachage différentes ? Oui, un nombre décimal positif et son opposé obtiennent des fonctions de hachage différentes car le signe du bit est compris dans le calcul du Hachage.
   Est-ce que la fonction de hachage d'un entier sur 32 bits et celle de ce même entier qui serait casé double sont les mêmes ? Non, parce que la représentation n'est pas la même.
   Et pour un cast vers un long ? Non, parce que lorsqu'on va caster notre int en long, on rajouter des zéros devant donc on perd le bit donnant l'info sur le signe et par conséquent, on obtient le même hash pour des signes opposés.

   Hint : `Long.toBinaryString(Double.doubleToRawLongBits(a))` permet d'afficher le tableau de bits utilisé pour la représentation d'un double.

   Q2 = TUYAU EXAM

2. La fonction de hachage pour un string donnée dans Algorithms-4 p460 est la suivante :

   ```
   int hash = 0;
   for (int i = 0; i < s.length(); i++)
   hash = (R * hash + s.charAt(i)) % M;
   ```

   Dans l'implémentation du livre la taille de M (le tableau) est une puissance de deux. La valeur suggérée pour R est "*un petit nombre premier tel que 31 de sorte que les bits de tous les caractères jouent un rôle.*" Supposons que R est un multiple de M. Que se passerait-il lors du calcul ? (a+b)%n = (a%n + b%n) %n
   h = (R * h + s) % M ; (avec R multiple de M)
   h_{intial} = (s_1) % M
   h_2 = (R * s_1 % M + s_2) % M
   = ((R * s_1 % M) % M + s_2 % M) % M
   On sait que (R* s_1 % M) % M = 0
   (R * s_1 % M) devient ((R % M) (s_1 % M)) mod M
   = O (zéro)
   h_n = (s_n % M) % M = s_n % M
   Seul le dernier caractère contribue aux hash et il risque donc d'y avoir beaucoup de collisions.

Supposons que R est un nombre pair. Que se passerait-il ? R est pair si et seulement si R est multiple de M puisque M est une puissance de 2.
Dans les deux cas, combien d'entrées du string détermineront effectivement le code de hachage ? Seule la dernière entrée déterminera effectivement le code de hachage.
Quels sont les risques en terme de collision ? On risque d'avoir beaucoup plus de collisions.
Est-ce que le contrôle du facteur de charge peut résoudre le problème ? Non, car la taille restera multiple de M.
Expliquez pourquoi utiliser 31 est un choix judicieux pour des tailles de tableau qui sont des puissances de deux ? Pas de risque de modulo nul et pas de risque d'overflow en utilisant un `int`.
Serait-ce aussi un bon choix pour une taille de tableau qui commencerait à 31 et qui serait multipliée par deux à chaque fois qu'il faut redimensionner ? Non, on se ramènerait au problème précédent car on obtiendrait encore des résultats nuls à nos opérations modulo.

3. Que suggèreriez-vous comme fonction de hachage pour l'identification de véhicules qui sont des strings de nombres et de lettres de la forme : "9X9XX99X9XX999999" où un 9 représente un chiffre et un "X" une lettre de A à Z. On propose d'utiliser une fonction de hachage via une méthode de Horner (cf. question précédente).
   Est-ce que votre fonction de hachage a la propriété que que pour une taille de N hypothétique de $10^{11} \cdot 26^6$ je n'ai jamais de collision ? Oui, on aura aucune collision si on prend un nombre premier suffisamment grand.