

## TD 1 Tables de Hachage

### 1 Boucherie & Table de Hachage.

Un ancien L3IF reconverti en boucher aimerait connaître l'état de ses stocks et le mettre à jour facilement. Il voudrait pouvoir connaître rapidement pour chaque morceau de viande la quantité dont il dispose encore. Il dispose en ce moment de 2 filets d'agneau, d' 1 bifteak de boeuf, de 4 côtes de boeuf, de 3 saucisses de porc, et de 5 côtes de porc.

**Préambule :** On suppose que notre boucher dispose bêtement d'une feuille de papier sur laquelle il note au fur et à mesure ses provisions sous la forme de couples

(morceau de viande, quantité restante)

Il peut ajouter un élément à la fin de la liste en temps  $O(1)$ , mais pour inclure un élément, il a besoin d'effacer et de réécrire les éléments suivants jusqu'à la fin de la liste.

**Exercice 1.1.** Estimer le coût de la recherche du nombre de saucisses en fonction du nombre d'éléments présents.

$O(n)$

**Exercice 1.2.** Au moment des livraisons, le boucher reçoit 2 saucisses. Quel est le coût de la mise à jour ? Et s'il reçoit 4 côtes d'agneau en plus ? Et lorsqu'il vend son dernier bifteak ?

$O(n), O(n) + O(1), O(n)$

**Exercice 1.3.** Et si la liste était maintenue triée dans l'ordre lexicographique ?

$O(\log(n)), O(\log(n)) + O(n), O(\log(n)) + O(n)$

À présent, étant un ancien L3IF, il veut utiliser son ordinateur pour que cela soit plus rapide. Sa première idée consiste à utiliser un objet de type *liste* où chaque élément contient le nom du morceau, la quantité restante, et l'adresse de l'élément suivant dans la liste.

**Exercice 1.4.** Quel est alors le coût d'une recherche ? D'un ajout ? D'une suppression ?

$O(n), O(n), O(n)$

**Exercice 1.5.** Et si la liste est triée?  $O(n), O(n), O(n)$

Déçu, il veut essayer de tout mettre dans un tableau de taille  $N$  ( $= 100$ ). Le tableau est suffisamment grand pour que chaque type de viande qu'il vendra un jour y trouve une case.

**Exercice 1.6.** Quel est alors le coût d'une recherche? D'un ajout? D'une suppression?  
 $O(N), O(N), O(N)$

**Exercice 1.7.** Et si le tableau est trié?  
 $O(\log N), O(\log N), O(\log N)$

Déjà familier avec la planche à découper, notre boucher aimerait à présent utiliser pour son inventaire une table de hachage simple pour améliorer encore les recherches. Une table de hachage est un ensemble composé d'un tableau de taille  $N$  et d'une fonction de hachage  $h(x)$  qui, à partir d'une *clé*  $x$  (ici, le nom du morceau de viande), renvoie en temps  $O(1)$  l'indice où stocker cet élément dans le tableau.

Notre boucher va utiliser la fonction de hachage suivante :

$$\begin{cases} h(\text{morceau}, \text{animal}) = h'(\text{morceau}) + h'(\text{animal}) * 10 \bmod N \\ h'(c) = |c| \text{ (nombre de caractères de la chaîne } c) \end{cases}$$

**Exercice 1.8.** Écrire la table correspondante.

indice	élément	
44	côte, porc	5
48	saucisse, porc	3
54	côte, boeuf	4
57	bifteak, boeuf	1
65	filet, agneau	2

**Exercice 1.9.** Calculer le coût de recherche d'un élément. D'ajout? De suppression?  
 $O(1), O(1), O(1)$

**Exercice 1.10.** Quel est le problème de ce genre de table?  
*le besoin en taille mémoire*

**Exercice 1.11.** Si on utilisait une liste plutôt qu'un tableau pour stocker les éléments, quel avantage y-a-t'il d'utiliser une fonction de hachage?  
*Comparer des clés / Comparer des entiers*

## 2 Voitures & Collision.

La préfecture voudrait maintenir une base de données des véhicules immatriculés et de leurs propriétaires.

Les plaques d'immatriculations sont de la forme  $(n_1, c_1 c_2 c_3, n_2)$ , où

- $n_1$  est un chiffre compris entre 0001 et 9999,
- $c_1, c_2, c_3$  sont des caractères compris entre  $A$  et  $Z$ , ( $c_3$  pouvant être  $\emptyset$ )
- $n_2$  est un chiffre compris entre 01 et 95 (mettons de côté les immatriculations d'outre-mer, ainsi que le  $2A$  et  $2B$  Corse)

Ils ont décidé d'utiliser pour leur base de donnée une table de hachage, dont la fonction de hachage est la suivante :

$$h(n_1, c_1, c_2, c_3, n_2) = n_1 + (h'(c_1) + h'(c_2) + h'(c_3)) + n_2$$
$$h'(c) = \begin{cases} 0 & \text{si } c = \emptyset \\ 1 & \text{si } c = A \\ \dots & \\ 26 & \text{si } c = Z \end{cases}$$

**Exercice 2.1.** Calculez la valeur de hachage des plaques suivantes :

1846 RC 69, 8626 BD 69, 2536 ARC 30, 1789 KTZ 90

Quel problème voyez-vous apparaître ?

*1936, 8701, 2588, 1936 : problème de collision*

**Exercice 2.2.** Trouvez une façon naïve de résoudre une collision. Calculer le temps de recherche d'un élément au pire cas.

*Hachage linéaire : on essaye de mettre la clé à l'indice suivant inoccupé.*

*Si la table est repliée sauf 1 case,  $O(N)$  au pire*

**Double Hachage** Une autre solution au problème de collision consiste à utiliser une seconde fonction de hachage  $h_2$ . S'il y a collision sur l'indice  $h(c)$ , on teste l'indice  $h(c, i) = h(c) + h_2(c) * (i - 1) \bmod N$  jusqu'à tomber sur une case du tableau inoccupée,  $i$  étant le nombre d'essai effectué.

**Exercice 2.3.** Donnez une condition nécessaire sur  $h_2$  pour que cette résolution des conflits soit efficace.

*$h_2(c)$  premiers avec  $N$  (sinon, on boucle sans tester les  $N$  cases (on teste seulement  $N/\text{pgcd}(h_2(c), p)$  cases)*

**Exercice 2.4.** Quel est la complexité de recherche dans le pire cas ?

$O(n)$

**Exercice 2.5.** Dans l'exercice précédent, trouvez 2 façons différentes de provoquer un problème de collision

*rôti porc, ou  $N \leq 10$*

En pratique, la méthode la plus utilisée pour résoudre les collisions est le *chainage*. Les indices de la table de hachage ne pointent plus sur des éléments à stocker, mais sur une liste chaînée d'éléments ayant la même valeur de hachage.

**Exercice 2.6.** Quel est le temps d'insertion d'un élément si on accepte la redondance ? Et dans le cas contraire ?

$O(1)$ , dépend de la longueur de la liste à l'indice  $h(c)$

On voit alors l'importance de répartir les éléments afin d'éviter le plus possible les collisions. Pour cela, il faut que chaque élément ait la même chance d'être haché vers l'un quelconque des indices. On dit alors que l'on a un hachage uniforme simple. Ainsi, le nombre moyen d'éléments de la liste chaînée de l'indice  $i$  est  $E[i] = \alpha = \frac{m}{N}$ .

**Exercice 2.7.** Quel est alors le temps moyen d'une recherche ? Prouvez-le de 2 manières différentes avant de passer à la question suivante

$O(1 + \alpha)$

**Exercice 2.8.** Trouvez une fonction de hachage uniforme simple si les clés sont

– des entiers  $h(k) = k \bmod N$

– des réels compris entre 0 et 1  $h(k) = \lfloor k * N \rfloor$

**Exercice 2.9.** Quelle est l'importance de la taille de la table ?

Pour le modulo, si  $N = 2^k$ , seul les bits de poids faibles rentrent en compte

**Exercice 2.10.** On propose pour hacher des chaînes de caractères la fonction suivante :

$$h(c_0c_1 \dots c_n) = h'(c_1) + 2^p h'(c_2) + \dots + 2^{ip} h'(c_i) + \dots + 2^{np} h'(c_n) \bmod N$$

avec  $p$  un nombre premier. Étant donné un livre, trouvez un moyen de trouver (très grossièrement) des rimes. Et pour trouver des anagrammes ?

Pour les rimes, on prend une fonction de hachage qui ne prend en compte que les derniers caractères.

Pour les anagrammes, on fixe  $N = 2^p - 1$ , pour ne pas avoir d'influence lors de la permutation des caractères sur les valeurs de hachage

### 3 Classe universelle de fonctions de hachage

Soit  $p$  un nombre premier assez grand pour que toute clé possible soit dans l'intervalle  $0..p-1$ . Et soit  $h_{a,b}$ , ( $a \in \{1, \dots, p-1\}$ ,  $b \in \{0, \dots, p-1\}$ ) la fonction de hachage suivante :

$$h_{a,b}(k) = ((ak + b) \bmod p) \bmod N$$

**Exercice 3.11.** Soit  $r = (ak + b) \bmod p$  et  $s = (al + b) \bmod p$ . Montrez que si  
 $k \neq l$  alors  $r \neq s$   
 $r - s = a(l - k) \bmod p$   
 $a \neq 0, p$  premier  
donc  $a(l - k) \not\equiv 0 \bmod p$

**Exercice 3.12.** Si  $k$  et  $l$  sont choisis aléatoirement sur  $0..p - 1$ , quelle est alors la probabilité d'une collision entre  $h(k)$  et  $h(l)$  ?  
*C'est égal à la probabilité que  $r \equiv s \pmod{m}$ ,  $r$  et  $s$  uniformément répartis sur  $\{1..p\}$ . Pour un  $r$  donné, le nombre d'éléments  $s$  pouvant entraîner une collision est  $\lceil \frac{p}{m} \rceil - 1 \leq \frac{p+m-1}{m} - 1 = \frac{p-1}{m}$  (1 élément tous les  $m$ ). Comme il y a  $p - 1$  choix possibles pour  $s$ , la probabilité de collision est  $\frac{1}{m}$  pour tout  $m$ .*

On dit de telles fonctions de hachage qu'elles sont uniformes. Cette propriété sera très utile dans la suite.

## 4 Gravure & Hachage

En plus d'avoir une très bonne performance en moyenne ( $O(1)$ ), la technique de hachage peut également avoir une très bonne efficacité **dans le pire cas**. On parle de **Hachage parfait** lorsque le nombre d'accès mémoire pour une recherche est au pire cas  $O(1)$ . On peut réaliser un hachage parfait lorsque l'ensemble des clés est *statique*, c'est-à-dire qu'il ne change plus une fois les clés stockées dans la table.

**Exercice 4.13.** Montrez que si  $h$  est une fonction de hachage uniforme choisie aléatoirement, alors la probabilité d'avoir des collision parmi  $n$  valeurs hachées contenues dans un tableau de taille  $n^2$  est inférieure à  $\frac{1}{2}$ .  
*nombre de paires pouvant entrer en collision :  $\frac{n(n-1)}{2}$ , proba collision :  $1/m = 1/n^2$*

**Exercice 4.14.** Comment réaliser un hachage parfait avec un petit nombre d'éléments? *On prend comme taille de table de hachage  $n^2$ , et on cherche par tâtonnement une fonction de hachage uniforme qui soit parfaite pour cet ensemble de clés.*

**Exercice 4.15.** \* Imaginez une méthode combinant la résolution de conflits par double hachage et celle par chaînage pour avoir un hachage parfait même dans le cas où il y a beaucoup d'éléments. *On prend une première table de hachage de taille  $n$  et une fonction de hachage uniforme quelconque. A l'indice  $i$ , il y a  $m_i$  éléments. On place ces éléments sur une table de hachage auxiliaire parfaite de taille  $m_i^2$  (de la même façon que dans la question précédente)*