

## EA4 – Éléments d’algorithmique

### TD n° 10 : hachage (suite)

#### Exercice 1 : tables de hachage et adressage ouvert

On considère une table de hachage  $T$  à adressage ouvert, de longueur  $m = 11$  avec la fonction de hachage  $h(k) = k \bmod 11$ .

1. Insérer successivement dans  $T$  les clés 5, 28, 19, 14, 20, 33, 37, 17, 26 et 10 en appliquant les méthodes de résolution des collisions suivantes :
  - a. sondage linéaire (autrement dit, on sonde la case  $h(k) \bmod m$  et si elle est déjà occupée on passe à la case  $h(k) + 1 \bmod m$  et ainsi de suite).
  - b. double hachage (autrement dit, on sonde successivement les cases  $h_1(k) + i \times h_2(k) \bmod m$ ) avec  $h_1(k) = h(k)$  et  $h_2(k) = 1 + (k \bmod (m - 1))$ .
2. Supprimer ensuite les clés 19, 37 et 17 en détaillant la suite de sondages réalisés.
3. Comment se passe ensuite l’insertion de 26 ? de 39 ? Préciser quels sont les sondages réalisés et la case finalement choisie pour l’insertion.
4. Plus généralement, quel est le bon paramètre à considérer pour décider quand procéder à un « redimensionnement » (qui peut éventuellement se borner à un simple nettoyage sans changement de dimension) ? Est-ce le même pour les insertions et les suppressions ?

#### Exercice 2 : hachage et doublons

Soit  $T$  un tableau de taille  $n$  possédant au plus  $m$  entiers différents. On veut calculer son nombre exact de valeurs différentes.

1. Donner une solution en  $O(n \log n)$  ne faisant pas appel au hachage.
2. Donner une solution faisant appel au hachage. Quelle sera sa complexité en mémoire ?
3. Avec l’hypothèse simplificatrice que l’on a accès à une fonction de hachage sans collision sur nos données (c’est-à-dire telle que  $h(x) \neq h(y)$  pour deux éléments distincts dès lors que la table est assez grande), quelle est la complexité de l’algorithme ?
4. Quelle est la complexité dans le pire cas si l’on suppose qu’on n’a jamais plus que des  $k$ -collisions (c’est-à-dire que jamais plus de  $k$  valeurs différentes ont la même image par la fonction de hachage) ?
5. Sous quelles hypothèses moins restrictives a-t-on une complexité *moyenne* du même ordre ?

#### Exercice 3 : borne en moyenne de plus long sondage

Le but de cet exercice est de prouver que dans une table à adressage ouvert, avec taux de remplissage  $\alpha$  borné et hachage supposé uniforme, la longueur moyenne de la plus longue suite de sondages est en  $O(\log n)$ .

On se place pour simplifier dans le cas  $\alpha = \frac{1}{2}$ , avec table initialement vide de longueur au moins  $2n$ , et on insère successivement  $n$  éléments. On note  $S_i$  le nombre de sondages nécessaires pour la  $i^{\text{e}}$  insertion.

1. Rappeler pourquoi  $\mathbb{P}(S_i > k) \leq 1/2^k$  pour tous  $i$  et  $k$ .
2. En déduire que  $\mathbb{P}(S_i > 2 \log n) \leq 1/n^2$  pour tout  $i$ .
3. Soit  $S = \max_{1 \leq i \leq n} S_i$ . Montrer que  $\mathbb{P}(S > 2 \log n) \leq 1/n$ .
4. En déduire que  $\mathbb{E}(S) \in O(\log n)$ .

**Exercice 4 : algorithme de Rabin et Karp (examen 2015)**

L'objectif de cet exercice est de donner une application des fonctions de hachage à la recherche d'un motif dans un texte.

On commence par proposer une méthode naïve.

1. Écrire une fonction `test_occurrence(m, t, i)` retournant True si un motif `m` apparaît à la position `i` d'un texte `t` (`m` et `t` étant représentés par des tableaux de caractères).
2. Écrire une fonction `recherche_naive(m, t)` effectuant une recherche naïve du motif `m` dans `t` à l'aide de `test_occurrence(m, t, i)`.
3. Quelle(s) opération(s) est-il raisonnable de prendre en compte pour le calcul de la complexité (en temps) des algorithmes précédents ? En déduire la complexité de l'algorithme `recherche_naive(m, t)`. Justifier.

Dans la suite on fera l'hypothèse que les chaînes de caractères ont été remplacées par des tableaux d'entiers en considérant la valeur entière du code binaire de chaque caractère (via la fonction `ord()` en Python, ou une conversion de format en Java).

L'algorithme de Rabin et Karp améliore l'algorithme `recherche_naive(m, t)` précédent. Il repose sur l'utilisation d'une fonction de hachage.

Soit  $b \geq 2$  un entier fixé ; pour tout entier  $k \geq 1$ , on considère la fonction  $h_k : \mathbb{N}^k \rightarrow \mathbb{N}$  telle que :

$$h_k(x_{k-1}, \dots, x_0) = x_{k-1} \cdot b^{k-1} + x_{k-2} \cdot b^{k-2} + \dots + x_2 \cdot b^2 + x_1 \cdot b + x_0$$

4. Rappeler le principe de la méthode de Horner pour évaluer un polynôme.
5. Soit `t` un tableau d'entiers, de longueur  $\ell \geq k$ . Décrire un algorithme `initialise_h(t, k)` permettant de calculer  $h_k(t[0], \dots, t[k-1])$  en  $\Theta(k)$  opérations arithmétiques sur des entiers (additions ou multiplications).
6. Si les entiers considérés sont quelconques, ce calcul est-il réellement de complexité linéaire ? Et si tous les calculs sont faits *modulo* un entier  $q$  fixé ?

*Dorénavant, tous les entiers sont considérés modulo q (inutile de réécrire `initialise_h(t, k)`).*

7. Quelle relation existe-t-il entre  $h_k(t[0], \dots, t[k-1])$  et  $h_k(t[1], \dots, t[k])$  ?
8. En déduire un algorithme `affiche_tous_h(t, k)` qui calcule puis affiche toutes les valeurs  $h_k(t[i], \dots, t[i+k-1])$  pour  $i \in [0, \text{len}(t) - k]$  avec une complexité totale en  $\Theta(\text{len}(t))$ .
9. Le principe de l'algorithme de Rabin et Karp est d'utiliser la fonction  $h_k$ , pour un  $k$  bien choisi, comme un filtre pour limiter le nombre d'appels à `test_occurrence(m, t, i)`. Écrire un algorithme `recherche_RK(t, m)` recherchant le motif `m` dans `t` selon ce principe.
10. Déterminer sa complexité dans le pire des cas, en donnant un exemple de tel pire cas.
11. En faisant l'hypothèse d'uniformité suivante :

*b et q peuvent être (et sont) choisis de telle manière que la répartition des images des k-uplets de lettres soit (quasi)-uniforme sur  $[0, q - 1]$*

déterminer la complexité en moyenne de `recherche_RK(t, m)` en fonction de  $\ell := \text{len}(t)$ , de  $k = \text{len}(m)$ , de  $q$  et du nombre  $p$  d'occurrences trouvées. En déduire que, sous des hypothèses raisonnables, cette complexité moyenne est en  $\Theta(\ell)$ .

# Table de Hashage

$h(x) \bmod m$

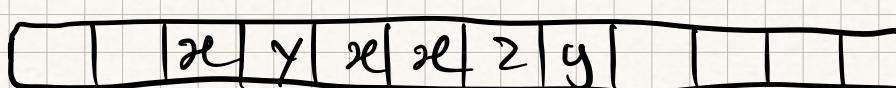


Adressage fermé

22

3

$h(x) \bmod m$



Adressage ouvert

1<sub>y</sub>  $h(y) \bmod m \rightarrow +1 \rightarrow +1\dots$  sondage linéaire

2<sub>y</sub>  $h(y) \bmod m \rightarrow + h_2(y) \rightarrow + h_2(y)$

Double hashage

Ex 1

$$m = 11$$

$$h(k) = k \bmod 11$$

$$h_2(k) = 1 + (k \bmod (m-1))$$

on revient

$$h(k) = k \bmod 11$$

$h_1$	0	1	2	3	4	5	6	7	8	9	10	$h(x)$	$h_2(x)$
1 <sub>y</sub>	33	10		14	37	5	28	14	19	20	26	22	

26?

17?

ya déjà qu'un

5

5

6

28

6

9

0	1	2	3	4	5	6	7	8	9	10
33	14	14	34	5	28	26	19	20	10	

5	6	7	8	1	2	3	4	5	6	7
8	1	2	3	4	5	6	7	8	1	2
7	4	5	6	7	8	9	10	1	2	3
6	7	8	9	10	1	2	3	4	5	6
7	1	2	3	4	5	6	7	8	9	10

$h_2$

19	8	13
14	3	5
20	9	1
33	0	4
34	4	8
14	6	8
26	4	4
10	10	1
pas		

Si  $h_2(\infty)$  est premier avec  $m$  on finira toujours par trouver une case vide si elle existe

On se décale de  $\times$  cases  
 $h_2(26)$

$$\begin{aligned} &= 1 + (26 \bmod 13) \\ &= 7 \end{aligned}$$

2, 19, 34 et 14

ay	33	10	14	34	5	28	14	X	20	26
----	----	----	----	----	---	----	----	---	----	----

0	1	2	3	4	5	6	7	8	9	10
33	14	14	34	5	28	26	14	20	10	
95 8 7 6 5 4 3 2 1 0	6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	18 17 16 15 14 13 12 11 10 9	1 2 3 4 5 6 7 8 9 10	9 8 7 6 5 4 3 2 1 0	3 2 1 0	14 13 12 11 10 9 8 7 6 5	1 2 3 4 5 6 7 8 9 10	36 35 34 33 32 31 30 29 28 27	47 46 45 44 43 42 41 40 49 48

$$w(18) = 8 \text{ on supp case } 8 \rightarrow 19$$

$$h(34) = 4 \text{ On supp } 34 \text{ case } 9$$

$h(14) = 6$  On supp. 6 cases et pas celui qu'en voulait donc décale 8 case  $\times$

3<sub>y</sub>

## Insertion avec suppressions

$$i = h(x)$$

tant que  $T[i]$  n'est pas vide ni  $\infty$

| et pas un élément supprimé

$$| \quad i = (i+1) \bmod m \quad \| i = (i+h_2(x)) \bmod m$$

$$T[i] = \infty$$

tant que  $T[i]$  n'est pas vide ou  $\infty$ :

$$i = (i+1) \bmod m$$

Si  $T[i]$  est  $\infty$ : marquer  $T[i]$  comme supp

	4	7	8	10
33   10   14   <del>26</del>   5   28   <del>26</del>   <del>26</del>   20   <del>26</del>				

	4	7	8	13
33   10   14   26   5   28   39   <del>26</del>   20   <del>26</del>				

	2	4	7	8
33   39   14   26   5   28   <del>26</del>   <del>26</del>   20   10				

Insérer 26 à  $j$  dans le tab. On supp et va que cas 4 est vide (3<sup>e</sup> supp)  
 On met 26 à case 4 unique  
 $h(26) = 4 \rightarrow$  case 4  
 On profite de rapprocher le chiffre à nous

4<sub>y</sub> Insertions : On regarde le nbre de "occupé"  
 Suppressions : On regarde le nbre de case marquée

Ex2

1) On tri le tab  $\mathcal{O}(n \log n)$

On parcourt tableau et on compte les valeurs diff dans  $n + n \log(n) \rightarrow \mathcal{O}(n \log n)$

Tri par fusion ou tri rapide

Parcourir le tab trié et compter les valeurs diff  $\rightarrow \mathcal{O}(n)$

$$\Rightarrow \mathcal{O}(n \log n + n) = \mathcal{O}(n \log n)$$

def count(T)

tri (T)

R = 0

$\mathcal{O}(n)$  (  $\left( \begin{array}{l} \text{for } i \text{ in range}(1, \text{len}(T)) \\ \text{if } T[i-1] \neq T[i] \\ R += 1 \end{array} \right) \mathcal{O}(n \log n)$

return R + 1

Quand le tableau est trié, les éléments identiques sont côte à côte, il suffit donc de compter le nb de régrains égaux

AFR :  $\mathcal{O}(n \log n)$

2x Parcourez le tab T et pour chaque élément, ajouter -le à l'ensemble de hachage sans double Il y a m éléments dans la table  $m < t_{\max}$  taille

taux remplissage >  $t_{\min}$

$$\frac{m}{taille} > \alpha \Rightarrow taille < \frac{m}{\alpha}$$

La complexité en mémoire est proportionnelle au nb de val diff dans tab,  $O(m)$

3x On utilise une table de hachage dont on effectue un redimensionnement en doublant la taille dès qu'il y a une collision. Ainsi chaque inserti on se fait en temps constant, et les redimensionnent ont un coût total de  $O(n)$