

# Algorithmique Avancée

Cours d'Antoine Genitrini, dispensé par Maryse Pelletier

`Maryse.Pelletier@lip6.fr`

Master Informatique 1

`https://www-master.ufr-info-p6.jussieu.fr/2020/ALGAV`

Année 2020-2021

# CHAPITRE 3

## MÉTHODES DE HACHAGE

### Plan du Chapitre 3

- Fonctions de hachage et gestion des collisions
- Hachage avec chaînage et hachage avec calcul
- Filtres de Bloom
- Hachage dynamique et hachage extensible
- Comparaison avec la recherche arborescente
- Hachage universel et hachage parfait
- Hachage cryptographique

# Rappels : Structures-Efficacité

Ensemble de  $n$  clés.

Chaque clé représentée sur  $\leq L$  caractères.

## Nombre de comparaisons pour une recherche/ajout

Structure	en Moyenne	au Pire	Mémoire
ABR	$O(\log n)$	$O(n)$	$n + 2n$ ref.
ABR-Equilibré	$O(\log n)$	$O(\log n)$	$n + 2n$ ref.
Arbre Digital <sup>1</sup>	$O(\log n)$	$L$	$n + 2n$ ref.
Trie <sup>1, 2</sup>	$O(\log n)$	$L$	$n + Ln$ ref.
Hachage <sup>3</sup>	$O(1)$	$O(n)$	$n$

- 
1. accès aux caractères
  2. comparaisons de caractères
  3. calcul fonction de hachage

# Plan du cours

- 1 Méthodes de hachage
- 2 Filtres de Bloom
- 3 Hachage externe
- 4 Hachage et randomisation
- 5 Hachage cryptographique

# Méthodes de hachage

Table  $T$  de taille  $m$  contenant des clés.

**Opérations** : Recherche, Insertion d'une clé  $x$  dans  $T$   
Suppression difficile

**Fonction de hachage**  $h : \mathcal{U} \mapsto \{0, \dots, m-1\}$

Accès direct dans  $T$  selon la valeur de hachage

**Transformation de la clé en une adresse**

(et pas d'accès aux bits de la clé comme dans les Tries)

---

```
def H-ajout(x, T, h):
    """ elt * Table * FonctionH -> Table
        Renvoie la table resultant de l'ajout de x. """
    v = h(x)
    if EstVide(T[v]):
        T[v] = x
    elif T[v] != x:
        GererCollision(x, T)

    return T
```

---

# Fonctions de hachage

Fonction de hachage  $h : \mathcal{C} \subset \mathcal{U} \mapsto \{0, \dots, m-1\}$

- calcul de fonction de hachage

- division :  $h(x) = x \bmod m$ , ( $m$  premier)

- multiplication :  $h(x) = \lfloor m \cdot (\lambda x - \lfloor \lambda x \rfloor) \rfloor$ ,  
 $\lambda = \frac{\sqrt{5}-1}{2}$

- but : obtenir *répartition uniforme des clés*

$$\forall x \in \mathcal{C}, \forall i \in \{0, \dots, m-1\}, \quad (h(x) = i) = \frac{1}{m}$$

- mais il y a *toujours des collisions* :  $x \neq y$  et  $h(x) = h(y)$ .

$$(0 \text{ collision}) = \frac{\# \text{injections de } [n] \text{ dans } [m]}{\# \text{fonctions de } [n] \text{ dans } [m]} = \frac{m(m-1) \dots (m-n+1)}{m^n}$$

$\Rightarrow$  nécessité de gérer les collisions

# Analyse du nombre de collisions primaires

- Hypothèse d'**uniformité** de la fonction de hachage

$$\forall x \in \mathcal{C}, \forall i \in \{0, \dots, m-1\}, \quad (h(x) = i) = \frac{1}{m}$$

- **Probabilité** que  $k$  clés aient même valeur de hachage  $v$

$$(X = k) = (|h^{-1}(v)| = k) = \binom{n}{k} \frac{1}{m^k} \left( \frac{m-1}{m} \right)^{n-k}.$$

- **Moyenne** nb clés par case :

$$\mathbb{E}(|h^{-1}(v)|) = \sum k \cdot (X = k) = \frac{n}{m}.$$

- **Variance** :

$$\mathbb{V}ar(|h^{-1}(v)|) = \mathbb{E}((X - \mathbb{E}(X))^2) = \frac{n}{m} \left( 1 - \frac{1}{m} \right).$$

# Paradoxe des anniversaires

: Probabilité pour qu'il y ait au moins 1 collision

$$\begin{aligned}
 &= 1 - \prod_{i=0}^{n-1} \left(1 - \frac{i}{m}\right) \\
 &= 1 - \exp \sum_{i=0}^{n-1} \log \left(1 - \frac{i}{m}\right) \sim 1 - e^{-\frac{n^2}{2m}}.
 \end{aligned}$$

$\Rightarrow$  pour  $= 0.5$

$$n \sim \sqrt{2m \log 2}, (m = 365 \rightarrow n = 23)$$

**Paradoxe des anniversaires** : étant donné un groupe de plus de 23 personnes, la probabilité qu'au moins 2 d'entre elles aient la même date anniversaire est supérieure à 1/2.



# Gestion des collisions

Différentes méthodes pour gérer les collisions

- **Hachage Chainage Séparé** : toutes les clés ayant la même valeur de hachage sont dans une structure extérieure (LC)
  - coût moyen d'une recherche négative :
 
$$\frac{1}{m} \sum_{i=1..m} L_i = \frac{n}{m} =: \alpha, \quad (\alpha : \text{taux de remplissage})$$
  - coût moyen d'une recherche positive :
 
$$\frac{1}{n} \sum_{i=0..n-1} 1 + \frac{i}{m} \sim 1 + \frac{\alpha}{2}$$
  - coût pire  $\sim n$  : toutes clés même valeur de hachage
- à l'**intérieur de la table**
  - par **calcul** :  $(h_i(x))_{i=1..m}$  suite d'essais, couvrant la table
    - HLinéaire :  $h_i(x) = h(x) + (i - 1) \bmod m$
    - Hquadratique :  $h_i(x) = h(x) + (i - 1)^2 \bmod m$
    - HDouble :  $h_i(x) = h(x) + d(x)(i - 1) \bmod m$
  - par **chaînage** (HCoalescent)

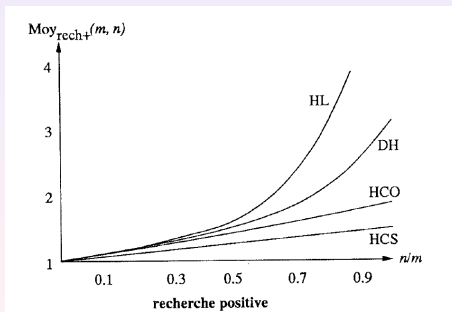
# Caractéristiques des méthodes de Hachage

- *Taux de remplissage*  $\alpha = \frac{n}{m} < 1$ , (sauf pour HCS).
- Très bien adaptées à la gestion d'ensembles statiques (choisir  $\alpha \sim 60\%$ )
- Mal adaptées aux suppressions
- Si table trop pleine il faut la réorganiser en augmentant la mémoire allouée, et en “rehachant” tous les éléments ( $\Rightarrow$  indisponibilité provisoire)

# Comparaison des performances

Opération fondamentale = comparaison entre clés (on ne compte pas coût hachage)  
Sous l'hypothèse de répartition uniforme des clés :

- HChainageSéparé a les meilleures performances, et la suppression est simple.
- HCoalescent très bon mais utilise place pour chaînages internes.
- DoubleH meilleur que HLinéaire, mais nécessite calcul de 2 fonctions de hachage.



Pour  $\alpha < 60\%$ , toutes méthodes en moyenne moins de 2 essais (mais au pire  $n$ ).

# Plan du cours

- 1 Méthodes de hachage
- 2 Filtres de Bloom
- 3 Hachage externe
- 4 Hachage et randomisation
- 5 Hachage cryptographique

# Exemple d'application : Filtres de Bloom

Si l'ensemble de clés à stocker est très grand (BD), alors il faut stocker en mémoire externe, et accès disque coûteux

→ rajouter structure intermédiaire pour déterminer appartenance d'un élément à l'ensemble, avant d'aller le chercher effectivement par hachage.

*Filtre de Bloom* : algo. probabiliste (basé aussi sur hachage !) avec faux positif possible (élément n'étant pas dans l'ensemble mais identifié comme y étant).

## Filtre de Bloom pour ensemble $S$

$F$  tableau de  $m$  bits, initialisé à 0.

$k$  fonctions de hachage uniformes indépendantes  $(h_i)_{i=1,\dots,k}$

$$h_i : \mathcal{U} \rightarrow \{0, \dots, m-1\}.$$

$S$  ensemble de  $n$  éléments de  $\mathcal{U}$  ( $\text{card}(\mathcal{U}) \gg n$ ).

Construction du filtre :  $\forall s \in S, \forall i = 1, \dots, k, F[h_i(s)] := 1$ .

## Test d'appartenance d'un élément $x$ à $S$

si  $\forall i = 1, \dots, k, F[h_i(x)] = 1$ , alors répondre OUI sinon NON

*Faux positif* :  $x \notin S$  déclaré dans  $S$  possible, **mais pas le contraire**.

# Probabilité d'erreur (faux positif)

Faux positif  $y$  :  $h_i(y) = 1, \forall i = 1, \dots, k$ , alors que  $y \notin S$

- La probabilité qu'un bit fixé de  $F$  soit toujours à 0 après avoir construit le filtre de Bloom associé à  $S$  est

$$\left(1 - \frac{1}{m}\right)^{kn} = \exp\left(kn \log\left(1 - \frac{1}{m}\right)\right) \sim e^{-kn/m}$$

- La probabilité que pour une clé  $y$ , on ait  $h_i(y) = 1, \forall i = 1, \dots, k$  est donc

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \sim (1 - e^{-kn/m})^k$$

La probabilité d'erreur est donc de l'ordre de  $(1 - e^{-kn/m})^k$ .  
Dimensionnement (voir TD)

# Plan du cours

- 1 Méthodes de hachage
- 2 Filtres de Bloom
- 3 Hachage externe**
- 4 Hachage et randomisation
- 5 Hachage cryptographique

# Hachage Externe

- Pour ne pas avoir à “rehacher”
- Pour traiter la recherche externe (mémoire secondaire)

La table est remplacée par un index (arbre lexicographique binaire), et les feuilles adressent vers des pages en mém. s.

## Hachage dynamique et Hachage extensible

- analogie avec les B-arbres : éléments dans pages, qui éclatent quand elles sont pleines
- analogie avec Accès Séquentiel indexé : on maintient un index pour guider vers la page (et si index trop grand, on le pagine lui aussi)
- méthodes de hachage (fonctions de hachage pour disperser les clés) : clé  $x \rightarrow h(x)$
- utilisent propriétés binaires des valeurs de hachage des clés (index sous la forme d'un arbre lexicographique)



# Accès Séquentiel Indexé

Liste des clés triée.

Éléments rangés séquentiellement dans les pages.

*Exemple* : Encyclopédie Universalis

**Index** : Page  $i$  contient clés jusqu'à  $F_i \Rightarrow$  Pour rechercher la clé  $x$  à partir de l'index  $D$ , on calcule (dichotomie) l'indice  $i$  tel que  $F_{i-1} < x \leq F_i$ , et on renvoie sur la page  $i$ .

Index sur plusieurs niveaux (Index des disques et sur chaque disque index des pages)

Performant pour la recherche, mais un ajout peut nécessiter le **recalcul de toutes les pages** !

# Hachage dynamique

- Table de hachage remplacé par index = arbre lexicographique binaire
- Index fabriqué par raffinements successifs d'une fonction de hachage. clé  $x \rightarrow h(x) \in \{0, 1\}^*$

Pour rechercher un élément  $x$

- on suit un chemin dans l'arbre lexicographique, qui mène à une feuille pointant sur la page contenant  $x$ ,
- le chemin suivi est guidé par la valeur de hachage de  $x$ .

Ajouts augmentent le nombre d'éléments dans une page

- allouer de nouvelles pages en mém sec
- répartir les éléments dans les pages en utilisant plus ou moins d'informations de la valeur binaire de leur fonction de hachage (selon qu'il y a plus ou moins de collisions)
- nouvelles pages référencées par nouvelles feuilles de l'arbre lexicographique (qui croît).

# Exemple

Hachage dynamique des clés  $E, X, T, F, R, N, C, L, S, G, B$ ,  
avec pages de capacité  $C = 4$

$h(E) = 00101, h(X) = 11000, h(T) = 10100, h(F) = 00110,$   
 $h(R) = 10010, h(N) = 01110, h(C) = 00011, h(L) = 01100,$   
 $h(S) = 10011, h(G) = 00111, h(B) = 00010$

Tri des éléments dans une page :  
temps négligeable par rapport au temps d'accès.

# Hachage Extensible

- index = arbre lexicographique parfait  $\Rightarrow$  représentable par un tableau de  $2^d$  mots
- chaque mot est une suite de  $d$  bits (nombre entre 0 et  $2^d - 1$ ) qui adresse vers une page
- Pour rechercher la clé  $x$ , on utilise les  $d$  premiers bits de  $h(x)$ , pour accéder à une page
- plusieurs mots peuvent adresser la même page : ( $2^{d-k}$  si les clés de cette page ont les  $k$  mêmes premiers bits pour leurs valeurs de hachage)

Lors d'un ajout, les modif. peuvent être à plusieurs niveaux :

- insertion dans une page (si elle n'est pas pleine)
- éclatement d'une page avec redistribution des clés (si plusieurs pointeurs de l'index vers cette même page) et l'index n'est pas modifié
- doublement de l'index

# Exemple

Hachage extensible des clés

$E, X, T, E, R, N, A, L, S, E, A, R, C, H, I, N, G, E, X, A, M, P, L, E,$   
avec pages de capacité  $C = 4$ .

$h(E) = 00101, h(X) = 11000, h(T) = 10100, h(R) = 10010,$   
 $h(N) = 01110, h(A) = 00001, h(L) = 01100, h(S) = 10011,$   
 $h(C) = 00011, h(H) = 01000, h(I) = 01001, h(G) = 00111,$   
 $h(M) = 01101, h(P) = 10000$

# Hachage/Arbres de Recherche

- Mémoire centrale : Complexité en nombre de comparaisons

	ABR-Equilibré	Hachage
Recherche, ajout	$O(\log n)$ // $O(\log n)$	$O(1)$ // $O(n)$
Suppression	$O(\log n)$	—
Recherche par intervalle (ou tris)	$O(\log n)$	—

- Mémoire secondaire paginée :
  - Complexité en nombre d'accès aux pages  
temps d'accès  $\gg$  temps de traitement
  - Taux de remplissage des pages ( $\alpha = n/Cm$ )

	B-arbres	H-Extensible
Recherche, ajout, suppression	$O(1)$	$O(1)$
Taux de remplissage	70%	70%
Lectures séquentielles (tris)	Oui	Non
Accès concurrents	complexes	plus simples

# Plan du cours

- 1 Méthodes de hachage
- 2 Filtres de Bloom
- 3 Hachage externe
- 4 Hachage et randomisation**
- 5 Hachage cryptographique

# Hachage universel

Choix aléatoire fonction de hachage  $\rightarrow \sim h$  uniforme

## Ensemble universel de fonctions de hachage

- $\mathcal{H} : \{h : \mathcal{U} \rightarrow \{0, \dots, m-1\}\}$  ensemble fini ;
- $\mathcal{H}$  est *universel* ssi  $\forall x, y \in \mathcal{U}, x \neq y,$   
 $|\{h \in \mathcal{H}; h(x) = h(y)\}| \leq \frac{|\mathcal{H}|}{m}$

D'où pour  $h \in \mathcal{H}$  aléatoire :  $\forall x \neq y \in \mathcal{U}^2, \mathbb{P}(h(x) = h(y)) \leq \frac{|\mathcal{H}|}{m} \cdot \frac{1}{|\mathcal{H}|} = \frac{1}{m}.$

## Propriété

Soit  $h$  fonction aléatoire dans  $\mathcal{H}$  universel ; si  $h$  répartit  $n$  clés dans une table de taille  $m$ , alors  $\forall x$ , le nombre moyen de clés  $y$  telles que  $h(y) = h(x)$  est inférieur à  $n/m$ .

$$\sum_{y \neq x} \mathbb{P}(h(x) = h(y)) \leq \frac{n-1}{m}$$



# Tirer aléatoirement une fonction de hachage

On dispose d'une fonction de hachage uniforme

$h : E \rightarrow [0, m - 1]$ ,

on peut créer alors une autre fonction  $h'$ ,

$$h'(x) = (a \times h(x) + b) \bmod m$$

avec  $a, b$  deux entiers tirés aléatoirement entre 0 et  $m - 1$

En pratique : technique peu coûteuse + bons résultats

On appelle ces fonctions 1-universelle (voir TD).

# Construction d'un ensemble universel

- on suppose  $m$  premier ;
- on écrit les clés en base  $m$  :  
 $\forall x \in \mathcal{U}, x = (x_0, x_1, \dots, x_r)$ , avec  $0 \leq x_i < m$
- randomisation : choisir  $a = (a_0, a_1, \dots, a_r)$ ,  
 avec chaque  $a_i$  aléatoire dans  $\{0, 1, \dots, m-1\}$

## Théorème

Soit  $h_a : x \rightarrow \sum_{i=0}^r a_i x_i \mod m$ .

L'ensemble  $\mathcal{H} = \{h_a\}$  est universel, de cardinal  $m^{r+1}$ .

Exemple : adresses IP – 132.227.074.253 – 4 champs de 8 bits :  $x = (x_1, x_2, x_3, x_4)$ .

Pour hacher  $\sim 250$  adresses, choisir  $m = 257$  (nb premier) et

$\mathcal{H} = \{h_a; h_a(x) = a_1 x_1 + a_2 x_2 + a_3 x_3 + a_4 x_4 \mod m, \quad \text{avec} \quad a_i \in [0..256]\}$

# Preuve

Soient  $x \neq y$ ,  $x = (x_0, \dots, x_r)$ ,  $y = (y_0, \dots, y_r)$ , avec  $x_0 \neq y_0$ .

Montrer que le nombre de  $h_a$  tq  $h_a(x) = h_a(y)$  est  $m^r$  (i.e.  $\|\mathcal{H}\|/m$ )

$$h_a(x) = h_a(y) \implies \sum_{i=0}^r a_i x_i \equiv \sum_{i=0}^r a_i y_i \pmod{m}$$

$$\implies a_0(x_0 - y_0) + \sum_{i=1}^r a_i(x_i - y_i) \equiv 0 \pmod{m}$$

$$\implies a_0 = \left(-\sum_{i=1}^r a_i(x_i - y_i)\right) \cdot (x_0 - y_0)^{-1} \pmod{m} \text{ (lemme)}$$

Donc  $\forall a_1, a_2, \dots, a_r$ , il existe un seul  $a_0$  tel que  $h_a(x) = h_a(y)$ .

Et le choix des  $a_1, a_2, \dots, a_r$  donne  $m^r$  fonctions possibles.

Donc  $\mathcal{H}$  est universel.

**Lemme** : Si  $m$  est premier, alors  $\forall z \neq 0 \in \mathbb{Z}/m\mathbb{Z}$ ,  $\exists ! z^{-1}$  tq.  $z \cdot z^{-1} = 1$

Ex. pour  $m = 7$ ,  $z = 1, 2, 3, 4, 5, 6$ ,  $z^{-1} = 1, 4, 5, 2, 3, 6$ .

# Hachage parfait

**Ensemble statique de  $n$  clés** : Pire cas  $O(1)$ , mémoire  $O(n)$

**Idée** : deux niveaux de hachage universel

- choisir  $m$  premier,  $m \sim n$  et  $h_1 \in \mathcal{H}$  universel.
- pour  $i = 1..m$ , soit  $n_i$  le nombre de clés tq  $h_1(x) = i$
- pour chaque  $i$ , choisir  $m_i$  premier,  $m_i \sim n_i^2$  et  $h_{2,i} \in \mathcal{H}$  universel.

**Mémoire totale**  $O(n)$  en moyenne car  $\sum \mathbb{E}(n_i^2) = O(n)$ .

$\forall i, n_i^2 = n_i + 2\binom{n_i}{2}$ , donc  $\mathbb{E}[\sum n_i^2] = n + 2\mathbb{E}[\sum \binom{n_i}{2}]$ . Or  $\sum \binom{n_i}{2} = \text{nb total collisions}$ . Et comme  $\mathcal{H}$  universel,  $\mathbb{E}[\text{nb total collisions}] = \frac{1}{m} \binom{n}{2} = (n-1)/2$ .

**Pas de collision au second niveau** (proba  $> 1/2$ )

## Théorème

Soit  $\mathcal{H}$  un ensemble universel pour une table de taille  $m \sim n^2$  ; le nb de collisions pour hacher les  $n$  clés est  $< \frac{1}{2}$  en moyenne.

$\forall (x, y), (h(x) = h(y)) \leq \frac{1}{m} \sim \frac{1}{n^2}$ . Et le nombre de couple est  $\binom{n}{2}$ . Donc le nombre moyen de collisions est  $\binom{n}{2} \frac{1}{n^2} < \frac{1}{2}$

## Corollaire

Dans ces conditions, (aucune collision)  $> \frac{1}{2}$ .

appliquer l'inégalité de Markov :  $\mathbb{P}(X > a) < \mathbb{E}[X]/a$ , avec  $X$  = nb collisions et  $a = 1$

Pour le hachage parfait il suffit donc d'essayer des fonctions de  $\mathcal{H}$  jusqu'à ce qu'il n'y ait pas de collision (prétraitement) ; et ensuite on travaille avec ensemble statique (recherches uniquement).

# Plan du cours

- 1 Méthodes de hachage
- 2 Filtres de Bloom
- 3 Hachage externe
- 4 Hachage et randomisation
- 5 Hachage cryptographique**

# Hachage cryptographique

Utilisation du hachage dans un contexte différent :

## Cryptage et Compression

but = sécuriser les messages

- $\mathcal{M}$  messages de tailles qcq,  $\mathcal{S}$  signatures (empreintes) de taille fixe  $m$  et fonction de hachage  $h : \mathcal{M} \rightarrow \mathcal{S}$
- objectif : "authentifier" le message et sa signature
- Applications
  - vérification de données (web, ftp)  
*Crypter  $M$ , archiver et protéger  $S$  (plus petit). Plus tard pour vérifier que  $M$  n'a pas été modifié, vérifier que même signature (bonne version, pas de virus ...)*
  - authentification de messages  
*Crypter  $M$  en  $M'$  ; signer  $S$  en  $S'$  ; envoyer  $M'$  et  $S'$ . Le destinataire décode la signature  $S'$  en  $S$  et vérifie que  $h(M) = S$ . (la fonction  $h$  est publique)*

# Propriétés Hachage cryptographique

- $h$  “compresse” :  $\{0, 1\}^* \rightarrow \{0, 1\}^m$
- $h$  à sens unique : *facile* à calculer et *très difficile* à inverser
  - *facile* : calculable en temps-mémoire polynomial
  - *très difficile* : techniquement impossible actuellement (pas de  $h$  prouvée mathématiquement non inversible)

## Propriétés recherchées

1. **Collisions faibles difficile** : étant donné  $x \in \{0, 1\}^*$ , il doit être *très difficile* de trouver  $x' \neq x$  tel que  $h(x) = h(x')$ .  
ex : vérification de données
2. **Collisions fortes difficile** : il doit être *très difficile* de trouver  $(x, x')$  tels que  $x \neq x'$  et  $h(x) = h(x')$ .  
paradoxe des anniversaires : si  $m = 2^k$ , il faut  $\sim 2^{k/2}$  clés pour avoir une collision avec proba  $> 1/2$ . Donc s'il est possible de stocker (et trier)  $2^{k/2}$  signatures, on a proba  $> 1/2$  de violer cette propriété.



# Méthodes de hachage cryptographique

- MD- Message Digest : Rivest
  - MD4 (1989), MD5 (1991) – 128 bits
  - faille en 96 et collision complète en 2004
  - vérification de téléchargements FTP
- SHA Secure Hash Algorithm : NSA
  - SHA-0 et SHA-1 – 160 bits
  - faille en 93 et collision complète en 2004
  - SHA-256, signature sur 256 bits ..., SHA-512
- RIPEMD-160, Whirlpool (512) : EU project