Algorithmique Avancée

Michèle Soria

Michele.Soria@lip6.fr

Master Informatique M1-STL

http://www-master.ufr-info-p6.jussieu.fr/2009

Année 2009-2010

Plan

Méthodes de hachage

Table T de taille m contenant des clés.
Opérations : Rechercher, Insérer, Supprimer une clé x dans T
Fonction de hachage $h : \mathcal{U} \longrightarrow \{0, \ldots, m-1\}$
Accès direct dans T selon la valeur de hachage

*H-ajout : elt * Table * fonctionH ⟶ Table*
***renvoie** la table résultant de l'ajout de x*
Fonction H-ajout (x, T, h)
  soit v = h(x)
    Si EstVide T(v) alors T(v) := x
    Sinon Si T(v)≠ x alors GérerCollision(x,T)
  Retourne T
Fin Fonction *H-ajout*

Fonctions de hachage

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

- calcul de fonction de hachage
  - division : $h(x) = x \mod m$, (m premier)
  - multiplication : $h(x) = \lfloor m.frac(\lambda x) \rfloor$, $\lambda = \frac{\sqrt{5}-1}{2}$
- but : obtenir *répartition uniforme* des clés
  $\forall x \in \mathcal{C}, \forall i \in \{0, \ldots, m-1\}, \quad \Pr(h(x) = i) = \frac{1}{m}$
- mais il y a *toujours des collisions* : $x \neq y$ et $h(x) = h(y)$.

$$\Pr(\textit{no collision}) = \frac{\#\textit{injections de } [n] \textit{ dans } [m]}{\#\textit{fonctions de } [n] \textit{ dans } [m]} = \frac{m(m-1)\ldots(m-n+1)}{m^n}$$

→ nécessité de gérer les collisions.

Analyse du nombre de collisions primaires

- Hypothèse d'uniformité de la fonction de hachage
  $\forall x \in \mathcal{C}, \forall i \in \{0, \dots, m-1\}, \quad \Pr(h(x) = i) = \frac{1}{m}$
- Probabilité que k clés aient même valeur de hachage v
  $\Pr(h^{-1}(v) = k) = \binom{n}{k} \frac{1}{m^k} \left(\frac{m-1}{m}\right)^{n-k}$
- Moyenne nb clés par case : $E(|h^{-1}(v)|) = \frac{n}{m}$
  Variance : $\frac{n}{m}\left(1 - \frac{1}{m}\right)$

Paradoxe des anniversaires $\quad p =$ Proba collision

$$p = 1 - \prod_{i=0}^{n-1}\left(1 - \frac{i}{m}\right) \sim 1 - e^{-\frac{n^2}{2m}}$$

$\Rightarrow$ pour $p = 0,5$, $n \sim \sqrt{1,38m}$, $(m = 365 \rightarrow n = 23)$

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 :
    $\frac{1}{m}\sum_{i=1..m} L_i = \frac{n}{m} = \alpha,$
  - 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 (HLinéaire, HDouble)
  - par chaînage (HCoalescent)

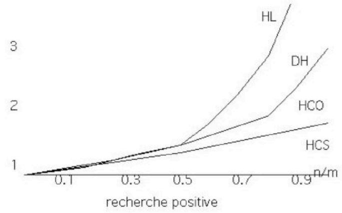
Caractéristiques des méthodes de Hachage

- *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 ($\Rightarrow$ sup "logiques")
- Si table trop pleine il faut 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).

Hachage Externe

- Pour ne pas avoir à "rehacher"
- Pour traiter la recherche externe (M.S.)

La table est remplacée par un index (arbre digital binaire), et les feuilles adressent vers des pages en mémoire secondaire.

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 digital)

Accès Séquentiel Indexé

Liste des clés triées.
Eléments rangés séquentiellement dans les pages.

Exemple : Encyclopedia 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** ! ! !

Arbre lexicographique

Représentation arborescente des mots d'un dictionnaire en évitant de répéter les préfixes communs
(Ex : complétion automatique, vérificateur d'orthographe)

Alphabet de 26 lettres -> arbres avec noeuds d'arité 26
Exemple : a, bac, balle, ballon, bas, base, bus, sac
Recherche, adjonction et suppression par parcours d'une branche en "épelant" le mot.

Alphabet binaire -> arbre binaire digital
Exemple : 0, 01,001, 01101, 11, 110, 111
Recherche, adjonction et suppression par parcours d'une branche (aiguillage à gauche si 0 et à droite si 1)

Hachage dynamique

- Table de hachage remplacé par index = arbre digital
- Index fabriqué par raffinnements 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 digital, 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 MS
- 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 digital (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 digital parfait ⇒ representable 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 (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	$O(\log n)$// $O(\log n)$	————
Recherche par intervalle (ou tris)	$O(\log n)$// $O(\log n)$	————

- Mémoire secondaire paginée :
  - Complexité en nombre d'accès aux pages temps d'accès >> 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

Hachage universel

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

Ensemble universel de fonctions de hachage

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

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

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.

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

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',

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

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 ;
- décomposition des clés en chiffres m-aires : $\forall x \in \mathcal{U}, x = (x_0, x_1, \dots, x_r)$,
- 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 – 132.227.74.253 – 4 champs de 8 bits : $x = (x_1, x_2, x_3, x_4)$.
Pour hacher $\sim$ 250 adresses, choisir $m = 257$ et
$\mathcal{H} = \{h_a; h_a(x) = a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4, \quad \text{avec} \quad a_i \in [0..256]\}$

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. $\|\mathcal{H}\|/m$)

$h_a(x) = h_a(y) \Longrightarrow \sum_{i=0}^{r} a_i x_i \equiv \sum_{i=0}^{r} a_i y_i (\mod m)$
$\Longrightarrow a_O(x_0 - y_0) + \sum_{i=1}^{r} a_i(x_i - y_i) \equiv O(\mod m)$
$\Longrightarrow a_0 = \left(-\sum_{i=1}^{r} a_i(x_i - y_i)\right).(x_0 - y_0)^{-1}(\mod m)$ (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. $z.z^{-1} = 1$
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.

Exemple

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

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 total de collisions pour hacher n clés est $< \frac{1}{2}$ en moyenne.

$\forall (x, y), \Pr(h(x) = h(y)) = \frac{1}{m} \sim \frac{1}{n^2}$. 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, $\Pr(\text{aucune collision}) > \frac{1}{2}$.

appliquer l'inégalité de Markov.

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).

Hachage cryptographique

- Utilisation du hachage dans un contexte différent : Cryptage et Compression
- $\mathcal{M}$ messages de tailles qques, $\mathcal{S}$ signatures (empreintes) de taille fixe m et $h : \mathcal{M} \rightarrow \mathcal{S}$
- "identifier" le message et sa signature
- représentation compacte et paradoxe des anniversaires : si m= 2^k, il faut $\sim 2^{k/2}$ messages pour avoir une collision avec proba 0.5
- Applications
  - vérification de données (web, ftp)
  - authentification de messages

Cryptosystème à clé publique

- Chaque utilisateur a 2 clés $u \rightarrow (P_u, S_u)$, avec
  - inverses : $P_u(S_u(T)) = S_u(P_u(T)) = T, \forall T$ texte
  - P_u publique (tables) et S_u secrète (seul u la connaît)
- Opérations possibles :
  - Crypter message T de A vers B : A transmet $P_B(T)$ à B
  - Signature de A vérif. par tous : A transmet T et $S_A(T)$
  - Crypter et signer : A transmet $P_B(T; S_A(T))$
- Algorithme RSA pour calculer (P_u, S_u) :
  - u choisit p et q deux (grands) nbres premiers ; et $n = pq$
  - u choisit e petit, premier avec $(p-1)(q-1)$ et calcule d inverse de e modulo $(p-1)(q-1)$
  - $P_u = (e, n)$ et $S_u = d$
  - $P_u(S_u(T)) = S_u(P_u(T)) = T, \forall T$

Propriétés Hachage cryptographique

- *h* compresse : $\{0,1\}^\star \to \{0,1\}^m$
- *h* à sens unique : facile à calculer et très difficile à inverser
  - *facile* : calculable en temps-mémoire polynomial
  - *très difficile* : techniquement impossible

Propriétés recherchées

1. Préimage difficile : pour presque tout $y \in \{0,1\}^m$, il doit être *très difficile* de trouver $x \in \{0,1\}^\star$ tel que $h(x) = y$.
2. Collisions faibles difficile : étant donné $x \in \{0,1\}^\star$, il doit être *très difficile* de trouver $x' \neq x$ tel que $h(x) = h(x')$.
3. Collisions fortes difficile : il doit être *très difficile* de trouver (x, x') tels que $x \neq x'$ et $h(x) = h(x')$.

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