# Organisation des données

## Structures internes des données. Tables de hachage

Mihaela JUGANARU-MATHIEU
mathieu@emse.fr

École Nationale Supérieure des Mines de St Etienne

2017-2018

# Dictionnaire et tableau associatif

Problème : garder de manière permanente des éléments en mémoire, pouvoir y accéder, ajouter de nouveaux éléments, supprimer certains.
Un "élément" doit être vu comme une brique de base, pas forcément une unique valeur, mais une structure de complexité variable ayant une composante particulière, appelée **clé**.
L'accès aux éléments stockés se fait par le biais de la valeur de la clé : on recherche un élément ayant une clé donnée, on insère un nouvel élément (qui a une clé), on supprime un élément dont on fournit la clé.
Si la clé est considérée unique, on parle d'un **dictionnaire**, sinon d'un **tableau associatif**.

# Structures simples

# Structures simples

Deux structures internes de données connues :

- liste chainée (simple ou double)
- tableau

Les éléments sont gardés sont dans un ordre quelconque (l'ordre d'apparition, le plus souvent) ou triés. La complexité des opérations dépend de l'ordre, du type de la structure.

La complexité des opération est :

| Structure | Recherche | Insertion | suppression |
|---|---|---|---|
| Vect. n-ord. | $O(N)$ | $O(1)$ | $O(1)$ |
| Liste n-ord. | $O(N)$ | $O(1)$ | $O(1)$ |
| Vect. ord. | $O(\log N)$ | $O(N)$ | $O(N)$ |
| Liste ord. | $O(N)$ | $O(1)$ | $O(1)$ |

$N$ est le nombre d'éléments gardés dans la structure de données.

L'opération de recherche est en $O(\log N)$ dans le meilleur des cas.
*On veut du* $O(1)$ *!*

# Table d'adressage direct

# Table d'adressage direct

Dans l'hypothèse que les éléments sont peu nombreux et que les clés sont des valeurs entières limitées par une valeur $M$ suffisamment petite, il est donc possible d'allouer un tableau de dimension $M$ (ou $M+1$), l'élément de clé $i$, $0 \leq i \leq M$ est gardé à la position $i$. Cette structure est appelée **table d'adressage direct**.

L'insertion, la suppression et la recherche d'un élément se font en temps constant, donc $O(1)$, si les clés sont uniques.

# Table d'adressage direct

Si les clés ne sont pas uniques, on construit plutôt un tableau de pointeurs, chaque pointeur indique le début de la liste chainée qui contient tous les éléments de clé $i$. La complexité des opération sera dépendante de la distribution des clés et de la taille de l'ensemble $E$ des éléments à stocker.

On mesure aussi le taux d'occupation de la table à adressage direct :

$$taux = \frac{|E|}{M}$$

Par définition l'ensemble $E$ des clés possibles est limité : $|U| \leq M$.

# Table de hachage

# Table de hachage

Si l'univers possible des clés $U$ est très grand et l'ensemble des clés qu'on manipule $E$ est relativement petit $E \subset U$ et $|E| \ll |U|$, on peut envisager à garder les éléments avec leur clés dans une **table de hachage** (table de dispersion).

Une table de hachage se compose de :

- un espace de stockage de taille raisonnable $M$
- une fonction de hachage (de dispersion)

$$h : U \to \{1, \ldots M\}$$

$$h(x) = i$$

l'élément $x$ sera casé à la position $i$.

Logiquement $|E| \leq M \ll |U|$.

# Difficultés de mise en oeuvre

- choisir $M$ la taille de la table
- choisir $f$ la fonction de hachage qui dépend de la nature de la clé (valeur numérique ou chaine de caractères/bytes) et ayant des bonnes propriétés

# Fonction de hachage - propriétés

Conditions sur $h$ :

- $h$ est déterministe (implicitement, car c'est une fonction)
- $h$ simple à calculer
- $h$ doit être uniforme :

$$\forall x \in U, \forall i \in \{1, 2, \ldots M\}, P([h(x) = i]) = 1/M$$

- Idéalement $h$ injective pour $E$ :

$$\text{si } h(x_1) = h(x_2) = i \text{ , alors } x_1 = x_2$$

# Fonction de hachage - autres propriétés

Autres conditions à imposer à $h$ selon le contexte :

- $h$ unidirectionnelle : si $h(x)$ est simple à calculer, il est difficile de trouver $x$ pour un $\alpha$ donné tel que $h(x) = \alpha$
- continuité : si $x_1$ et $x_2$ sont proches (selon une quelconque définition), alors $h(x_1)$ et $h(x_2)$ sont proches aussi.
- stricte non continuité (en cryptographie)
- résistante - première préimage et seconde préimage (toujours en cryptographie)
  - si $h(x) = M$ il est difficile de trouver $x'$ tel que $h(x) = h(x')$
  - il est difficile de trouve $x$ et $x'$ avec $x \neq x$‘ tel que $h(x) = h(x')$
- résistante au calcul des nouvelles valeurs : si on connait une suite $(x_i, h(x_i))$, avec $i = 1, K$ on ne peut pas déduire $h(xx)$ pour un nouveau $xx$.

# Fonction de hachage pour nombre entiers

Deux grandes classes de fonction de hachage :

- hachage par division :

$$h(x) = x \; mod \; M$$

ou

$$h(x) = (k\,x) \; mod \; M$$

- hachage par multiplication :

$$h(x) = [M * (x\,A \; mod \; 1)]$$

# Fonction de hachage par division

Version 1 :

$$h(x) = x \; mod \; M$$

On remarque que si $M$ est pair, alors la parité $x$ et de $h(x)$ est la même.
Aussi si $x$ est une représentation en base $b$ et si $M$ est un multiple de $b$ alors on prendra pas en compte les chiffre de poids fort.
Il convient de prendre $M$ un nombre premier assez éloigné des valeurs $b^k + a$, avec $k$ et $a$ petites valeurs.

Version 2 :

$$h(x) = (kx + a) \; mod \; M$$

Cette formule "cache" la polarisation des valeurs, mais les valeurs $k$ et $M$ doivent être premières entre elles.

# Fonction de hachage par multiplication

$$h(x) = [M \times (xA \bmod 1)]$$

$A$ est une valeur réelle (facile à calculer)
$y \bmod 1$ signifie la partie fractionnaire $y \bmod 1 = y - [y]$
Suggestion de D. Knuth : $A = \frac{\sqrt{5}-1}{2}$ (inspiré du nombre d'or) ou un autre nombre irrationnel.

# Fonction de hachage pour chaines de caractères

1. On fait la somme des codes des caractères qui composent la chaine. Par contre, deux permutations ont la même valeur de hachage.
2. On considère la chaine de caractères comme une représentation numérique en base $b$ :

   $$x = \overline{s_1 s_2 \dots s_l}_{(b)}$$

   avec les symboles $s_i$ en base $b$ ($b$ est la taille de l'alphabet). On calcule $xb = s_1 \times b^{l-1} + s_2 \times b^{l-2} + \dots s_l$

Pour des longues chaines de caractères et la représentation en base $b$, on peut appliquer l'opération *mod* plus souvent afin d'éviter le débordement ou l'utilisation des grands nombres.

# Hachage - opérations au niveau de bits

Au lieu de l'addition et le modulo on utilise les opérations au niveau de bit surtout XOR et les décalages.
Opérations binaires (leur codage en langage C) :

- et (AND) &, ou (OR) |, ou exclusif (XOR) ^

Opération unaires :

- complément à 1 ~ *exemple :* $\sim 0101 \rightarrow 1010$
- complément à 2 - *exemple :* $-00010110 \rightarrow 11101010$
- décalage à gauche << $k$ et à droite >> $k$
  *exemples :* $00010110 >> 2 \rightarrow 00000010$ ,
  $00010110 << 3 \rightarrow 10110000$
- rotations à gauche et à droite

# Rappel - opérations au niveau de bits

En langage C les opérations de `~`, `-`, `<<`, `>>`, `&` , `|` et `^` sont proposées pour les types entiers (`char`, `short`, `int`, `long int`), et elles sont suffisantes pour implémenter la rotation, l'accès à un bit bien précis, etc. On peut aussi écrire de fonction pour travailler avec d'autres longueurs de représentation.

On représente souvent une suite en bits en hexadécimal.
*Exemple :* $00010110 = 0x16$.

# Exemple 1 - fonction de Jenkins One-at-a-time

```
uint32_t jenkins_one_at_a_time_hash(char *key, size_t len)
{
    uint32_t hash, i;
    for(hash = i = 0; i < len; ++i)
    {
        hash += key[i];
        hash += (hash << 10);
        hash ^= (hash >> 6);
    }
    hash += (hash << 3);
    hash ^= (hash >> 11);
    hash += (hash << 15);
    return hash;
}
```

# Exemple 1 - fonction de Pearson

```
void Pearson16(const unsigned char *x, size_t len,
          char *hex, size_t hexlen)
{
    size_t i, j;
    unsigned char hh[8];
    static const unsigned char T[256] = {
 // 256 values 0-255 in any (random) order suffices
    98,  6, 85,150, 36, 23,112,164,135,207,169,  5, 26, 64,165,219, //  1
    61, 20, 68, 89,130, 63, 52,102, 24,229,132,245, 80,216,195,115, //  2
.....  };
   for (j = 0; j < 8; j++) {
     unsigned char h = T[(x[0] + j) % 256];
     for (i = 1; i < len; i++) {
        h = T[h ^ x[i]];
     }
     hh[j] = h;
   }
    snprintf(hex, hexlen, "%02X%02X%02X%02X%02X%02X%02X%02X",
    hh[0], hh[1], hh[2], hh[3],
        hh[4], hh[5], hh[6], hh[7]);
 }
```

# La recherche dans une table de hachage

Si $h$ injective, la recherche d'un élément $x$ est la suivante :

- calcul de $h(x)$, valeur $i$
- recherche dans l'alvéole (case) $i$ du tableau. Si case vide, recherche échouée. Soit $e$ l'élément situé en $i$.
- Si $x = e$, succès
- Si $x \neq e$, recherche échouée

Si on veut ajouter $x$, dans le deuxième cas, ($h$ n'est plus injective !), on doit résoudre une **collision primaire**.
Le temps de recherche en absence des collision est de $O(1)$.

# Collisions dans la table de hachage

La condition de "$h$ fonction injective", est trop fort. En réalité, $h$ n'est pas injective sur $E$, si $M$ n'est pas trop grand par rapport à $|E|$, même si $h$ est uniforme.

Les collisions doivent être prise en compte
Deux classes de méthodes de hachage :

- Méthodes de résolution des collisions par chaînage **adressage fermé** : une alvéole contient non pas un élément mais un pointeur vers une liste chainée (On empile les bocaux)
- Méthodes de résolution des collisions par calcul **adressage ouvert** (On décale le bocal)

# Adressage ouvert (universel)

Dans une table de hachage à **adressage ouvert** tous les éléments sont gardés dans la table.
On ne dispose pas d'une seule fonction de hachage, mais d'une famille de fonctions de hachage :

$$h : U \times 1, \ldots M \rightarrow 1, \ldots M$$

Avec la propriété : $(h(x,1)h(x,2), \ldots h(x,M))$ est une permutation de $1, \ldots M$.

On calcule successivement $h(x,1)$, puis $h(x,2)$, ... jusqu'à rencontrer une alvéole vide.

# Sondage linéaire

Pour une fonction de hachage ordinaire $h'$ on construit une fonction de hachage auxiliaire par la méthode du sondage linéaire :

$$h(x, i) = (h'(x) + i) \, mod \, M$$

Cette est un adressage universel, toutefois on a tendance à occuper des zones contigües (grappes ou clusters) dans la table de hachage en cas de collision.

# Sondage quadratique

$$h(x, i) = (h'(x) + c_1\, i + c_2\, i^2)\ mod\ M$$

Avec $c_1$ et $c_2$ des constantes auxiliaires.
Ce n'est pas forcément un bon choix.

# Hachage ouvert basé sur deux fonctions

Les meilleures performances sont obtenues pour un adressage ouvert en considérant deux fonctions de hachage auxiliaire $h_1$ et $h_2$ :

$$h(x, i) = h_1(x) + i\, h_2(x)) \; mod \; M$$

# Double hachage

Dans un **double hachage** lors d'une collision sur une alvéole $i$, au lieu de considérer une liste chainée, on considère une nouvelle table de hachage de taille $m$ ($m \leq M$) et une seconde fonction de hachage $h_2$.
Ou pour une alvéole $j$ on considère une fonction de hachage secondaire $h'_j$ et une taille $m_j$ adaptée aux nombre de collisions possibles.

Un double hachage permet de se rapprocher d'un hachage idéal, à savoir une fonction de hachage injective.

# Complexité moyenne des opérations dans une table de hachage

| Structure | Recherche et Insertion | Suppression | Vider |
|---|---|---|---|
| Idéal (sans collisions) | $O(1)$ | $O(1)$ | $O(M)$ |
| Avec adressage fermé | $O(1 + N/M)$ | $O(1 + N/M)$ | $O(M + N)$ |
| Avec adressage ouvert | $O(\frac{M}{M - N})$ | ? | $O(M)$ |

La suppression est difficile car il faut faire la différence entre une alvéole devenue vide et une disponible.

$N = |E|$ est nombre d'éléments gardés dans la table de hachage.

Dans le pire des cas, $O(N)$ pour la table à adressage fermé et $O(M)$ pour la table à adressage direct.

# Bibliographie

Pour lire d'avantage :


- Donald E. Knuth. *The art of computer programming*, volume 3. Sorting and Searching, *Chapter 6. Hashing* Addison-Wesley, 1974.
- Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction à l'algorithmique*. DUNOD, Paris, 2003. (ou éditions plus récente)