Algorithmique et structures de données

CM 10 – Tables de hachage et fonctions de hachage

Plan du CM 10

Plan du CM 10

Table de hachage (en anglais hash table)

Ensemble de paires (clef, valeur)

- on accède à une valeur à partir d'une clef
- à chaque clef correspond une seule valeur

Accès rapide

- on veut accéder à une valeur en $\Theta(1)$
- vrai en moyenne
- dans le pire des cas, on peut y accéder en $O(n)$

Implémentation facile

Mémoire limitée

Exemple

Paires (clef,valeur)

- la clef est le nom d'une personne
- la valeur est un numéro de téléphone

Problème

- stockage dans une structure de donnée
  - il faut stocker les numéros de téléphone
- recherche
  - il faut retrouver un numéro de téléphone à partir d'une clef

Table de hachage – construction

Eléments présents dans la table

- on place les paires (clef,valeur) dans une table (tableau)

Recherche d'un élément dans la table

- il faut trouver *i* à partir du nom de la personne

Exemple de table

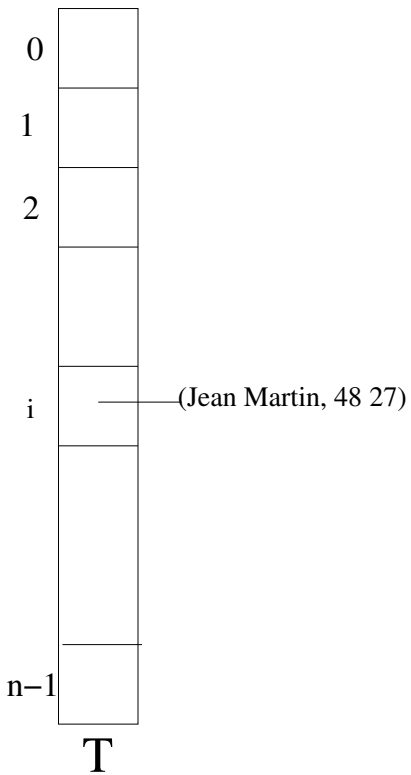
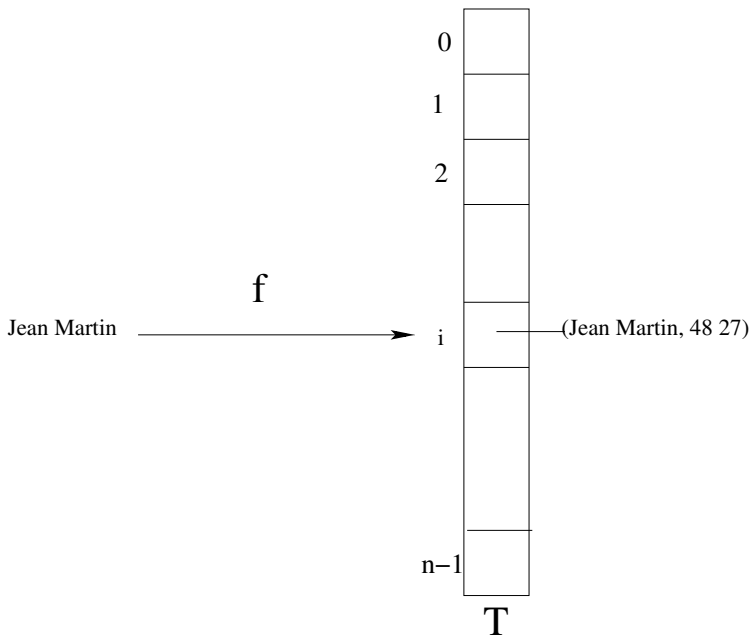


Table de hachage – construction

Fonction de hachage

- on utilise une fonction de hachage f
- à chaque personne, elle associe un indice i, valeur de hachage appelé alvéole
- le tableau est un tableau associatif, les éléments ne sont pas ordonnées
- f doit être facile à calculer (en temps et en mémoire)

Exemple



Plan du CM 10

Fonction de hachage

L'objectif est de transformer une chaîne de caractères en entier.

Exemple simple de fonction de hachage

On transforme chaque caractère avec le code ascii

LSB \ MSB		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	}
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	{
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

Fonctions de hachage

Il existe de nombreuses fonctions de hachage

MD5 – Message Digest 5

- fonction de hachage cryptographique
- crée une empreinte numérique d'un fichier (intégrité d'un document)
- inventé par Ronald Rivest en 1991
- aujourd'hui dépassé pour la cryptographie

SHA-0 et SHA-1

- fonction de hachage cryptographique
- inventé par la NSA (National Security Agency)
- SHA-1 (1993) et SHA-1 (1995)
- récemment rejeté en cryptographie, successeur SHA-2
- très bon générateur pseudo-aléatoire

Espace de hachage

Définition

- rappel, indice = alvéole
- n est le nombre d'alvéoleson, on choisit souvent $n = 2^k$
- on veut $f : \{0,1\}^* \to \{0,1\}^k$
- $f(x) = hash(x) \mod n$, pour une fonction de hachage *hash*

Compromis nombre de clefs/espace de hachage

- nous devons estimer N le nombre de paires (clef,valeur) que nous allons créer
- nous choisissons ensuite n convenablement en fonction de cette estimation

Dict – Dictionnaire en python

Création d'un dictionnaire

```
#création d'un dictionnaire
dico = {}

#création d'une paire (clef,valeur)
dico["Jean Martin"] = "48 27"

#accès à une valeur
print(dico["Jean Martin"])
```

Accès rapide à une clef et à une valeur

```
#accès rapide à une clef
print(clef in dico)                          O(1)
#accès rapide à une valeur
print(dico[clef])                            O(1)

#accès lent à une clef
print(clef in dico.keys())                   O(n)
print(clef in list(dico.keys())              O(n)
print(valeur in dico.values())               O(n)
print(valeur in list(dico.values())          O(n)
```

set – Ensemble en Python

Gestion d'un ensemble – éléments non ordonnés

Python utilise une table de hachage pour gérer les éléments d'un ensemble.

Complexité des opérations sur set

Opération	Exemple	Classe de complexité	Remarque
longueur	len(s)	O(1)	
ajout	s.add(5)	O(1)	
appartenance	x in/not in s	O(1)	O(n) pour list et tuple
suppression	s.remove(5)	O(1)	O(n) pour list et tuple
suppression conditionnelle	s.discard(5)	O(1)	
suppression aléatoire	s.pop	O(1)	O(n) pour list
réinitialisation	s.clear	O(1)	identique à s = set()
construction	set(L)	len(L)	
égalité	s != t	O(min(len(s),len(t))	
sous-ensemble	s <= t	O(len(s)	
réunion	s | t	O(len(s)+len(t))	
intersection	s & t	O(min(len(s),len(t))	
différence	s - t	O(len(t))	
différence symétrique	s ^ t	O(len(s))	
itération	for el in s	O(n)	
copie	t = s.copy	O(n)	

Dict – Dictionnaire en python

Méthode *hash*

La méthode *hash* permet de retourner le haché d'un objet.

```
>>>hash("Jean Martin")
>>>5359551642767892649
>>>T = (1,2,3)
>>>hash(T)
>>>2528502973977326415
>>> L = [1,2,3]                                    list est mutable
>>> hash(L)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Dict est dynamique

- comme la classe *list*, la classe *dict* est gérée dynamiquement
- l'espace de hachage peut être augmenté
- la réallocation (comme la désallocation) s'effectue au fur et a mesure que l'on ajoute (ou supprime) des paires (clef,valeur).

Hashtable de Java

Exemple

```
import java.util.Hashtable;
import java.util.Map;

public class HashtableExample{
    public static void main(final String argv[]) {
        Map<String, String> ht = new Hashtable<String,String>();
        ht.put("Ba", "Bah"); ht.put("Aa", "aha");
        ht.put("BB", "bebe");
        for( String k : ht.keySet() ) {
            System.out.println("Cle : "+ k + " Hash : "+ k.hashCode()
                    + " Valeur : " + ht.get(k));
        }
    }
}
```

Exécution du programme

Cle : Ba Hash : 2143 Valeur : Bah
Cle : BB Hash : 2112 Valeur : bebe
Cle : Aa Hash : 2112 Valeur : aha

Plan du CM 10

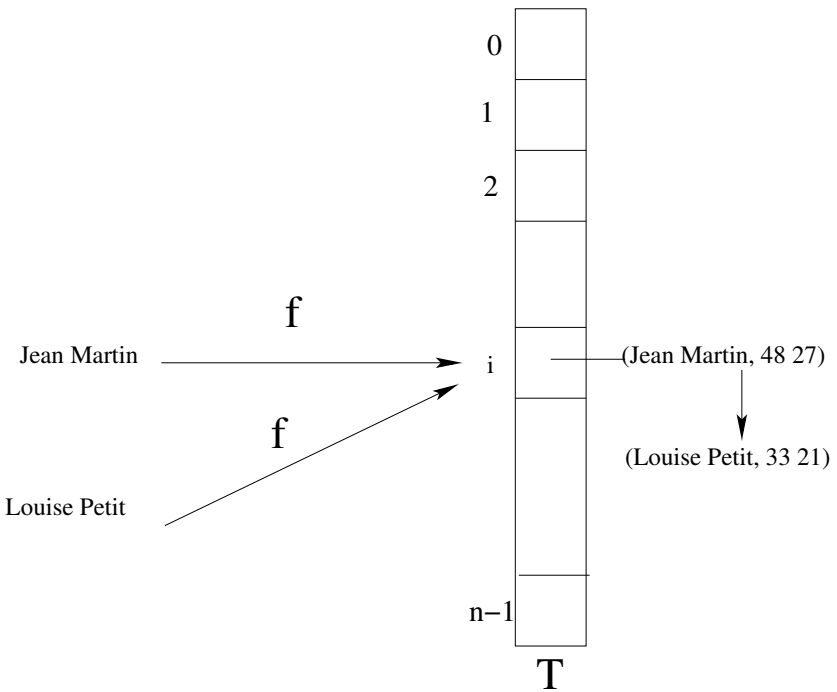
Principe des tiroirs (Pigeon Hole Principle)

Principe des tiroirs

- nous avons n tiroirs
- $N > n$ chaussettes rangés dans ces tiroirs
- nous avons au moins deux chaussettes dans le même tiroir

Solution pour la table de hachage

On associe une liste chaînée aux clefs donnant la même valeur.



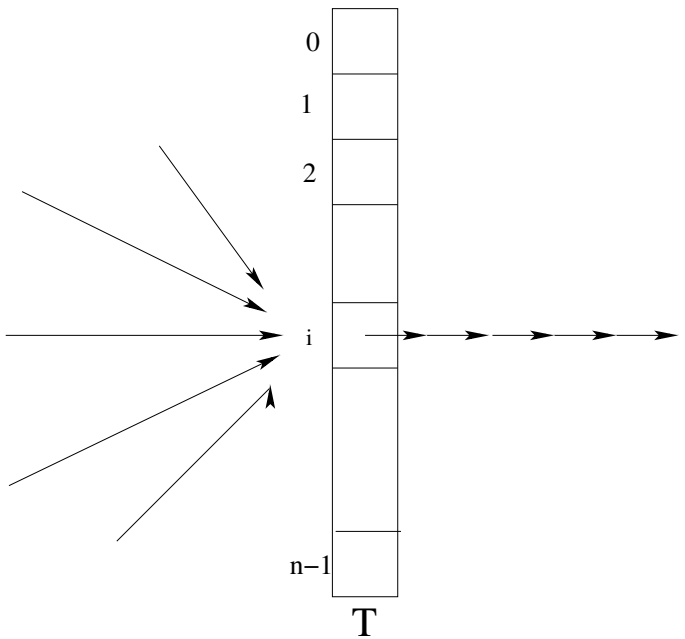
Principe des tiroirs (Pigeon Hole Principle)

Nombre de collisions

- n alvéoles et $N = n + p$ clefs
- au moins p collisions

Conséquence

- si nous avons trop de collisions, nous devons créer des listes chaînées plus grandes
- nous ne pouvons pas garantir l'accès à une valeur en $O(1)$ dans le pire des cas



Paradoxe des anniversaires

Problème

- dans une classe de 23 élèves
- quelle est la probabilité d'avoir deux élèves nés le même jour ?

Réponse (étonnante)

Nous avons une chance sur deux que deux élèves soient nés le même jour !

Conséquence

- les collisions existent même lorsque N est beaucoup plus petit que n.
- la gestion des collisions est donc toujours nécessaire

Paradoxe des anniversaires

Problème étendu

- à chaque individu correspond une valeur
- le nombre de valeurs possibles est m
- on considère k individus

Calcul de la probabilité

la probabilité que k individus successifs soient tous distincts vaut

$$1 \times \frac{m-1}{m} \times \frac{m-2}{m} \times \ldots \times \frac{m-k+1}{m}.$$

Donc la probabilité que deux individus aient la même valeur vaut

$$P(m,k) = 1 - 1 \times \frac{m-1}{m} \times \frac{m-2}{m} \times \ldots \times \frac{m-k+1}{m}.$$

Paradoxe des anniversaires

Choix de k

- n calcule alors k_0 la plus petite valeur k telle que $P(m, k) \geq 1/2$.
- on peut considérer d'autres probabilités p que $1/2$.

Application numérique

- pour $m = 365$ et $p = 1/2$, on obtient $k_0 = 23$.

Applications en informatique

- ce problème intervient très souvent en cryptographie et en sécurité informatique.
- dans l'analyse de la complexité d'une attaque

Paradoxe des anniversaires

Probabilité de collision

- n nombre de personnes
- $p(n)$ probabilité de collision

n	$p(n)$
10	$11,69\%$
15	$25,29\%$
20	$41,14\%$
23	$50,73\%$
25	$56,87\%$
30	$70,63\%$
40	$89,12\%$
50	$97,04\%$
60	$99,41\%$
80	$99,99\%$
100	$99,99997\%$
300	$1-(7.10^{-73})$

Plan du CM 10

Utilisation des fonctions de hachage

Les fonctions de hachage ne sont pas utilisées uniquement pour stocker des valeurs dans une table de hachage

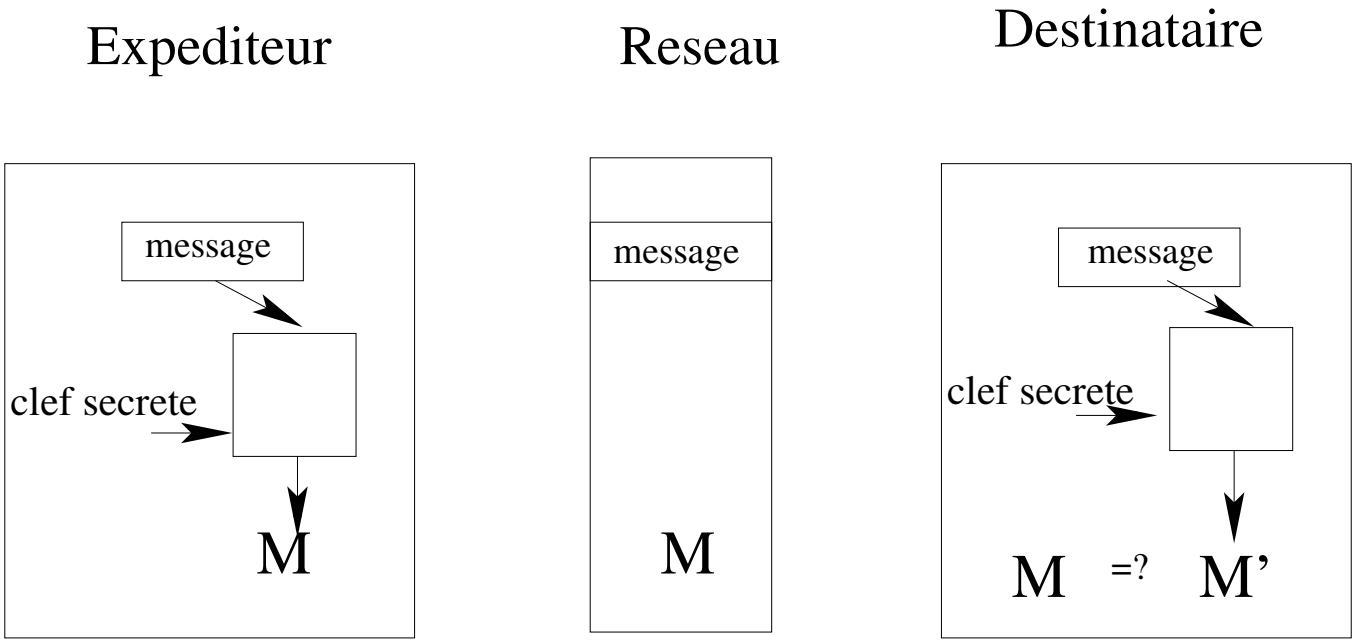
Il existe beaucoup d'autres utilisations

- Intégrité d'un fichier
- stockage des mots de passe
- code d'authentification de messages (MAC)
- signature, chiffrement en cryptographie

code d'authentification de messages (MAC)

Principe

- proche d'une fonction de hachage,
- mais associé à une clef secrète
- assure l'intégrité d'un message



Critères en cryptographie

Problème 1 Résistance au calcul de préimage

- on possède un haché h
- on cherche une valeur x tel que $f(x) = h$.

Problème 2 Résistance au calcul de seconde préimage

- on possède x et $h = f(x)$
- on cherche y différent de x tel que $f(y) = h$

Critères en cryptographie

Propriétés mathématiques attendues

Complétude

Chaque bit de sortie dépend simultanément de chaque bit en entrée

Avalanche

L'effet d'avalanche strict assure que l'inversion d'un seul bit en entrée inverse statistiquement chaque bit de sortie avec une probabilité $1/2$.

```
>>> hash("Jean Dupond")
8592085748933861834
>>> hash("Jean Dupont")
3821719789437734537
```

Stockage de mots de passe

Le serveur stocke les login et les mots de passe dans une table de hachage.

Paires (login, haché du mot de passe)

```
l1      h1 = f(m1)
l2      h2 = f(m2)
l3      h3 = f(m3)
```

Authentification

- l'utilisateur tappe son login l et son mot de passe m
- le serveur récupère la paire $(l, h = f(m))$
- il vérifie que la paire est dans sa table de hachage

Stockage de mots de passe

Fuite de bases de données de mots de passe

De nombreuses bases de mots de passe ont été piratées.
Voici trois exemples de bases qui ont fuitées :

Nom de la base	type de base	taille de la base	année du piratage
RockYou	jeux vidéos et réseau social	> 32 millions	2009
LinkedIN	réseau social professionnels	> 160 millions	2012
Myspace	création blog et musique	> 320	2008

Croisements

- les hachés ne sont pas rattachés à une personne
- beaucoup de gens utilisent un même mot de passe pour plusieurs services
- les mots de passe les plus faciles à trouver sont très souvent utilisés loi de Zipf

Stockage de mots de passe

Paires (login, haché du mot de passe+sel)

- pour ne pas retrouver les hachés d'une base à une autre, on sale les mots de passe.

```
l1 sel      h1 = f(m1||sel)
l2 sel      h2 = f(m2)||sel
l3 sel      h3 = f(m3||sel)
```

- mieux encore : ne pas mettre le sel sur la table.
- poivre comme le sel met différent pour chaque mot de passe

Pas de corrélation entre les mots de passe

- soit h_1 le haché de 12345 dans la base B_1
- soit h_2 le haché de 12345 de la base B_2
- il n'est pas possible de relier h_1 et h_2
- on ne peut pas voir que le mot de passe appartient aux deux bases

Fonctions de hachage lentes ou rapides

Pour certaines applications, il est préférable d'utiliser une fonction de hachage lente pour freiner un attaquant.

Applications où les fonctions lentes sont utiles

- stockage des mots de passe
- minage dans la blockchain (voir l'exemple du Bitcoin)

Bilan sur les fonctions et les tables de hachage

Beaucoup d'utilisation possibles

- stockage de données
- indexation dans les bases de données
- cryptographie
- sécurité informatique

Alternatives possibles pour les deux premières

- ABR
- arbres-B (arbres équilibrés)
- arbres rouge-noir (ABR équilibrés)