

# Table de Hachage

Vincent Aranega  
vincent.aranega@genymodel.com

22 février 2018

1 / 28

## Outline

- 1 Pourquoi une nouvelle structure ?
- 2 Table à adressage direct
- 3 Table de hashage
- 4 Collisions

2 / 28

## Temps d'accès aux éléments

Pour trouver la position d'un élément  $e$  dans une structure de donnée de  $n$  éléments :

- 1 Liste : comparaison de la valeur des éléments de la liste avec  $e$ 
  - au pire : comparaison jusqu'au dernier élément
  - recherche en  $O(n)$
- 2 Arbre : comparaison de la valeur des éléments de l'arbre avec  $e$ 
  - au pire : comparaison jusqu'à une feuille
  - recherche en  $O(\log(n))$

→ dépend du nombre d'éléments dans le TDA  
→ si  $n \nearrow$  alors temps de la recherche  $\nearrow$

3 / 28

## Dans l'idéal

Pour trouver la position d'un élément  $e$  dans un ensemble de  $n$  éléments :

- accès direct à  $e$
- un seul accès pour accéder à  $e$
- recherche en  $O(1)$

→ ne dépend pas de nombre d'éléments  $n$   
→  $\forall n$  temps de la recherche rapide  
→ même si  $n \nearrow$  alors temps de la recherche = 1

4 / 28

## Outline

- 1 Pourquoi une nouvelle structure ?
- 2 Table à adressage direct
- 3 Table de hashage
- 4 Collisions

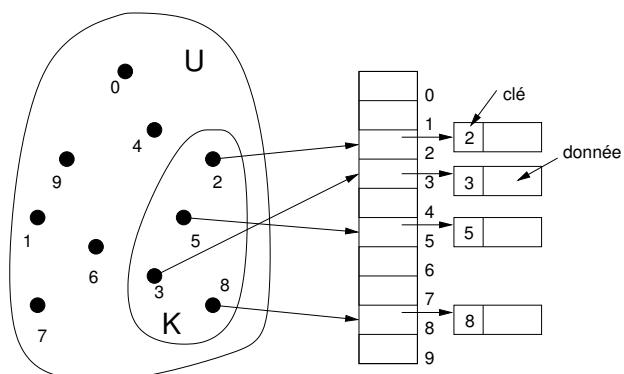
5 / 28

## Principe

- $U = \{0, 1, \dots, m-1\}$  est l'univers de tout les éléments/clés
  - $K$  est un sous ensemble dynamique de  $U$  représentant les clés effectivement recherchées/manipulées
- Technique simple si l'univers des clés  $U$  est petit
- Représentation : table à adressage directe  $T[0 \dots m-1]$
  - Chaque indice de  $T$  correspond à une clé dans  $U$
  - 2 éléments ne peuvent avoir la même clé

6 / 28

## Exemple



7 / 28

## Problèmes

- Si  $U$  est grand,  $T$  ne tient pas en mémoire
  - Si  $|K| \ll |U|$  alors gaspillage de la mémoire
- En pratique, quasi impossible à utiliser, solution à conserver lorsque  $U$  est petit et  $|K|$  est sensiblement égal à  $|U|$

8 / 28

- 1 Pourquoi une nouvelle structure ?
- 2 Table à adressage direct
- 3 Table de hachage
- 4 Collisions

- la place d'un élément dans la table est calculée à partir de sa propre valeur
- calcul réalisé par une fonction de hachage : transforme la valeur de l'élément en une adresse dans un tableau
- recherche d'un élément : nombre constant de comparaisons  $O(1)$ . Ne dépend pas du nombre d'éléments dans le tableau

- transforme la valeur d'un élément en position
- doit être facilement calculable (temps d'exécution de la fonction rapide sinon on perd le bénéfice de l'accès en  $O(1)$ )
- Pour une table  $T$  et un élément  $e$   
 $\exists$  une fonction de hachage  $h$  telle que  $T[h(e)] = e$  (si  $e \in T$ )

Attention

$h$  est une fonction déterministe sinon on ne pourrait pas retrouver nos données

- Ensemble  $K$ 
  - Ensemble des éléments à stocker
  - { serge, odile, luc, anne, annie, julie, basile, paula, marcel, elise }
- Table  $T$  avec  $n$ 
  - taille de la table
  - 13

Rôle de la fonction de hachage  $h$

→ associer à chaque élément  $e$  une position  $h(e) \in [0..12]$

Exemple d'algorithme de fonction  $h$  :

- 1 Attribuer aux lettres  $a, b, \dots, z$  les valeurs  $1, 2, \dots, 26$
- 2  $res = \sum$  valeurs des lettres de  $e$
- 3  $res = res + \text{nombre de lettres de } e$
- 4  $res = res \bmod(n)$  (ici  $n = 13$ )

La position de l'élément *serge* est donnée par  $h(\text{serge})$

- $h(\text{serge}) = (54 + 5) \bmod 13 = 7$
- *serge* est à la position 7 dans la table de hachage

De même :

- $h(\text{odile}) = (45 + 5) \bmod 13 = 11$
- $h(\text{luc}) = (36 + 3) \bmod 13 = 0$
- $h(\text{anne}) = (34 + 4) \bmod 13 = 12$
- $h(\text{annie}) = (43 + 5) \bmod 13 = 9$
- $h(\text{jean}) = 8, h(\text{julie}) = 10, h(\text{basile}) = 2,$   
 $h(\text{paule}) = 4, h(\text{elise}) = 3, h(\text{marcel}) = 6$

0	Luc
1	
2	Basile
3	Elise
4	Paula
5	
6	Marcel
7	Serge
8	Jean
9	Annie
10	Julie
11	Odile
12	Anne

serge ?

0	luc
1	
2	basile
3	elise
4	paula
5	
6	marcel
7	serge
8	jean
9	annie
10	julie
11	odile
12	anne

## Exemple

serge?  $\rightarrow h(\text{serge}) = 7$

0	luc
1	
2	basile
3	elise
4	paula
5	
6	marcel
7	serge
8	jean
9	annie
10	julie
11	odile
12	anne

17 / 28

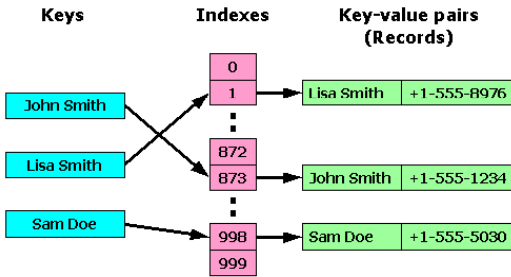
## Opération sur les tables de hachage

- `put(T, e)`
  - insère une valeur  $e$  dans la table  $T$
  - $T[h(e)] = e$ ;
- `get(T, e)`
  - retourne la valeur  $e$  si elle est présente dans  $T$ ,  $NULL$  sinon (en considérant que chaque case du tableau à été init à  $NULL$ )
  - $\text{return } T[h(e)]$ ;
- `remove(T, e)`
  - supprime l'entrée  $e$  de la table  $T$
  - $T[h(e)] = NULL$ ;

18 / 28

## Stockage d'informations complémentaires

- La clé sert à rechercher l'indice dans le tableau  $T$
- Actuellement clé = valeur stockée
- Possible d'associer d'autres informations à la clé



19 / 28

## Opération sur les tables de hachage

- On considère une structure  $\{key, value\}$  comme élément de la table
- `put(T, key, val)`
  - insère un couple  $key, val$  dans la table  $T$
  - $T[h(key)] = \text{couple}(key, val)$ ;
- `get(T, key)`
  - retourne la valeur  $val$  associée à  $key$  si elle est présente dans  $T$ ,  $NULL$  sinon
  - $\text{return } T[h(key)] \neq NULL ? T[h(key)].value : NULL$ ;
  - ( $T[h(key)]$  retourne un couple)
- `remove(T, key)`
  - supprime l'entrée  $key$  de la table  $T$
  - $T[h(key)] = NULL$ ;

20 / 28

## Outline

- 1 Pourquoi une nouvelle structure ?
- 2 Table à adressage direct
- 3 Table de hashage
- 4 Collisions

21 / 28

## Collision

Comme  $|U| \gg n$  ( $n$  taille de  $T$ )  $\rightarrow$  collision :

- $\exists k, k' \in U \mid h(k) = h(k') \wedge k \neq k'$
- Important de trouver la bonne fonction  $h$
- $h$  est dépendant des valeurs à stocker

22 / 28

## Résolution

Diverses solutions :

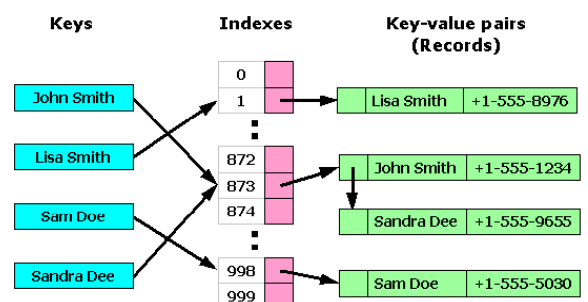
- Chaînage
- Adressage ouvert

### Solution par chaînage

- Chaque valeur hachée est placée dans une liste
- La case  $T[i]$  contient le pointeur vers la liste des éléments de clés  $k$
- Si  $T[i]$  ne désigne rien, alors il pointe sur  $NULL$

23 / 28

## Exemple



24 / 28

## Opération sur les tables de hachage

- On considère une structure  $\{key, value\}$  comme élément de la liste contenue par chaque case de  $T$
- `put(T, key, val)`
  - on ajoute le couple  $(key, val)$  en tête de la liste chaînée à la position  $h(k)$  du tableau
  - `ajout_tete(T[h(key)], couple(key, val));`
- `get(T, key)`
  - on cherche la  $key$  dans la liste située à  $h(key)$  (recherche dans liste de couple)
  - `couple = list_find(T[h(key)], key);`
  - `return couple != NULL ? couple.value : NULL;`
- `remove(T, key)`
  - on supprime le couple  $(key, val)$  situé de la liste contenue en  $h(key)$
  - `supp(T[h(key)], key);`

25 / 28

## Conclusion

- Ajout d'informations et recherche efficace
- Association clé, valeur
- Valeurs ordonnable ou non (pas obligatoire)
- Dépendant d'une fonction de hachage
- Difficulté → déterminer fonction de hachage

26 / 28

## Comparaison

	recherche	Moyenne	
		insertion	suppression
liste	$O(n)$	$O(1)$	$O(n)$
arbres binaires	$O(\log n)$	$O(\log n)$	$O(\log n)$
tables de hachage	$O(1)$	$O(1)$	$O(1)$

	recherche	Pire cas	
		insertion	suppression
liste	$O(n)$	$O(1)$	$O(n)$
arbres binaires	$O(n)$	$O(n)$	$O(n)$
tables de hachage	$O(n)$	$O(n)$	$O(n)$

27 / 28

## Références

- Algorithmes et Structure de données – Jean-Charles Régis
- Les tables de hachage – Christophe Gonzales, Pierre-Henri Wuillemin
- Tables de hachage – ENSIEE
- Table de hachage – B. Jacob
- Les Tables de Hachage – Jean-March Nicod

28 / 28