Table de Hachage

Vincent Aranega
vincent.aranega@genmymodel.com

22 février 2018

Outline

Temps d'accés 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 $\nearrow$

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 $= 1$

Outline

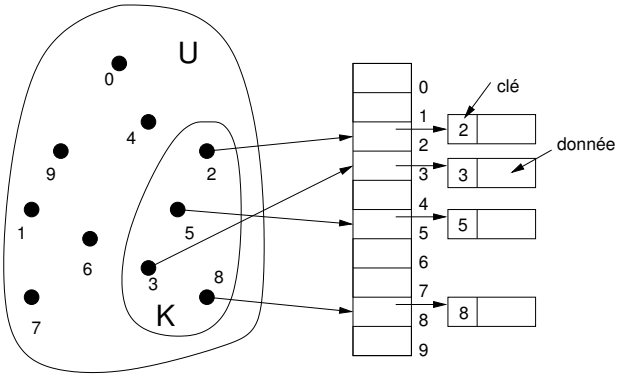
Principe

- $U = \{0, 1, \ldots, m-1\}$ est l'univers de tout les éléments/clés
- K est un sous ensemble dynamique de U représentant les clés effectivement cherchées/manipulées

→ Technique simple si l'univers des clés U est petit

- Représentation : table à adressage directe $T[0 \ldots m-1]$
- Chaque indice de T correspond à une clé dans U
- 2 éléments ne peuvent avoir la même clé

Exemple



Problèmes

- Si U est grand, T ne tient pas en mémoire
- Si $|K| << |U|$ alors gaspillage de la mémoire

→ En pratique, quasi impossible à utiliser, solution à conserver lorsque U est petit et $|K|$ est sensiblement égal à $|U|$

Outline

Principe

- 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

Fonction de hachage

- 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

Exemple

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

Exemple

Exemple d'algorithme de fonction h :

1. Attribuer aux lettres $a, b, \ldots, z$ les valeurs $1, 2, \ldots, 26$
2. $res = \sum$ valeurs des lettres de e
3. $res = res +$ nombre de lettres de e
4. $res = res \ mod(n)$ (ici $n = 13$)

Exemple

La position de l'élément *serge* est donnée par $h(serge)$

- $h(serge) = (54 + 5) mod 13 = 7$
- *serge* est à la position 7 dans la table de hachage

De même :

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

Exemple

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 ?

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 ? $\rightarrow h(serge) = 7$

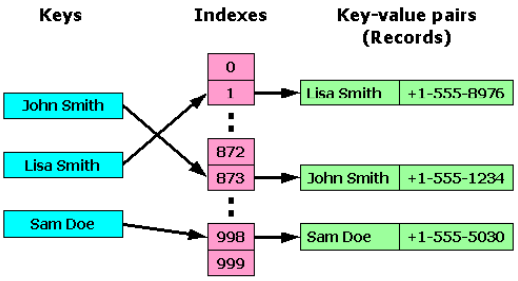
0	luc
1	
2	basile
3	elise
4	paula
5	
6	marcel
7	serge
8	jean
9	annie
10	julie
11	odile
12	anne

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*
  - `return T[h(e)];`
- `remove(T, e)`
  - supprime l'entrée e de la table T
  - `T[h(e)] = NULL;`

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



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)] = couple(key, val);`
- `get(T, key)`
  - retourne la valeur *val* associée à *key* si elle est présente dans T, *NULL* sinon
  - `return T[h(key)] != 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;`

Outline

Collision

Comme $|U| >> n$ (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

Résolution

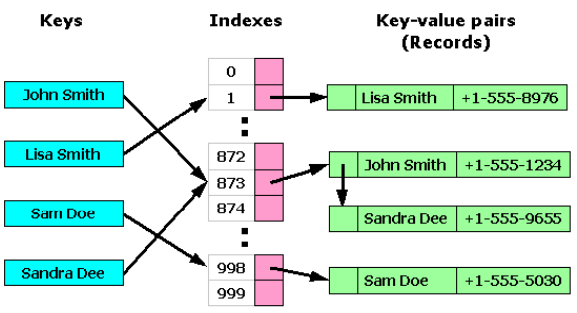
Diverses solutions :

- 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*

Exemple



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);`

Comparaison

Moyenne

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

Pire cas

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

Conclusion

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

Références

- Algorithme et Structure de données – Jean-Charles Régin
- 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
    ```