1.1

Algorithmique et Structures de Données

Jean-Charles Régin

Licence Informatique 2ème année

1.2

Table de hachage

Jean-Charles Régin

Licence Informatique 2ème année

Table de hachage

- Une **table de hachage** (*hash table* en anglais) est
  - structure de données qui permet une association clé-élément,
  - implémentation du *type abstrait* table de symboles.
- On accède à chaque élément de la table via sa **clé**.
- L'accès à un élément se fait en transformant la clé en une valeur de hachage (ou simplement hachage) par l'intermédiaire d'une **fonction de hachage**.
- Le hachage est un nombre qui permet la localisation des éléments dans le tableau, typiquement le hachage est l'index de l'élément dans le tableau.
- Une case dans le tableau est appelée **alvéole**.

Table de hachage

- **insérer(H,elt,clé)** : insère dans la table de hachage H un élément elt dont la clef est clé
- **elt recherche(H,clé)** : recherche dans la table de hachage H si un élément est associé à la clef clé et renvoie cet élément
- **booléen appartient(H,clé)** : recherche dans la table de hachage H si un élément est associé à la clef clé

Table de hachage

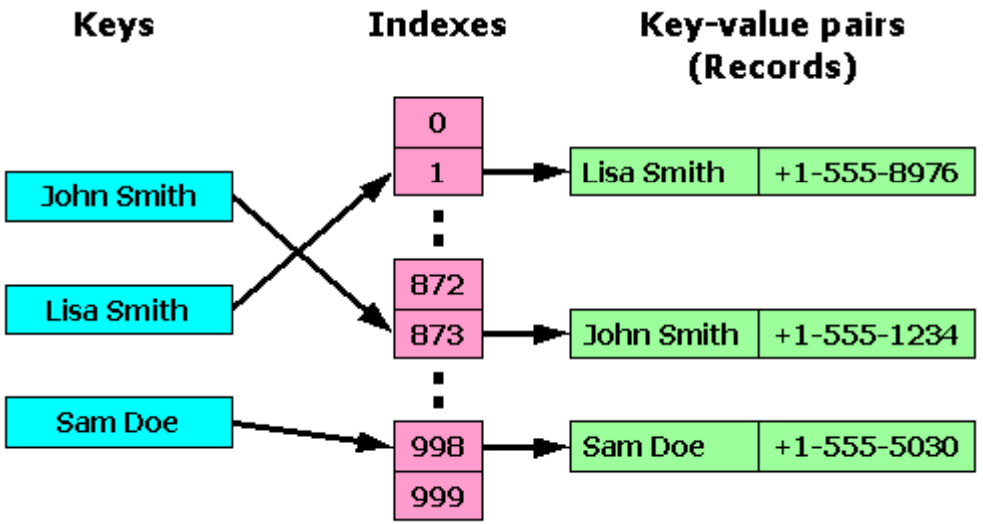
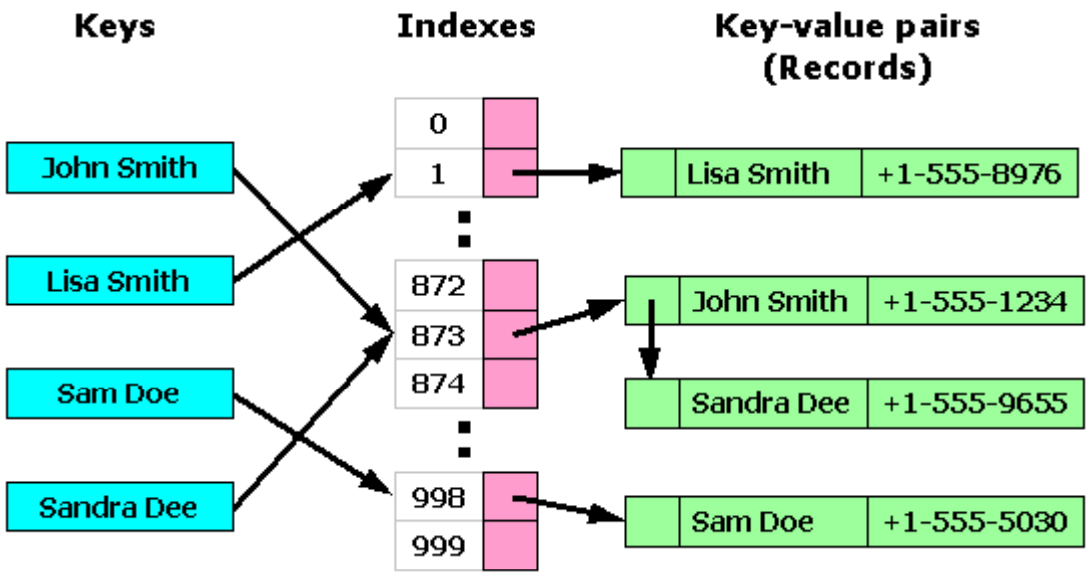


Table de hachage

- Les tables de hachage permettent
  - un accès en O(1) en moyenne, mais
  - le temps d'accès dans le pire des cas peut être de O(n).
- Une table de hachage n'est pas ordonnée
- Le fait de créer un hash à partir d'une clé peut engendrer un problème de **collision** :
  - à partir de deux clés différentes, la fonction de hachage peut renvoyer la même valeur de hash, et donc donner accès à la même position dans le "tableau".
- Pour minimiser les risques de collisions, il faut donc choisir soigneusement sa fonction de hachage.

Collision


Keys
Indexes
Key-value pairs (Records)
John Smith
Lisa Smith
Sam Doe
Sandra Dee
0
1
872
873
874
998
999
Lisa Smith
+1-555-8976
John Smith
+1-555-1234
Sandra Dee
+1-555-9655
Sam Doe
+1-555-5030

Résolution des collisions

- Lorsque deux clés ont la même valeur de hachage, les deux éléments associés ne peuvent être stockés à la même position, on doit alors employer une stratégie de résolution des collisions.
- Les collisions sont fréquentes (voir transparent suivant)
- De nombreuses stratégies de résolution des collisions existent mais les plus connues et utilisées sont
  - le chainage
  - l'adressage ouvert.

Collisions : probabilités

- Similaire au problème des anniversaires :
  - Dans un groupe de personnes, quelle est la probabilité que deux soient nées le même jour ?
- On raisonne à l'aide du complémentaire : probabilité que les personnes soient toutes nées des jours différents
  - pour la première personne : 365 choix / 365
  - pour la 2eme personne : 364 / 365
  - pour la ieme personne : (365 – i +1) /365
- Pour n : $[365*364*363*\ldots*(365-n+1)]/365^n$
- Donc 2 le même jour :
  - $1 - [365*364*363*\ldots*(365-n+1)]/365^n$

Collisions : probabilités

5	2,71%
10	11,69%
15	25,99%
20	41,14%
25	56,87%
30	70,63%
40	89,12%
50	97,04%
60	99,41%
80	99,99%
100	99,99997%

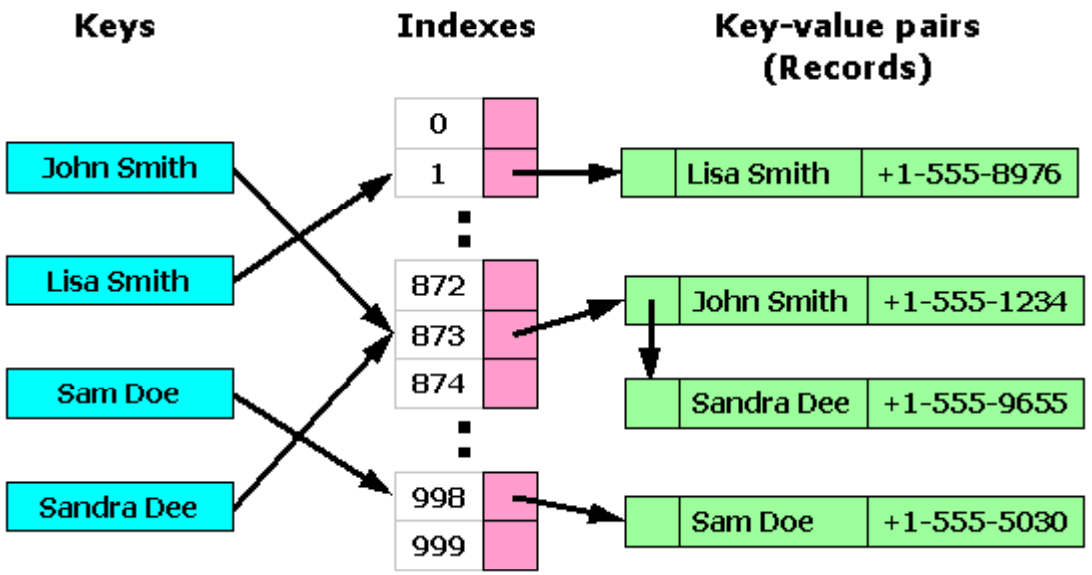
Collisions : probabilités

- Probabilités d'avoir une collision dans une table de taille 1 million avec 2500 éléments : 95%
- Remarques
  - Cela suppose une répartition uniforme des données : c'est rarement le cas en pratique

Résolution des collisions : chainage

- Méthode la plus simple.
- Chaque case de la table est en fait une liste chaînée d'éléments dont les clés ont le même hachage.
  - Une fois la case trouvée, la recherche est alors linéaire en la taille de la liste chaînée.
  - Pire des cas (la fonction de hachage renvoie toujours la même valeur) en O(n).
- Avantages
  - la suppression d'un élément est facile,
  - l'agrandissement de la table peut être retardé

Résolution des collisions



Résolution des collisions : adressage ouvert

- L'adressage ouvert consiste dans le cas d'une collision à stocker les éléments dans d'autres alvéoles de la table.
- La position de ces alvéoles est déterminée par une méthode de 'sondage' :
  - Lors d'une recherche, si la case obtenue par hachage direct ne permet pas d'obtenir le bon élément, une recherche sur les cases obtenues par une méthode de sondage est effectuée jusqu'à trouver l'élément, ou non, ce qui indique qu'aucun élément de ce type n'appartient à la table.

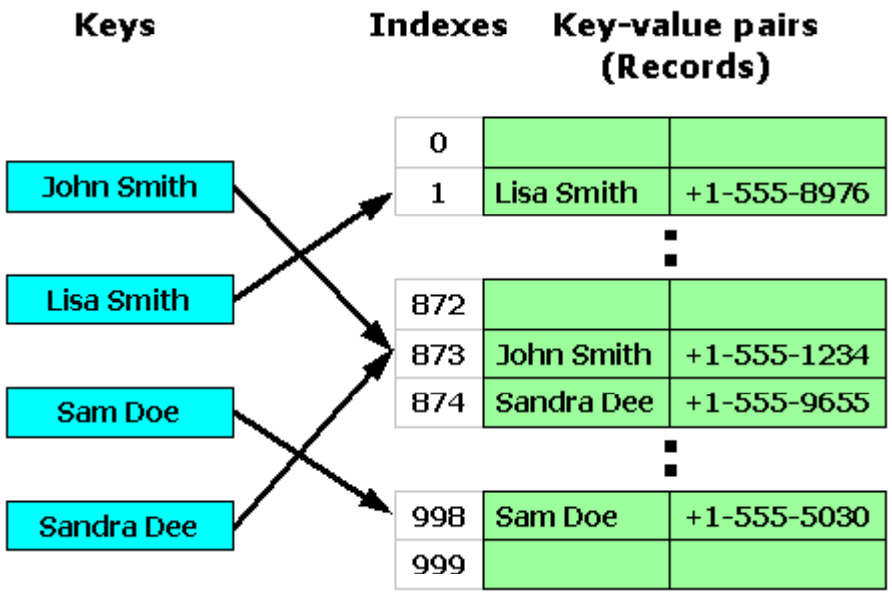
Résolution des collisions : adressage ouvert

- Les méthodes de sondage courantes sont:
  - **Sondage linéaire** (*linear probing*) : l'intervalle entre les cases est fixe, souvent 1.
    - next(clé,i) = (hash(clé) + i) mod m
    - Ordre de parcours des alvéoles
      - T[hash(clé)], T[hash(clé)+1], T[hash(clé)+2] …
  - **Sondage quadratique** (*quadratic probing*) : l'intervalle entre les cases augmente linéairement. Les indices des cases augmentent donc quadratiquement : +3 , +6, +12, +20 …
    - next(clé,i) = (hash(clé) + i + i^2) mod m
    - Ordre de parcours des alvéoles
      - T[hash(clé)], T[hash(clé)+2], T[hash(clé)+6] …

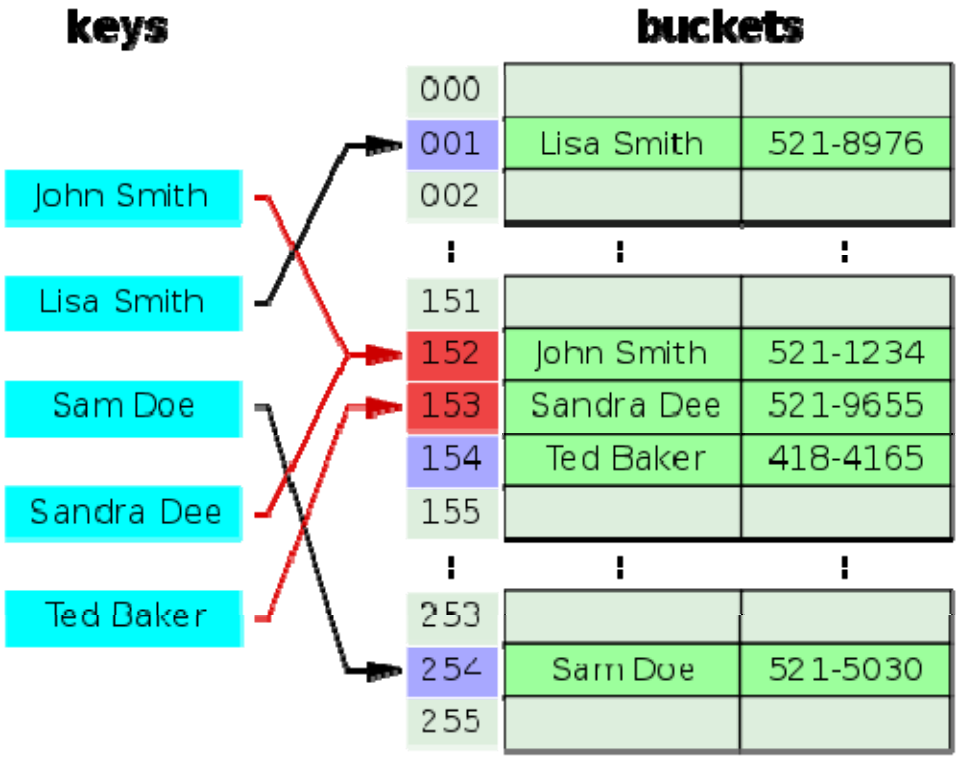
Résolution des collisions : adressage ouvert

- Les méthodes de sondage courantes sont:
  - **Double hachage** : l'adresse de la case est donnée par une deuxième fonction de hachage, ou hachage secondaire.
    - next(clé,i) = ($hash_1$(clé) + i $hash_2$(clé)) mod m

Adressage ouvert



Adressage ouvert



Résolution des collisions : adressage ouvert

- Avantages et inconvénients :
  - Sondage linéaire : possède la meilleure performance en termes de cache mais est sensible à l'effet de clustering (regroupement de données dans la même partie de la table).
  - Le double hachage ne permet pas d'utiliser le cache efficacement mais permet de réduire presque complètement les problèmes de clustering, en contrepartie d'une complexité plus élevée.
  - Le sondage quadratique se situe entre le linéaire et le double hashing au niveau des performances.
  - **la suppression d'un élément n'est pas gérable facilement**

Résolution des collisions

- Chainage avec d'autres structures de données que les listes chaînées :
  - arbre équilibré : le coût théorique de recherche dans le pire des cas est en O(log n). En général peu efficace car liste courte et structure lourde
  - tableau dynamique. Problème avec la suppression d'éléments

Fonction de hachage

- Une fonction de hachage permet de transformer une clé en une valeur de hachage (un index), donnant ainsi la position d'un élément dans le tableau.
- Si la clé n'est pas un entier naturel, il faut trouver un moyen de la considérer comme un entier naturel.
  - si la clé est de type *Chaine de caractères*, on peut ajouter la position dans l'alphabet de chaque lettre pour obtenir un entier naturel. Il suffit ensuite de transformer cet entier en index par différentes façons. Par exemple, en prenant le reste de la division de l'entier par le nombre d'éléments dans le tableau.

Fonction de hachage

- Fonction de hachage de Daniel J. Bernstein :

```
entier hash(str[]) { // str est une chaine de
caractères
    hash = 5381
    i  ← 0
    tant que(str[i] ≠'\0') {
         // hash=hash*33+str[i]
         hash ← ((hash << 5)+hash)+str[i]
         i ← i + 1
    }
    renvoyer hash
}
```

Fonction de hachage

- Fonction de hachage de Pearson
- Requière une “look-up” table T qui contient une permutation des nombres de 0 à 255
- Un message est découpée en n octets : le tableau M représente ces octets
- La table fait une taille m
- ```
  hash ← 0
  pour i de 1 à n {
       hash ← T[(hash + M[i]) % m]
  }
  renvoyer hash
  ```
- Avantages
  - Simple
  - Rapide
  - Peu de collisions espérées
  - Permet d’engendrer des tables de hachage parfaites
  ```

Fonction de hachage

- Robert Jenkins

```
x = (x+0x7ed55d16) + (x<<12);
x = (x^0xc761c23c) ^ (x>>19);
x = (x+0x165667b1) + (x<<5);
x = (x+0xd3a2646c) ^ (x<<9);
x = (x+0xfd7046c5) + (x<<3);
x = (x^0xb55a4f09) ^ (x>>16);
return x;
```

Fonction de hachage

- Bonne fonction de hachage
  - Engendre le moins de collisions
  - Possède une complexité faible
- Le calcul du hachage se fait parfois en 2 temps :
  - Une fonction de hachage particulière à l'application est utilisée pour produire un nombre entier à partir de la donnée d'origine.
  - Ce nombre entier est converti en une position possible de la table, en général en calculant le reste modulo la taille de la table.
- Taille de la table de hachage :
  - **en théorie** : souvent des nombres premiers pour éviter les problèmes de diviseurs communs, qui créeraient un nombre important de collisions.
  - **en pratique :** on utilise une puissance de deux, ce qui permet de réaliser l'opération modulo par de simples décalages, et donc de gagner en rapidité.

Hachage et Tri par base

- Quand on sélectionne un digit dans la tri par base, d'une certaine manière on fait une fonction de hachage simple : le hachage retourne un seul digit.
- Les éléments ayant le même digit sont en collision.

- On comprend que l'on n'est pas obligé de trier, on a juste besoin de différencier les éléments qui sont en collision

Clustering

- Regroupement des valeurs de hachage de façon côte à côte dans la table = **cluster**
- Le « **clustering** » est très pénalisant pour les techniques de résolution des collisions par adressage ouvert.
- Les fonctions de hachage réalisant une distribution uniforme des hachages sont donc les meilleures, mais sont en pratique difficile à trouver.

Facteur de charge

- Une indication critique des performances d'une table de hachage est le **facteur de charge** qui est la proportion de cases utilisées dans la table.
- Le facteur de charge est en général limité à 80%, même en disposant d'une bonne fonction de hachage.
- Des facteurs de charge faibles ne sont pas pour autant significatifs de bonne performances, en particulier si la fonction de hachage est mauvaise et génère du clustering.

Hachage parfait

- Une fonction de hachage est parfaite si elle ne génère pas de collisions
- Si on connait à l'avance les éléments alors on peut calculer une telle fonction
- On peut aussi chercher celle qui a besoin de la table la plus petite.

Hachage parfait

- Si on connait l'ensemble des éléments (donc des clés) à l'avance, alors la fonction qui retourne la position d'une clé dans l'ensemble trié est une fonction de hachage parfaite
  - Parfaite = pas de collisions
  - Complexité = celle du tri (on doit à chaque fois trié pour trouver la position d'un élément)
  - Minimale : n'occupe que n cases mémoires
- Recherche dichotomique = Fonction de hachage parfaite
- Inconvénient : lent O(log(n))

Hachage parfait

- En TD vous verrez une méthode