Adressage dispersé

Hachage

Introduction

- Recherche d’un élément dans un ensemble : jusqu’ici,
  - Meilleure solution = recherche dichotomique dans un ensemble trié
- Peut-on faire mieux ?
- L’adressage dispersé a pour but d’améliorer encore les performances
- C'est une méthode d'accès direct aux éléments de l’ensemble géré

Principe de l'adressage dispersé

- On associé à chaque élément de l'ensemble une clé, partie de la valeur de l'élément
- On range l'élément en fonction de sa clé
  - On calcule l'indice de l'emplacement de tableau dans lequel on doit ranger l'élément
- L'indice de l'emplacement dans le tableau est *fonction de la valeur l'élément (la clé)*

Fonction de dispersion

- Pour déterminer l'emplacement d'un élément dans la table, on définit une fonction appelée fonction de hachage ou fonction de dispersion, qui calcule un indice sur l'intervalle de définition de la table à partir de la valeur de la clé

  $$h : C^* \rightarrow [binf...bsup]$$

- Exemples :
  - Rang de la première lettre d'un mot ou d'une chaîne : **'a'** → **binf, 'b'** → **binf+1, etc.**
  - Somme des codes ASCII des lettres d'une chaîne modulo *M où M est la taille du tableau*
  - Pour un mot x de longueur lg écrit en ASCII,

    $$h(x) = \left(\sum_{i=0}^{lg-1} x[i] * B^{lg-i-1}\right) \bmod N$$

    avec B une puissance de 2 et N premier,

    par exemple B=256 et N=251

Une bonne fonction de dispersion

- Permet d'atteindre toutes les cases du tableau
- Réalise une bonne dispersion :
  - Distribution uniforme sur les clés
  - Exemples précédent :
    - La 2ème fonction est meilleure que la première
    - Beaucoup de chaînes commencent par la même lettre
    - De façon inverse, certaines lettres ne seront guère ou jamais utilisées
- Le but est d'avoir des opérations de recherche, d'insertion et de suppression en temps constant.

Méthode

- Pour insertion, recherche ou suppression
  - On calcule la valeur h(x) associée à la clé x de l'élément à traiter
  - On va a l'indice correspondant à h(x)
  - On réalise l'opération
- Si chaque h(x) se réfère à un élément unique, on a effectivement un traitement en temps constant

Collision

- En général, il existe des cas de clés différentes pour lesquelles la fonction de hachage donne le même résultat:

  $x1 \neq x2$ et $h(x1) = h(x2)$

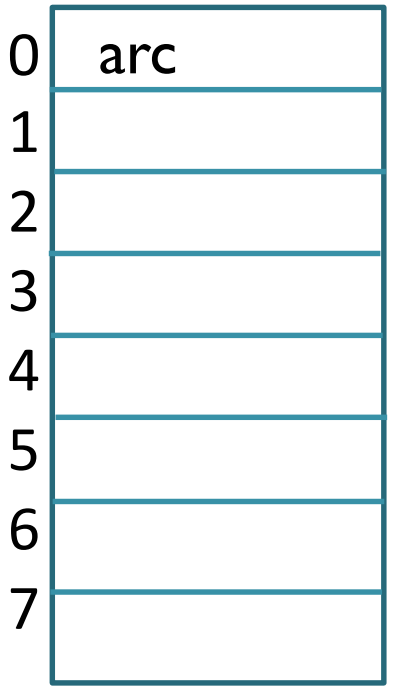
- On appelle ce cas de figure une *collision primaire*

Exemple de collision

- On a des mots à insérer dans un tableau de taille *M = 8*
- La fonction de hachage *h est donnée* par le rang de la première lettre du mot
  - h(‘a’) = 0, h(‘b’) = 1, etc. (modulo la taille du tableau)
  - Les mots sont **arc, chat, lisse, roux** et **case**

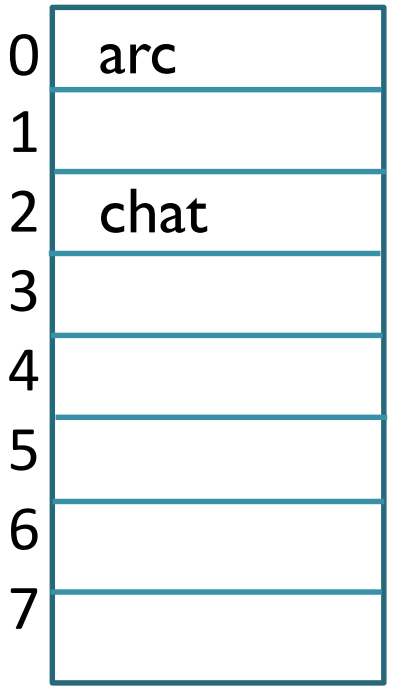
Exemple de collision

- Les mots sont **arc, chat, lisse, roux** et **case**
- On insère arc
- h("arc") = 0



Exemple de collision

- Les mots sont **arc, chat, lisse, roux** et **case**
- On insère chat
- h("chat") = 2



Exemple de collision

- Les mots sont **arc, chat, lisse, roux** et **case**
- On insère lisse
- h("lisse") = 11 mod 8 = 3

Exemple de collision

- Les mots sont **arc, chat, lisse, roux** et **case**
- On insère roux
- h("roux") = 17 mod 8 = 1

Exemple de collision

- Les mots sont **arc, chat, lisse, roux** et **case**
- On insère case
- h("case") = 2

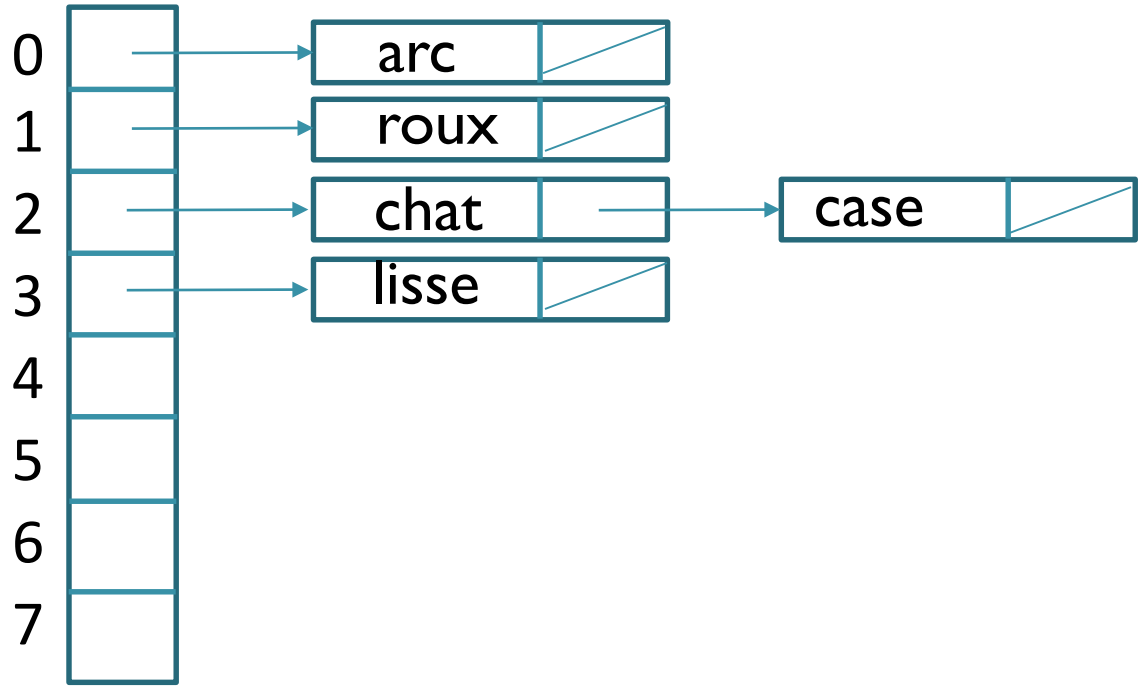
=> Collision

Gestion des collisions

- Que faire si la fonction de hachage de deux clés différentes produit la même adresse ?
  - Première solution: à chaque adresse correspond une liste chaînée d'enregistrements
  - Solution appelée « chainage externe »

Chainage externe

- Pour notre exemple on aurait :



Analyse de performance du chaînage externe

- En utilisant M liens, la technique du chaînage externe réduit le temps de recherche par rapport à la recherche séquentielle d'un facteur M
- Pour une application, M doit être choisi
  - suffisamment petit pour ne pas utiliser trop de mémoire
  - suffisamment grand pour éviter que les listes soient trop longues

Analyse de performance du chaînage externe

- En pratique, on choisira comme valeur de M un dixième du nombre d‘éléments supposés être mis dans la table
- Le chaînage externe s'applique bien lorsque
  - le nombre d'enregistrements peut être estimé avec suffisamment de précision,
  - suffisamment de mémoire est à disposition.

Gestion des collisions

- Que faire si la fonction de hachage de deux clés différentes produit la même adresse ?
  - Deuxième solution: remplacer les liens du chaînage externe par un adressage interne
  - Technique appelée parfois « hachage linéaire »
- Soit N le nombre maximal d'éléments, on réserve une table de M entrées avec M > N
- Principe : on ne gère pas les collisions
- Idée (pour recherche et ajout) :
  - Si la case est occupée, on en tente une autre
  - Si celle-ci est occupée, on en tente encore une autre, etc.

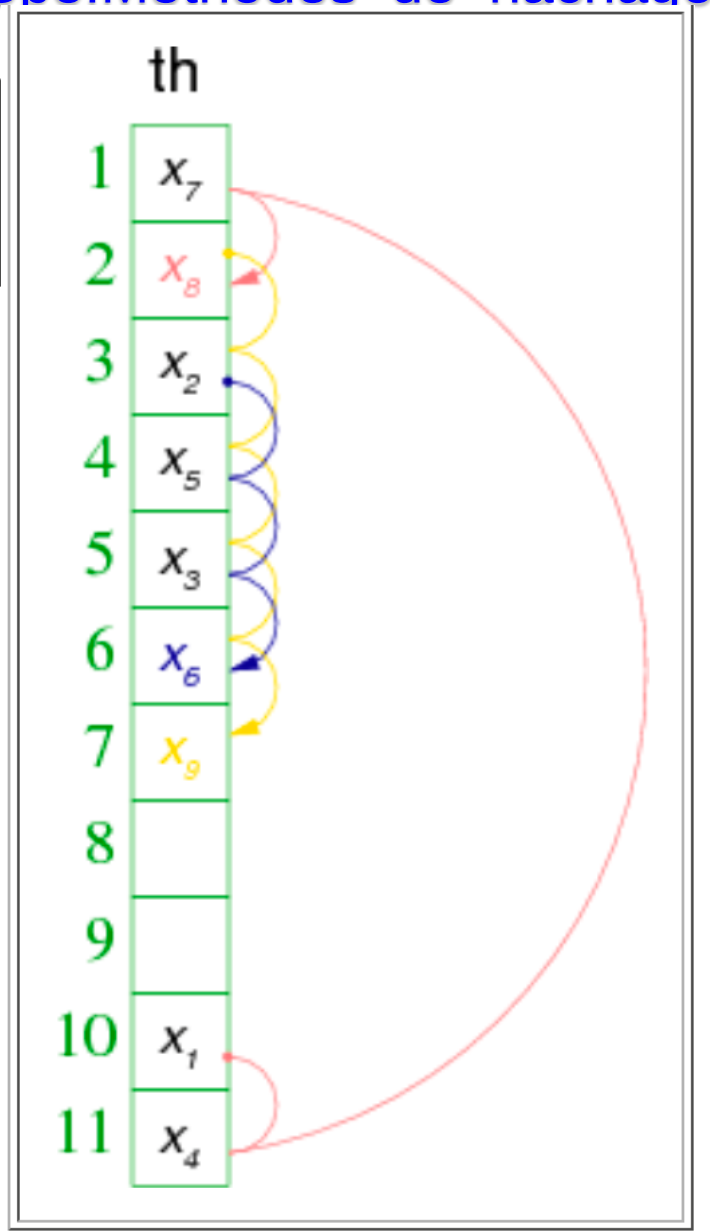
Hachage linéaire

- Lors de la recherche d'une clé, on parcours séquentiellement la table, à partir de l'adresse retournée par la fonction de hachage en distinguant les cas suivants :
  - l'adresse contient la clé cherchée => recherche fructueuse
  - l'adresse est inoccupée => recherche infructueuse
  - l'adresse contient une autre clé et la recherche est poursuivie séquentiellement à l'adresse suivante

Hachage linéaire – Exemple

d'après Epita:Algo:Cours:Info-Spe:Méthodes_de_hachage

éléments	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
valeurs de hachage	10	3	5	11	4	3	1	10	2



Evaluation du hachage linéaire

- Avantage : on garde une structure unique simple
- Inconvénients :
  - Risque de collisions secondaires
  - Il faut marquer les cases inoccupées
- Complexité de la recherche
  - Pour une table à moitié pleine, quelle que soit la taille (résultat calculé empiriquement) :
  - En moyenne 2 accès si absent
  - En moyenne 3,387 accès si présent

Comparaison du chainage externe et du hachage linéaire

- les tables utilisées pour le hachage linéaire sont bien plus grandes que celles utilisées pour le chaînage externe
- par contre, il ne faut pas de mémoire additionnelle pour gérer les collisions
- le hachage linéaire s'applique lorsque le nombre d'enregistrements peut être estimé, que suffisamment de mémoire est disponible, et que les opérations de suppressions sont rares (complexes à programmer).

Conclusion

- En pratique, il ne faudrait jamais dépasser un taux de remplissage de 90% de la table.
- Le choix de la meilleure méthode d'adressage dispersé pour une application est difficile.