4.0 Hashing

N. F. Stewart

Université de Montréal

Adressage dispersé

Je conçois cela comme une modification de la méthode de “Pigeonholing” avec ajout d’une compression randomisée du champs d’adressage.

À lire: *W.* Sections 5.1-5.5, pp. 171-189 (3ième éd.)

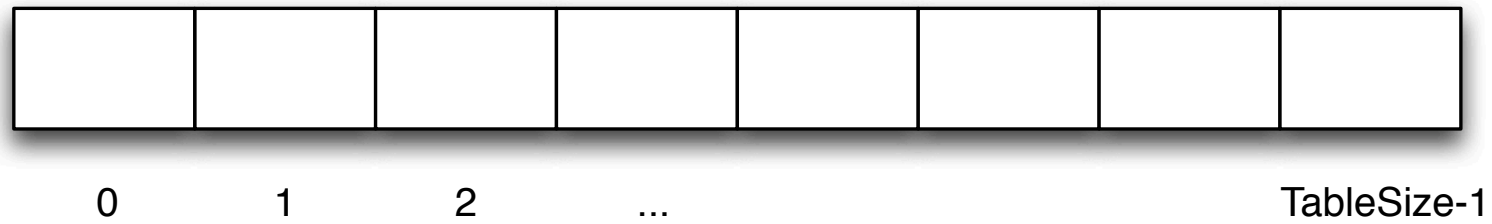
L'idée de base

- Cette méthode a déjà été mentionnée dans l'introduction. L'idée de base:

L'idée de base

- Cette méthode a déjà été mentionnée dans l'introduction. L'idée de base:
- On exprime la clef x en bits, et on mélange pour avoir

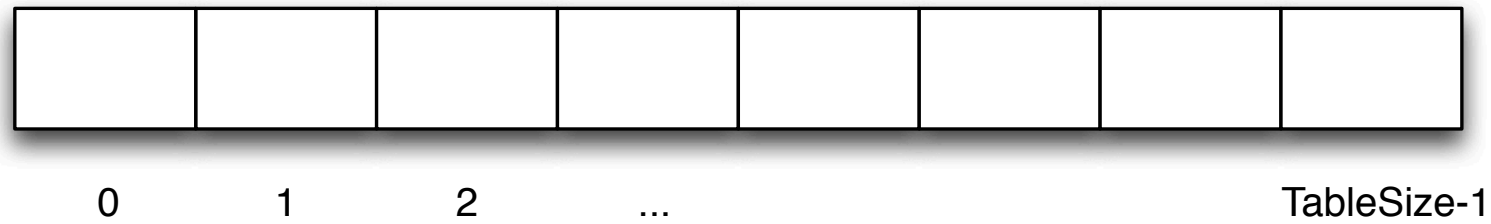
$$h(x) = hashVal(x) \in \{0, \ldots, TableSize - 1\}$$



L'idée de base

- Cette méthode a déjà été mentionnée dans l'introduction. L'idée de base:
- On exprime la clef x en bits, et on mélange pour avoir

$$h(x) = hashVal(x) \in \{0, \ldots, TableSize - 1\}$$

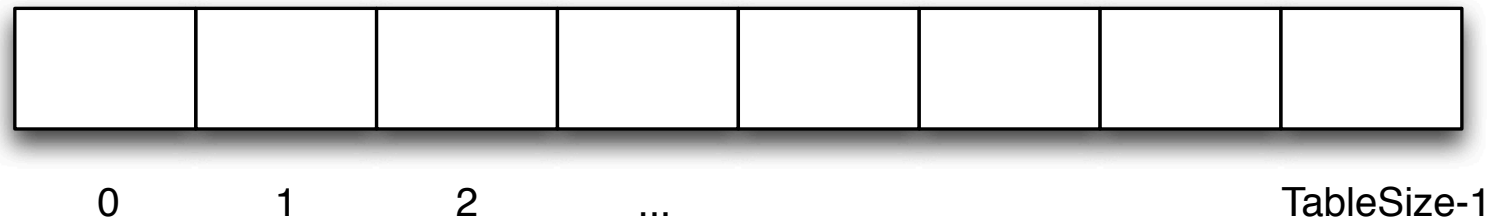


- En recalculant $h(x)$ plus tard, on peut avoir un accès qui est $O(1)$.

L'idée de base

- Cette méthode a déjà été mentionnée dans l'introduction. L'idée de base:
- On exprime la clef x en bits, et on mélange pour avoir

$$h(x) = hashVal(x) \in \{0, \ldots, TableSize - 1\}$$



- En recalculant $h(x)$ plus tard, on peut avoir un accès qui est $O(1)$.
- (Bon, il y a quand même quelques petites difficultés à rêgler!)

- Nous considérons cela comme une méthode pour *TAD Table*:

  $(x, I(x))$ $x =$ la clef (“Key”) $I =$ Information

  Opérations:

- Nous considérons cela comme une méthode pour *TAD Table*:

  $(x, I(x))$ $x =$ la clef (“Key”) $I =$ Information

  Opérations:

  - Opérations “update the data structure”:

- Nous considérons cela comme une méthode pour *TAD Table*:

  $(x, I(x))$ $x =$ la clef (“Key”) $I =$ Information

  Opérations:

  - Opérations “update the data structure”:
    - Insérer $(x, I(x))$

- Nous considérons cela comme une méthode pour *TAD Table*:

  $(x, I(x))$ $x =$ la clef (“Key”) $I =$ Information

  Opérations:

  - Opérations “update the data structure”:
    - Insérer $(x, I(x))$
    - Retirer $(x, I(x))$

- Nous considérons cela comme une méthode pour *TAD Table*:

  $(x, I(x))$ $\quad x =$ la clef ("Key") $\quad I =$ Information

  Opérations:

  - Opérations "update the data structure":
    - Insérer $(x, I(x))$
    - Retirer $(x, I(x))$
  - x donné: Trouver $I(x)$

- Nous considérons cela comme une méthode pour *TAD Table*:

  $(x, I(x))$ $\quad x =$ la clef ("Key") $\quad I =$ Information

  Opérations:

  - Opérations "update the data structure":
    - Insérer $(x, I(x))$
    - Retirer $(x, I(x))$
  - x donné: Trouver $I(x)$
  - x donné: Mettre $I(x)$ à jour ("update $I(x)$")

- Nous considérons cela comme une méthode pour *TAD Table*:

  $(x, I(x))$ $\quad x =$ la clef ("Key") $\quad I =$ Information

  Opérations:

  - Opérations "update the data structure":
    - Insérer $(x, I(x))$
    - Retirer $(x, I(x))$
  - x donné: Trouver $I(x)$
  - x donné: Mettre $I(x)$ à jour ("update $I(x)$")
  - Énumérer les éléments dans l'ordre des clefs,

    i.e., trouver $j_1, \ldots, j_N$

    tels que $x_{j_1} \leq x_{j_2} \leq \ldots \leq x_{j_N}$

    ce qui donne $I_{j_1}, I_{j_2}, \ldots I_{j_N}$.

- Le Hashing comme méthode pour *TAD Table* (suite):

  Opérations: on peut distinquer entre

- Le Hashing comme méthode pour *TAD Table* (suite):

  Opérations: on peut distinquer entre

  - Trouver (le ou les) I_j t. q. $x_j = x$ ("Exact match")

- Le Hashing comme méthode pour *TAD Table* (suite):

  Opérations: on peut distinquer entre

  - Trouver (le ou les) I_j t. q. $x_j = x$ ("Exact match")
  - Trouver les $I_{j_1}, I_{j_2}, \ldots, I_{j_n}$ pour les clefs telles que

    $x_{min} \leq x_{j_\ell} \leq x_{max}$, $\ell = 1, \ldots, n$ ("Range Query")

- Le Hashing comme méthode pour *TAD Table* (suite):

  Opérations: on peut distinquer entre

  - Trouver (le ou les) I_j t. q. $x_j = x$ ("Exact match")
  - Trouver les $I_{j_1}, I_{j_2}, \ldots, I_{j_n}$ pour les clefs telles que

    $x_{min} \leq x_{j_\ell} \leq x_{max}$, $\ell = 1, \ldots, n$ ("Range Query")

- Nous verrons que les opérations (Énumérer, "Range Query", ...) qui font référence à *l'ordre* des clefs ne sont pas efficacement implantées par le Hashing.

Dépendant de la taille de N, et de la fréquence relative des opérations, on pourrait utiliser:

- Des méthodes très simples: $O(N)$, $O(\log N)$, $O(1)$ selon le cas:

Dépendant de la taille de N, et de la fréquence relative des opérations, on pourrait utiliser:

- Des méthodes très simples: $O(N)$, $O(\log N)$, $O(1)$ selon le cas:
  - Tableau trié

Dépendant de la taille de N, et de la fréquence relative des opérations, on pourrait utiliser:

- Des méthodes très simples: $O(N)$, $O(\log N)$, $O(1)$ selon le cas:
  - Tableau trié
  - Tableau non-trié

Dépendant de la taille de N, et de la fréquence relative des opérations, on pourrait utiliser:

- Des méthodes très simples: $O(N)$, $O(\log N)$, $O(1)$ selon le cas:
  - Tableau trié
  - Tableau non-trié
  - Liste chaînée

Dépendant de la taille de N, et de la fréquence relative des opérations, on pourrait utiliser:

- Des méthodes très simples: $O(N)$, $O(\log N)$, $O(1)$ selon le cas:
  - Tableau trié
  - Tableau non-trié
  - Liste chaînée
- Les arbres binaires de recherche, *p.e. Arbre AVL:* $O(\log N)$

Dépendant de la taille de N, et de la fréquence relative des opérations, on pourrait utiliser:

- Des méthodes très simples: $O(N)$, $O(\log N)$, $O(1)$ selon le cas:
  - Tableau trié
  - Tableau non-trié
  - Liste chaînée
- Les arbres binaires de recherche, *p.e. Arbre AVL:* $O(\log N)$
- Les *Skiplist:* aussi $O(\log N)$

Dépendant de la taille de N, et de la fréquence relative des opérations, on pourrait utiliser:

- Des méthodes très simples: $O(N)$, $O(\log N)$, $O(1)$ selon le cas:
  - Tableau trié
  - Tableau non-trié
  - Liste chaînée
- Les arbres binaires de recherche, *p.e. Arbre AVL:* $O(\log N)$
- Les *Skiplist:* aussi $O(\log N)$
- Les *Splay-tree:* coût ammortisé $O(\log N)$

Dépendant de la taille de N, et de la fréquence relative des opérations, on pourrait utiliser:

- Des méthodes très simples: $O(N)$, $O(\log N)$, $O(1)$ selon le cas:
  - Tableau trié
  - Tableau non-trié
  - Liste chaînée
- Les arbres binaires de recherche, *p.e. Arbre AVL:* $O(\log N)$
- Les *Skiplist:* aussi $O(\log N)$
- Les *Splay-tree:* coût ammortisé $O(\log N)$
- Le *Hashing:* une méthode $O(1)$ pour

Dépendant de la taille de N, et de la fréquence relative des opérations, on pourrait utiliser:

- Des méthodes très simples: $O(N)$, $O(\log N)$, $O(1)$ selon le cas:
  - Tableau trié
  - Tableau non-trié
  - Liste chaînée
- Les arbres binaires de recherche, *p.e. Arbre AVL:* $O(\log N)$
- Les *Skiplist:* aussi $O(\log N)$
- Les *Splay-tree:* coût ammortisé $O(\log N)$
- Le *Hashing:* une méthode $O(1)$ pour
  - Insérer, retirer

Dépendant de la taille de N, et de la fréquence relative des opérations, on pourrait utiliser:

- Des méthodes très simples: $O(N)$, $O(\log N)$, $O(1)$ selon le cas:
  - Tableau trié
  - Tableau non-trié
  - Liste chaînée
- Les arbres binaires de recherche, *p.e. Arbre AVL:* $O(\log N)$
- Les *Skiplist:* aussi $O(\log N)$
- Les *Splay-tree:* coût ammortisé $O(\log N)$
- Le *Hashing:* une méthode $O(1)$ pour
  - Insérer, retirer
  - Trouver $I(x)$

Dépendant de la taille de N, et de la fréquence relative des opérations, on pourrait utiliser:

- Des méthodes très simples: $O(N)$, $O(\log N)$, $O(1)$ selon le cas:
  - Tableau trié
  - Tableau non-trié
  - Liste chaînée
- Les arbres binaires de recherche, *p.e. Arbre AVL:* $O(\log N)$
- Les *Skiplist:* aussi $O(\log N)$
- Les *Splay-tree:* coût ammortisé $O(\log N)$
- Le *Hashing:* une méthode $O(1)$ pour
  - Insérer, retirer
  - Trouver $I(x)$
  - Mettre $I(x)$ à jour

Dépendant de la taille de N, et de la fréquence relative des opérations, on pourrait utiliser:

- Des méthodes très simples: $O(N)$, $O(\log N)$, $O(1)$ selon le cas:
  - Tableau trié
  - Tableau non-trié
  - Liste chaînée
- Les arbres binaires de recherche, *p.e. Arbre AVL:* $O(\log N)$
- Les *Skiplist:* aussi $O(\log N)$
- Les *Splay-tree:* coût ammortisé $O(\log N)$
- Le *Hashing:* une méthode $O(1)$ pour
  - Insérer, retirer
  - Trouver $I(x)$
  - Mettre $I(x)$ à jour
  - Hmm, pourquoi pas mettre toutes les autres méthodes à la poubelle?

Parce qu'il y a quand même des petits problèmes:

- (Version adressage ouvert): Retrait un peu difficile

Parce qu'il y a quand même des petits problèmes:

- (Version adressage ouvert): Retrait un peu difficile
- Le "load factor" λ doit rester petit:

Parce qu'il y a quand même des petits problèmes:

- (Version adressage ouvert): Retrait un peu difficile
- Le "load factor" λ doit rester petit:
  - Hypothèses au sujet de la mémoire disponible (version adressage ouvert)

Parce qu'il y a quand même des petits problèmes:

- (Version adressage ouvert): Retrait un peu difficile
- Le "load factor" λ doit rester petit:
  - Hypothèses au sujet de la mémoire disponible (version adressage ouvert)
  - $O(1)$ est dans la réalité $O(\lambda)$ (version chaînage extérieur)

Parce qu'il y a quand même des petits problèmes:

- (Version adressage ouvert): Retrait un peu difficile
- Le "load factor" λ doit rester petit:
  - Hypothèses au sujet de la mémoire disponible (version adressage ouvert)
  - $O(1)$ est dans la réalité $O(\lambda)$ (version chaînage extérieur)
- Les opérations qui dépendent de *l'ordre* des clefs ne sont pas efficacement implantées

Parce qu'il y a quand même des petits problèmes:

- (Version adressage ouvert): Retrait un peu difficile
- Le "load factor" λ doit rester petit:
  - Hypothèses au sujet de la mémoire disponible (version adressage ouvert)
  - $O(1)$ est dans la réalité $O(\lambda)$ (version chaînage extérieur)
- Les opérations qui dépendent de *l'ordre* des clefs ne sont pas efficacement implantées

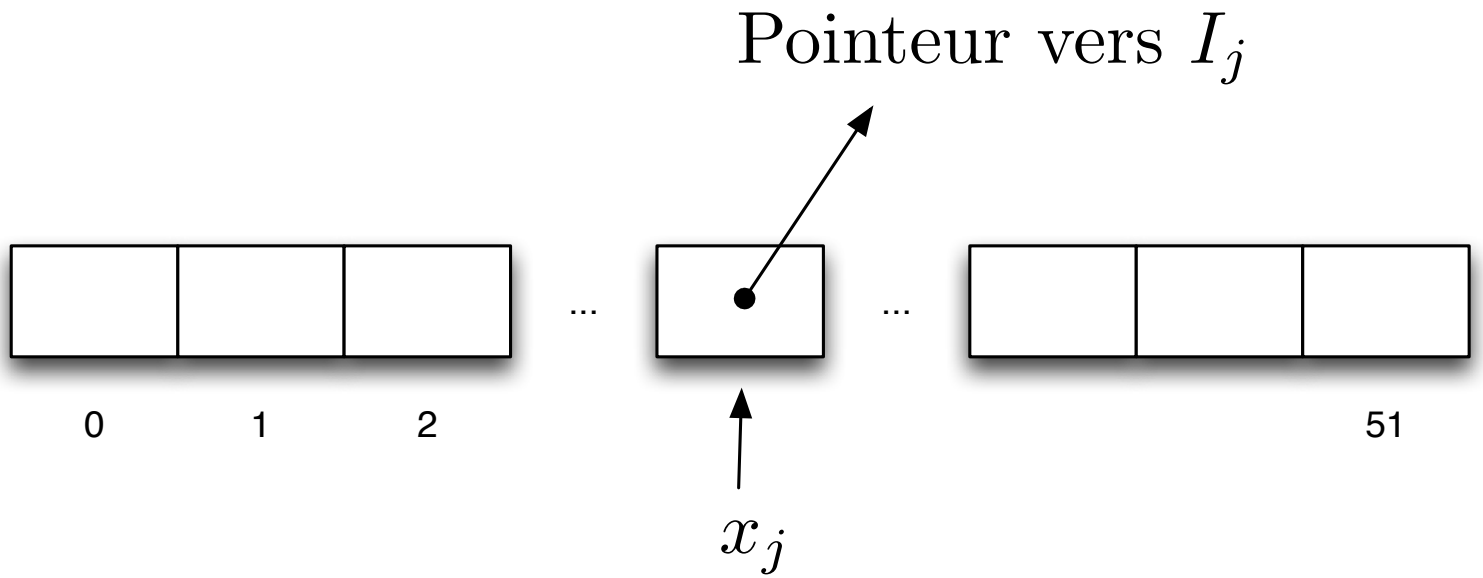
- Cela reste quand même une méthode très attrayante.

Movitation: méthode de “Pigeonholing”

- Supposons que je n’ais que 52 clefs: $2\clubsuit$, $3\clubsuit$, . . . , $A\spadesuit$

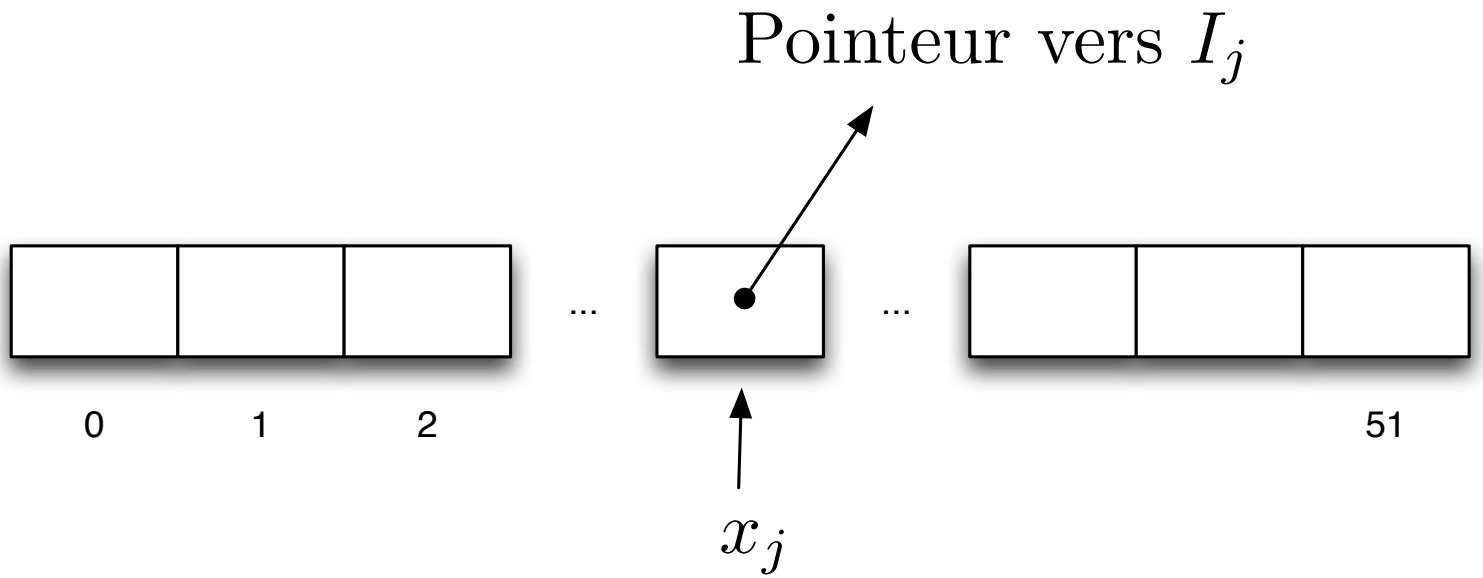
Movitation: méthode de “Pigeonholing”

- Supposons que je n’ais que 52 clefs: $2\clubsuit$, $3\clubsuit$, . . . , $A\spadesuit$
- Je pourrais allouer une case pour chaque clef: $0 \leq x_j \leq 51$



Movitation: méthode de “Pigeonholing”

- Supposons que je n’ais que 52 clefs: $2\clubsuit$, $3\clubsuit$, . . . , $A\spadesuit$
- Je pourrais allouer une case pour chaque clef: $0 \leq x_j \leq 51$



- Cela pourrait être très commode (indexage directe, séquences, couleurs, . . .), et si par exemple 47 cases sur 52 sont toujours vides, ce n’est pas grave.

Méthode de “Pigeonholing” (suite)

- Mais si par contre les clefs sont les noms de variable dans un langage de programmation (application compilateur):

Clef (x_j)	Informations (I_j)
SOMME	classe, espace alloué,...

Méthode de “Pigeonholing” (suite)

- Mais si par contre les clefs sont les noms de variable dans un langage de programmation (application compilateur):

Clef (x_j)	Informations (I_j)
SOMME	classe, espace alloué,...

- Si l’identificateur (exemple *SOMME*) avait même un maximum de 20 caractères, des lettres majuscules seulement, ça ferait déjà 26^{20} cases.

  C’est déjà un peu beaucoup.

Méthode de “Pigeonholing” (suite)

- Mais si par contre les clefs sont les noms de variable dans un langage de programmation (application compilateur):

Clef (x_j)	Informations (I_j)
SOMME	classe, espace alloué,...

- Si l’identificateur (exemple *SOMME*) avait même un maximum de 20 caractères, des lettres majuscules seulement, ça ferait déjà 26^{20} cases.

  C’est déjà un peu beaucoup.

- **Note!** C’est le nombre de clefs *possibles* qui doit être petit (et non pas le nombre de clefs utilisées dans l’application).

- Pour illustrer, restons dans le cas du compilateur, avec des identificateurs d'un seul caractère:

$$A, B, C, \ldots, Z$$

- Pour illustrer, restons dans le cas du compilateur, avec des identificateurs d'un seul caractère:

$$A, B, C, \ldots, Z$$

- On va "mélanger" les bits de la clef en utilisant

$$\begin{array}{l} \textit{public int } h(\textit{Lettre } \ell) \\ \{h \leftarrow \textit{position}(\ell) \bmod 7 \end{array}$$

quitte à les mettre dans une table plus petite de 7 cases.

- Pour illustrer, restons dans le cas du compilateur, avec des identificateurs d'un seul caractère:

$$A, B, C, \ldots, Z$$

- On va "mélanger" les bits de la clef en utilisant

$$\textit{public int } h(\textit{Lettre } \ell)$$
$$\{h \leftarrow \textit{position}(\ell) \bmod 7$$

quitte à les mettre dans une table plus petite de 7 cases.

- Quelle est le danger? **Les collisions.**
(En effet, la raison pour vouloir "mélanger" de façon aléatoire est d'éviter autant que possible les collisions.)

- Pour illustrer, restons dans le cas du compilateur, avec des identificateurs d'un seul caractère:

$$A, B, C, \ldots, Z$$

- On va "mélanger" les bits de la clef en utilisant

$$\textit{public int } h(\textit{Lettre } \ell)$$
$$\{h \leftarrow \textit{position}(\ell) \bmod 7$$

quitte à les mettre dans une table plus petite de 7 cases.

- Quelle est le danger? **Les collisions.**
(En effet, la raison pour vouloir "mélanger" de façon aléatoire est d'éviter autant que possible les collisions.)

- [Exemple au tableau]

Politique de collision

Nous avons besoin, donc, d'une **politique de collision**

- Il y a deux approches:

Politique de collision

Nous avons besoin, donc, d'une **politique de collision**

- Il y a deux approches:
  - Adressage ouvert *W.* Section 5.4, p. 179

Politique de collision

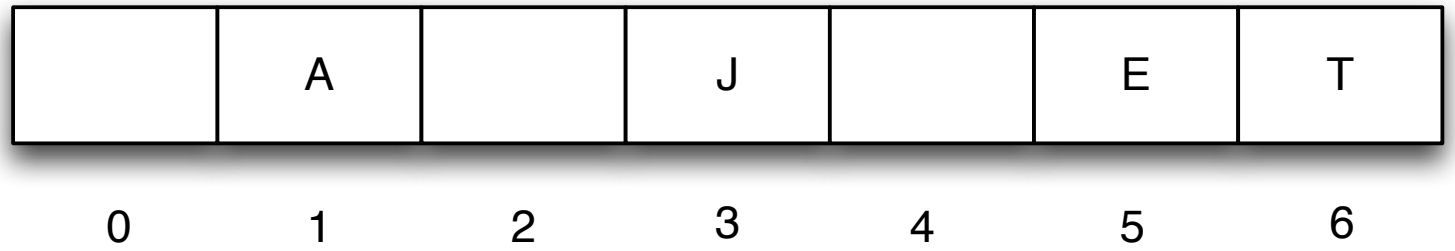
Nous avons besoin, donc, d'une **politique de collision**

- Il y a deux approches:
  - Adressage ouvert *W.* Section 5.4, p. 179
  - Chaînage Externe

Adressage ouvert: ciblage linéaire

- **Adressage ouvert.** Exemple simple: “linear probing”.
  Mettre la clef accidentée dans la première case disponible:

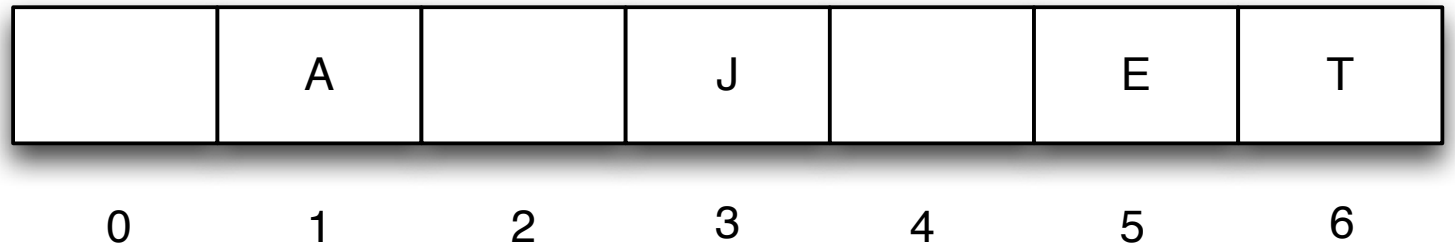
“Probe sequence”: $i \leftarrow (i - 1) \bmod 7$



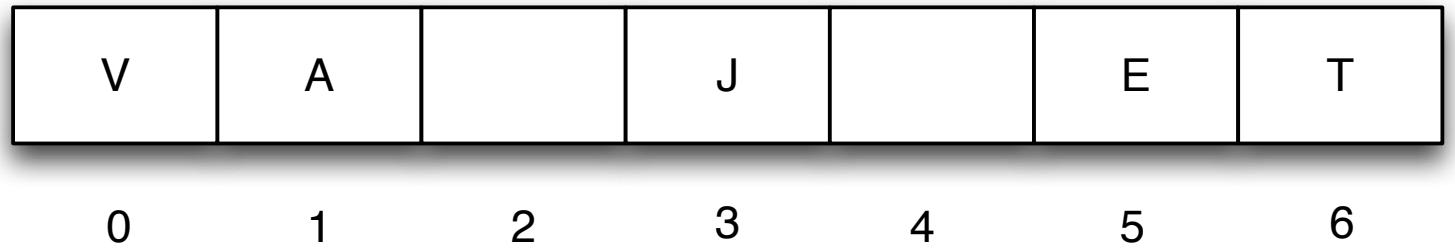
Adressage ouvert: ciblage linéaire

- **Adressage ouvert.** Exemple simple: “linear probing”.
  Mettre la clef accidentée dans la première case disponible:

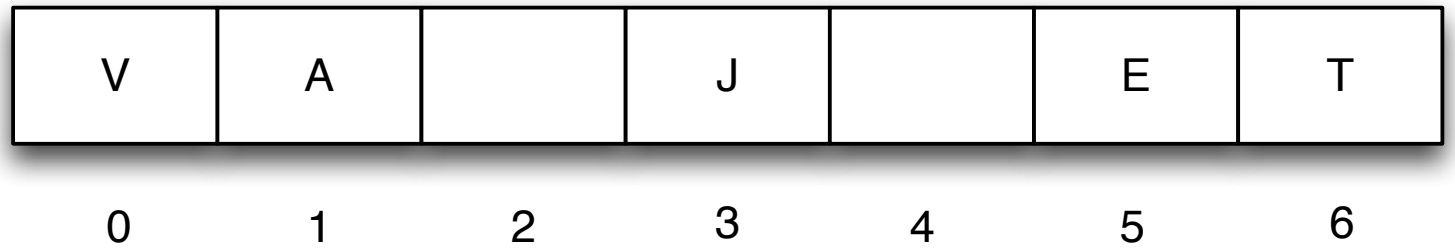
  “Probe sequence”: $i \leftarrow (i - 1) \bmod 7$



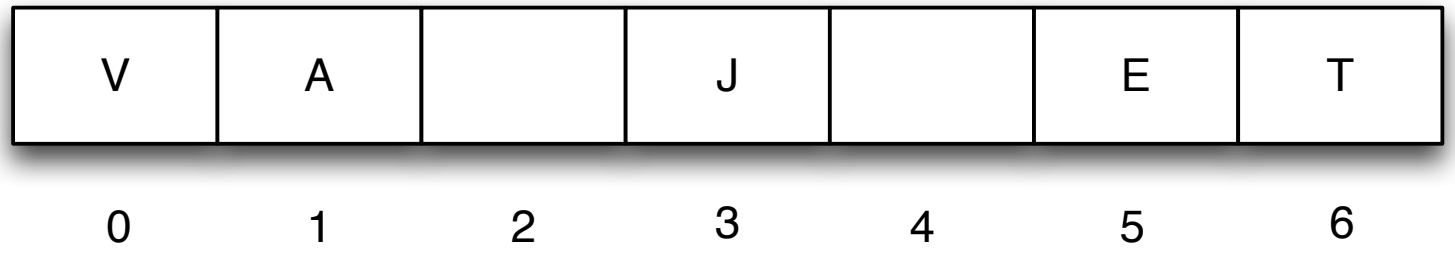
- $p(V) = 22 \Rightarrow h(V) = 1$. Donc:



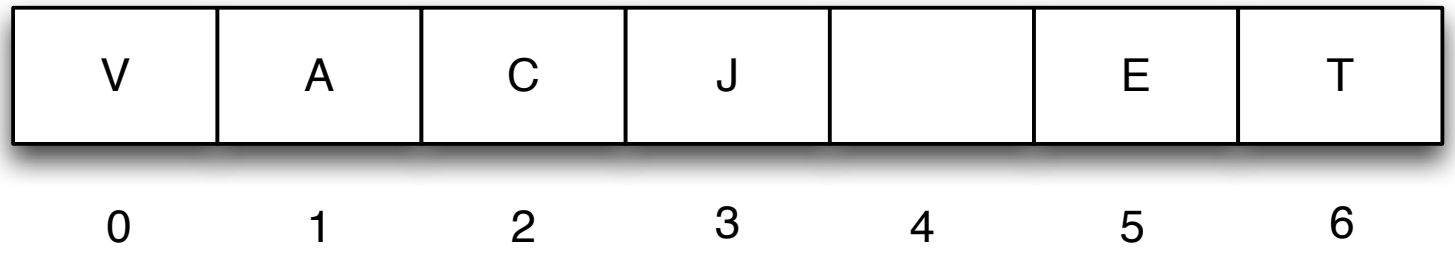
- **Adressage ouvert.** Exemple simple: “linear probing” (suite).



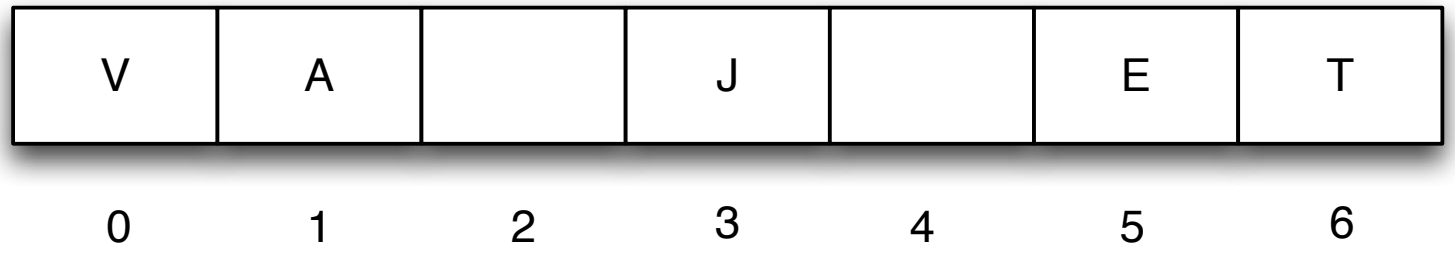
- **Adressage ouvert.** Exemple simple: “linear probing” (suite).



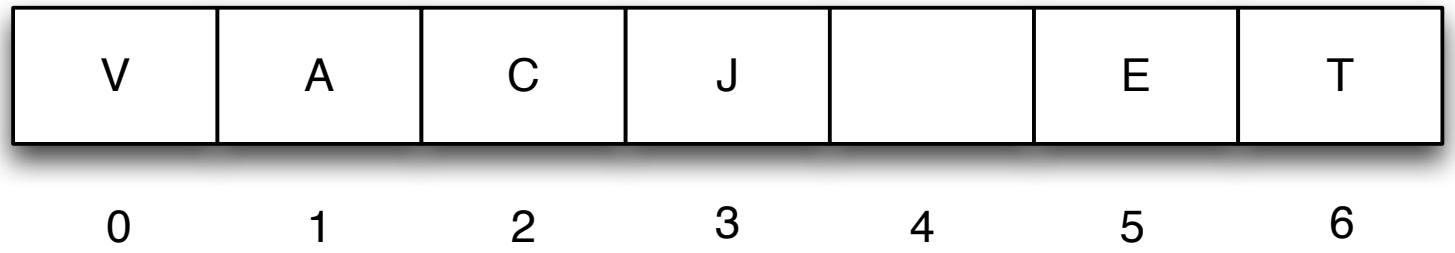
- $p(C) = 3 \Rightarrow h(C) = 3$. Donc:



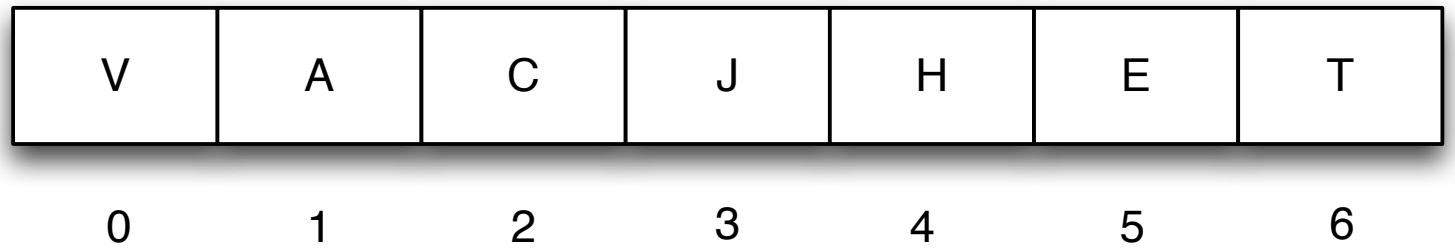
- **Adressage ouvert.** Exemple simple: “linear probing” (suite).



- $p(C) = 3 \Rightarrow h(C) = 3$. Donc:



- $p(H) = 8 \Rightarrow h(H) = 1$. Donc:



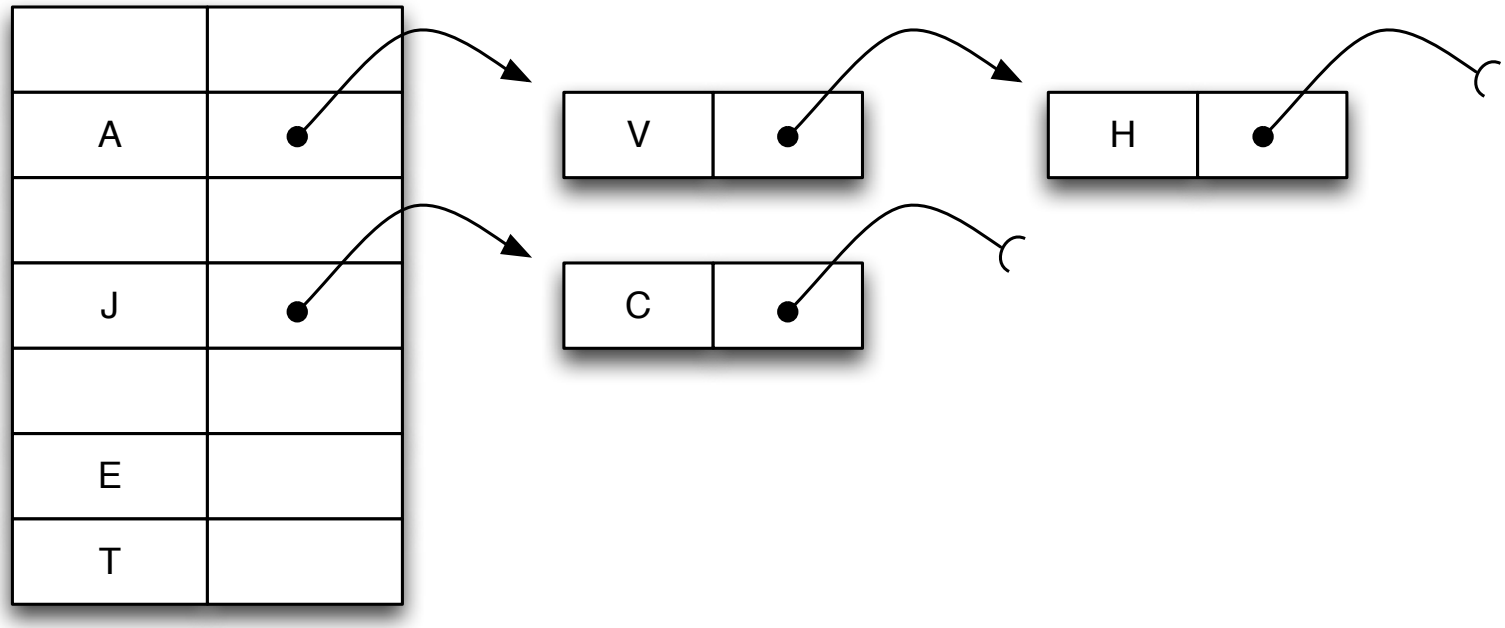
Ciblage linéaire: possibilité d'amas?

- **Adressage ouvert.** Exemple simple: "linear probing" (suite et fin).

  En regardant ce qui s'est passé à la fin de l'exemple, on peut se poser la question suivante. Est-il possible que nous aurons des "cluster" de clefs, c'est-à-dire des *amas*, qui vont impliquer des longues recherches linéaires?

Politique de collision

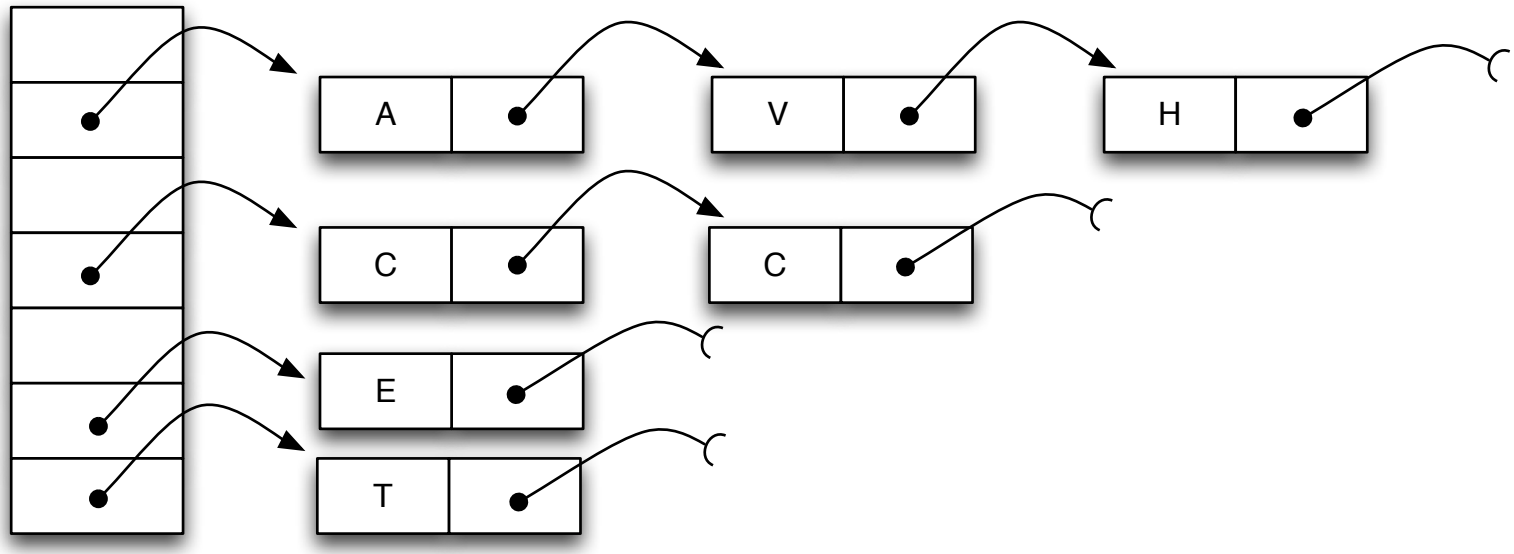
- Deuxième approche: **Chaînage Externe** Weiss, Sec. 5.3



Ou bien, pour que la recherche au démarrage soit similaire à la recherche plus loin:

Politique de collision

- Deuxième approche: **Chaînage Externe** (Weiss, Fig. 5.5)



Coût de régler les collisions

- Deux questions (heureusement différentes):

Coût de régler les collisions

- Deux questions (heureusement différentes):

  1. Quelle est la probabilité d'une collision?

Coût de régler les collisions

- Deux questions (heureusement différentes):

  1. Quelle est la probabilité d'une collision?
  2. Quel est le *coût* de régler les collisions?

Coût de régler les collisions

- Deux questions (heureusement différentes):
  1. Quelle est la probabilité d'une collision?
  2. Quel est le *coût* de régler les collisions?
- La probabilité dépend de

$$\lambda = N/M.$$

$N =$ nombre d'éléments

$M =$ nombre de cases (7 dans notre exemple)

Coût de régler les collisions

- Deux questions (heureusement différentes):
  1. Quelle est la probabilité d'une collision?
  2. Quel est le *coût* de régler les collisions?
- La probabilité dépend de

$$\lambda = N/M.$$

$N =$ nombre d'éléments

$M =$ nombre de cases (7 dans notre exemple)

- Notons que λ peut très bien être supérieur à 1 dans le cas de Chaînage Externe.

Coût de régler les collisions

- On pourrait penser que peut-être n'utiliser que la moitiée de la table ($\lambda = 1/2$) dans le cas d'adressage ouvert serait suffisant pour éviter la majorité de collisions, ou sinon peut-être $\lambda = 1/3$?

Coût de régler les collisions

- On pourrait penser que peut-être n'utiliser que la moitiée de la table ($\lambda = 1/2$) dans le cas d'adressage ouvert serait suffisant pour éviter la majorité de collisions, ou sinon peut-être $\lambda = 1/3$?
- "Utiliser" veut dire allouer assez de mémoire pour la table pour que λ reste inférieur à cette valeur. Par exemple, on pourrait aller jusqu'à $\lambda = 1/4$, c-à-d allouer 4 fois plus de cases que le nombre attendu de valeurs dans la table, en supposant ce nombre connu.

Coût de régler les collisions

- On pourrait penser que peut-être n'utiliser que la moitiée de la table ($\lambda = 1/2$) dans le cas d'adressage ouvert serait suffisant pour éviter la majorité de collisions, ou sinon peut-être $\lambda = 1/3$?
- "Utiliser" veut dire allouer assez de mémoire pour la table pour que λ reste inférieur à cette valeur. Par exemple, on pourrait aller jusqu'à $\lambda = 1/4$, c-à-d allouer 4 fois plus de cases que le nombre attendu de valeurs dans la table, en supposant ce nombre connu.
- Mais, en fait, c'est sans espoir: *Paradoxe des journées de fête.*

  [Explication au tableau.]

Coût de régler les collisions

- Mais, heureusement, cette mauvaise nouvelle ne dérange pas plus que cela, parce que le coût de trouver la clef restera petit: la longueur des recherches, dans le cas d'une collision, restera *courte.* Voici des résultats sans preuve:

Coût de régler les collisions

- Mais, heureusement, cette mauvaise nouvelle ne dérange pas plus que cela, parce que le coût de trouver la clef restera petit: la longueur des recherches, dans le cas d'une collision, restera *courte.* Voici des résultats sans preuve:
- Chaînage Externe

Coût de régler les collisions

- Mais, heureusement, cette mauvaise nouvelle ne dérange pas plus que cela, parce que le coût de trouver la clef restera petit: la longueur des recherches, dans le cas d'une collision, restera *courte.* Voici des résultats sans preuve:
- Chaînage Externe
  - Recherche non-réussie: $C'_N = \lambda = N/M$
    (Beaucoup de recherches vont terminer tout de suite, avec une référence nulle.)

Coût de régler les collisions

- Mais, heureusement, cette mauvaise nouvelle ne dérange pas plus que cela, parce que le coût de trouver la clef restera petit: la longueur des recherches, dans le cas d'une collision, restera *courte.* Voici des résultats sans preuve:
- Chaînage Externe
  - Recherche non-réussie: $C'_N = \lambda = N/M$
    (Beaucoup de recherches vont terminer tout de suite, avec une référence nulle.)
  - Recherche réussie: $C_N \cong 1 + \lambda/2$

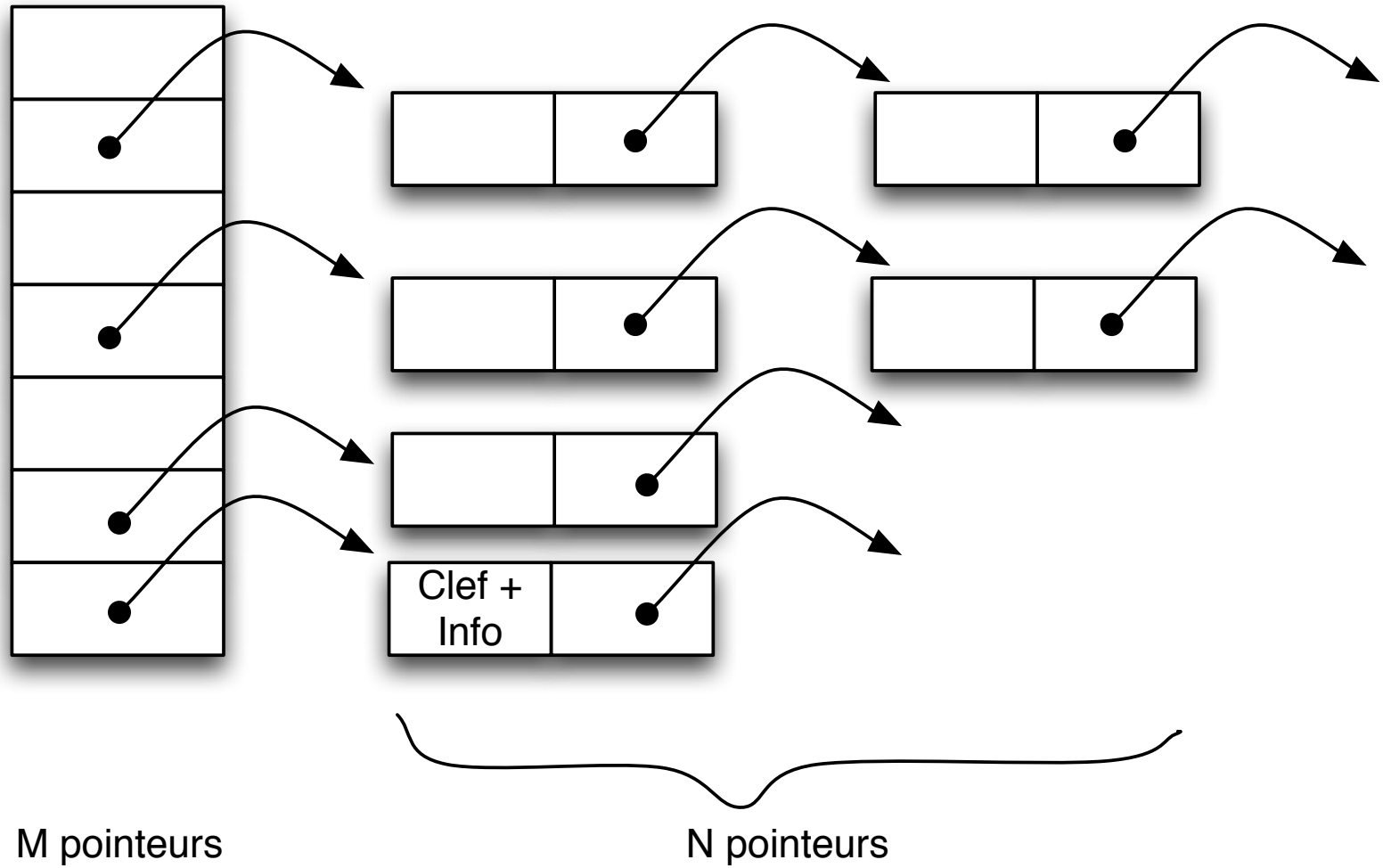
Coût de régler les collisions

- Mais, heureusement, cette mauvaise nouvelle ne dérange pas plus que cela, parce que le coût de trouver la clef restera petit: la longueur des recherches, dans le cas d'une collision, restera *courte.* Voici des résultats sans preuve:
- Chaînage Externe
  - Recherche non-réussie: $C'_N = \lambda = N/M$
    (Beaucoup de recherches vont terminer tout de suite, avec une référence nulle.)
  - Recherche réussie: $C_N \cong 1 + \lambda/2$
- Adressage ouvert: résultats théoriques satisfaisants pour C_N et C'_N à condition que $\lambda << 1$ (par exemple, $\lambda \cong 0.5$).

  Voir Weiss Fig. 5.12, p. 181.

Comparaison espace (Adr. Ouvert *vs.* Ch. Externe)

- Le chaînage externe peut prendre beaucoup d'espace si les records (les informations) sont courts par rapport aux pointeurs.



Comparaison espace (Adr. Ouvert *vs.* Ch. Externe) suite

- Le *chaînage externe* prend donc $N + M$ pointeurs pour enregistrer N records d'information (y compris les clefs).

  Dans le cas extrême où l'information est la seule présence de la clef, cela donne $N + M$ pointeurs, et l'information prend N fois l'espace pour enregistrer une clef.

Comparaison espace (Adr. Ouvert *vs.* Ch. Externe) suite

- Le *chaînage externe* prend donc $N + M$ pointeurs pour enregistrer N records d'information (y compris les clefs).

  Dans le cas extrême où l'information est la seule présence de la clef, cela donne $N + M$ pointeurs, et l'information prend N fois l'espace pour enregistrer une clef.

- *L'adressage ouvert* n'utilise que la mémoire allouée statiquement. Parce que $\lambda < 1$, il y a donc gaspillage de la mémoire.

  S'il y a beaucoup d'incertitude autour de la valeur de N, cette méthode devient moins intéressante. (C'est le compromis classique entre tableau de taille fixe et liste chaînée.)