Hachage

Jérémie Cabessa

Laboratoire DAVID, UVSQ

Introduction

- Les **tables de hachage** sont des structures de données qui permettent de stocker des éléments de manière efficace.
- Si on voulait stocker des entiers de 1 à N, alors un tableau de taille N serait optimal pour cela.
- Les **tables de hachage** généralisent le principe du tableau (*dictionnaires* ou *ensembles* en python).
- En pratique, ce ne sont pas des nombres qui sont stockés mais des données instanciées par des *clés*: une donnée qui ne change pas dans le temps (nombres, chaînes de caractères, etc.)
- Les requêtes que l'on veut faire rapidement sont: RECHERCHER, INSERER, SUPPRIMER.

Introduction

- Les **tables de hachage** sont des structures de données qui permettent de stocker des éléments de manière efficace.
- Si on voulait stocker des entiers de 1 à N, alors un tableau de taille N serait optimal pour cela.
- Les **tables de hachage** généralisent le principe du tableau (*dictionnaires* ou *ensembles* en python).
- En pratique, ce ne sont pas des nombres qui sont stockés mais des données instanciées par des *clés*: une donnée qui ne change pas dans le temps (nombres, chaînes de caractères, etc.)
- Les requêtes que l'on veut faire rapidement sont: RECHERCHER, INSERER, SUPPRIMER.

Introduction

- Les **tables de hachage** sont des structures de données qui permettent de stocker des éléments de manière efficace.
- Si on voulait stocker des entiers de 1 à N, alors un tableau de taille N serait optimal pour cela.
- Les **tables de hachage** généralisent le principe du tableau (*dictionnaires* ou *ensembles* en python).
- En pratique, ce ne sont pas des nombres qui sont stockés mais des données instanciées par des *clés*: une donnée qui ne change pas dans le temps (nombres, chaînes de caractères, etc.)
- Les requêtes que l'on veut faire rapidement sont: RECHERCHER, INSERER, SUPPRIMER.

Introduction

- Les **tables de hachage** sont des structures de données qui permettent de stocker des éléments de manière efficace.
- Si on voulait stocker des entiers de 1 à N, alors un tableau de taille N serait optimal pour cela.
- Les **tables de hachage** généralisent le principe du tableau (*dictionnaires* ou *ensembles* en python).
- En pratique, ce ne sont pas des nombres qui sont stockés mais des données instanciées par des *clés*: une donnée qui ne change pas dans le temps (nombres, chaînes de caractères, etc.)
- Les requêtes que l'on veut faire rapidement sont: RECHERCHER, INSERER, SUPPRIMER.

Introduction

- Les **tables de hachage** sont des structures de données qui permettent de stocker des éléments de manière efficace.
- Si on voulait stocker des entiers de 1 à N, alors un tableau de taille N serait optimal pour cela.
- Les **tables de hachage** généralisent le principe du tableau (*dictionnaires* ou *ensembles* en python).
- En pratique, ce ne sont pas des nombres qui sont stockés mais des données instanciées par des *clés*: une donnée qui ne change pas dans le temps (nombres, chaînes de caractères, etc.)
- Les requêtes que l'on veut faire rapidement sont: RECHERCHER, INSERER, SUPPRIMER.

Table de hachage

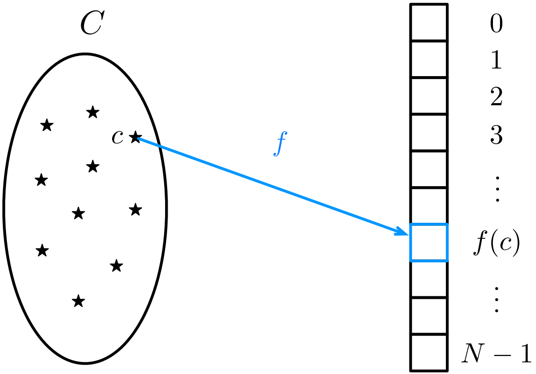


Table de hachage

- Soit un ensemble de clés C que l'on souhaite stocker.
- Pour cela, dispose d'une **table de hachage**, i.e.:
  - Un *tableau* de taille N
  - Une *fonction de hachage* calculable $f : C \to \{0, 1, \ldots, N-1\}$

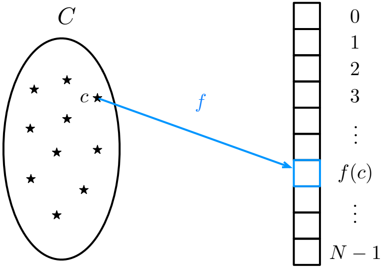


Table de hachage

- Soit un ensemble de clés C que l'on souhaite stocker.
- Pour cela, dispose d'une **table de hachage**, i.e.:
  - Un *tableau* de taille N
  - Une *fonction de hachage* calculable $f : C \to \{0, 1, \dots, N-1\}$

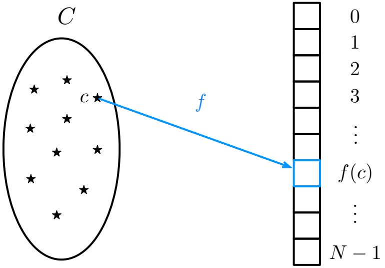


Table de hachage

- Soit un ensemble de clés C que l'on souhaite stocker.
- Pour cela, dispose d'une **table de hachage**, i.e.:
  - Un *tableau* de taille N
  - Une *fonction de hachage* calculable $f : C \to \{0, 1, \ldots, N-1\}$

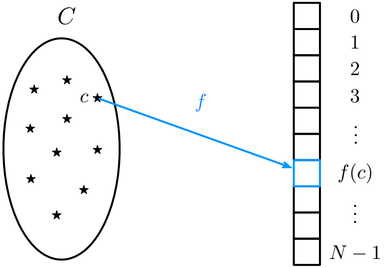


Table de hachage

- Soit un ensemble de clés C que l'on souhaite stocker.
- Pour cela, dispose d'une **table de hachage**, i.e.:
  - Un *tableau* de taille N
  - Une *fonction de hachage* calculable $f : C \to \{0, 1, \ldots, N-1\}$

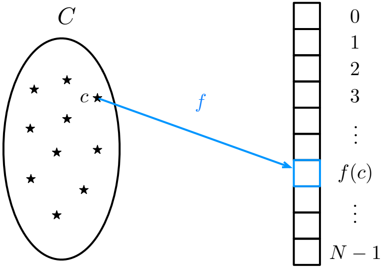


Table de hachage

- *Insérer* une clé c: on calcule l'indice $f(c)$ et on place la clé dans le tableau à cet indice.
- *Rechercher* une clé c: on calcule $f(c)$ et on cherche dans le tableau à cet indice.
- *Supprimer* une clé c: on calcule $f(c)$ et on supprime dans le tableau à l'élément qui se trouve à cet indice.

Table de hachage

- *Insérer* une clé c: on calcule l'indice $f(c)$ et on place la clé dans le tableau à cet indice.
- *Rechercher* une clé c: on calcule $f(c)$ et on cherche dans le tableau à cet indice.
- *Supprimer* une clé c: on calcule $f(c)$ et on supprime dans le tableau à l'élément qui se trouve à cet indice.

Table de hachage

- *Insérer* une clé c: on calcule l'indice $f(c)$ et on place la clé dans le tableau à cet indice.
- *Rechercher* une clé c: on calcule $f(c)$ et on cherche dans le tableau à cet indice.
- *Supprimer* une clé c: on calcule $f(c)$ et on supprime dans le tableau à l'élément qui se trouve à cet indice.

Exemple

- On aimerait stocker les fiches des étudiants dans la base de données.
- Les fiches sont indexées par leurs clés qui sont les numéro étudiants à 10 chiffres (donc 10^{10} possibilités).
- Il y a 1000 étudiants à stocker, donc un tableau de taille $N = 10000$ semble largement suffisant.
- Pour la fonction de hachage f, on peut prendre:
  - les 5 derniers chiffres du numéro étudiant
  - la somme des 5 premiers et derniers chiffres modulo 10000
  - toute autre formule?

Exemple

- On aimerait stocker les fiches des étudiants dans la base de données.
- Les fiches sont indexées par leurs clés qui sont les numéro étudiants à 10 chiffres (donc 10^{10} possibilités).
- Il y a 1000 étudiants à stocker, donc un tableau de taille $N = 10000$ semble largement suffisant.
- Pour la fonction de hachage f, on peut prendre:
  - les 5 derniers chiffres du numéro étudiant
  - la somme des 5 premiers et derniers chiffres modulo 10000
  - toute autre formule?

Exemple

- On aimerait stocker les fiches des étudiants dans la base de données.
- Les fiches sont indexées par leurs clés qui sont les numéro étudiants à 10 chiffres (donc 10^{10} possibilités).
- Il y a 1000 étudiants à stocker, donc un tableau de taille $N = 10000$ semble largement suffisant.
- Pour la fonction de hachage f, on peut prendre:
  - les 5 derniers chiffres du numéro étudiant
  - la somme des 5 premiers et derniers chiffres modulo 10000
  - toute autre formule?

Exemple

- On aimerait stocker les fiches des étudiants dans la base de données.
- Les fiches sont indexées par leurs clés qui sont les numéro étudiants à 10 chiffres (donc 10^{10} possibilités).
- Il y a 1000 étudiants à stocker, donc un tableau de taille $N = 10000$ semble largement suffisant.
- Pour la fonction de hachage f, on peut prendre:
  - les 5 derniers chiffres du numéro étudiant
  - la somme des 5 premiers et derniers chiffres modulo 10000
  - toute autre formule?

Exemple

- On aimerait stocker les fiches des étudiants dans la base de données.
- Les fiches sont indexées par leurs clés qui sont les numéro étudiants à 10 chiffres (donc 10^{10} possibilités).
- Il y a 1000 étudiants à stocker, donc un tableau de taille $N = 10000$ semble largement suffisant.
- Pour la fonction de hachage f, on peut prendre:
  - les 5 derniers chiffres du numéro étudiant
  - la somme des 5 premiers et derniers chiffres modulo 10000
  - toute autre formule?

Exemple

- On aimerait stocker les fiches des étudiants dans la base de données.
- Les fiches sont indexées par leurs clés qui sont les numéro étudiants à 10 chiffres (donc 10^{10} possibilités).
- Il y a 1000 étudiants à stocker, donc un tableau de taille $N = 10000$ semble largement suffisant.
- Pour la fonction de hachage f, on peut prendre:
  - les 5 derniers chiffres du numéro étudiant
  - la somme des 5 premiers et derniers chiffres modulo 10000
  - toute autre formule?

Exemple

- On aimerait stocker les fiches des étudiants dans la base de données.
- Les fiches sont indexées par leurs clés qui sont les numéro étudiants à 10 chiffres (donc 10^{10} possibilités).
- Il y a 1000 étudiants à stocker, donc un tableau de taille $N = 10000$ semble largement suffisant.
- Pour la fonction de hachage f, on peut prendre:
  - les 5 derniers chiffres du numéro étudiant
  - la somme des 5 premiers et derniers chiffres modulo 10000
  - toute autre formule?

Table de hachage: collision!

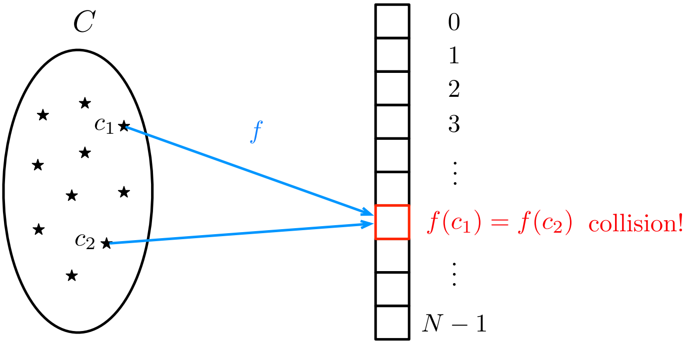


Table de hachage: collisions!

- Lorsque deux clés distinctes c_1 et c_2 ont la même adresse de hachage $f(c_1) = f(c_2)$, il y a collision.
- Cela arrive obligatoirement si l'univers des clés potentielles C est plus grand que la taille N de la table.
- Pour les éviter, on essaie de trouver des fonctions de hachage qui répartissent les clés uniformément dans la table.

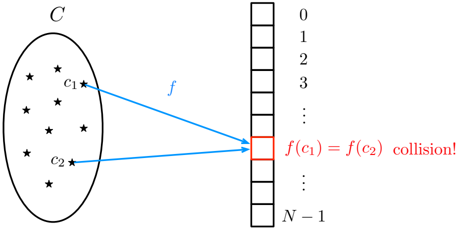


Table de hachage: collisions!

- Lorsque deux clés distinctes c_1 et c_2 ont la même adresse de hachage $f(c_1) = f(c_2)$, il y a collision.
- Cela arrive obligatoirement si l'univers des clés potentielles C est plus grand que la taille N de la table.
- Pour les éviter, on essaie de trouver des fonctions de hachage qui répartissent les clés uniformément dans la table.

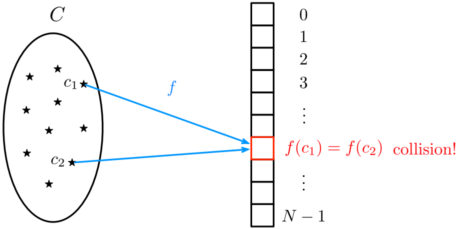
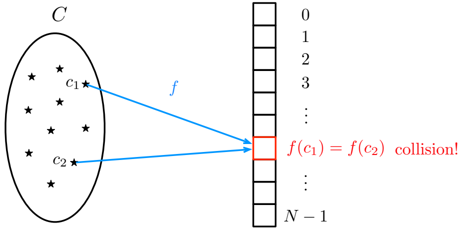


Table de hachage: collisions!

- Lorsque deux clés distinctes c_1 et c_2 ont la même adresse de hachage $f(c_1) = f(c_2)$, il y a collision.
- Cela arrive obligatoirement si l'univers des clés potentielles C est plus grand que la taille N de la table.
- Pour les éviter, on essaie de trouver des fonctions de hachage qui répartissent les clés uniformément dans la table.



Collisions: paradoxe des anniversaires

- Dans un groupe de k personnes (tirées uniformément au hasard), quelle est la probabilité que deux personnes aient la même date d'anniversaire?

$$P(k \text{ dates différentes}) = \frac{365 \cdot 364 \cdots (365-k+1)}{365^k}$$

$$P(\text{au moins 2 dates égales}) = 1 - \frac{365 \cdot 364 \cdots (365-k+1)}{365^k}$$

- Pour $k = 10$ personnes, la probabilité est de 12%
- Pour $k = 23$ personnes, la probabilité est de [illegible]
- Pour $k = 50$ personnes, la probabilité est de [illegible]

Collisions: paradoxe des anniversaires

- Dans un groupe de k personnes (tirées uniformément au hasard), quelle est la probabilité que deux personnes aient la même date d'anniversaire?

$$
\begin{aligned}
P(k \text{ dates différentes}) &= \frac{365 \cdot 364 \cdots (365-k+1)}{365^k} \\
P(\text{au moins 2 dates égales}) &= 1 - \frac{365 \cdot 364 \cdots (365-k+1)}{365^k}
\end{aligned}
$$

- Pour $k = 10$ personnes, la probabilité est de 12%
- Pour $k = 23$ personnes, la probabilité est de 50%
- Pour $k = 60$ personnes, la probabilité est de 99%

Collisions: paradoxe des anniversaires

- Dans un groupe de k personnes (tirées uniformément au hasard), quelle est la probabilité que deux personnes aient la même date d'anniversaire?

$$
\begin{aligned}
P(k \text{ dates différentes}) &= \frac{365 \cdot 364 \cdots (365-k+1)}{365^k} \\
P(\text{au moins } 2 \text{ dates égales}) &= 1 - \frac{365 \cdot 364 \cdots (365-k+1)}{365^k}
\end{aligned}
$$

- Pour $k = 10$ personnes, la probabilité est de 12%
- Pour $k = 23$ personnes, la probabilité est de 50%
- Pour $k = 60$ personnes, la probabilité est de 99%

Collisions: paradoxe des anniversaires

- Dans un groupe de k personnes (tirées uniformément au hasard), quelle est la probabilité que deux personnes aient la même date d'anniversaire?

$$
\begin{aligned}
P(k \text{ dates différentes}) &= \frac{365 \cdot 364 \cdots (365-k+1)}{365^k} \\
P(\text{au moins } 2 \text{ dates égales}) &= 1 - \frac{365 \cdot 364 \cdots (365-k+1)}{365^k}
\end{aligned}
$$

- Pour $k = 10$ personnes, la probabilité est de 12%
- Pour $k = 23$ personnes, la probabilité est de 50%
- Pour $k = 60$ personnes, la probabilité est de 99%

Collisions: paradoxe des anniversaires

- Dans un groupe de k personnes (tirées uniformément au hasard), quelle est la probabilité que deux personnes aient la même date d'anniversaire?

$$
\begin{aligned}
P(k \text{ dates différentes}) &= \frac{365 \cdot 364 \cdots (365-k+1)}{365^k} \\
P(\text{au moins } 2 \text{ dates égales}) &= 1 - \frac{365 \cdot 364 \cdots (365-k+1)}{365^k}
\end{aligned}
$$

- Pour $k = 10$ personnes, la probabilité est de 12%
- Pour $k = 23$ personnes, la probabilité est de 50%
- Pour $k = 60$ personnes, la probabilité est de 99%

Collisions

- De même, étant données k clés $c_1, \ldots, c_k$ dont les adresses $f(c_1), \ldots, f(c_k)$ sont réparties uniformément dans une table de taille N, quelle est la probabilité de collision (i.e. $f(c_i) = f(c_j)$ pour $i \neq j$)?
- Pour k et N assez grands, on estime la probabilité de non-collision à $\exp(-k^2/2N)$
- Application numérique: pour $k = 1000$, si on veut moins de 1% de collisions, alors il faut alors $N > k^2/2\ln(\frac{1}{0.99}) \approx 10^8$, à savoir une mémoire énorme !
- Donc il faut gérer les collisions !

Collisions

- De même, étant données k clés $c_1, \ldots, c_k$ dont les adresses $f(c_1), \ldots, f(c_k)$ sont réparties uniformément dans une table de taille N, quelle est la probabilité de collision (i.e. $f(c_i) = f(c_j)$ pour $i \neq j$)?
- Pour k et N assez grands, on estime la probabilité de non-collision à $\exp(-k^2/2N)$
- Application numérique: pour $k = 1000$, si on veut moins de 1% de collisions, alors il faut alors $N > k^2/2\ln(\frac{1}{0.99}) \approx 10^8$, à savoir une mémoire énorme !
- Donc il faut gérer les collisions !

Collisions

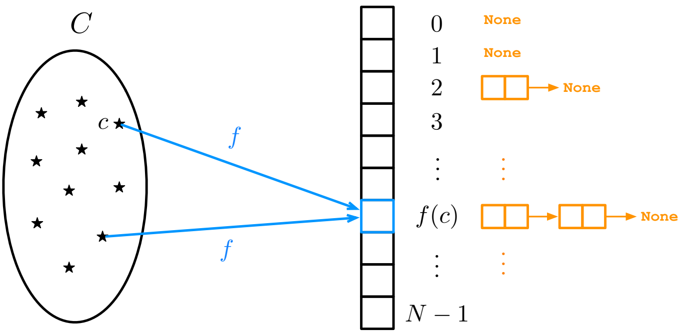
- De même, étant données k clés $c_1, \ldots, c_k$ dont les adresses $f(c_1), \ldots, f(c_k)$ sont réparties uniformément dans une table de taille N, quelle est la probabilité de collision (i.e. $f(c_i) = f(c_j)$ pour $i \neq j$)?
- Pour k et N assez grands, on estime la probabilité de non-collision à $\exp(-k^2/2N)$
- Application numérique: pour $k = 1000$, si on veut moins de 1% de collisions, alors il faut alors $N > k^2/2\ln(\frac{1}{0.99}) \approx 10^8$, à savoir une mémoire énorme !
- Donc il faut gérer les collisions !

Collisions

- De même, étant données k clés $c_1, \ldots, c_k$ dont les adresses $f(c_1), \ldots, f(c_k)$ sont réparties uniformément dans une table de taille N, quelle est la probabilité de collision (i.e. $f(c_i) = f(c_j)$ pour $i \neq j$)?
- Pour k et N assez grands, on estime la probabilité de non-collision à $\exp(-k^2/2N)$
- Application numérique: pour $k = 1000$, si on veut moins de 1% de collisions, alors il faut alors $N > k^2/2\ln(\frac{1}{0.99}) \approx 10^8$, à savoir une mémoire énorme !
- Donc il faut gérer les collisions !

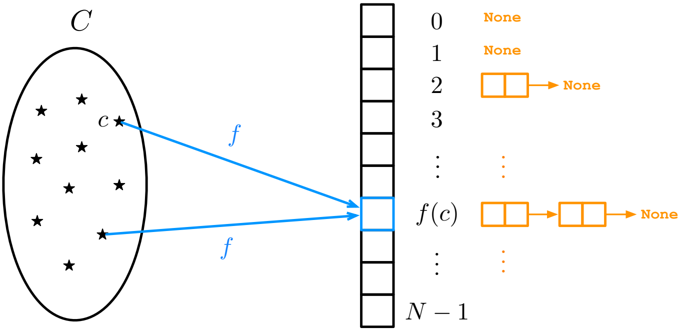
Résolution par chaînage externe

- Chaque case du tableau pointe vers une liste chaînée.
- Les nouvelles clés sont insérées en début de liste chaînée, et il faut la parcourir pour rechercher / supprimer une clé.



Résolution par chaînage externe

- Chaque case du tableau pointe vers une liste chaînée.
- Les nouvelles clés sont insérées en début de liste chaînée, et il faut la parcourir pour rechercher / supprimer une clé.



Résolution par chaînage interne avec adressage ouvert

- Au lieu d'avoir une seule fonction de hachage f, on se donne une suite de fonctions de hachage $f_0, f_1, f_2, \ldots$
- Pour insérer une clé c, on calcule l'indice $f_0(c)$. Si cette case est déjà occupée, on calcule l'indice $f_1(c)$. Si cette case est déjà occupée, on calcule l'indice $f_2(c)$. Etc. Jusqu'à trouver un indice $f_i(c)$ qui correspond à une case de libre. On insère alors la clé dans cette case.
- La fonction f_i représente la i-ème tentative d'insertion.

Résolution par chaînage interne avec adressage ouvert

- Au lieu d'avoir une seule fonction de hachage f, on se donne une suite de fonctions de hachage $f_0, f_1, f_2, \ldots$
- Pour insérer une clé c, on calcule l'indice $f_0(c)$. Si cette case est déjà occupée, on calcule l'indice $f_1(c)$. Si cette case est déjà occupée, on calcule l'indice $f_2(c)$. Etc. Jusqu'à trouver un indice $f_i(c)$ qui correspond à une case de libre. On insère alors la clé dans cette case.
- La fonction f_i représente la i-ème tentative d'insertion.

Résolution par chaînage interne avec adressage ouvert

- Au lieu d'avoir une seule fonction de hachage f, on se donne une suite de fonctions de hachage $f_0, f_1, f_2, \ldots$
- Pour insérer une clé c, on calcule l'indice $f_0(c)$. Si cette case est déjà occupée, on calcule l'indice $f_1(c)$. Si cette case est déjà occupée, on calcule l'indice $f_2(c)$. Etc. Jusqu'à trouver un indice $f_i(c)$ qui correspond à une case de libre. On insère alors la clé dans cette case.
- La fonction f_i représente la i-ème tentative d'insertion.

Résolution par chaînage interne avec adressage ouvert

- **Sondage linéaire:** On prend $f_i = (f_0 + i) \bmod N$, ce qui signifie qu'on va essayer toutes les cases dans l'ordre jusqu'à en trouver une libre.
- Le problème de cette technique est que cela crée de grappes de valeurs qui rendent compliquées les recherches de clés (on risque de tomber au milieu d'une grappe et de devoir la traverser).
- **Sondage quadratique:** On prend par exemple $f_i = (f_0 + i + 3i^2) \bmod N$.

Résolution par chaînage interne avec adressage ouvert

- **Sondage linéaire:** On prend $f_i = (f_0 + i) \mod N$, ce qui signifie qu'on va essayer toutes les cases dans l'ordre jusqu'à en trouver une libre.
- Le problème de cette technique est que cela crée de grappes de valeurs qui rendent compliquées les recherches de clés (on risque de tomber au milieu d'une grappe et de devoir la traverser).
- **Sondage quadratique:** On prend par exemple $f_i = (f_0 + i + 3i^2) \mod N$.

Résolution par chaînage interne avec adressage ouvert

- **Sondage linéaire:** On prend $f_i = (f_0 + i) \mod N$, ce qui signifie qu'on va essayer toutes les cases dans l'ordre jusqu'à en trouver une libre.
- Le problème de cette technique est que cela crée de grappes de valeurs qui rendent compliquées les recherches de clés (on risque de tomber au milieu d'une grappe et de devoir la traverser).
- **Sondage quadratique:** On prend par exemple $f_i = (f_0 + i + 3i^2) \mod N$.

Résolution par chaînage interne avec adressage ouvert

- Lorsque on supprime un élément, la case devient vide. Toutefois, il ne faut pas la marquer vide mais "vidée", car à cause d'elle un élément a pu être caché plus loin. Si la table a besoin de nombreuses insertions / suppressions il faut utiliser une technique plus élaborée.
- Une table à adressage ouvert ne peut pas pas contenir plus d'éléments que de cases !

Résolution par chaînage interne avec adressage ouvert

- Lorsque on supprime un élément, la case devient vide. Toutefois, il ne faut pas la marquer vide mais "vidée", car à cause d'elle un élément a pu être caché plus loin. Si la table a besoin de nombreuses insertions / suppressions il faut utiliser une technique plus élaborée.
- Une table à adressage ouvert ne peut pas pas contenir plus d'éléments que de cases !