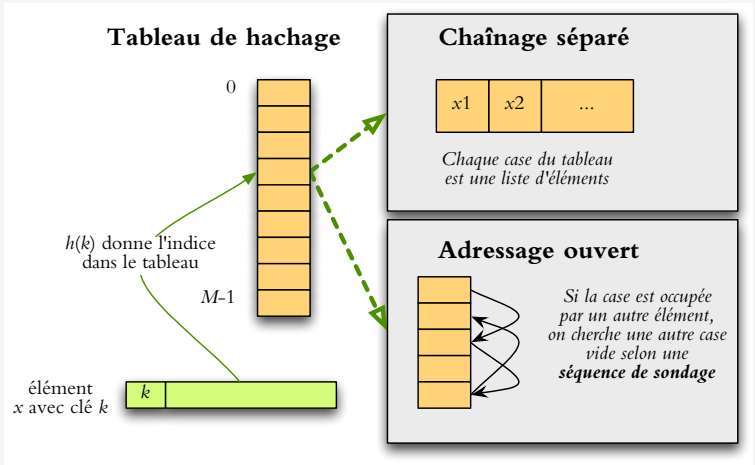
# 14 Hachage

Une autre structure pour implanter le TA table de symboles : **tableau de hachage**. W(fr)




On utilise une **fonction de hachage** $h\colon \mathcal{U} \mapsto \{0, 1, \ldots, M-1\}$ pour définir l'emplacement d'une clé $k \in \mathcal{U}$ dans le tableau.

En **chaînage séparé**, chaque case du tableau contient une liste d´éléments avec clés qui donnent la même valeur de hachage.

En **adressage ouvert**, tous les éléments sont placés directement dans le tableau. La suite de cases à essayer pour le placement est définie par une séquence de sondage qui commence avec l'indice $h(k)$.

Un tableau de hachage performe bien dans **le cas moyen** — dans le pire cas, la performance est comme pour une liste chaînée ($\Theta(n)$ pour insérer ou rechercher). Idéalement, on veut utiliser une fonction de hachage qui mène à une **distribution uniforme** de $h(k)$. La performance moyenne est déterminée par la **facteur de charge** $\alpha = n/M$ quand on a $n$ éléments dans le tableau.

## 14.1 Chaînage séparé

On utilise une liste chaînée (chaque case donne la tête de sa liste), ou un tableau pour stocker les éléments à chaque indice. Les éléments sont non-triés en général (surtout si les clés ne sont pas comparables), parce que $\alpha$ (longueur moyenne d'une liste) reste assez petite dans toutes les implantations efficaces.

## 14.2 Fonctions de hachage

On veut assurer que $h(k)$ est assez uniforme — mais on ne connaît pas la distribution des clés en général !

**Méthode de la division.** On utilise $h(k) = k \bmod M$. Il faut bien choisir $M$ pour éviter la réduction de l'espace de valeurs de hachage à cause des clés non-aléatoires :
$\Rightarrow$ choisir un nombre premier loin de $2^j$ et $10^j$.

**Méthode de la multiplication.** On utilise $h(k) = \lfloor M\{\gamma k\} \rfloor$ avec une valeur flottante $\gamma$ (partie fractionnaire : $\{x\} = x - \lfloor x \rfloor$). La dispersion des valeurs de hachage dépend principalement de $\gamma$. Un bon choix est $\gamma = \frac{\sqrt{5}-1}{2}$. Ici, on choisit une taille $M = 2^p$ pour un calcul rapide avec opérations entières (multiplication, décalage de bits). Calcul rapide pour un entier $k$ representé sur $w$ bits [p.e., $w = 32$ pour `int` de Java] : écrire $\gamma = A/2^w$, alors `h = (A*k) >>> (w-p)` (`>>>` dénote décalage de bits vers la droite).

**Hachage universel.** On a une clé de $r$ caractères : $k = \langle k_1, k_2, \ldots, k_r \rangle$ (p.e., `String`). Fonction de hachage $h^{(r)}(k) = \left( \sum_{i=1}^{r} h_i(k_i) \right) \bmod M$ avec des fonctions de hachage $h_i(x)$ choisies «au hasard». En pratique, on utilise une règle approximative simple comme $h_i(x) = a^{r-i} \cdot x$ (où $a$ est un nombre premier) :

initialiser $h \leftarrow 0$ ; **for** $i \leftarrow 1, 2, \ldots, r$ **do** $h \leftarrow (a \cdot h + k_i) \bmod M$

P.e., **hashcode**() de `String` (Java) utilise $a = 31$ et $M = 2^{32}$ (32-bit entiers).

**Définition 14.1.** *Deux clés $k, k'$ sont **en collision** si $h(k) = h(k')$.*

Pour une bonne performance, on veut éviter des collisions entre clés. W(fr)

**Théorème 14.1.** ***[Birthday paradox]*** *Avec des clés uniformement distribués, on a au moins une collision avec probabilité $> 1/2$ quand $n > 1.18\sqrt{M}$.*

*Démonstration.* Probabilité d'aucune collision : $p = 1\left(1 - \frac{1}{M}\right)\left(1 - \frac{2}{M}\right) \cdots \left(1 - \frac{n-1}{M}\right) < \prod_{i=0}^{n-1} e^{-i/M} = \exp\left(-\frac{(n-1)n}{2M}\right)$. Avec $n/\sqrt{M} > \sqrt{2 \ln 2} = 1.177\cdots$, on a $p < 1/2$. ■

**Théorème 14.2.** *En hachage universel défini par les fonctions $h_i(x) = a_i x$, avec $a_i \in \{0, \ldots M-1\}$ choisis au hasard, la probabilité de collision entre deux clés est $1/M$.*

*Démonstration.* Clés fixées $\langle x_1, \ldots x_r \rangle$ et $\langle y_1, \ldots, y_r \rangle$ sont en collision si $\sum_{i=1}^{r} a_i(x_i - y_i) \equiv 0 \pmod{M}$. Il existe $M^{r-1}$ choix de $\langle a_1, \ldots a_r \rangle$ avec égalité $\Rightarrow$ collision avec probabilité $M^{r-1}/M^r = 1/M$. ■

## 14.3 Adressage ouvert

En adressage ouvert, on fait la **sondage** (*probing*) d'une séquence de positions : dépend de la clé à insérer. L'adressage ouvert ne permet pas $\alpha > 1$. On examine les cases $h_0(k), h_1(k), \ldots$ avec une fonction $f : h_i(k) = h(k) + f(i,k) \bmod M$. La fonction $f$ représente la **stratégie de résolution de collision**. Méthodes :
⋆ sondage linéaire $f(i,k) = i$ ou $f(i,k) = ai + b$.
⋆ double hachage $f(i,k) = ih'(k)$ avec fonction de hachage auxiliaire $h'$

**Suppression.** On utilise souvent une **approche paresseuse** (*lazy deletion*) : suppression = remplacement par une sentinelle. search doit passer les sentinelles, insert peut les recycler.

**Sondage linéaire.** (*Linear probing*) : $h(k), h(k)+1, h(k)+2, \ldots$. Mène à la **grappe forte** (*primary clustering*) — blocs de cases occupées.

**Double hachage.** Généralisation de sondage linéaire : $h(k), h(k)+c, h(k)+2c, h(k)+3c, \ldots$. Ici, $c$ dépend de la clé $k$ : $c = h'(k)$. Cette méthode est très proche d'une résolution idéale. Exemples : $h'(x) = 1 + x \bmod M'$ avec $M' < M$ ou $h'(x) = M' - (x \bmod M')$.

## 14.4 Performances : temps moyen en fonction de la facteur de charge $\alpha$

| | **chaînage séparé** | **sondage linéaire** | **double hachage** |
|---|---|---|---|
| | (nœuds) | (cases examinées dans la séquence de sondage) | |
| recherche fructueuse | $\frac{1+\alpha}{2}$ | $\approx \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$ | $\approx \frac{1}{n}\sum_{i=0}^{n-1} \frac{1}{1-i/M} \approx \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$ |
| recherche infructeuse | $1 + \alpha$ | $\approx \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$ | $\approx 1 + \alpha + \alpha^2 + \cdots \approx \frac{1}{1-\alpha}$ |
| insertion | 1 | *même que recherche infructeuse* | |