# Double hachage

Généralisation de sondage linéaire : $h(k), h(k)+c, h(k)+2c, h(k)+3c, \ldots$
$c$ dépend de la clé $k$ : $c = h'(k)$

Exemples (éviter $c = 0$ !) : $h'(x) = 1 + x \bmod M'$ avec $M' < M$
$h'(x) = M' - (x \bmod M')$

Très proche d'une résolution idéale

Nombre moyen de sondages lors d'insertion ou recherche infructueuse :
$\approx 1 + \alpha + \alpha^2 + \cdots \approx \frac{1}{1-\alpha}$

sondages lors de recherche fructueuse :
$\approx \frac{1}{N} \sum_{i=0}^{N-1} \frac{1}{1-i/M} \approx \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

# Deletion

suppression avec adressage ouvert : très difficile

utiliser suppression paresseuse (*lazy deletion*) : marquer la case «supprimée»
remplacer l'élément supprimé par une sentinelle

search doit passer par les sentinelles
insert peut utiliser la case d'une sentinelle

$\Rightarrow$ facteur de remplissage inclut les éléments supprimés

# Un autre usage de hachage

Problème : comment compter le nombre d'éléments différents sur une grande liste ?

Exemples : nombre de pages Web, nombre d'addresses IP différents dans un ensemble de paquets envoyés (diagnostique d'attaques sur l'internet)

Idée d'un compteur probabiliste : si on veut compter jusqu'à $n$, on a besoin de $\lg(n+1)$ bits :

```
1 Compteur
2 faire n ← n + 1
```

Est-ce qu'on peut compter avec $O(\log \log n)$ bits jusqu'à $n$ ?

# Compteur probabiliste

On veut juste estimer la grandeur de $n$ : on va essayer de stocker $\lg n$ seulement.

Idéalement, on aura $X = 0$ pour $n = 1$, $X = 1$ pour $n = 2..3$, $X = 2$ pour $n = 4..7$ etc.
Donc il faut attendre l'arrivée de $2^X$ éléments avant d'incrémenter $X$.
On ne veut pas compter jusqu'à $2^X$...

```
1 Compteur
2 faire X ← X + 1 avec probabilité 2^{-X}
```

On a que $n \approx 2^X$ à la fin.

Pour stocker $X$, on a besoin de $\lg \lg N$ bits, et on peut compter jusqu'à $N$ !

# Nombre d'éléments différents

On a une grande liste $x_1, \ldots, x_n$ et on voudrait savoir le nombre d'éléments différents (**cardinalité**). On sait que $x_i \in U$ ou $U$ est un ensemble fini (p.e., addresses IP)

Solution possible : tableau booléen de taille $|U|$ — prend trop de mémoire

Une approche approximative : utilisons une bonne fonction de hachage $h$. La valeur de hachage $h(x_i)$ est un nombre binaire sur $t$ bits.

Chaque bit de $h(x_i)$ est 0 ou 1 avec des probabilités 50–50%
Nombre de bits 0 au début : égal à $k$ avec probabilité $0.5^k$. Soit $\rho(h(x))$ le nombre de 0s au début de $h(x)$.

Si $x_i = x_j$, alors $h(x_i) = h(x_j)$. Supposons qu'on a $m$ éléments différents parmi les $n$. Si on regarde $\rho(h(x_i))$, on trouvera en moyenne $m/2^k$ éléments différents avec $\rho(h(x_i)) = k$

$\Rightarrow$ le maximum $\max_{i=1,\ldots n} \rho(h(x_i))$ devrait être à peu près $\lg m$.

# Loglog counting

1 **Cardinalité**$(x)$
2 faire $\rho_{\mathsf{max}} \leftarrow \mathsf{max}\{\rho_{\mathsf{max}}, \rho(h(x))\}$

Initialiser avec $\rho_{\mathsf{max}} \leftarrow 0$.

Nombre d'éléments différents est estimé par

$$m = \frac{2^{\rho_{\mathsf{max}}}}{\phi}$$

avec $\phi = e^{-\gamma}\sqrt{2} \approx 0.78$ (le $\phi$ vient de l'analyse précis de la distribution de $\rho_{\mathsf{max}}$).

Si $\rho_{\mathsf{max}} \leq r$, on peut le stocker sur $\lg(r+1)$ bits $\Rightarrow$ avec $O(\log\log M)$ bits on peut estimer une cardinalité jusqu'à $M$