Tables de hachage

Sets
Ensembles (finis) :
- Insertion d'un élément (qui n'existe pas encore)
- Recherche d'un élément
- Suppression d'un élément

Maps
Tableaux associatifs (dictionnaires) : ensembles de paires (clef ↘, valeur ↓) (k, v)
- Insertion d'une paire (dont la clef n'existe pas encore)
- Recherche d'une paire par sa clef
- Suppression d'une paire (par sa clef)

En Python : set pour les ensembles (x in s)
dict pour les tableaux associatifs
(implémentés par une table de hachage)

Implémentations naïves

1) Tableau dynamique contenant les éléments de l'ensemble
- Insertion : $O(1)$ amorti
- Recherche : $\Theta(n)$
- Suppression : $O(1)$ amorti après recherche (en l'échangeant avec le dernier élément du tableau)

2) Tableau dynamique trié selon les éléments / les clefs
- Insertion : $\Theta(n)$
- Recherche : $\Theta(\log n)$
- Suppression : $\Theta(n)$

3) Liste doublement chaînée
- Insertion : $O(1)$
- Recherche : $\Theta(n)$
- Suppression : $O(1)$ après recherche

Tables de hachage

Espace de hachage : $\{0, \dots, m-1\}$

Univers U : ensemble des (éléments possibles / clefs)

n : nombre d'éléments à stocker

En général $|U| \gg n$ et même parfois $|U| = \infty$

Aparté : si $|U|$ est proche de n, on peut utiliser pour représenter un ensemble un tableau de $|U|$ bits (un bitmap) dont le i-ème bit est à 1 ssi l'objet appartient à l'ensemble.

- Insertion : $O(1)$
- Recherche : $O(1)$
- Suppression : $O(1)$

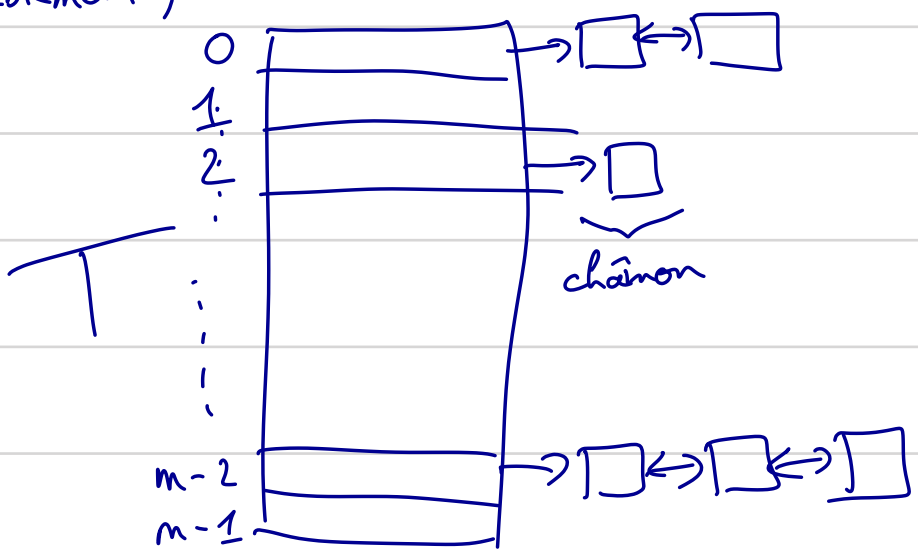
Complexité (de la construction en espace) $O(|U|)$

On va prendre m proche de n.

$\alpha = \frac{n}{m}$ facteur de charge (load factor)

On va prendre $\alpha = O(1)$

Table de hachage : tableau fixe de m éléments, chaque case du tableau étant soit vide soit un pointeur vers une liste chaînée (doublement) des éléments de l'ensemble.



On suppose l'existence d'une fonction de hachage

$$h : U \longrightarrow \{0, \dots, m-1\}$$

calculable en $O(1)$.

Construire la table de hachage : $O(m)$

Insérer un nouvel élément $x \in U$

→ Calcule $j = h(x)$

→ Accède à $T[j]$

$O(1)$ → Ajoute x au début de la liste chaînée pointée par $T[j]$ (si elle existe ; sinon on la crée)

Rechercher si un élément $x \in U$ est dans l'ensemble

→ Calcule $j = h(x)$

→ Accède à $T[j]$

$\Theta(n_j)$ → Recherche si x est dans la liste chaînée pointée par $T[j]$

($n_j = \#\{x \in S \mid h(x) = j\}$)

Suppression (après recherche)

$O(1)$ → Suppression ds liste dblt chaînée

Dans le pire des cas

ensemble à stocker ↓

$h(x) = h(x')$ pour tous $x, x' \in S$

collision

$n_{h(x)} = n$ et donc complexité en $\Theta(n)$

Cela se produit si :

- h est "mauvaise" (elle répartit les éléments de l'univers non uniformément dans l'espace de hachage)
- on est "malchanceux" : la fonction de hachage est uniforme mais les éléments à stocker ont quand même tous la même valeur par la fonction de hachage

Exemple de fonction de hachage

$U = \mathbb{N}$ $\quad h : x \longmapsto x \bmod m$

Par exemples, si $m = 256 = 2^8$

Si x est un entier sur 8 octets (entre 0 et $2^{64} - 1$) alors $h(x)$ est l'octet de poids faible

Octet de poids faible : 8 bits les plus à droite dans la représentation binaire

$$\underbrace{10100101}_{\text{octet de poids faible}}\,|\,01100111$$

* Si $m = 1000$

Si on stocke des chiffres de population souvent arrondies au millier le plus proche, alors on a souvent $h(x) = 0$.

Du coup, on va choisir m un nombre premier sans signification particulière pour les éléments de l'univers; et on va essayer de choisir h un peu plus complexe que le modulo. Ne résout pas toutes les situations problématiques.

Hypothèse d'uniformité $\Pr(h(x) = j) = \frac{1}{m}$ ($x \in$ éléments de l'ensemble)

pour $0 \leq j \leq m-1$

$$E[n_j] \left(= \sum_{k=0}^{n} k \Pr(n_j = k)\right)$$

$$= E\left[\mathbb{1}_{\{h(S_1)=j\}} + \mathbb{1}_{\{h(S_2)=j\}} + \dots + \mathbb{1}_{\{h(S_n)=j\}}\right]$$

$$= \sum_{i=1}^{n} \Pr(h(S_i) = j) = \sum_{i=1}^{n} \frac{1}{m} = \frac{n}{m} = \alpha$$

α : facteur de charge

Si :
→ l'hypothèse d'uniformité est vérifiée
→ $\alpha = O(1)$

Alors Recherche en $O(1)$ en moyenne

Hachage universel

On se donne une famille de fonctions de hachage $H = \{h_1, \dots, h_w\}$ avec la propriété suivante :

$$\text{Pour } x \neq y \in U \quad \#\{h(x) = h(y) \mid h \in H\} \leq \frac{|H|}{m}$$

On construit une table de hachage comme avant, mais en tirant la fonction de hachage h uniformément aléatoirement parmi H

On veut montrer que $E[n_j] = O(\alpha)$

On fixe $k, l \in U$ avec $k \neq l$

On introduit
- la variable aléatoire $X_{kl} = \mathbb{1}_{h(k)=h(l)}$
- la variable aléatoire Y_k = le nombre d'éléments dans la liste chaînée à la case $h(k)$

$$E[Y_1] \leq E\left[\sum_{\substack{l \in S \\ l \neq k}} X_{kl}\right] + 1$$

$$\leq \sum_{\substack{l \in S \\ l \neq 1}} \underbrace{E[X_{1l}]}_{\leq \frac{1}{m} \text{ par hypothèse sur ma famille } H} + 1$$

$$\leq \sum_{\substack{l \in S \\ l \neq 1}} \frac{1}{m} + 1$$

$$\leq \frac{n}{m} + 1 = O\left(\frac{n}{m}\right) = O(\alpha)$$

Choix d'un ensemble de fonctions de hachage universel

Fixons $p > m$ un nombre premier.
On suppose aussi $U = \{0, \dots, |U|-1\}$ avec $|U| < p$
On considère pour $0 < a < p$ et $0 \leq b < p$
la fonction de hachage $h_{a,b}$ définie par :

$$h_{a,b}(k) = \Big[(ak + b) \mod p \Big] \mod m$$

$$H = \{ h_{a,b} \mid 0 < a < p, \quad 0 \leq b < p \}$$

$$|H| = (p-1) \times p$$

Mq $\#_h \{ h(x) = h(y) \} \leq \dfrac{p(p-1)}{m}$

pour tous $x \neq y$

Soit $k \neq l \in U$

$$r = (ak + b) \mod p$$
$$s = (al + b) \mod p$$

$$r - s = a(k-l) \mod p \neq 0 \mod p \text{ donc } r \neq s$$

($a \neq 0$, $(k-l) \neq 0$)

$$\begin{cases} a = (r-s)\big[(k-l)^{-1} \mod p\big] \mod p \\ b = r - ak \mod p \end{cases}$$

Il y a une bijection entre les (a,b) avec $0 < a < p$ et $0 \leq b < p$ et les (r,s) avec $r \neq s$ et $0 \leq r < p$, $0 \leq s < p$.

Donc, étant donné $k \neq l$ en entrée, on obtient toutes les paires (r,s) avec $r \neq s$ avec la même probabilité $\dfrac{1}{p(p-1)}$ en choisissant uniformément aléatoirement une paire (a,b) avec $0 < a < p$ et $0 \leq b < p$.

$$\Pr_h[h(k)=h(\ell)] = \Pr_{\substack{0\le r,s<p \\ r\neq s}} \underbrace{[r \equiv s \bmod m]}_{(*)}$$

Pour un $0 \le r < p$ fixé, le nombre de $s \neq r$ t.q. $0 \le s < p$ et $r \equiv s \bmod m$:

$$\le \left\lceil \frac{p}{m} \right\rceil - 1 \le \frac{p-1}{m}$$

choix de r → p ← choix de s

$$(*) \le \frac{(p)\times\frac{p-1}{m}}{(p)\times(p-1)} = \frac{1}{m}$$

$$\#_h\{h(k)=h(\ell)\} \le \frac{p(p-1)}{m}$$ □

En pratique
→ soit en fait du hachage universel
→ soit on prend une "bonne" fonction de hachage h et on fait l'hypothèse que les éléments à stocker ne sont pas biaisés pour h

Ça marche en pratique : $O(1)$ pour recherche, insertion, suppression constatés en pratique

---

Jusqu'ici, on a parlé de table de hachage avec chaînage.

Inconvénient : la table de hachage ne tient pas dans une zone contiguë de mémoire.

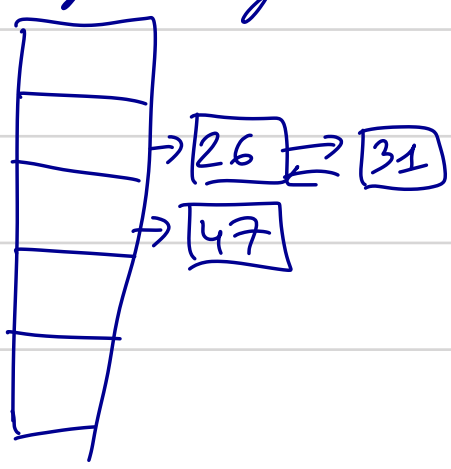
Alternative : hachage ouvert → tous les éléments sont directement dans la table de hachage, sans liste chaînée.

$S = \{26, 31, 47\}$

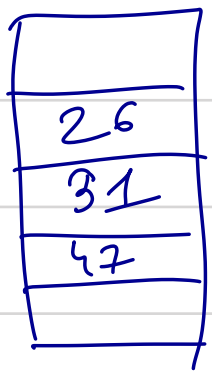
$h(x) = x \bmod 5$

$m = 5$

hachage/chaînage



hachage ouvert (sondage linéaire)



Sondage linéaire → ajoter l'élément ds la prochaine case libre (parcours linéaire)

Suppression : remplacer pa une valeur spéciale indiquant une suppression

Pb: a tendance a faire apparaître de longues zones contiguës pleines → dégradation des performances

Hachage ouvert par double hachage

h_1, h_2 2 fonctions de hachage

$$h(k, i) = h_1(k) + i \times h_2(k) \mod m$$

(on veut $h_2(k)$ premier avec m).

On essaie d'insérer k en $h(k,0)$
si déjà rempli, en $h(k,1)$
en $h(k,2)$
⋮
jusqu'à trouver une case libre.

[Hachage linéaire : $h_2(k) = 1$]

En pratique, très bonnes performances.

Tables de hachage dynamiques (dont la taille n'est pas fixée à l'avance).

hachage ouvert : plus de cases disponible
hachage chaînage $\alpha >$ seuil fixé

* Faire comme les tableaux dynamiques
(réallouer la table de hachage quand elle est pleine avec $m' = 2 \times m$ ou similaire)
On prend une nouvelle fonction de hachage $h' : U \to \{0, \dots, m'-1\}$
et on réinsère les valeurs unes à unes.
Coûteux ($O(n+m')$) mais rare.
Analyse amortie → $O(1)$ amorti pour toutes les opérations

* Faire plus intelligent
→ hachage extensible
→ hachage linéaire