LU2IN006

SDA table de hachage

"Structures de données"

Pierre-Henri Wuillemin
2025fev
pierre-henri.wuillemin@lip6.fr

1. table de symboles

Question : Comment retrouver facilement les données associés à un symbole, une clé ?

$\Longrightarrow$ {
on ne connaît pas le nombre d'éléments a priori,
on doit accéder rapidement aux éléments.
les clés n'ont pas à être ordonnées.

Intérêts ?

- Compilateurs [tables de variables],
- Application complexes [CAO, etc.],
- Algorithmes nécessitant des caches sophistiqués, etc.

Table de symboles

structure abstraite pour effectuer rapidement :

- l'insertion
- la recherche
- la suppression

d'informations dynamiques, identifiées par une clé quelconque.

Problématique des tables de hachage (1/2)

Soit un langage (type ocaml) :

```
let aaa = 3 and aba = 4 and abb = 5 in
let abc = aaa + 2 and abd = aaa + 3 in
let acc = ...
```

- Représentation de la table des variables par une liste linéaire ordonnée ?

$$\texttt{aaa} \longrightarrow \texttt{aba} \longrightarrow \texttt{...} \longrightarrow \texttt{acc}.$$

- Représentation de la table des variables par un table ordonné ?

aaa	aba	...	acc

$\Longrightarrow$ trop de comparaisons

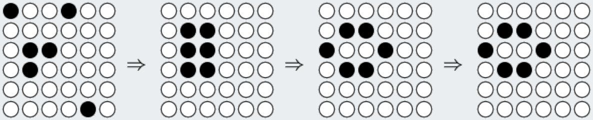
L'ordre est une hypothèse trop forte !

Problématique des tables de hachage (2/2)

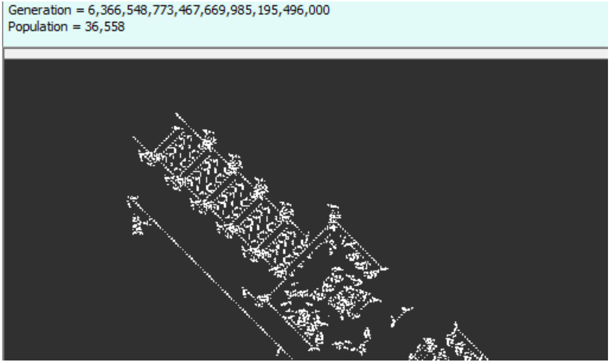
Game of life

- Une cellule a huit voisines.
- L'état d'une cellule à $t+1$ dépend de son état et de celui de ses voisines à l'instant t :

État à t	État à $t+1$
Cellule morte, 3 voisines vivantes	Vivante
Cellule vivante, 2 ou 3 voisines vivantes	Vivante
Sinon	Morte



Golly : hashlife



Utilisation d'un cache élaboré pour éliminer le plus de calculs redondants possibles : portion de plan à $t+1$ indexée par une autre portion de plan à t.

Adressage direct (1/3)

Principe des tables de hachage

- on associe à chaque information à stocker un entier
- on recherche les informations via leur entier associé

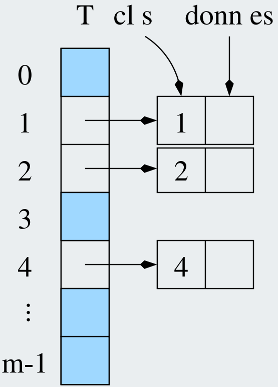
(0,aaa)	(1,aba)		(3,abb)		(5,abc)	(6,abd)	(7,acc)

Adressage direct (2/3)

U : l'univers des clés

Tables à adressage direct

- $U = \{0, 1, \ldots, M - 1\}$
- Toutes les clés ont des entiers associés différents
- un tableau T contient les données



Adressage direct (3/3)

Problèmes des tables à adressage direct :

1. $U \neq \{0, 1, \ldots, M-1\}$ ou, pire, U n'est pas un ensemble d'entiers positifs
2. $|U|$ est très grand $\Longrightarrow$ T trop grand pour être stocké en mémoire
3. plusieurs données ont la même clé

Solutions :

1. créer une fonction `hash` : $U \mapsto \{0, 1, \ldots, m-1\}$
2. si tout l'univers des clés est utilisé (m=M), pas de solution
   sinon :
   soit K l'ensemble des clés réellement utilisées($|K| << |U|$).
   avec $|K|$ du même ordre (même si plus grand) que m
   $\Longrightarrow$ table de hachage
3. *collisions* dans une table de hachage.

Solutions dans les transparents su

Tables de hachage (1/2)

Définition

Table (comme pour l'adressage direct) mais au lieu de stocker l'élément de clé k à l'indice k de la table, on le stocke à l'indice $h(k)$, où h est une *fonction de hachage* : $U \mapsto \{0, 1, \ldots, m-1\}$.
$|U|$ est souvent plus grand que m.

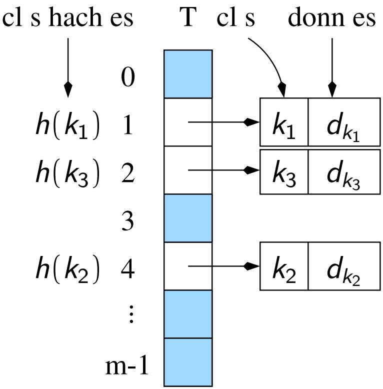
$\Longrightarrow$ répond aux points 1 et 2

Exemples

- Annuaire téléphonique :
  clé = nom des personnes, donnée = numéro de téléphone
- Nom des variables dans un compilateur :
  clé = nom de la variable, donnée = adresse en mémoire
- Gestion d'un ensemble de voitures :
  clé = numéro d'immatriculation, donnée = données sur la voiture

Tables de hachage (2/2)

donnée	clé	clé hachée
d_{k_1}	k_1	$h(k_1) = 1$
d_{k_2}	k_2	$h(k_2) = 4$
d_{k_3}	k_3	$h(k_3) = 2$



Problèmes des tables de hachage

- Comment choisir la fonction h ?
  - le calcul de $h(k)$ doit être rapide
  - h doit éviter au maximum les collisions
- Que faire quand il y a collision ? ($h(k_1) = h(k_2)$ pour $k_1 \neq k_2$)
  - résolution par chaînage
  - résolution par adressage ouvert

problème des collisions

soit $h(k) = 0 \ \forall k \in U \implies$ collision pour tout $k_1 \neq k_2$

soit $h(k) = k \ \forall k \in U \implies$ moins de collisions

$\implies$ le choix de h permet d'éviter des collisions

Propriété : Il est impossible d'éviter totalement les collisions

table de hachage : stocke des données ayant $|K|$ clés possibles dans une table de longueur $m << |K| \implies$ il n'y a pas bijection entre K et $\{0, 1, \ldots, m-1\}$

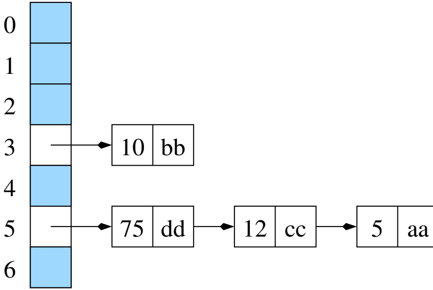
Résolution par chaînage

Principe

Chaque élément de la table est une liste chaînée et tous les éléments ayant la même clé hachée sont dans la même liste chaînée.

Exemple :

- $h(k) = k \ \% \ 7$
- «aa» $\Rightarrow k = 5 \ \Rightarrow h(k) = 5$
- «bb» $\Rightarrow k = 10 \Rightarrow h(k) = 3$
- «cc» $\Rightarrow k = 12 \Rightarrow h(k) = 5$
- «dd» $\Rightarrow k = 75 \Rightarrow h(k) = 5$



$\Longrightarrow$ nécéssité de stocker les clés dans la liste chaînée

Analyse de la résolution par chaînage (1/4)

Hypothèse : le calcul de $h(k)$ s'effectue en O(1)

Analyse de l'insertion d'un élément

- calcul de $h(k) = O(1)$
- insertion dans la liste chaînée appropriée : O(1)

Résultat : insertion en O(1)

Analyse de la suppression d'un élément

si la liste est doublement chaînée : suppression d'un élément en O(1)

Analyse de la résolution par chaînage (2/4)

Supposons que la table t soit de taille m et contienne n éléments

Analyse de la recherche d'un élément

Dans le pire des cas, tous les éléments de la table appartiennent à la même liste chaînée $\Rightarrow$ recherche en $\Theta(n)$.

$\Longrightarrow$ Les tables de hachage sont moins performantes que les listes chaînées dans le pire des cas

facteur de charge

Le facteur de charge d'une table de hachage : $\alpha = n/m$.

Hypothèse (hachage uniforme simple) : chaque liste de la table a la même chance de recevoir un élément tiré au hasard

Analyse de la résolution par chaînage (3/4)

Recherche d'un élément n'appartenant pas à la table

Sous l'hypothèse de hachage uniforme simple, la recherche d'un élément n'appartenant pas à la table est en $\Theta(1+\alpha)$ en moyenne.

Démonstration : Sous l'hypothèse de hachage uniforme simple, $h(k)$ a une chance égale de correspondre à n'importe quelle liste de la table.

Donc le temps moyen pour la recherche d'une clé k est le temps moyen pour arriver à la fin d'une des listes chaînées.

La longueur moyenne d'une liste est α. Donc la recherche s'effectue en 1 (calcul de $h(k)$) $+ \alpha$.

Analyse de la résolution par chaînage (4/4)

Recherche d'un élément appartenant à la table

Sous l'hypothèse de hachage uniforme simple, la recherche d'un élément appartenant à la table est en $\Theta(1+\alpha)$ en moyenne.

Démonstration : Hypothèse de hachage uniforme simple $\Longrightarrow$ la longueur moyenne des listes après insertion de $i-1$ éléments est $(i-1)/m$. Supposons que les éléments sont insérés à la fin des listes. Le nombre moyen d'éléments examinés pendant la recherche du $i$ème élément inséré est 1 + le nombre
d'éléments de la liste avant d'insérer cet élément $= 1 + (i-1)/m$
$\Longrightarrow$ Celui de la recherche d'un elt quelconque est en moyenne :

$$\frac{1}{n}\sum_{i=1}^{n}\left(1+\frac{i-1}{m}\right) = 1 + \frac{\alpha}{2} - \frac{1}{2m}.$$

Donc la recherche est en $\Theta(2+\alpha/2-1/2m) = \Theta(1+\alpha)$.

Conclusions

Conclusions des analyses

Si le nombre de cases du tableau est au moins proportionnel au nombre d'éléments à stocker, $\alpha = n/m = O(m)/m = O(1)$.

- l'ajout d'éléments est en O(1)
- la suppression est en O(1) si les listes sont doublement chaînées
- la recherche d'éléments est en O(1)

Choix des fonctions de hachage (1/2)

Problème : qu'est-ce qu'une bonne fonction de hachage ?

une fonction :

- qui se calcule rapidement (en O(1))
- qui minimise les collisions autant que possible

Minimisation des collisions = hachage uniforme simple

$$\sum_{k:h(k)=i} P(k) = \frac{1}{m} \quad \forall i \in \{0, \ldots, m-1\}$$

P connue $\Longrightarrow$ on peut trouver h qui minimise les collisions

En pratique P est inconnue $\Longrightarrow$ on utilise des heuristiques

Choix des fonctions de hachage (2/2)

Principe des fonctions de hachage usuelles

1. transformer la clé k en un entier via une fonction $f : U \mapsto \mathbb{N}$
2. transformer cet entier en un entier entre 0 et $m-1$ via une fonction $g : \mathbb{N} \mapsto \{0, \ldots, m-1\}$

Autrement dit, $h(k) = g \circ f(k)$.

Exemple : la fonction f `: string -> int` *d'ocaml*

```
#define Beta 19

unsigned long hash_accu = 0;
unsigned int i = strlen(k);
for (char *p = k; i > 0; i--, p++)
   hash_accu = hash_accu * Beta + *p;
```

Hachage par division

Définition

$g(x) = x \ \% \ m$

Choix de m

- éviter les collisions $\Longrightarrow$ utiliser dans g tous les bits de x
  $\Longrightarrow$ éviter les puissances de 2
- ≪bon choix≫ : nombre premier pas trop proche d'une puissance de 2

Hachage par multiplication

Définition

$g(x) = \lfloor m(xA - \lfloor xA \rfloor) \rfloor$, où $A \in]0, 1[$.
Avantage par rapport au hachage par division : choix de m non critique

Choix de A

Knuth propose d'utiliser le nombre d'or (-1) : $\dfrac{\sqrt{5}-1}{2}$

Propriété : pour tout A irrationnel, $g(x), g(x+1), \ldots, g(x+k)$ sont éloignés les uns des autres et $g(x+k+1)$ appartient au plus grand des segments $[g(x+i), g(x+i+1)]$.

adressage ouvert (1/2)

Constat : la résolution par chaînage peut être coûteuse à cause des pointeurs des listes chaînées

Peut-on éviter les pointeurs ?

Oui : en stockant directement les éléments dans la table et non dans des listes chaînées

Avantage : utilise moins de place que la résolution par chaînage $\Longrightarrow$ on peut utiliser des tables plus grandes

Problème : gestion des collisions

adressage ouvert (2/2)

Principe

Pour effectuer une insertion, on scanne la table jusqu'à ce que l'on trouve une case vide dans laquelle insérer l'élément. La séquence de cases scannées dépend de la clé de l'élément à insérer.

fonction de hachage

$$h : U \times \{0, \ldots, m-1\} \mapsto \{0, \ldots, m-1\}$$

Séquence de scans : $\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$

$\Longrightarrow \langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ doit être une permutation de $\{0, \ldots, m-1\}$

Exemple d'insertion

$$h(k,i) = \left(\left\lfloor 8 \left(\frac{\pi k}{4} - \left\lfloor \frac{\pi k}{4} \right\rfloor \right) \right\rfloor + \frac{i}{2} + \frac{i^2}{2} \right) \ \% \ 8$$

0	8	dd
1		
2	3	aa
3	12	cc
4		
5	11	bb
6		
7		

$x = \text{“dd”},\ k = 8$

$h(k,3) = 0$

Autres opérations

- Recherche d'un élément,
- Suppression d'un élément.

⚠ La suppression d'élément brise la chaîne d'éléments. Il est nécessaire d'avoir une représentation de 'élément supprimé'.

Malheureusement les analyses de complexité ne dépendent plus seulement du facteur de charge α

$\Longrightarrow$ en principe, on utilise plutôt la résolution par chaînage

Analyse de l'adressage ouvert (1/2)

Hypothèse (hachage uniforme) : chaque clé a une chance égale d'avoir n'importe laquelle des $m!$ permutations de $\{0, \ldots, m-1\}$ comme séquence de scans

En pratique, cette hypothèse n'est jamais vérifiée : on n'en a que des approximations

Analyse d'une recherche infructueuse

Soit une table de hachage dans laquelle aucune suppression n'est permise. Soit $\alpha = n/m < 1$ le facteur de charge de la table. Alors l'espérance du nombre de scans lors de la recherche infructueuse d'un élément est au plus de $1/(1-\alpha)$.

α	50 %	80 %	90 %	99 %
scans	2	5	10	100

Analyse de l'adressage ouvert (2/2)

Analyse d'une recherche couronnée de succès

Si $\alpha < 1$ alors l'espérance du nombre de scans pour trouver l'élément recherché est au plus de :

$$\frac{1}{\alpha} \ln \frac{1}{1-\alpha} + \frac{1}{\alpha}.$$

α	50 %	80 %	90 %	99 %
scans	3,396	3,26	3,67	5,66

Analyse d'une insertion

Si $\alpha < 1$ alors l'espérance du nombre de scans nécessaires à l'insertion d'un élément est au plus de $1/(1-\alpha)$.

fonctions de hachage (1/2)

Probing linéaire

$h(k,i) = (h'(k) + i) \ \% \ m$, où $h' : U \mapsto \{0,\ldots,m-1\}$.

Problème : formation de clusters de cases remplies

$\Longrightarrow$ les scans peuvent être longs

Probing quadratique

$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \ \% \ m$, où $h' : U \mapsto \{0,\ldots,m-1\}$.

Problème : 2 clés k_1, k_2 telles que $h'(k_1) = h'(k_2)$ auront la meme séquence de scans

fonctions de hachage (2/2)

double hachage

$h(k, i) = (h_1(k) + ih_2(k)) \ \% \ m$, où h_1 et h_2 sont des fonctions de hachage de $U \mapsto \{0, \ldots, m-1\}$.

Propriété : le double hachage approche l'hypothèse de hachage uniforme.

Le double hachage est plus performant que les 2 autres fonctions de hachage

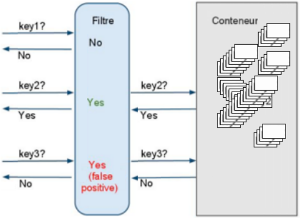
4. Filtre de Bloom : tester si un élément n'est pas dans un ensemble.

Un filtre de Bloom permet de savoir :

- avec certitude que l'élément est absent de l'ensemble

  pas de faux négatif
- avec une certaine probabilité que l'élément peut être présent

  faux positifs possible



but : éviter des accès inutiles à des ressources *lentes*.

- BigTable (base de données distribuées – Google)
- inspection de paquets en profondeur.
- Chrome évite des milliers d'appels réseau en stockant une blacklist.
- Correcteurs orthographiques.

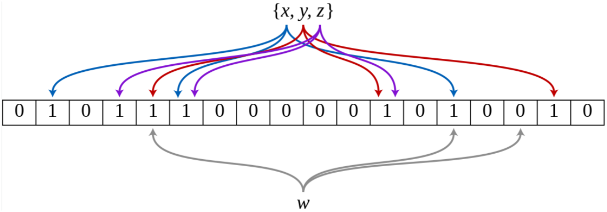
Principe du filtre de Bloom

Filtre de Bloom pour un sous-ensemble de U

- $T[0 \ldots m-1]$ tableau de booléens initialisé à 0
(très compact si le langage le permet)
- $\{h_i, 0 \leq i < k\}$ k fonctions de hachage de $U \mapsto \{0, 1, \ldots, m-1\}$

Opérations :

- $\texttt{Insertion}(e \in U)$: $\forall i < k$, $T[h_i(e)] \leftarrow 1$
- $\texttt{EstPossiblementPrésent}(e \in U) \iff \bigwedge_{i<k} (T[H_i(e)])$



vressort

un filtre de Bloom est une structure légère.

$log_2(m)$ octets en mémoire.

filtre de Bloom : probabilité de faux-positifs

Soit un filtre de Bloom de taille m, avec k fonctions de hachage, contenant déjà les éléments $\{e_j\}_{j<n}$.

En supposant les $\{h_i\}_{i<k}$ des fonctions de hachage uniforme

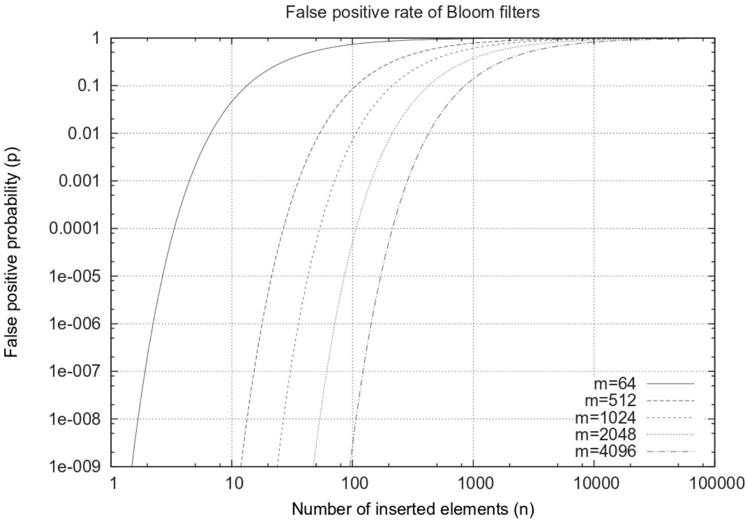
- Proba que $h_i(e)$ mette le bit b à 1 :
- Proba que $h_i(e)$ laisse b à 0 :
- Proba que $\{h_i(e)\}_{i<k}$ laisse b à 0 :
- Proba que $\{h_i(e_j)\}_{i<k,\ j<n}$ laisse b à 0 :
- Proba que $\{h_i(e_j)\}_{i<k,\ j<n}$ mette b à 1 :

Proportion de faux-positifs

Proba. que les k bits représentant un élément x soit à 1 à cause des n clés déjà dans le filtre de Bloom :

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$$

Filtre de Bloom : taux de faux-positifs

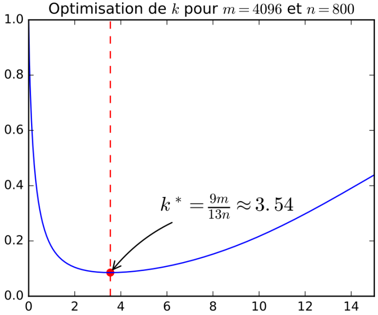


hressort

Filtre de Bloom : Nombre de fonction de hachage

$$FP(n, m, k) = \left(1 - e^{-\frac{kn}{m}}\right)^k$$

peut être optimisé pour k (en annulant la dérivée).



Optimisation de k

$$k^* = \ln 2 \cdot \frac{m}{n} \approx \frac{9m}{13n}$$

$$FP(n, m, k^*) \approx 0.6185^{\frac{m}{n}}$$

Pour garder un taux constant de faux positif, il faut faire grandir linéairement le filtre de Bloom.

Bonus : Hachage universel (1/3)

Retour sur le hachage : *En fonction de la taille et de la fonction, le comportement de la SDA peut être très mauvais*.

But : éviter qu'un utilisateur mal intentioné ne choisisse que des clés ayant la même valeur hachée

Principe : choisir la fonction g au hasard indépendamment des clés lors de chaque exécution du programme

$\Longrightarrow$ La vitesse change à chaque exécution
mais en moyenne, quelque soit l'ordre d'insertion, accès en O(1).

Hachage universel (2/3)

Ensemble universel

Soit $\mathcal{H}$ un ensemble fini de fonctions de hachage de $U \mapsto \{1, \ldots, m-1\}$.
$\mathcal{H}$ est universel si $\forall\ k_1, k_2 \in U$ tels que $k_1 \neq k_2$,
$|\{h \in \mathcal{H} : h(k_1) = h(k_2)\}| = |\mathcal{H}|/m$

$\Longrightarrow$ pour une fonction $h \in \mathcal{H}$ donnée, la probabilité de collision entre 2 clés est de $1/m$

Théorème

Si h est choisie dans un ensemble universel et est utilisée pour hacher n clés dans une table de taille m, avec $n \leq m$, alors l'espérance du nombre de collisions avec une clé k donnée est inférieure strictement à 1.

Hachage universel (3/3)

Choix d'un ensemble universel

On suppose que m est un nombre premier

On définit un mot comme tout ensemble de nombres entiers de 0 à p, avec $p < m$

On décompose une clé $k \in U$ en une séquence de $r+1$ mots : $k = \langle k_0, k_1, \ldots, k_r \rangle$

On note $a = \langle a_0, a_1, \ldots, a_r \rangle$ une séquence où les a_i sont choisis au hasard dans $\{0, \ldots, m-1\}$

On définit $\mathcal{H} = \bigcup_a \{h_a\}$ avec

$$h_a(k) = \sum_{i=0}^{r} a_i k_i \ \% \ m.$$

Alors $\mathcal{H}$ est un ensemble universel.