

M02 : Table de hachage, dictionnaire et un peu de probabilité

Michel Van Caneghem

Novembre 2004

Le problème

Une compagnie de téléphone veut fournir un service d'identifiant de l'appelant. C'est à dire à partir du numéro de téléphone retrouver le nom de l'appelant. Le nombre de numéros de téléphone est $R = 10^{10} - 1$ et il y a n abonnés. Comment faire cela de la manière la plus efficace possible.

- ✓ **Un arbre balancé (AVL, red-black) avec le numéro de téléphone comme clé : $O(n)$ espace mémoire et $O(\log n)$ accès.**
- ✓ **Un tableau avec R entrées : temps d'accès optimal $O(1)$ par contre utilisation d'une mémoire gigantesque $O(R)$.**

La solution

Une table de hachage ("hash-code") est la solution avec un temps d'accès de $O(1)$ et une utilisation de $O(n + N)$ en mémoire, ou N est la taille de la table.

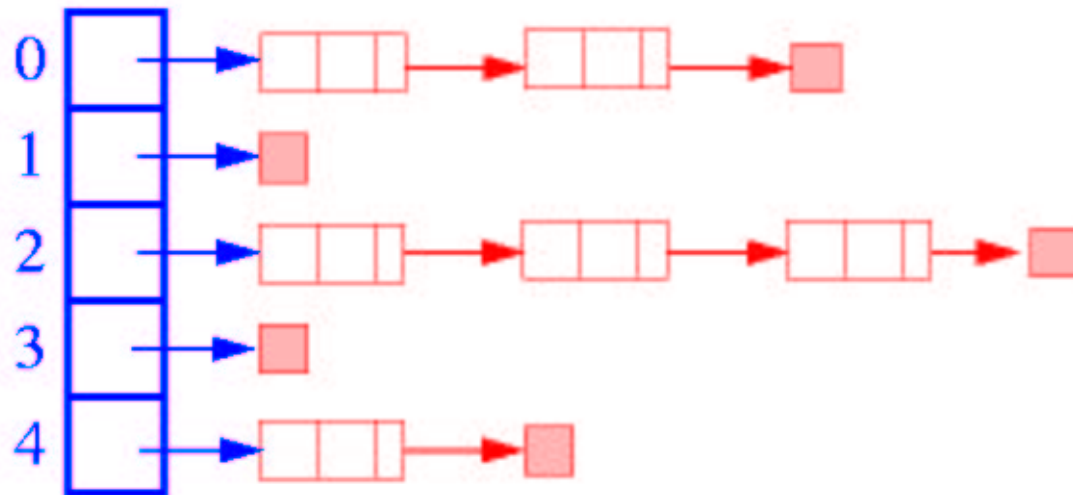
La fonction de hachage transforme l'espace des clés en un espace plus petit. Supposons que l'on prenne le numéro de téléphone modulo 5. Si Michel a comme numéro de téléphone 04 91 26 00 03, alors il sera rangé dans la case 3.

0	1	2	3	4
		Gilles	Michel	

Gilles (04 91 26 00 02) dans la case 2, mais ou ranger Julien (04 91 26 00 63) ?

Comment résoudre les conflits ?

Il suffit de construire une liste chaînée qui contient toutes les entrées ayant la même valeur de hachage.



Le temps d'accès moyen sera $O(n/N)$ et donc réglable en fonction de N .

Comment construire une bonne fonction de hachage ?

Dans ce qui suit nous allons supposer que la taille de la table est N et que le nombre d'entrées est M . Il faut alors que chaque entrée soit répartie le plus uniformément possible.

Cela dépend de la nature de la clé. Si la clé est numérique alors.

✌ Si c'est un entier k alors on choisit $h(k) = k \pmod{N}$.

Pourquoi a-t-on intérêt à choisir N premier ?

✌ Si k est un nombre flottant $0 \leq k < 1$ alors on choisit $h(k) = \lfloor N * k \rfloor$.

Une bonne fonction de hachage (2)

Si k est une chaîne $k = c_0c_1 \dots c_{n-1}$ (c_i code numérique du i ème caractère de k). Il suffit de choisir un nombre a et de calculer avec le schéma de Horner :

$$h(k) = c_0 + ac_1 + a^2c_2 + \dots + a^{n-1}c_{n-1} \pmod{N}$$

```
k = 0;
for (int i = n - 1; i >= 0; i--)
    k = k*a + c[i] % M;
```

Java a choisit $a = 31$, vous pouvez prendre pour a n'importe quelle valeur, sauf une puissance de 2, si le codage d'un caractère est basée sur une puissance de 2.

Une bonne fonction de hachage (3)

Mieux ? (a vous de me le prouver dans le devoir) : fonction de hachage universelle.

```
a = 31415; b = 27183;
k = 0;
for (int i = 0; i < n; i++) {
    k = k*a + c[i] % N;
    a = a*b % (N - 1);
}
```

On peut espérer que la probabilité de conflit entre deux chaînes est $1/N$, si les clés sont réparties uniformément. *Qu'est-ce que l'on peut en déduire ?*

Le problème du remplissage

En anglais : "Occupancy problem". On veut placer au hasard M balles dans N boites. On cherche à savoir comment les différentes boites vont être remplies. De manière plus précise : soit X_i la variable aléatoire qui compte le nombre de balles dans la boite i . On souhaite connaître :

- ✦ La moyenne et la variance de cette loi
- ✦ La probabilité qu'il y ait k balles dans la boite i :
 $Pr[X_1 = k]$ (y compris pour $k = 0$)
- ✦ La probabilité pour qu'une boite contienne plus de k balles : $Pr[X_1 > k]$
- ✦ Le nombre moyen de tirages pour avoir une boite avec deux balles.
- ✦ Le nombre moyen de tirages pour remplir toutes les boites "coupon collector"

Moyenne et variance

Comme toutes les boites sont équivalentes, on peut raisonner sur la boite 1. Grace à la linéarité des espérances, on a :

$$E\left[\sum_{i=1}^N X_i\right] = M = \sum_{i=1}^N E[X_i] = N E[X_1] \text{ donc } E[X_1] = M/N = \alpha$$

Ceci n'est pas suffisant pour connaitre la répartition, car toutes les balles pourraient être dans la même boite. Il faut calculer la variance : $\sigma^2 = E[(X_1 - E[X_1])^2] = E[X_1^2] - E[X_1]^2$.

Pour cela il faut connaitre la loi de probabilité. Cependant dans ce cas on peut s'en passer, en remarquant que : X_1 est la somme de M variables aléatoires x_i indépendantes : x_i vaut 1 si la boite 1 a été choisie au tirage i et 0 sinon.

Moyenne et variance (2)

Pour cette loi on a : $P[x_i = 1] = 1/N$ et $P[x_i = 0] = 1 - 1/N$

On peut alors calculer la moyenne et la variance :

$$E[x_i] = 1 \times P[x_i = 1] + 0 \times P[x_i = 0] = 1/N$$

$$\sigma_i^2 = E[(x_i - 1/N)^2] = (1 - 1/N)^2 P[x_i = 1] + (1/N)^2 P[x_i = 0]$$

$$= \frac{(1 - 1/N)^2}{N} + \frac{(1 - 1/N)}{N^2} \simeq \frac{1}{N}$$

$$E[X_1] = E\left[\sum_{i=1}^M x_i\right] = \sum_{i=1}^M E[x_i] = M/N = \alpha$$

$$\sigma^2 = \sum_{i=1}^M \sigma_i^2 = M/N = \alpha$$

Car les variables x_i sont indépendantes.

La loi de probabilité

Cherchons maintenant la répartition dans chaque boîte. Pour cela nous allons calculer quelle est la probabilité que la boîte 1 contienne exactement k boules.

Si on considère la boîte 1, pour chaque tirage d'une boule, on a une probabilité de $\frac{1}{N}$ que la boule soit dans la boîte 1 et $1 - 1/N$ que la boule soit ailleurs que dans la boîte 1. Il s'agit d'un schéma de Bernoulli (loi binomiale).

$$Pr[X_1 = k] = C_M^k \left(\frac{1}{N}\right)^k \left(1 - \frac{1}{N}\right)^{M-k}$$

On pose toujours $\alpha = M/N$.

La loi de probabilité (2)

Si $k = 0$ alors : $Pr[X_1 = 0] = \left(1 - \frac{1}{N}\right)^M$ **si M et N sont grand**
alors : $\left(1 - \frac{1}{N}\right)^M = \left(\left(1 - \frac{1}{N}\right)^N\right)^{M/N} = e^{-\alpha}$. **Donc premier résultat :**

$$Pr[X_1 = 0] = e^{-\alpha}$$

De manière générale on trouve, si k est petit et M et N grand :

$$Pr[X_1 = k] = \frac{\alpha^k}{k!} e^{-\alpha}$$

Ce qui n'est rien d'autre qu'une loi de Poisson. (Problème des files d'attente, ...)

Vérification expérimentale

Il s'agit de voir comment les clé se répartissent dans les différentes entrées de la table. On a $N = 342538$. J'ai choisit $M = 600000$. On a donc $\alpha = 0.57$. Voici les résultats expérimentaux et théoriques :

k	nb de boites mesurée	nb de boite théorique
0	338982	339011
1	193485	193540
2	55447	55246
3	10400	10513
4	1492	1500
5	175	171
6	17	16
7	2	1

L'occupation maximale d'une boîte

On peut se demander quel est le nombre maximal de balles que l'on peut trouver dans une boîte. Ce n'est pas la bonne question !! Il faut se fixer une probabilité à priori, par exemple $p_0 = \frac{1}{1000}$. Et se demander quel est le plus petit k tel que $Pr[X_1 > k] < p_0$:

$$Pr[X_1 > k] = \sum_{i=k+1}^M C_M^i \left(\frac{1}{N}\right)^i \left(1 - \frac{1}{N}\right)^{M-i}$$

On va utiliser la loi de Poisson. Remarquons tout d'abord que (formule de Taylor) :

$$e^\alpha \leq 1 + \frac{\alpha}{1!} + \dots + \frac{\alpha^k}{k!} + \frac{\alpha^{k+1}}{(k+1)!} \times e^\alpha$$

L'occupation maximale d'une boîte (2)

$$Pr[X_1 > k] = 1 - e^{-\alpha} \sum_{i=0}^k \frac{\alpha^i}{i!} < 1 + e^{-\alpha} \left(\frac{\alpha^{k+1}}{(k+1)!} e^{\alpha} - e^{\alpha} \right) = \frac{\alpha^{k+1}}{(k+1)!}$$

On va alors utiliser la formule de Stirling pour calculer la factorielle : $k! = \sqrt{2\pi k} \left(\frac{k}{e}\right)^k$

$$Pr[X_1 > k] = \sqrt{2\pi(k+1)} \times \left(\frac{e\alpha}{k+1}\right)^{k+1} < \left(\frac{e\alpha}{k}\right)^k$$

$$Pr[X_1 > t\alpha] = \left(\frac{e}{t}\right)^{t\alpha}$$

L'occupation maximale d'une boîte (3)

Prenons deux exemples : $\alpha = 0.5$ et $\alpha = 2$.

t	$\alpha = 0.5$ $Pr[X > 0.5t]$	$\alpha = 2$ $Pr[X > 2t]$
3	0.86	0.55
4	0.46	0.045
5	0.22	0.002
8	0.01	$< 10^{-6}$
16	$< 10^{-6}$	

On en déduit que si $\alpha = 0.5$ alors on a une probabilité de $\frac{1}{1000000}$ d'avoir une liste de longueur supérieure à 8, et si $\alpha = 2$, une liste supérieure à 16.

Le paradoxe de l'anniversaire

Dans un amphi de n étudiants quel est la probabilité d'avoir 2 personnes nées le même jour. Ou à partir de quel nombre d'étudiants est-ce intéressant de parier qu'il y a deux personnes nées le même jour ?

Nous allons faire l'essai.....

Le paradoxe de l'anniversaire (2)

Nous avons $N = 365$ boîtes et on peut supposer que chaque étudiant représente un tirage d'un nombre aléatoire entre 1 et 365. Quel est la probabilité qu'il y ait une boîte contenant 2 étudiants au bout de n tirages ? Cherchons plutôt la probabilité contraire \bar{P} .

$$\begin{aligned}\bar{P} &= 1 \times \left(1 - \frac{1}{N}\right) \times \left(1 - \frac{2}{N}\right) \times \cdots \times \left(1 - \frac{n-1}{N}\right) \\ &= \prod_{i=1}^{n-1} \left(1 - \frac{i}{N}\right) < \prod_{i=1}^{n-1} e^{-\frac{i}{N}} \\ &\leq e^{-\frac{1}{N} \sum_{i=1}^{n-1} i} \leq e^{-\frac{n(n-1)}{2N}}\end{aligned}$$

Le paradoxe de l'anniversaire (3)

$$P(n) = 1 - e^{-(n-1)n/(2N)}$$

Si $n = 90$ et $N = 365$, Alors la probabilité que deux étudiants soient nés le même jour est de : 0,9999828. Cela vaut le coup de parier !!!

Si $n = 24$ et $N = 365$ alors $p = 0,53$.

Au fait est-ce problème que nous avons résolu dans l'amphi ?

Je prends des risque : je prédis qu'au bout de $q = \frac{1}{1-e^{-\alpha}}$ essais il y aura 2 étudiants ayant la même date de naissance. ($\alpha = 90/365 = 0.25$ d'ou $q \simeq 5$)

"Coupon Collector"

On a toujours N boîtes, combien faut-il tirer de boules aléatoirement pour que dans chaque boîte il y ait au moins une boule. On trouve ce problème également dans des livres ou on achète des images à coller : combien faut-il acheter d'images pour remplir le livre ? [250 => 1525]

On note t_1, t_2, \dots, t_x les tirages ou on remplit une boîte qui était vide auparavant. On pose $X_i = t_{i+1} - t_i$. Les X_i satisfont une distribution géométrique de probabilité : $(N - i)/N$ d'où $E(X_i) = N/(N - i)$. On a donc :

$$\begin{aligned} X &= \sum_{i=0}^{N-1} X_i & E(X) &= \sum_{i=0}^{N-1} E(X_i) = \sum_{i=0}^{N-1} \frac{N}{N-i} \\ &= N \sum_{i=0}^{N-1} \frac{1}{N-i} = N \sum_{i=1}^N \frac{1}{i} = N \times H_N = N \log N \end{aligned}$$

Méthode A : Liste chaînées

Toutes les clés ayant le même hachage (conflit) sont chaînées entre-elles. Cherchons le nombre moyen d'éléments à comparer en cas de succès (le mot est dans le dictionnaire) et en cas d'échec.

En cas de succès : il y a M mots dans le dictionnaire et N entrées. Pour une boîte si p_i est la probabilité d'avoir exactement i éléments dans la boîte i . La longueur moyenne d'une liste $E(S)$ est donné par :

$$E(S) = 1 \times p_1 + 2 \times p_2 + \dots + M \times p_M$$

$$E(X) = \alpha = 0 \times p_0 + 1 \times p_1 + 2 \times p_2 + \dots + M \times p_M$$

$$E(S) = \alpha$$

Le nombre moyen de comparaisons est donc $1 + \alpha/2$

Méthode A : Liste chaînées (2)

En cas d'échec, c'est plus simple : soit le hachage n'est pas dans la table, soit il faut parcourir toute la liste chaînée jusqu'au bout. On a vu que la longueur moyenne d'une liste était de α donc le nombre moyen de comparaison est α

Occupation mémoire : N mots pour la table + M mots pour les clés + M mots pour les liens. Ce qui donne en tout :

$$N + M + M = N + 2N\alpha = N(1 + 2\alpha)$$

Exemple si $M = 300000$, $N = 100000$, $\alpha = 3$: occupation mémoire = 700000 et nombre moyen de comparaisons par consultation : 2,5

Méthode A : Liste chaînées (3)

Voici un test avec le dictionnaire du français et le texte de Proust du Devoir1

M = 342538, nbMots = 79019, inDico = 77784,
outDico = 1235, nbEchecs = 4756

N	alpha	lgSu	lgEc
100000	3.425	2.671	3.797
200000	1.713	1.822	1.904
300000	1.142	1.625	1.33
400000	0.856	1.473	0.989
500000	0.685	1.241	0.667
600000	0.571	1.342	0.635
700000	0.489	1.273	0.567
800000	0.428	1.214	0.453

Méthode B : Linear Probing

Dans ce cas on met les entrées directement dans la table. Quand il y a un conflit, alors on parcourt la table circulairement et on met la nouvelle entrée dans la première case libre. Quand on recherche un élément on fait la même chose, si on tombe sur une case vide, alors l'élément n'est pas dans la table.

```
public boolean containsKey(String s) {
    int i = hash(s);
    while (table[i] != null) {
        if (s.equals(table[i])) return true;
        i = (i + 1) % table.length;
    }
    return false;
}
```

Méthode B : Linear Probing (2)

Il se crée des "clusters" et l'analyse est beaucoup plus complexe. Il faudrait connaître la répartition statistique des longueurs des cluster.

alpha = 0.57, N = 600 000, M = 342538

Nb de clusters = 104311, lg moy. d'un cluster = 3.284,

carré moyen de la taille d'un cluster = 4.706

taille	Nombre	taille	Nombre
1	43258	8	1901
2	20741	9	1485
3	11989	10	1041
4	7328	
5	5026	73	3
6	3732	74	0
7	2608	75	1

Méthode B : Linear Probing (3)

Je ne sais pas démontrer ces formules. Mais voici les nombres de comparaisons moyens (dans ce cas $\alpha < 1$) :

En cas de succès :

$$\frac{1}{2} \left(1 + \frac{1}{1 - \alpha} \right)$$

En cas d'échec :

$$\frac{1}{2} \left(1 + \frac{1}{(1 - \alpha)^2} \right)$$

Occupation mémoire : N et c'est tout !!

Méthode B : Linear Probing (4)

Un essai :

M = 342538, nbMots = 79019, inDico = 77784, outDico = 1235

N	alpha	lgSu	lgSuT	lgEc	lgEcT	nbCmpT
400000	0.856	4.684	3.972	26.761	24.613	491618
500000	0.685	2.35	2.087	4.188	5.539	202704
600000	0.571	1.834	1.666	2.24	3.217	153287
700000	0.489	1.594	1.478	1.323	2.415	130296
800000	0.428	1.33	1.374	0.945	2.028	107937
900000	0.381	1.326	1.308	1.109	1.805	108411
1000000	0.343	1.259	1.261	0.595	1.658	100729
1100000	0.311	1.247	1.226	0.867	1.553	101141
1200000	0.285	1.2	1.199	0.453	1.478	95482

Méthode C : Double Hashing

Comme la méthode précédente sauf que quand il y a un conflit, on ne cherche pas la prochaine case libre, mais on fait un second hachage et on cherche ainsi la prochaine case libre. Le but est de diminuer la taille des clusters.

```
public boolean containsKey(String s) {
    int i = hash(s);
    while (table[i] != null) {
        if (s.equals(table[i])) return true;
        i = (i + hash2(s)) % table.length;
    }
    return false;
}
```

Méthode C : Double Hashing (2)

On essayera de montrer en TD comment trouver ces formules :

En cas de succès :

$$\frac{1}{\alpha} \ln \left(\frac{1}{1 - \alpha} \right)$$

En cas d'échec :

$$\frac{1}{1 - \alpha}$$

Occupation mémoire : N et c'est tout !!

Méthode C : Double Hashing (3)

Mes résultats

M = 342538, nbMots = 79019, inDico = 77784, outDico = 1235

N	alpha	lgSu	lgSuT	lgEc	lgEcT	nbCmpT
400000	0.856	2.329	2.264	8.238	6.944	220311
500000	0.685	1.766	1.686	2.177	3.175	147722
600000	0.571	1.537	1.482	1.574	2.331	127064
700000	0.489	1.356	1.373	1.083	1.957	110607
800000	0.428	1.244	1.305	0.816	1.748	100682
900000	0.381	1.203	1.259	0.712	1.616	96994
1000000	0.343	1.207	1.225	0.569	1.522	96583
1100000	0.311	1.183	1.198	0.525	1.451	94527
1200000	0.285	1.144	1.177	0.397	1.399	90903

Qui est le meilleur ?

Si on veut faire des comparaisons équitables, il faut se mettre dans les mêmes conditions au niveau de l'occupation mémoire. Par exemple si on choisit dans le cas du chaînage d'avoir une mémoire totale de 900 000 mots on trouve alors ($M = 342538 \Rightarrow N = 214924 \Rightarrow \alpha = 1.59$)

Méthode	α	IgSu	IgEch
SeparateChaining	1.59	1.708	1.592
LinearProbing	0.381	1.326	1.109
DoubleHashing	0.381	1.203	0.712

Donc double hashing est la meilleure méthode sauf si le temps du double hachage est trop long !!

Le Devoir 1

- ① Etude de la fonction de hachage.
- ② Mesure des performances en utilisant les listes chaînées pour résoudre les conflits.
- ③ Mesures des performances en utilisant le "linear probing".
- ④ Mesures des performances en utilisant le "double hashing".
- ⑤ Concluez, en indiquant quel est selon vous la meilleure méthode ?
- ⑥ Comment pourrait-on faire quand un mot n'est pas dans le dictionnaire pour suggérer une orthographe ?

Document écrit (au plus 10 pages), A rendre le 22 Novembre 2004