IFT2015 :: hiver 2020

Cours de structures de données, Université de Montréal

MARS 10, 2020MARS 10, 2020 *by* CSUROSM

Tableau de hachage: notes de cours et exercices

- EXERCICES, NOTES DE COURS
- LAISSER UN COMMENTAIRE

Tableau de hachage: 10note-hachage.pdf (https://ift2015h20code.files.wordpress.com/2020/03/ift2015h20-10note-hachage.pdf) (notes de cours)

Exercices

h.0 Une mauvaise fonction de hachage

Pourquoi $h(x) = (x^2 + 1) \bmod 11$ n'est pas une bonne fonction de hachage pour un tableau de taille 11?

Indice. Examinez si le critère d'uniformité est satisfait. *Après avoir trouvé la réponse, un article Wikipédia intéressant sur le sujet et le résidu quadratique (https://fr.wikipedia.org/wiki/Résidu_quadratique).*

h.1 Éléments distincts

On veut compter le nombre d'éléments distincts dans un tableau A de nombres n nombres entiers. On n'a pas le droit de changer l'ordre dans le tableau.

- **a. Hachage.** Donner un algorithme à temps linéaire au moyen, à l'aide de mémoire de travail pour 2n+O(1) nombres entiers.
- **b. Tri.** Donner un algorithme à temps $O(n \log n)$ au pire, à l'aide de mémoire de travail pour n+O(1) nombres entiers.

h.2 L'ordre n'a pas d'importance

On se demande s'il est possible de réarranger les éléments dans le tableau de hachage (on reste avec la même fonction de hachage) avec sondage linéaire afin d'améliorer l'efficacité de la recherche. Démontrer que cela ne vaut pas la peine: la recherche fructueuse a le même coût (coût = nombre de cases examinées dans le tableau) moyen après l'insertion des éléments $\{x_1, x_2, \ldots, x_n\}$ dans n'importe quel ordre.

Indice. Considérez comment on change le coût de la recherche si on insère dans l'ordre $x_1, x_2, \ldots, x_{i+1}, x_i, \ldots, x_n$ au lieu de $x_1, x_2, \ldots, x_i, x_{i+1}, \ldots, x_n$, et argumentez qu'on peut produire toute permutation par échanges de voisins.

h.3 Recherche fructueuse

(https://ift2015h19.files.wordpress.com/2019/03/pooh-northpole.jpg)

Supposons qu'on quête un tableau de hachage T[0..M-1] rempli en employant sondage linéaire et une fonction de hachage h:

```
1  Value search(Key x) { // recherche de clé x
2      int i= h(x); // indice de case 0<=i<M
3      while (T[i]!=null && !T[i].key.equals(x)) { i = (i+1) % M;}
4      if (T[i]==null) return null; // échec: clé ne se trouve pas
5      else return T[i].value; // succès
6  }
```

Dans cet exercice, on veut étudier le *coût moyen de recherche fructueuse*, ou le nombre de cases examinées(`T[i]` dans la boucle `while` en Ligne 3) en moyenne quand on cherche une clé existante. Dans d'autres mots, si c(i) dénote le nombre d'itérations quand on cherche la clé qui a été placée dans cellule i, on considère $\bar{c} = \frac{1}{n}\sum_{i:\, T[i]\neq \text{null}} c(i)$ comme le coût moyen (n est le nombre des éléments).

	0	1	2	3	4	5	6	7	8	9	10
T	L	G	R	H	I	O	M	T			A
c	2	2	3	2	1	1	6	1			1

SOUS CHAQUE CASE AVEC CLÉ X, C(X) EST LE NOMBRE DE CELLULES EXAMINÉES DANS LA RECHERCHE POUR X.

Exemple: supposons qu'on a M=10, h(`A`)=10, h(`L`)=10, h(`G`)=0, h(`O`)=5, h(`R`)=0, h(I)=4, h(T)=7, h(H)=2, h(M)=1, et on insère `A`,`L`,`G`,`O`,`R`,`I`,`T`,`H`,`M` dans cet ordre, résultant dans le tableau illustré ici. Alors $\bar{c} = (2+2+3+2+1+1+6+1+1)/9 = 2\frac{1}{9}$.

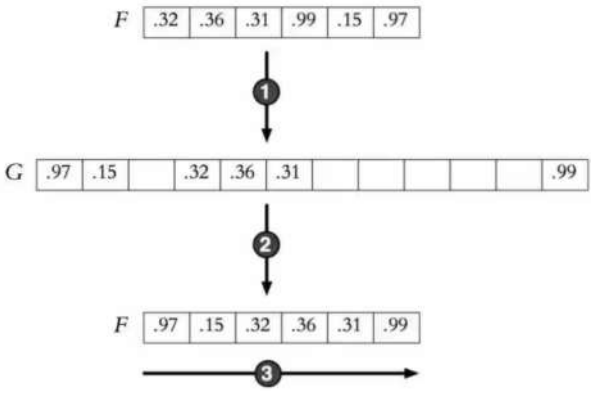
a. Démarrage à froid. Donner un algorithme qui calcule $\bar{c}$ sur un tableau T de capacité M fourni à l'entrée. (L'algorithme connaît la fonction de hachage h, et il n'a pas le droit de changer T.) L'algorithme doit prendre un temps linéaire en M, et utiliser juste O(1) espace de travail.

b. Entretien. On veut maintenant récuperer le coût moyen à tout temps pendant la vie du tableau de hachage. Expliquer comment amender la structure, et modifier les opérations d'insertion et de suppression pour supporter le calcul de $\bar{c}$ en O(1) à tout temps. Le temps d'exécution des autres opérations ne peut être affecté que par une facteur constante. Vous avez le droit d'introduire O(1) nouvelles variables.

Indice. Introduire des variables auxiliaires comme n, $\bar{c}$ ou le compte de cellules vides, et montrer comment les mettre à jour lors d'une insertion ou suppression.

h.4 Tri par hachage

Supposons qu'on veut trier un tableau $F[0..n-1]$ de nombres flottants $0 \leq F[i] < 1$. Considérer la démarche suivante:



- **(0)** Créer un tableau $G[0..2n-1]$, avec cellules vides au début (en Java, on peut mettre `G[i]=Double.NaN`).
- **(1)** Utiliser G comme un tableau de hachage avec la fonction de hachage $h(x) = \lfloor 2n \cdot x \rfloor \in \{0, 1, \ldots, 2n-1\}$: insérer les $F[i]$ un-à-un avec sondage linéaire.
- **(2)** Copier le contenu des cases occupées de G à F dans le même ordre, et
- **(3)** trier F par insertion.

a. Donner le code pour un tel algorithme avec tous les détails (incluant tri par insertion et

insertion dans le tableau de hachage; on peut même combiner (2)+(3) dans une seule boucle).
b. Démontrer que l'algorithme de **a** prend un temps linéaire à la moyenne (assumer que les $F[i]$ sont uniformement distribués) et quadratique au pire.

Indice. Comparez la boucle interne du tri par insertion dans F et la recherche dans G qui est rempli à une facteur de $\alpha = 1/2$. Pour b, comparez le déroulement du tri par insertion dans F avec la recherche dans le tableau G.

h.5 Sondage dubieux 🏆

Considérer un tableau de hachage H[0..m-1] initialement vide ($\forall i: H[i] = \text{null}$), qu'on remplit en utilisant l'adressage ouvert avec la fonction de hachage h (connue). Après n insertions ($0 \leq n<m$), H contient n éléments non-nulls. Un tableau est *valide* si les éléments ont été insérés par le code suivant:

```
1  insert(x)
2  {
3     j ← h(x);
4     for (k ← j+1,... ,m-1) do
5        if H[k]=null then H[k] ← x; return
6     for k\setto 0, 1,...,j-1 do
7        if H[k]=null then H[k] ← x; return
8     erreur de débordement: trop d'éléments dans le tableau
```

Donner un algorithme `valid`(H[0..m-1])
qui vérifie si le tableau est valide.

Indice. Il faut parcourir les indices pour vérifier si h(H[i]) est possible pour tout i avec H[i] ≠ `null`. Faites attention aux cas où H[m-1] ≠ `null`.

Publicités