

EA4 – Éléments d’algorithmique

TD n° 9 : tables de hachage

Exercice 1 : échauffement

On considère une table de hachage T à adressage fermé avec résolution des collisions par chaînage, de longueur $m = 11$, utilisant la fonction de hachage $h(k) = k \bmod m$.

Insérer successivement les clés 5, 28, 19, 15, 20, 33, 37, 17, 26 et 10 dans T , puis supprimer ensuite les clés 19, 37 et 17.

On fixe maintenant pour la table un taux de remplissage maximal de $\frac{3}{4}$. Comment sont alors réalisées ces insertions et suppressions ?

Exercice 2 : hachage et statistiques

On souhaite déterminer quels sont les mots utilisés par Proust dans la *Recherche du temps perdu*, et leur fréquence. Proposer une solution utilisant un dictionnaire Python, en supposant qu’on dispose d’un générateur `proust()` qui permet d’itérer sur tous les mots du roman, l’un après l’autre, par une simple boucle `for mot in proust()`.

Estimer la complexité de cette solution, sachant que la longueur du texte est d’environ 10 millions de caractères pour 1.5 million de mots ? Comparer avec d’autres méthodes.

Exercice 3 : hachage et (autres) opérations ensemblistes

On représente ici des ensembles d’entiers naturels par des couples (T, h) , où T est une table de hachage à adressage fermé avec résolution des collisions par chaînage, et h la fonction de hachage associée.

1. Écrire une fonction `liste_elements(E)` qui renvoie une liste constituée de tous les éléments de l’ensemble E . Quelle est sa complexité ?
2. Écrire une fonction `union(E, F)` prenant en paramètre deux ensembles et renvoyant leur union. On supposera qu’on dispose d’une fonction `ensemble_vide()` qui initialise et renvoie un ensemble vide, et d’une fonction `ajoute(E, elt)` qui ajoute un `elt` dans la table représentant E . Quelle est sa complexité ?
3. Écrire une fonction `intersection(E, F)` prenant en paramètre deux ensembles et renvoyant leur intersection. On supposera qu’on dispose d’une fonction `contient(E, elt)` qui recherche `elt` dans la table représentant E et renvoie `True` ou `False` selon les cas. Quelle est sa complexité ?

Exercice 4 : redimensionnement de table de hachage

Une table de hachage est vue ici comme un objet ayant six attributs :

- l’attribut `cles` est un tableau de (listes de) clés,
- l’attribut `h` est une fonction de hachage,
- l’attribut `taille` est la longueur du tableau `cles`,
- les attributs `tmax` et `tmin` sont les taux maximal et minimal de remplissage, et
- l’attribut `nbCles` est le nombre de clés du tableau.

Parmi ces six attributs, trois ne varient jamais : la fonction de hachage h et les taux de remplissage t_{max} et t_{min} . La fonction de hachage h associe à une clé un (grand) entier naturel, qui est considéré modulo taille pour obtenir le hachage de la clé.

1. Écrire une fonction `redimensionne(table, t)` qui prend en paramètre une `table` de hachage et une (nouvelle) taille t , et redimensionne `table` à la taille t .

Pour quelles valeurs de t cette fonction est-elle utilisée en général ? Dans quelles circonstances ?

2. Quelle est la complexité de cet algorithme ?

3. Écrire une fonction `ajoute(table, elt)` qui ajoute un élément dans la `table` en effectuant un redimensionnement si nécessaire.

4. Soit $2m$ la longueur d'une table après une succession d'insertions et de suppressions ; des redimensionnements ont été effectués à chaque fois que le taux de remplissage a atteint $t_{max} = \alpha$. Combien de clés au moins étaient présentes dans la table au moment de son remplissage maximal ?

En déduire une borne inférieure pour le nombre d'opérations d'insertion et de suppression effectuées depuis la création de la table.

Si la longueur de la table au moment de sa création était ℓ , quel est le coût cumulé des redimensionnements effectués ?

Déduire des résultats précédents une borne supérieure pour la complexité amortie de ces redimensionnements.

5. Écrire une fonction `supprime(table, elt)` qui supprime `elt` de la `table`, en procédant à un redimensionnement si nécessaire.

Exercice 5 : analyse du hachage par chaînage

On considère une table de hachage de m boîtes contenant n éléments, supposés insérés dans un ordre aléatoire. On suppose par ailleurs que la résolution des collisions est faite par chaînage, et que chaque élément a une probabilité uniforme d'être haché vers l'une des m boîtes (hypothèse de *hachage uniforme simple*).

1. Quel est le nombre moyen d'éléments par boîte ?
2. Quel est alors le nombre moyen de tests faits lors d'une recherche *infructueuse* ?
3. Quelle est donc la complexité moyenne d'une recherche infructueuse ?
4. Lors d'une recherche réussie, quels éléments sont examinés avant l'élément cherché ?
5. Si k éléments ont été insérés dans la table après l'élément cherché, quel est le nombre moyen d'insertions dans une boîte donnée ?
6. Quel est alors le nombre moyen d'éléments insérés après l'élément cherché dans la même boîte que lui ?
7. Quelle est donc la complexité moyenne d'une recherche réussie ?

Table de hachage :

$$h(13) \bmod m$$

clés	0	1	2	3	4	5	6	7	8	9	$m-1$
	j	j	j	j	j	j	j	j	j	j	j

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

4 2 12 X 19 2e 2e 2e 2e 2e

taille : m , ici $m = 9$

nb clés : n , ici $n = 4$

t_{\min} , t_{\max} :

On veut avoir :

$$t_{\min} < \frac{n}{m} < t_{\max}$$

$$\frac{1}{2} < \frac{3}{4}$$

liste chaînée : L

i) for e in L

ii) $L.append(e)$

Ex 4

0x Faire un algo qui parcourt tous les éléments d'une table et renvoie une liste qui les contient tous

def parcourir(tab):

L = new ListeChaînée()

for l2 in tab: $\Theta(1)$

 for e in l2:
 L.append(e)
return L

$\Theta(n)$

$\Theta(1 + \text{nbr élé}) = \Theta(n)$

2x Insérer élément:

def insert(tab, x):

tab[h(x) % len(tab)].append(x)

1x Def redimensionne(table, f) \underbrace{m}

table = [list() for i in range(f)]

elems = parcourir(table)) $\Theta(m + n)$

for x in elements:

 insert(tab, x)

return tab

2x $\Theta(m + n)$

3y Def ajoute(table, elt)

if ($\frac{\text{table.n+1}}{\text{table.m}} \geq t_{\max}$)

table = resize (table, 2xtable.m)

insert (table, elt)

table.n += 1

return table

4y Table de taille 2m

1y Quelle était sa taille avant redimensionnement ?
table.m

2y Quel était son taux de remplissage à son dernier redimensionnement ?

$\frac{n}{m} > t_{\max}$: $\frac{\text{nbr élé}}{\text{tab.length}}$

3y Combien d'éléments avait-elle à ce moment-là ?
 $n \geq 9m$

$$\frac{4}{3} > t_{\max}$$

$$\frac{3}{4}$$

$$4 > t_{\max} \times \frac{1}{3}$$

$$\frac{3}{4}$$

4) n insertions coûtent $\Theta(n)$ au total
 $\leadsto \Theta(1)$ "en moyenne"
"multi"

Ex 5

nombre de boîte n nb de élé

1) $\frac{n}{m}$

2) Chn de comparaisons sans le retrouver

Algo recherche

i) calcul : $h(x) \% m$

ii) On cherche se dans
tab[i] \rightarrow court
taille de tab[i]

3) $\Theta\left(\frac{n}{m}\right)$
taille moyenne d'une
Liste

4) Ceux qui arrivent avant dans la liste : | Ceux qui arrivent après

5) en moyenne $\frac{k}{m}$ éléments insérés par boîte

6) Dans $\frac{k}{m}$ dans la boîte de x

7) On parcourt tous les élé de la liste de x qui

ont été insérés après x en moyenne

il y en a $\frac{k}{m}$ \Rightarrow complexité $O(l + \frac{k}{m})$