EA4 – Éléments d'algorithmique
TD n° 10 : hachage (suite)

Exercice 1 : tables de hachage et adressage ouvert

On considère une table de hachage `T` à adressage ouvert, de longueur $m = 11$ avec la fonction de hachage $h(k) = k \mod 11$.

1. Insérer successivement dans `T` les clés 5, 28, 19, 14, 20, 33, 36, 48, 26 et 10 en appliquant les méthodes de résolution des collisions suivantes :
   a. sondage linéaire (autrement dit, on sonde la case $h(k) \mod m$ et si elle est déjà occupée on passe à la case $h(k) + 1 \mod m$ et ainsi de suite).
   b. double hachage (autrement dit, on sonde successivement les cases $h_1(k)+i\times h_2(k) \mod m$) avec $h_1(k) = h(k)$ et $h_2(k) = 1 + (k \mod (m-1))$.
2. Supprimer ensuite les clés 19, 36 et 48 en détaillant la suite de sondages réalisés.
3. Comment se passe ensuite l'insertion de 26 ? de 39 ? Préciser quels sont les sondages réalisés et la case finalement choisie pour l'insertion.

Exercice 2 : redimensionnement de table de hachage II

On considère ici des tables de hachage à adressage ouvert avec des taux de remplissage minimal (β) et maximal (α), $\alpha > \beta$.

1. Quels sont le bons paramètres à considérer pour décider quand procéder à un « redimensionnement » (qui peut éventuellement se borner à un simple nettoyage sans changement de dimension) ? Est-ce le même pour les insertions et les suppressions ?
2. Trouver un exemple d'une table de hachage où, si on redimensionne en dupliquant la taille et ne recopiant que les clés presents, le taux de remplissage ($\frac{\texttt{nbCles}}{\texttt{taille}}$) est inférieur au taux de remplissage minimal.
3. Quel serait le taux de remplissage "idéal" dans une table de hachage ?
4. Déduire le valeur de redimensionnement pour une table avec `nbCles` assurant que la table obtenue ait ce taux.

Exercice 3 : algorithme de Rabin et Karp (examen 2015)

L'objectif de cet exercice est de donner une application des fonctions de hachage à la recherche d'un motif dans un texte.
On commence par proposer une méthode naïve.

1. Écrire une fonction `test_occurrence(m, t, i)` retournant `True` si un motif `m` apparaît à la position `i` d'un texte `t` (`m` et `t` étant représentés par des tableaux de caractères).
2. Écrire une fonction `recherche_naive(m, t)` effectuant une recherche naïve du motif `m` dans `t` à l'aide de `test_occurrence(m, t, i)`.
3. Quelle(s) opération(s) est-il raisonnable de prendre en compte pour le calcul de la complexité (en temps) des algorithmes précédents ? En déduire la complexité de l'algorithme `recherche_naive(m, t)`. Justifier.

Dans la suite on fera l'hypothèse que les chaînes de caractères ont été remplacées par des tableaux d'entiers en considérant la valeur entière du code binaire de chaque caractère (via la fonction `ord()` en Python, ou une conversion de format en Java).

L'algorithme de Rabin et Karp améliore l'algorithme `recherche_naive(m, t)` précédent. Il repose sur l'utilisation d'une fonction de hachage.

Soit $b \geqslant 2$ un entier fixé ; pour tout entier $k \geqslant 1$, on considère la fonction $h_k : \mathbb{N}^k \to \mathbb{N}$ telle que :

$$h_k(x_{k-1}, \ldots, x_0) = x_{k-1} \cdot b^{k-1} + x_{k-2} \cdot b^{k-2} + \cdots + x_2 \cdot b^2 + x_1 \cdot b + x_0$$

4. Rappeler le principe de la méthode de Horner pour évaluer un polynôme.

5. Soit `t` un tableau d'entiers, de longueur $\ell \geqslant k$. Décrire un algorithme `initialise_h(t, k)` permettant de calculer $h_k(\texttt{t[0]}, \ldots, \texttt{t[}k-1\texttt{]})$ en $\Theta(k)$ opérations arithmétiques sur des entiers (additions ou multiplications).

6. Si les entiers considérés sont quelconques, ce calcul est-il réellement de complexité linéaire ? Et si tous les calculs sont faits *modulo* un entier q fixé ?

Dorénavant, tous les entiers sont considérés modulo q (inutile de réécrire `initialise_h(t, k)`).

7. Quelle relation existe-t-il entre $h_k(\texttt{t[0]}, \ldots, \texttt{t[}k-1\texttt{]})$ et $h_k(\texttt{t[1]}, \ldots, \texttt{t[}k\texttt{]})$?

8. En déduire un algorithme `affiche_tous_h(t, k)` qui calcule puis affiche *toutes* les valeurs $h_k(\texttt{t[}i\texttt{]}, \ldots, \texttt{t[}i+k-1\texttt{]})$ pour $i \in [\![0, \texttt{len(t)} - k]\!]$ avec une complexité *totale* en $\Theta(\texttt{len(t)})$.

9. Le principe de l'algorithme de Rabin et Karp est d'utiliser la fonction h_k, pour un k bien choisi, comme un filtre pour limiter le nombre d'appels à `test_occurrence(m, t, i)`. Écrire un algorithme `recherche_RK(t, m)` recherchant le motif `m` dans `t` selon ce principe.

10. Déterminer sa complexité dans le pire des cas, en donnant un exemple de tel pire cas.

11. En faisant l'hypothèse d'uniformité suivante :

> *b et q peuvent être (et sont) choisis de telle manière que la répartition des images des k-uplets de lettres soit (quasi)-uniforme sur* $[\![0, q-1]\!]$

déterminer la complexité en moyenne de `recherche_RK(t, m)` en fonction de $\ell := \texttt{len(t)}$, de $k = \texttt{len(m)}$, de q et du nombre p d'occurrences trouvées. En déduire que, sous des hypothèses raisonnables, cette complexité moyenne est en $\Theta(\ell)$.

Exercice 4 : une application en bioinformatique

En bioinformatique, l'un des domaines principaux de recherche tourne autour de l'analyse de séquences ADN (représentées par des chaînes de caractères sur l'alphabet {`A, T, C, G`}). Par exemple, on peut chercher à repérer un motif codant pour une certaine protéine.

On utilise la fonction de hachage d'un mot $w = w_0 \ldots w_n$ définie de la façon suivante :

$$h(w) = \sum_{i=0}^{n} c_i 4^{n-i} \mod 48, \text{ où } c_i = \begin{cases} 0 & \text{si } w_i = \texttt{A} \\ 1 & \text{si } w_i = \texttt{C} \\ 2 & \text{si } w_i = \texttt{G} \\ 3 & \text{si } w_i = \texttt{T} \end{cases}$$

1. Calculer $h(\texttt{CCAGAT})$.

2. Appliquer l'algorithme de Rabin et Karp pour rechercher le mot `CCAGAT` dans le texte `TGCATACGAAATTGCACCAGATCA`.