Algorithmique Avancée Examen Réparti 2

AVEC CORRECTION

Les seuls documents autorisés sont les polys de cours, ainsi que la copie personnelle. Le barème donné est indicatif.

Exercice 1 : Questions diverses [5 points]

Question 1 On considère une table de hachage de taille $m = 2n$ dans laquelle on insère n clés, avec une fonction de hachage uniforme. Prouver que la proportion de cases vides dans la table vaut $\exp(-1/2)$ asymptotiquement, lorsque n tend vers l'infini.

Question 2 On veut incrémenter un compteur binaire représenté par un tableau $A[0..k-1]$ de k bits (le tableau contenant le bit b_i en case i représente l'entier $\sum b_i 2^i$).

1. Écrire l'algorithme pour passer de x à $x+1$ dans le tableau ($0 \leq x < 2^k - 2$).
   Quel est le nombre maximum de flips (passage d'un bit de 1 à 0 ou de 0 à 1) pour une opération d'incrément ?
2. On montre ici que le coût amorti, en nombre de flips, pour incrémenter le compteur de 0 à $n = 2^k - 1$ est de $O(1)$ pour une opération d'incrément.
   - Méthode par agrégat : montrer que le nombre total de flips pour la suite des n incréments est $2n(1 - 1/n)$.
   - Méthode de potentiel : on considère la fonction qui à un tableau associe son nombre de bits à 1. Montrer que c'est une fonction de potentiel. Quel est le coût amorti de l'opération d'incrément de i vers $i + 1$?

<u>Solution</u>

1. proba qu'une case soit vide après l'insertion de 1 clé : $m - 1/m$
après l'insertion de n clés : $((m-1)/m)^n$
nombre de cases vides est $m((m-1)/m)^n$ d'où proportion $((m-1)/m)^n = e^{n\log(1-1/m)} \sim e^{-n/m}$

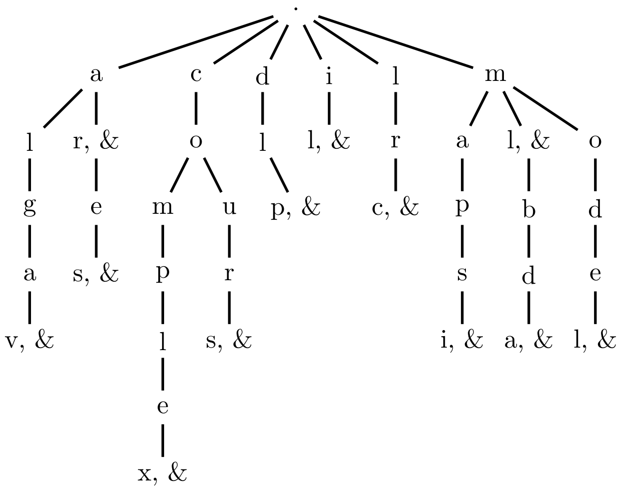
2. (a) A[0] := A[0] + 1 ; i :=0
TQ A[i] = 2
A[i] := 0, A[i+1] := A[i+1] +1
i := i+1
FTQ
pour incrementer de $2^{k-1} - 1$ à 2^{k-1}, il faut modifier tous les bits : k flips

(b) Méthode par agrégat : le bit sur A[0] change à chaque itération (n fois) le bit sur A[1] change 1 fois sur 2, le bit sur A[2] change 1 fois sur 4 Donc nombre total = n + n/2 + n/4 + n/8 + ... = 2n (1- 1/n) Donc le cout amorti en clips, sur n opérations d'incrément est O(1)
Soit opération d'incrément de i vers i+1, k =nb de retenues le nombre de flips est k+1 et la différence de potentiel Phi(i+1) - Phi(i) est -k+1 (k bits changes de 1 à O et 1 bit change de 0 à 1) donc cout amorti de l'opération d'incrément de i vers i+1 est k+1 + -k + 1 = 2 et donc cout totat < cout amorti total <2n

Exercice 2 : R-Trie et Patricia Trie [15 points]

Dans cet exercice, on considère deux structures de tries, la seconde permettant d'optimiser l'utilisation mémoire des structures. On se place dans l'ensemble des mots écrits en minuscules sans accent, donc on utilise les 26 caractères de a à z (et donc nous considérons des 26-tries).

Voilà un exemple, dans lequel seuls les sous-arbres contenant des suffixes de mots sont représentés. La racine est un nœud particulier puisqu'elle ne contient pas de lettre.



Chaque nœud (en dehors de la racine) contient un caractère et une information valant '&' si un mot se termine dans ce nœud et une liste de 26 éléments tel que le premier descendant est un 26-trie contenant les sous-mots commençant par a, le deuxième descendant est un 26-trie contenant les sous-mots commençant par b. Les sous arbres vides ne sont pas représentés sur la figure.

Voilà les primitives que vous devez utilisez. Toute autre fonction-outil nécessaire sera spécifiée et décrite.

```
def prem(cle):
    """ S -> str
        Renvoie le premier caractere de la cle."""
```

```
def reste(cle):
        """ S -> str
            Renvoie la cle privee de son premier caractere."""
```

```
def TrieVide():
        """  -> R-trie
            Renvoie le trie a 1 noeud vide avec R liens vides."""
```

```
def EstVide(A):
        """ R-trie -> booleen
            Renvoie vrai ssi A est vide."""
```

```
def Val(A):
        """ R-trie -> elt
            Renvoie la cle de la racine du trie."""
```

```
def SousArbre(A, i):
        """ R-trie * entier -> R-trie
            Renvoie une copie du i-eme sous-arbre de A."""
```

```
def FilsSauf(A, i):
        """ R-trie * entier -> liste[R-trie]
            Renvoie la liste des sous-arbres du trie privee du i-eme sous-arbre."""
```

Question 1 Donner la liste des mots stockés dans l'arbre précédent.

Question 2 Donner le pseudo-code d'un algorithme d'insertion d'un mot dans un 26-trie.

Question 3 Donner le pseudo-code d'un algorithme de suppression d'un mot dans un 26-trie.

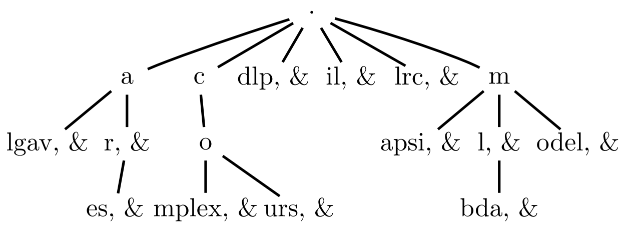
Question 4 Donner le pseudo-code d'un algorithme qui construit la liste des mots stockés dans un 26-trie, triée dans l'ordre alphabétique.

Question 5 Donner le pseudo-code d'un algorithme qui étant donné un 26-trie et un préfixe p, construit la liste des mots stockés dans le 26-trie qui ont pour préfixe p.

Question 6 Donner le pseudo-code d'un algorithme qui calcule la longueur du plus long mot stocké dans un 26-trie. On fera particulièrement attention à l'efficacité de l'algorithme.
Indiquer la mesure de complexité prise en compte et la complexité dans le pire des cas de l'algorithme.

Comme on le voit dans l'exemple ci-dessus, certains nœuds sont unaires, c'est-à-dire qu'ils ne contiennent qu'un seul descendant non vide (et pas de symbole de fin de mot &). Afin d'éviter cette perte de place (tous les descendants sauf un sont vides), on va fusionner les nœuds internes unaires (sans symbole &) avec leur fils non nul jusqu'à ce qu'il n'y ait plus de nœud unaire dans l'arbre (sans symbole &). Dès lors, les nœuds internes ne contiennent plus uniquement un caractère mais éventuellement un sous-mot de plusieurs caractères. Ces arbres sont appelés Patricia-tries.

Le premier exemple donne le Patricia-trie suivant.



Question 7 Décrire la structure pour les nœud permettant de stocker des Patricia-tries.

Question 8 Étant donnée la liste de mots suivante :
`[lou, leve, les, loups, dans, le, lourd, tapis, de, luxe, vert, olive]`,
dessiner le Patricia-trie qui lui correspond.

Question 9 Donner le pseudo-code d'un algorithme d'insertion d'un mot dans un Patricia-trie.

Question 10 Donner le pseudo-code d'un algorithme de fusion de deux Patricia-tries en un seul.

Solution

1.

Exercice 3 *: Hachage coucou [6 points]*

Le *hachage coucou* est une technique de hachage qui utilise deux fonctions de hachage h_1 et h_2. Ces deux fonctions sont définies sur un univers de n clés et sont à valeurs dans $\{1, \ldots, r\}$, avec $r > n$. On suppose que :

— h_1 et h_2 répartissent uniformément les clés, *i.e.*
pour tout $i \in \{1, \ldots, r\}$ on a $\Pr(h_1 = i) = \Pr(h_2 = i) = \frac{1}{r}$;

— h_1 et h_2 sont indépendantes, *i.e.*
pour tous $i, j \in \{1, \ldots, r\}$, on a $\Pr(h_1 = i, h_2 = j) = \Pr(h_1 = i)\Pr(h_2 = j)$.

Les clés sont réparties dans une table de hachage $T[1..r]$, chaque clé x peut être placée à la position $h_1(x)$ ou à la position $h_2(x)$ dans la table T. Pour insérer une clé x dans la table T, on calcule sa position $i = h_1(x)$. Si la case $T[i]$ est vide on y met la clé x, sinon on éjecte la clé y déjà présente et on la remplace par x. Il faut maintenant placer la clé y dans la table. Pour cela on calcule l'autre position j de y (si $i = h_1(y)$ alors $j = h_2(y)$ sinon $j = h_1(y)$). On recommence avec y ce qu'on a fait avec x (si $T[j]$ est vide on y met y, sinon...). Le processus s'arrête lorsqu'on tombe sur une case vide ou lorsqu'on a atteint le nombre maximal d'itérations, que l'on a fixé au préalable (égal au nombre n de clés). Dans ce dernier cas deux nouvelles fonctions de hachage sont choisies et on reconstruit toute la table (on dit qu'il y a un *re-hachage*). Il est possible que ce re-hachage n'aboutisse pas lui non plus, auquel cas on a recours à un deuxième re-hachage (et éventuellement à un troisième, etc).

Voici un pseudo-code pour la procédure d'insertion d'une clé x dans une table T.

```
Inserer(T, x)
    Si T[h1(x)] <> x Et T[h2(x)] <> x Alors
        pos <- h1(x)
        Repeter n fois
            Si T[pos] est vide Alors
                 T[pos] <- x
                 Exit Inserer
            Sinon
                 echanger x et T[pos]
                 Si pos = h1(x) Alors pos <- h2(x) Sinon pos <- h1(x) Fin Si
            Fin Si
        Fin Repeter
        Re-hacher(T)
        Inserer(T,x)
Fin Inserer
```

Remarque :

Au départ la procédure n'examine pas les deux positions $h_1(x)$ et $h_2(x)$ de la clé à insérer, mais seulement la position $h_1(x)$. La position $h_2(x)$ sera éventuellement examinée si l'on retombe sur la position $h_1(x)$ lors du passage dans la boucle `Repeter`. Cette situation se présentera dans les exemples.

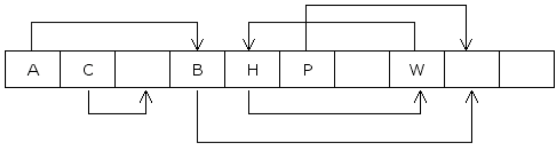


Figure 1 – Hachage coucou

La figure 1 représente le hachage coucou des clés A, B, C, H, P, W dans une table $T[1..10]$. Chaque clé x est placée à une position correspondant à l'une de ses deux valeurs de hachage et une flèche indique l'autre position possible de x dans la table (correspondant à l'autre valeur de hachage de x).

Question 1 Réaliser l'insertion de la clé Z ayant comme valeurs de hachage $h_1(Z) = 5$ et $h_2(Z) = 1$.

Question 2 Peut-on insérer une clé V ayant comme valeurs de hachage $h_1(V) = 5$ et $h_2(V) = 8$?

Si x est un entier, on désigne par $b_0(x), b_1(x), \ldots, b_k(x)$ les bits de x dans l'écriture binaire de x. Autrement dit, $x = b_k(x)2^k + \ldots b_1(x)2^1 + b_0(x)2^0$, avec $b_i(x) = 0$ ou 1. On veut réaliser le hachage coucou de clés entières en utilisant les opérations & et `rot` ainsi définies :

— si x et y sont deux entiers naturels alors $x\&y$ est l'entier z tel que $b_i(z) = 1$ ssi $b_i(x) = 1$ et $b_i(y) = 1$.

— si x est un entier naturel alors $\texttt{rot}(x, j)$ est l'entier obtenu en faisant une rotation circulaire de j bits vers la droite dans la représentation binaire de x. Autrement dit, si $x = a_k 2^k + \ldots a_1 2^1 + a_0 2^0$ et si $z = \texttt{rot}(x, j)$, avec $j \leq k$, alors $z = a_{j-1}2^k + \ldots a_0 2^{k-j+1} + a_k 2^{k-j} + \ldots + a_j 2^0$.

Question 3 Calculer 13 & 23 et $\texttt{rot}(100, 4)$.

Question 4 On considère les deux fonctions de hachage suivantes, à valeurs dans $\{1, \ldots, 8\}$:

— $h_1(x) = [(x^2 \bmod 17) \& 7] + 1$

— $h_2(x) = [(\texttt{rot}(x, 4) \bmod 33) \& 7] + 1$

On peut remarquer que x & 7 est le reste de la division par 8.
Effectuer le hachage coucou des clés 14, 100, 1000, 31, 117, dans cet ordre.
Aide : $14^2 \bmod 17 = 9$, $100^2 \bmod 17 = 4$,$1000^2 \bmod 17 = 9$, $31^2 \bmod 17 = 9$, $117^2 \bmod 17 = 4$.

Solution

1. $h_1(Z) = 5$, on éjecte H et on le remplace par Z, donc $T[5] = Z$.
L'autre position de H est 8, on éjecte W et on le remplace par H, donc $T[8] = H$.
L'autre position de W est 5, on éjecte Z et on le remplace par W, donc $T[5] = W$.
L'autre position de Z est $h_2(Z) = 1$, on éjecte A et on le remplace par Z, donc $T[1] = Z$.
L'autre position de A est 4, on éjecte B et on le remplace par A, donc $T[4] = A$.
L'autre position de B est 9, on tombe sur une case vide et on y place B donc $T[9] = B$.
On obtient donc la table :

Z	C		A	W	P		H	B	

2. Non, car H et W qui occupent les cases 5 et 8 ont comme seules positions possibles 5 et 8.

3. $13 \,\&\, 23 \equiv 1101 \,\&\, 10111 = 101 \equiv 5$. $\texttt{rot}(100, 4) \equiv \texttt{rot}(1100100, 4) = 0100110 \equiv 38$.

4. Valeurs de hachage :

	14	100	1000	31	117
h_1	2	5	2	2	5
h_2	7	6	6	8	7

Table de hachage :

position	1	2	3	4	5	6	7	8
clé		1000			117	100	14	31