Note : toutes les réponses doivent être justifiées. Les parties I, II et III sont indépendantes.

Partie I - Algorithmique

Notations : les intervalles utilisés dans cette partie sont des intervalles de $\mathbb{N}$.

Dans cette partie, on s'intéresse au problème de la recherche d'un élément dans un tableau (ou vecteur) de taille M indexé dans l'intervalle $[0, M-1]$ de $\mathbb{N}$. Soit E l'ensemble des éléments possibles, et K un ensemble de clés permettant d'identifier les éléments.

On note $k : E \to K$ l'application qui associe à chaque élément sa clé. **On supposera dans toute la partie I que l'application k est bijective**. Chaque entrée du tableau peut contenir soit un élément de E, soit le marqueur $\perp$ qui dénote l'absence d'élément. Dans ce dernier cas, l'entrée est dite libre. Un élément de E ne peut pas apparaître plus d'une fois dans le tableau. On note $t : [0, M-1] \to E \cup \{\perp\}$ l'application associant à un indice l'élément du tableau se trouvant à cet indice, ou $\perp$ si l'entrée correspondant à cet indice est libre. La recherche d'un élément consiste donc, connaissant sa clé, à trouver cet élément dans le tableau. On note f la fonction de recherche $f : K \to [0, M-1]$ qui à chaque clé associe l'indice de l'élément correspondant dans le tableau. Si l'élément de clé k n'est pas présent dans le tableau, $f(k)$ est l'indice d'une case libre où l'on peut placer l'élément de clé k. On note $\mathbb{B}$ l'ensemble des booléens {vrai, faux}.

Si a et b sont deux entiers avec $b \neq 0$, on désigne par a mod b le reste de la division entière de a par b. De plus, la représentation du nombre a en base b est considérée comme un mot de l'alphabet $\{0, 1, \ldots, b-1\}$: c'est le mot $a_m a_{m-1} \ldots a_0$ si

$$a = \sum_{i=0}^{m} a_i b^i \text{ avec } a_m \neq 0 .$$

Si x est un nombre réel $\lfloor x \rfloor$ désigne la partie entière de x.

I.A - Notations

Les notations suivantes seront utilisées pour exprimer les algorithmes. Pour donner un algorithme, le candidat devra l'exprimer soit avec ces notations, soit en Pascal, soit en CAML.

Définir f : $x \in A \mapsto y \in B$ corps de l'algorithme **FinDéfinir**	Définit la fonction f de l'ensemble A dans l'ensemble B qui à un élément $x \in A$ associe un élément $y \in B$ selon les instructions données dans le corps de l'algorithme. La valeur retournée par l'appel de $f(x)$ est la dernière valeur prise par y dans le corps de l'algorithme.
Si condition **alors** liste 1 d'instructions **Sinon** liste 2 d'instructions **FinSi**	Si condition est vraie, exécute liste 1 d'instructions sinon, exécute liste 2 d'instructions.
Si condition **alors** liste d'instructions **FinSi**	Si condition est vraie, exécute liste d'instructions, sinon, ne fait rien.
TantQue condition **faire** liste d'instructions **FinTantQue**	Si condition est vraie, exécute liste d'instructions jusqu'à ce que condition soit fausse. Si condition est fausse au départ, ne fait rien.

Les instructions sont séparées par une virgule ou un passage à la ligne. On dispose des opérateurs de comparaison usuels ($<, \leq, =, \neq, \geq, >$) lorsqu'ils ont un sens, ainsi que des opérateurs booléens, **et**, **ou** et **non**. Tous les symboles utilisés doivent être déclarés, les arguments et le résultat d'une fonction étant déclarés dans sa définition. Un symbole est déclaré en indiquant à quel ensemble appartient l'élément qu'il représente. Ainsi, $x \in A$ déclare un symbole x représentant un élément de A. L'élément que représente un symbole est choisi grâce à l'opérateur d'affectation $\leftarrow$. Après l'instruction $x \leftarrow y$ le symbole x représente l'élément que représentait le symbole y juste avant l'instruction. Par exemple, après $x \in \mathbb{N}$, $y \in \mathbb{N}$, $y \leftarrow 0$, $x \leftarrow y$, $y \leftarrow 1$ le symbole x représente l'entier 0, et le symbole y représente l'entier 1.

I.B - Recherche séquentielle

L'algorithme suivant recherche un élément en parcourant séquentiellement les entrées du tableau. On suppose que certains éléments de E peuvent ne pas être dans le tableau, c'est-à-dire que la recherche peut échouer.

```
Définir recherche : clef ∈ K ⟼ index ∈
   i ∈ [0, M − 1], i ← 0
   fini ∈ IB, fini ← faux
   Tant que non fini faire
      Si t(i) ≠ ⊥ alors
         Si k(t(i)) = clef alors
            index ← i
            fini ← vrai
         FinSi
      Sinon
            index ← i
      FinSi
      Si i = M − 1 alors
         fini ← vrai
      Sinon
         i ← i + 1
      FinSi
   FinTantQue
FinDéfinir
```

I.B.1) Quelle hypothèse doit-on faire sur le tableau pour que cet algorithme donne une valeur à index dans tous les cas ? On se place jusqu'à la fin du problème dans le cas où la fonction de recherche donne toujours une valeur à index.

I.B.2) Quelle propriété a la valeur que rend cette fonction lorsqu'il n'existe pas d'élément de clé clef dans le tableau ?

I.C - Fonctions de hachage

On suppose ici que $K \subset [0, b^n - 1]$, c'est-à-dire que les clés peuvent être considérées soit comme des nombres d'au plus n chiffres en base b soit comme des mots de longueur au plus n de l'alphabet $\{0, 1,\ldots,b-1\}$. Afin d'éviter de comparer la clé de l'élément recherché à toutes les clés des entrées présentes dans le tableau, on place les éléments à un indice calculable à partir de leur clé. Ainsi, la donnée de la clé permet de calculer l'indice auquel se trouve l'entrée correspondante si elle est présente dans le tableau. Si l'entrée n'est pas présente, l'indice obtenu est celui auquel il faudrait la ranger dans le tableau. On note $h{:}K \to [0, M-1]$ l'application associant à chaque clé un indice du tableau.

I.C.1) L'application h doit-elle être surjective, injective ou bijective ? Combien y-a-t-il de fonctions de K dans $[0, M-1]$ ayant cette propriété ? Donner un majorant dépendant de $\mathrm{Card}(K)$ et de M, du rapport entre le nombre de fonctions ayant cette propriété et le nombre de fonctions de K dans $[0, M-1]$. Évaluer ce majorant pour $\mathrm{Card}(K) = 3M/4$ supposé très grand devant 1.

I.C.2) Donner un algorithme de recherche simple utilisant h. Cet algorithme doit donner les renseignements suivants :

L'élément de clé k est-il dans le tableau ?

- Si oui, quel est l'indice de l'élément de clé k ?
- Si non, à quel indice peut-on placer l'élément de clé k ?

I.C.3) La fonction h ne peut être construite que si l'on connaît toutes les clés, et même si on les connaît, sa construction n'est pas triviale. Pour éviter ce problème, on accepte que h puisse donner le même indice pour deux clés différentes. On utilise alors une fonction $s:[0, M-1] \to [0, M-1]$ pour obtenir un nouvel indice lorsque celui que l'on a utilisé correspond à une entrée de clé différente de la clé recherchée. Ainsi, la fonction h ne donne plus directement l'indice de l'élément recherché dans le tableau, mais l'indice à partir duquel on doit le chercher. L'élément, s'il est présent, se trouve à un indice $s^j(h(k))$. S'il est absent, il existe j tel que $t(s^j(h(k))) = \perp$.

Donner un algorithme de recherche utilisant les fonctions h et s.

I.C.4) On appelle code de hachage d'une clé k la valeur $h(k)$. Lorsque deux clés ont le même code de hachage, on dit qu'il y a collision. Si c clés donnent le même résultat par h, il peut être nécessaire d'effectuer c comparaisons de clés lors d'une recherche, ce qui dégrade les performances de l'algorithme. De plus, on peut avoir $s^j(h(k)) = h(k')$: même s'il n'y a pas de collision sur k', la recherche de cette clé est perturbée par les collisions sur d'autres clés. On cherche donc à construire des fonctions de hachage qui donnent le plus rarement possible des résultats identiques pour des clés différentes.

Si l'on suppose que les éléments de K sont choisis au hasard dans $[0, b^n - 1]$ et que $M = b^m$ avec $m \leq n$, montrer qu'on construit une fonction h générant peu de collisions, en choisissant $h:k \longmapsto k \bmod M$.

I.C.5) **Hachage par division**

On choisit effectivement la fonction $h:k \longmapsto k \bmod M$. L'indice est donc le reste de la division entière de la clé par M.

a) Quelle relation lie les représentations en base b de k et de $h(k)$ si $M = b^m$?

b) Cent personnes ont leurs identités rangées dans un tableau (de taille 100). On souhaite effectuer une recherche dans ce tableau, les clés étant les dates de naissance que l'on suppose réparties uniformément. Évaluer la qualité d'une fonction de hachage par division selon que les clés sont de la forme JJMMAA, MMAAJJ ou AAJJMM. Les mots JJ, MM et AA désignent respectivement le jour, le mois et les deux derniers chiffres de l'année de naissance. On a $b = 10$, $m = 2$, $n = 6$ et l'on suppose que deux personnes ne peuvent avoir la même date de naissance.

c) On suppose que les cent personnes de la question précédente sont des élèves d'une même classe. Quel est le meilleur format à utiliser pour les clés ?

I.C.6) **Hachage par multiplication**. Nous venons de voir que le hachage par division est sensible au format des clés car il n'en exploite qu'une partie. Le hachage par multiplication permet d'éviter ce problème. On pose $w = b^n$, ainsi, la représentation d'un entier i en base b sur n chiffres devient celle du rationnel i/w si l'on place la virgule à gauche de son premier chiffre. On considère la fonction de hachage suivante, a étant un entier fixé :

$$h(k) = \left\lfloor \frac{M(ak \bmod w)}{w} \right\rfloor$$

$\lfloor x \rfloor$ désignant la partie entière de x. Comment calculer rapidement h si $M = b^m$ avec $m < n$?

I.C.7) **Clés multi-mots**. Dans la pratique, les clés sont rarement des entiers, mais plutôt des chaînes de caractères (par exemple des noms de personnes). Le codage numérique de telles clés peut nécessiter plus de n chiffres en base b. Pour pouvoir utiliser des fonctions de hachage de $[0, b^n - 1]$ dans $[0, M-1]$, on décompose chaque clé en mots de n chiffres en base b, et on les combine pour obtenir un seul mot de $[0, b^n - 1]$. Dans ce qui suit, si X est un ensemble, on note

$$X^* = \bigcup_{i=1}^{\infty} X^i$$

l'ensemble des suites de longueur quelconque d'éléments de X. On note $c : [0, b^n - 1]^* \to [0, b^n - 1]$ la fonction qui combine les mots d'une clé en un seul mot. Si h est une fonction de hachage de $[0, b^n - 1]$ dans $[0, M-1]$ on définit une fonction de hachage $h' : [0, b^n - 1]^* \to [0, M-1]$ permettant d'utiliser des clés multi-mots par $h' = h \circ c$. la fonction de combinaison c doit être rapide à calculer, tout comme la fonction de hachage. On choisira par exemple l'addition modulo b^n ou le **ou exclusif** bit à bit sur la représentation en base 2 des mots de la clé.

On rappelle les tables de vérité suivantes, $\vee$ désignant l'opérateur ou, $\wedge$ l'opérateur et, et $\oplus$ l'opérateur ou exclusif :

a	b	$a \vee b$
0	0	0
0	1	1
1	1	1
1	0	1

a	b	$a \wedge b$
0	0	0
0	1	0
1	1	1
1	0	0

a	b	$a \oplus b$
0	0	0
0	1	1
1	1	0
1	0	1

a) Pourquoi le **et** et le **ou** bit à bit, qui sont pourtant rapides à calculer, ne sont pas de bons choix pour la fonction de combinaison ?

b) L'addition modulo w et le **ou exclusif** sont commutatifs. Une clé composée des mots X et Y donnera donc le même résultat par c qu'une clé composée des mots Y et X, ce qui est gênant dans les cas pratiques où les clés ne sont pas uniformément réparties sur leur domaine. Quelle solution peut-on apporter à ce problème ?

I.D - Résolution des collisions

Pour résoudre les collisions (clés distinctes donnant le même résultat par la fonction de hachage), on introduit une fonction $h_2 : K \to [0, M-1]$, puis pour tout $k \in K$ une fonction $s_k : [0, M-1] \to [0, M-1]$ définie par $s_k(i) = (i + h_2(k)) \bmod M$.

I.D.1) Quel problème apparaît si h_2 ne donne pas toujours un nombre premier avec M ?

I.D.2) Donner une fonction h_2 sachant que $M = 2^m$ et que les clés sont des entiers de $[0, b^n - 1]$.

I.D.3) **Résolution de collisions par chaînage externe.** Nous avons vu que la collision de plusieurs clés sur un même indice du tableau peut perturber la recherche d'éléments dont la clé n'entre pas en collision avec cet indice. En effet, il se peut que pour deux clés k et k' distinctes, $s^j(h(k)) = h(k')$. Pour y remédier, on ne stocke plus les éléments eux-mêmes dans le tableau, mais les listes d'éléments dont la clé donne le même code de hachage. Ainsi, à l'indice i du tableau, on trouvera la liste (éventuellement vide) des éléments de clé k telle que $h(k) = i$. Soit $L = E^* \cup \{\perp\}$ l'ensemble des listes d'éléments de E, $\perp$ désignant la liste vide. On suppose que l'on dispose des fonctions suivantes :

- longueur : $L \to \mathbb{N}$ qui associe à chaque liste le nombre de ses éléments. On a longueur($\perp$) = 0.
- élément : $L \times \mathbb{N} \to E$. Si λ est une liste et i un entier, élément (λ, i) n'est définie que si $i <$ longueur (λ) et rend dans ce cas l'élément numéro i de la liste λ. Le premier élément d'une liste est à l'indice 0.

Donner un algorithme de recherche utilisant cette organisation des données. Cet algorithme calculera un couple (`indice`, `position`) d'entiers, le premier étant l'indice de la liste dans le tableau, le second étant la position de l'élément dans la liste. Si l'élément n'est pas dans la liste, `position` sera la longueur de la liste donc la position à laquelle l'élément pourra être placé.

I.E - Application

On se place dans les conditions définies au I.C en utilisant des mots binaires de deux bits $(b = 2, n = 2)$ avec un hachage par multiplication $(w = 4, a = 3)$. Les clés sont des chaînes de caractères sur l'alphabet {C,A,M,L}, et on associe les valeurs 00_2, 01_2, 10_2, 11_2 aux caractères C, A, M, L. Pour obtenir une clé sur deux bits, on combine la valeur des caractères par un ou exclusif ($\oplus$) précédé d'une rotation

d'un bit vers la gauche ($1\leftarrow$) de la valeur courante de la clé. On rappelle que la rotation d'un bit vers la gauche d'un mot de n bit $b_{n-1}b_{n-2}\ldots b_1b_0$ donne le mot $b_{n-2}b_{n-3}\ldots b_1b_0b_{n-1}$. On a $m = 2$, $M = 2^m = 4$, et on utilise une fonction $h_2(k) = (2k+1) \bmod M$ pour calculer les déplacements de la fonction $s_k(i) = (i + h_2(k)) \bmod M$. Par exemple, pour obtenir h(LMAC) on calculera :

$$L(11_2)\ 1\leftarrow \oplus\ M(10_2) = 01_2$$

$$01_2\ 1\leftarrow \oplus\ A(01_2) = 11_2$$

$$11_2\ 1\leftarrow \oplus\ C(00_2) = 11_2$$

$$(11_2 \times a) \bmod M = 9_{10} \bmod 4_{10} = 1$$

La clé calculée à partir des quatre caractères de LMAC est donc $11_2 = 3_{10}$ et h(LMAC) = 1. Si l'entrée 1 du tableau est occupée par un élément autre que LMAC, il faut calculer $h_2(11_2) = (3_{10} \times 2_{10} + 1) \bmod 4_{10} = 3_{10}$ pour obtenir $s_{11_2}(1) = (1 + h_2(11_2)) \bmod 4_{10} = 0_{10}$.

Donner la position des éléments de clés CAML, CLAM et CALM dans le tableau après leur insertion, dans cet ordre. Le tableau est initialement vide.

Partie II - Automates

II.A - Définitions et notations employées

- Soit A un alphabet, on note A^* l'ensemble des mots sur A.
- Soit u un mot de A^*, $|u|$ est la longueur du mot u.
- On note ε le mot vide (de longueur nulle).
- Le symbole de la concaténation est $\cdot$ (le point).
- Un mot u est un sous-mot de v quand il existe deux mots x et y, éventuellement vides tels que $v = x \cdot u \cdot y$.
- Un mot u est préfixe de v quand il existe un mot w tel que $v = u \cdot w$.
- Un mot u est suffixe de v quand il existe un mot w tel que $v = w \cdot u$.

Pour un automate fini déterministe $A = \langle E, e_0, F, A, \delta \rangle$:

- E est l'ensemble (fini) des états,
- e_0 est un élément distingué de E appelé état initial,
- F est un sous ensemble de E appelé ensemble des états finals,
- A est l'alphabet d'entrée,
- δ est une fonction de $E \times A$ dans E, appelée fonction de transition,
- δ^* est la fonction de $E \times A^*$ dans E définie à partir de δ par :
  - $\delta^*(e, \varepsilon) = e$

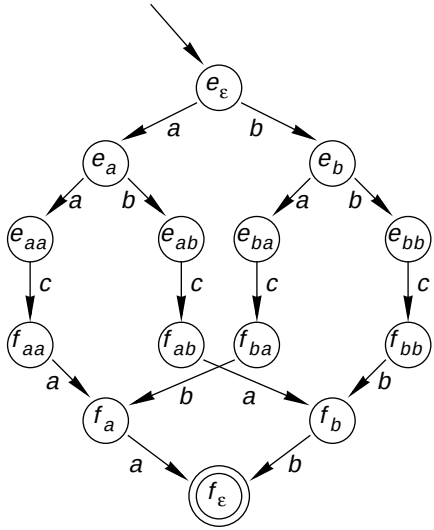
  - si $\delta^*(e,u)$ est défini, avec $\delta^*(e,u) = e_1$ et si $\delta(e_1,a)$ est défini avec $\delta(e_1,a) = e_2$, alors $\delta^*(e,u \cdot a)$ est défini et $\delta^*(e,u \cdot a) = \delta(\delta^*(e,u),a) = e_2$
- un mot u est reconnu par l'automate A quand $\delta^*(e,u)$ est défini, et $\delta^*(e,u) \in F$,
- le langage reconnu par l'automate A est l'ensemble des mots reconnus par A.

Pour un automate fini non déterministe $A = \langle E,e_0,F,A,\delta \rangle$.

- E est l'ensemble (fini) des états,
- e_0 est un élément distingué de E appelé état initial,
- F est un sous-ensemble de E appelé ensemble des états finals,
- A est l'alphabet d'entrée,
- δ est une application de $E \times A$ dans $P(E)^*$, appelée fonction de transition,
- soit $u = u_1u_2...u_n$ un mot de A^*, une suite d'états $s = s_0s_1u_s...s_n$ est compatible pour u quand : $s_0 = e_0$, et $s_i \in \delta(s_{i-1},u_i)$ pour $1 \le i \le n$,
- une suite compatible est acceptante quand elle se termine par un état final,
- un mot u est accepté par A quand il existe une suite acceptante pour u,
- un état e est accessible par u quand il existe une suite d'états compatibles pour u se terminant par e,
- un état e est accessible quand il existe un mot u tel que e est accessible par u,
- un mot $u = u_1u_2...u_n$ fait passer de e à e' quand il existe une suite d'états $s = s_0s_1...s_n$, $s_0 = e$, $s_n = e'$, et $s_i \in \delta(s_{i-1},u_i)$ pour $1 \le i \le n$.

II.B - Soit l'alphabet $A = \{a, b, c\}$. Montrer que l'automate déterministe, donné par le graphe ci-contre, reconnaît le langage L_2 :

$$L_2 = \{u \cdot c \cdot u | u \in \{a,b\}^*, |u|=2\}.$$



Dans les deux questions suivantes, on veillera à démontrer la validité des constructions proposées.

II.C - Généralisation : soit $n \geq 1$ un entier, construire un automate déterministe qui reconnaît le langage :

$$L_n = \{u \cdot c \cdot u | u \in \{a,b\}^*, |u| = n\}.$$

Quel est, en fonction de n, le nombre d'états de cet automate ?

II.D - Construire un automate non déterministe, qui reconnaît le langage suivant :

$$L'_n = \{v \cdot c \cdot w | \exists v_1, v_2, w_1, w_2, u \in \{a,b\}^*, |u|=n, v=v_1 \cdot u \cdot u_2, w=w_1 \cdot u \cdot w_2\}.$$

Donner le nombre d'états de cet automate en fonction de n.

II.E - Donner le graphe d'un automate reconnaissant L'_2 (cas où $n = 2$).

II.F - Soit A un automate fini non déterministe reconnaissant L'_n, δ la fonction de transition de A, e_0 l'état initial de A. Montrer que dans toute suite compatible acceptante pour un mot de longueur $2n+1$ tous les états sont distincts.

II.G - Montrer que $\text{Card}(\{e | \exists u \in \{a,b\}^*, e \text{ accessible par } u\}) \geq 2^{n+1} - 1$.

II.H - Montrer que tout automate déterministe ou non déterministe qui reconnaît L'_n a au moins $2^{n+2} - 2$ états.

II.I - Donner une expression rationnelle associée à L'_2.

II.J - Donner une expression rationnelle associée à $A^* - L'_1$.

Partie III - Expressions booléennes

On appelle ici expression booléenne toute expression construite avec les variables booléennes $a,b,\ldots,z$, les connecteurs binaires $\vee$ (OU) et $\wedge$ (ET), le connecteur unaire $\neg$ (NON) et les deux connecteurs 0-aires (c'est-à-dire les constantes booléennes) T (VRAI) et F (FAUX).

III.A - Montrer que toute expression booléenne est équivalente à une expression utilisant uniquement $\wedge$ et $\neg$ comme connecteurs.

III.B - Soit le connecteur ternaire $G(a,b,c)$ défini par :

$$G(a,b,c) = (\neg(a) \wedge b \wedge c) \vee (a \wedge (\neg(b))) \vee (\neg(b) \wedge \neg(c)) .$$

Montrer que toute expression booléenne est équivalente à une expression utilisant uniquement G comme connecteur.

III.C - Soit le connecteur binaire $-$ défini par

$$b - a = (\neg(a) \wedge b) .$$

Montrer que toute expression booléenne est équivalente à une expression utilisant uniquement $-$ et T comme connecteurs.

III.D - Toute expression booléenne est-elle équivalente à une expression utilisant uniquement $-$ comme connecteur ?

••• FIN •••
