**ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARIS,
TÉLÉCOM PARIS, MINES PARIS,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT ATLANTIQUE, ENSAE PARIS,
CHIMIE PARISTECH - PSL.**

**Concours Mines-Télécom,
Concours Centrale-Supélec (Cycle International).**

CONCOURS 2021

ÉPREUVE D'INFORMATIQUE MP

Durée de l'épreuve : 3 heures

L'usage de la calculatrice et de tout dispositif électronique est interdit.

Cette épreuve concerne uniquement les candidats de la filière MP.

Les candidats sont priés de mentionner de façon apparente sur la première page de la copie :

INFORMATIQUE - MP

L'énoncé de cette épreuve comporte 11 pages de texte.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Préliminaires

L'épreuve est composée d'un problème unique, comportant 30 questions. Après cette section de préliminaires et une section de présentation du *jeu du solitaire*, le problème est divisé en deux sections indépendantes, pages 4 et 8. Dans la première section, nous étudions le jeu du solitaire sur un tablier de forme classique par des méthodes de théorie des graphes. Dans la seconde section, nous étudions le jeu du solitaire sur un tablier en bande unidimensionnelle par des méthodes de théorie des automates. Pour répondre à une question, un candidat pourra réutiliser le résultat d'une question antérieure, même s'il n'est pas parvenu à établir ce résultat.

Concernant la programmation

Il faudra coder des fonctions à l'aide du langage de programmation OCaml, en reprenant l'entête de fonction fournie par le sujet, sans nécessairement reprendre la déclaration des types. Pour écrire une fonction, on pourra faire appel à d'autres fonctions définies dans les questions précédentes ; on pourra aussi définir des fonctions auxiliaires. Quand l'énoncé demande de coder une fonction, il n'est pas nécessaire de justifier que celle-ci est correcte, sauf si l'énoncé le demande explicitement. Si les paramètres d'une fonction à coder sont supposés vérifier certaines hypothèses, il ne sera pas utile de tester si les hypothèses sont bien vérifiées dans le code de la fonction.

Dans tout l'énoncé, un même identificateur écrit dans deux polices de caractère différentes désignera la même entité, mais du point de vue mathématique pour la police en italique (par exemple n) et du point de vue informatique pour celle en romain avec espacement fixe (par exemple `n`).

Aide à la programmation en OCaml

Opérations sur les listes : Sans qu'il ne soit imposé de coder dans un style de programmation les utilisant, on pourra s'appuyer sur les fonctions suivantes :

- `append`, de type `'a list -> 'a list -> 'a list`, qui concatène deux listes en une seule liste ;
- `filter`, de type `('a -> bool) -> 'a list -> 'a list`, telle que `filter p l` renvoie la liste des éléments x de la liste ℓ tels que le prédicat $p(x)$ vaut `true`, en respectant l'ordre de la liste ;
- `flatten`, de type `'a list list -> 'a list`, qui concatène une liste de listes en une seule liste ;
- `fold`, de type `('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`, telle que `fold f a [b1; ..; bn]` renvoie `f(..(f(f a b1) b2)..) bn` ;
- `map`, de type `('a -> 'b) -> 'a list -> 'b list`, telle que `map f [a1; ..; an]` renvoie la liste `[f a1; ..; f an]`.

Opérations sur les couples : On rappelle que

— la fonction `fst`, de type `'a * 'b -> 'a`, renvoie le premier terme d'un couple ;

— la fonction `snd`, de type `'a * 'b -> 'b`, renvoie le second terme d'un couple.

Opérateurs logiques sur les entiers : Nous supposons que les entiers naturels, de type `int` en OCaml, sont systématiquement codés à l'aide de 62 bits. Dans un texte mathématique, on pourra signaler la représentation binaire par une barre horizontale. Pour tous entiers n et k avec $0 \leqslant n < 2^{62}$ et $0 \leqslant k < 62$, nous notons $[n]_k$ le k^{e} bit de poids le plus faible de la représentation binaire de n, ce qui permet d'écrire $n = \overline{[n]_{61}[n]_{60}\cdots[n]_0}$.

Le tableau ci-dessous rappelle le résultat de quelques-uns des opérateurs logiques de OCaml qui agissent bit à bit sur deux entiers naturels a et b. Tous ces opérateurs renvoient un élément de type `int`. Dans ce tableau, les symboles $\wedge$, $\vee$, $\oplus$ et $\neg$ désignent respectivement le « et », le « ou », le « ou exclusif » entre deux booléens et la négation d'un booléen.

Opérateur	Nom	Résultat exprimé sur 62 bits
`a land b`	Et logique bit à bit	$\overline{([a]_{61} \wedge [b]_{61})\cdots([a]_1 \wedge [b]_1)([a]_0 \wedge [b]_0)}$
`a lor b`	Ou logique bit à bit	$\overline{([a]_{61} \vee [b]_{61})\cdots([a]_1 \vee [b]_1)([a]_0 \vee [b]_0)}$
`a lxor b`	Xor logique bit à bit	$\overline{([a]_{61} \oplus [b]_{61})\cdots([a]_1 \oplus [b]_1)([a]_0 \oplus [b]_0)}$
`lnot a`	Non logique bit à bit	$\overline{(\neg[a]_{61})\cdots(\neg[a]_1)(\neg[a]_0)}$
`a lsl b`	Décalage vers la gauche	$\overline{[a]_{61-b}\cdots[a]_1[a]_0\underbrace{0\cdots0}_{b}}$ b zéros à droite
`a lsr b`	Décalage vers la droite	$\overline{\underbrace{0\cdots0}_{b}[a]_{61}\cdots[a]_{b+1}[a]_b}$ b zéros à gauche

Par exemple, lorsque $a = 6$ et $b = 3$, les écritures sur 62 bits de a et b sont $a = \overline{0\cdots0110}$ et $b = \overline{0\cdots0011}$ et

— le calcul de `a land b` vaut l'entier 2, dont l'écriture sur 62 bits est $\overline{0\cdots0010}$;

— le calcul de `a lor b` vaut l'entier 7, dont l'écriture sur 62 bits est $\overline{0\cdots0111}$;

— le calcul de `a lxor b` vaut l'entier 5, dont l'écriture sur 62 bits est $\overline{0\cdots0101}$;

— le calcul de `a lsl b` renvoie 48, dont l'écriture sur 62 bits est $\overline{0\cdots0110000}$;

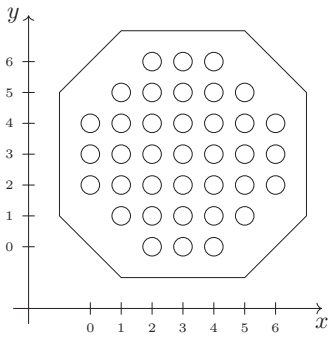
— le calcul de `a lsr b` renvoie 0 puisque les deux bits non nuls de a sont sortis de la représentation.

L'entier 2^k, avec $0 \leqslant k < 62$, peut commodément se définir en OCaml par `1 lsl k`.

Le jeu du solitaire

Le *jeu du solitaire* se joue sur un tablier percé d'encoches disposées en quadrillage et remplissant une certaine forme géométrique.

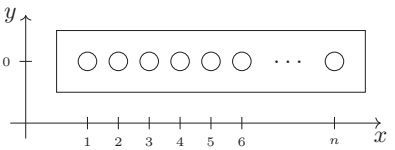
Parmi les tabliers les plus fréquents, le *tablier européen* est formé de 37 encoches organisées en octogone



dont on peut décrire les coordonnées par

$$\Big\{(x,y)\in\mathbb{N}^2\,;\ 0\leqslant x\leqslant 6,\ 0\leqslant y\leqslant 6 \text{ et } |x-3|+|y-3|\leqslant 4\Big\}.$$

D'autres tabliers existent tels que le tablier rectangulaire de dimension $n\times 1$, où $n\geqslant 1$ est un entier,



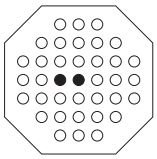
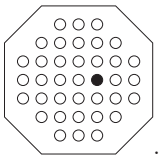
et dont on peut décrire les coordonnées par

$$\Big\{(x,y)\in\mathbb{N}^2;\ 1\leqslant x\leqslant n \text{ et } y=0\Big\}.$$

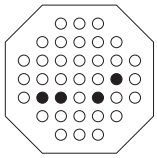
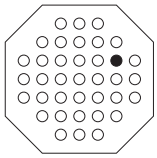
Sur un tablier, toute encoche de coordonnées (x,y) possède au plus quatre encoches *voisines*, à savoir les encoches de coordonnées $(x+1,y)$, $(x-1,y)$, $(x,y+1)$ et $(x,y-1)$ si elles existent.

Nous appellons *motif* tout sous-ensemble d'encoches dans le tablier. Nous disons qu'un motif est *ponctuel* s'il est de cardinal 1.

En début de jeu, des fichets (ou des billes) viennent se loger dans les encoches d'un motif initial. En cours de jeu, un *coup simple* consiste à déplacer l'un des fichets en sautant par-dessus l'une des encoches voisines pour atteindre l'encoche immédiatement après. Il peut être joué à condition que l'encoche voisine en question soit occupée et que le fichet déplacé retombe dans une encoche inoccupée. À l'issue d'un coup simple, le fichet planté dans l'encoche au-dessus de laquelle a eu lieu le saut est retiré. Par exemple,

 peut devenir  .

En cours de jeu, un *coup composé* consiste à jouer zéro, un ou plusieurs coups simples en déplaçant le même fichet. Par exemple,

 peut devenir  .

Une suite de motifs obtenus en jouant une suite de coups s'appelle une *partie*. L'objectif de la joueuse ou du joueur est de transformer un motif en un autre motif par une suite de coups.

1 Partie jouée sur le tablier européen en un minimum de coups

Dans toute la section 1 du sujet, une joueuse joue sur le tablier européen. Elle souhaite transformer un motif initial en un motif final par un minimum de coups composés.

☐ 1 – À titre préliminaire, nommer le type de parcours de graphe le plus approprié pour déterminer un chemin entre deux sommets qui minimise le nombre d'arcs traversés. Quelle politique de mise en attente de sommets caractérise ce parcours ?

1.1 Numérotation du tablier

Nous numérotons l'encoche de coordonnées $(x, y) \in \mathbb{N}^2$ dans le tablier européen par l'entier $z = 8y + x \in \mathbb{N}$ (voir figure 1, page 5). Nous notons l'ensemble des numéros d'encoches du tablier européen par

$$\mathcal{E} = \{8y + x \,;\, (x, y) \in \mathbb{N}^2 \text{ avec } 0 \leqslant x, y \leqslant 6 \text{ et } |x - 3| + |y - 3| \leqslant 4\} \subseteq \mathbb{N}.$$

Indication OCaml : En OCaml, on peut retrouver les coordonnées (x, y) d'une encoche à partir de son numéro z en écrivant `z mod 8` pour obtenir l'abscisse x et `z/8` pour obtenir l'ordonnée y. On peut calculer la valeur absolue d'un entier avec `abs`.

☐ 2 – Écrire une fonction `numero_interieur (z:int) : bool`, qui teste si un entier naturel z est le numéro d'une encoche du tablier européen $\mathcal{E}$.

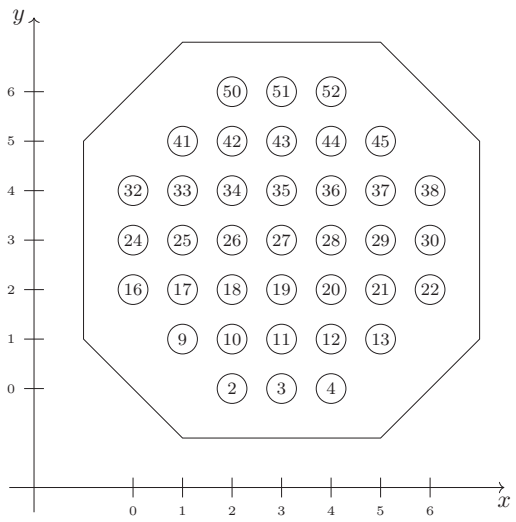


FIGURE 1 – Numérotation des encoches du tablier européen

☐ 3 – En énumérant les entiers naturels inférieurs à 52 et à l'aide de la fonction `numero_interieur` introduite à la question 2, définir une constante globale `numeros_europeens`, de type `int list`, égale à la liste des numéros d'encoches du tablier européen.

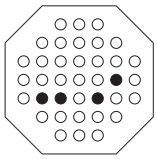
☐ 4 – En supposant qu'elles existent dans le tablier, donner par une expression mathématique le numéro des quatre encoches voisines de l'encoche de numéro z.

1.2 Motifs

Pour représenter un motif $M \subseteq \mathcal{E}$, nous utilisons un entier naturel m dont nous exploitons une partie des bits de l'écriture binaire comme suit : pour tout numéro d'encoche $z \in \mathcal{E}$, nous fixons

$$[m]_z = \begin{cases} 1 & \text{si } z \in M \\ 0 & \text{si } z \notin M \text{ ou si } z \notin \mathcal{E} \end{cases}.$$

Lorsqu'un motif est ponctuel, c'est-à-dire qu'il ne contient qu'une seule encoche, sa représentation est simplement une puissance de 2.
Par exemple, le motif à quatre encoches suivant



se représente par le nombre entier

$$m_0 = 2^{29} + 2^{20} + 2^{18} + 2^{17} \ = 538312704 \in \mathbb{N}.$$

Indication OCaml : Pour représenter les motifs et les motifs ponctuels, nous définissons les types `motif` et `ponctuel` par la déclaration suivante :

```
type motif = int;;
type ponctuel = motif;;
```

L'exemple ci-dessus se déclare à l'aide des opérateurs logiques en OCaml par
`let m0 = (1 lsl 29) lor (1 lsl 20) lor (1 lsl 18) lor (1 lsl 17);;`

☐ 5 – Écrire une fonction `numero_vers_ponctuel (z:int) : ponctuel` qui renvoie le motif ponctuel $\{z\} \subseteq \mathcal{E}$ formé d'une seule encoche de numéro z.

☐ 6 – Écrire une fonction `numeros_vers_motif (l:int list) : motif` qui renvoie le motif formé des encoches de numéro appartenant à la liste ℓ.

☐ 7 – En s'appuyant sur la constante `numeros_europeens` introduite à la question 3, définir une constante globale `motif_europeen`, de type `motif`, égale au motif formé de toutes les encoches du tablier européen.

☐ 8 – Démontrer que la fonction suivante

```
let est_ponctuel (m:motif) : bool =
  (m>0) && ((m land (m-1)) = 0) ;;
```

renvoie `true` si le motif m est un motif ponctuel et `false` sinon.

☐ 9 – Écrire, à l'aide des opérateurs logiques sur les entiers, une fonction `inclus (m:motif) (p:ponctuel) : bool` qui renvoie `true` si le motif ponctuel p est inclus dans le motif m ou `false` dans le cas opposé. Donner une preuve mathématique du résultat.

☐ 10 – Écrire, à l'aide des opérateurs logiques sur les entiers, une fonction `voisin_g (p:ponctuel) : ponctuel` qui calcule la translation par une encoche vers la gauche du motif ponctuel p. Si le motif ponctuel p sort du tablier, la valeur de retour est le motif vide 0.

Dans la suite du problème, nous supposerons avoir codé des fonctions similaires `voisin_d`, `voisin_h` et `voisin_b` qui calculent les décalages d'un motif ponctuel respectivement vers la droite, vers le haut et vers le bas. Nous déclarons une constante globale par `let voisins = [voisin_g; voisin_d; voisin_h; voisin_b];;`.

1.3 Coups simples et composés

☐ 11 – Écrire une fonction `coup_simple ((m, p) : motif * ponctuel) : (motif * ponctuel) list` qui prend en entrée un motif quelconque m et un motif ponctuel p contenu dans m et qui construit la liste des couples (m', p') où m' est un motif obtenu à partir de m à la suite du déplacement par un coup simple du fichet initialement placé dans p et où p' est le motif ponctuel repérant le même fichet après son déplacement.

☐ 12 – Écrire une fonction `coup_compose ((m, p) : motif * ponctuel) : (motif * ponctuel) list` qui prend en entrée un motif quelconque m et un motif ponctuel p contenu dans m et qui construit la liste des couples (m', p') où m' est un motif obtenu à partir de m à la suite du déplacement par un coup composé du fichet initialement placé dans p et où p' est le motif ponctuel repérant le même fichet après son déplacement.

☐ 13 – Écrire une fonction `mouvements (m:motif) : motif list` dont la valeur de retour est la liste de tous les motifs que l'on peut obtenir en jouant un coup composé depuis le motif m, n'importe quel fichet pouvant être déplacé.

1.4 Partie de longueur minimale

Indication OCaml : Nous utilisons dans cette sous-section une structure de données de dictionnaire, avec modification en place, permettant d'associer des clés de type `motif` et des valeurs de type `int`. Nous notons le type de cette structure `dico`. Pour utiliser ces dictionnaires, nous disposons des fonctions suivantes :

- la fonction `create`, de type `unit -> dico`, qui permet de créer un dictionnaire vide ;
- la fonction `add`, de type `dico -> motif -> int -> unit`, telle que `add d m n` permet d'ajouter au dictionnaire d une association entre la clé m et l'entier n (si une association de clé m existe déjà, elle est remplacée) ;
- la fonction `mem`, de type `dico -> motif -> bool`, telle que `mem d m` renvoie le booléen `true` si la clé m est membre du dictionnaire d ou `false` sinon.
- la fonction `find`, de type `dico -> motif -> int`, telle que `find d m` renvoie la valeur associée à la clé m si la clé est membre du dictionnaire d ou une erreur sinon.

☐ 14 – Écrire une fonction `add_and_mem (d:dico) (s:int) (m:motif) : bool` qui ajoute l'association entre le motif m et l'entier s dans le dictionnaire d si le motif m est initialement absent du dictionnaire d. La valeur de retour de la fonction est `true` si le dictionnaire a été modifié et `false` sinon.

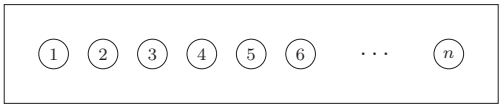
☐ 15 – Écrire une fonction `strate (d:dico) (s:int) (l:motif list) : motif list` qui, pour tout motif m que l'on peut obtenir après un coup composé à partir d'un motif de la liste ℓ, ajoute l'association entre le motif m et l'entier s au dictionnaire d si le motif m n'est pas présent dans le dictionnaire d. La valeur de retour de la fonction est la liste des motifs que la fonction ajoute.

☐ 16 – Dans cette question, nous supposons qu'il est possible de passer d'un motif m_i à un motif m_f par une suite de coups. Écrire une fonction `partie_minimale (mi:motif) (mf:motif) : int` qui calcule le nombre minimal de coups composés à jouer pour passer du motif initial m_i au motif final m_f. On pourra utiliser un dictionnaire qui associe des motifs au nombre de coups minimal pour les atteindre depuis le motif m_i.

☐ 17 – Peut-on considérer que la réponse donnée à la question 16 observe le parcours nommé à la question 1 ? Introduire un graphe puis argumenter.

2 Reconnaissance des motifs résolubles sur le tablier unidimensionnel

Dans toute la section 2 du sujet, une joueuse joue sur des tabliers rectangulaires de dimension $n \times 1$, où n est un entier naturel pouvant varier d'une partie à une autre.



Dans cette section, le but de la joueuse est de faire disparaître tous les fichets sauf un par une suite de coups, autrement dit de jouer une partie qui amène le motif initial à un motif ponctuel. Lorsque ceci est possible, nous disons que le motif initial est *résoluble* et que la partie est *gagnante*.

Nous introduisons l'alphabet binaire $\Sigma = \{0, 1\}$ et notons Σ^* l'ensemble des mots sur Σ. Pour tout mot $w = \sigma_1 \dots \sigma_n \in \Sigma^*$, nous notons $[w]_k$ le k^{e} symbole σ_k de w.

Le *mot d'un motif* M du tablier rectangulaire de dimension $n \times 1$ est le mot w formé de n symboles de l'alphabet Σ tel que, pour k variant entre 1 et n, le symbole $[w]_k$ vaut 1 si la k^{e} encoche à partir de la gauche appartient au motif M et vaut 0 sinon. Par exemple, sur ce tablier, le motif $\bullet\circ\bullet\bullet\circ$ est décrit par le mot $10110 \in \Sigma^*$.

Dans toute cette section, nous ne considérons que des coups simples. Une partie est par conséquent une suite finie W de mots w_0, w_1, …, w_m de même longueur telle que pour tout indice i compris entre 0 et $m-1$, le mot w_{i+1} est obtenu à partir du mot w_i en remplaçant dans le mot w_i un facteur 110 par les symboles 001 ou bien en remplaçant un facteur 011 par les symboles 100. Dans le premier cas, nous parlons de coup vers la droite ; dans le second cas, nous parlons de coup vers la gauche. Une partie est gagnante si, de plus, le dernier mot w_m de la partie ne comprend le symbole 1 qu'une seule fois.

L'objectif de cette section est d'étudier le langage $\mathcal{R} \subseteq \Sigma^*$ des mots de longueur quelconque de motifs résolubles.

2.1 Mots de motifs ponctuels

Nous notons $\mathcal{P} \subseteq \Sigma^*$ le langage des mots de longueur quelconque de motifs ponctuels.

☐ 18 – Donner, sans justification, une expression rationnelle qui décrit le langage $\mathcal{P}$.

☐ 19 – Dessiner, sans justification, un automate fini qui reconnaît le langage $\mathcal{P}$.

☐ 20 – Le langage $\mathcal{P}$ est-il un langage local ?

2.2 Bascules de l'état d'une encoche

Soient $W = (w_0, w_1, \ldots, w_m) \in (\Sigma^n)^{m+1}$ une partie en m coups simples joués sur le tablier de dimension $n \times 1$ et k un entier compris entre 1 et n. Dans cette sous-section, nous souhaitons démontrer qu'il existe une constante α indépendante des entiers n, m ou k telle la suite des $(m+1)$ symboles $[w_0]_k$, $[w_1]_k$, …, $[w_m]_k$ ne change jamais de valeur à plus de α reprises.

Nous notons $\phi = \frac{\sqrt{5}-1}{2}$ l'une des racines du polynôme X^2+X-1. On pourra utiliser sans preuve l'inégalité $\sum_{j=1}^{+\infty} \phi^j \leqslant 2$. Pour tout mot $w \in \Sigma^n$ de longueur n, nous introduisons le variant $V_k(w)$ par la somme

$$V_k(w) = \sum_{j=1}^{n} \phi^{|j-k|}[w]_j \in \mathbb{R}.$$

☐ 21 – Montrer qu'il existe un réel S, indépendant des entiers n et k, tel que, pour tout mot $w \in \Sigma^n$, on ait l'inégalité $V_k(w) \leqslant S$.

☐ 22 – Montrer que la suite des $(m+1)$ variants $V_k(w_0)$, $V_k(w_1)$, …, $V_k(w_m)$ est une suite de réels décroissante.

☐ 23 – Montrer que si, pour un indice i compris entre 0 et $m-1$, le symbole $[w_i]_k$ vaut 1 et le symbole $[w_{i+1}]_k$ vaut 0, alors on a $V_k(w_{i+1}) \leqslant V_k(w_i) - 1$.

☐ 24 – En déduire qu'il existe une constante α, indépendante des entiers n, m et k, telle que la suite $[w_0]_k$, $[w_1]_k$, …, $[w_m]_k$ ne change jamais de valeur à plus de α reprises.

2.3 Trace d'une partie

À partir d'une partie $W = (w_0, w_1, \ldots, w_m) \in (\Sigma^n)^{m+1}$ en m coups simples joués sur le tablier de dimension $n \times 1$, nous construisons la *trace T de la partie W* en suivant la procédure suivante. Pour plus de clarté, un exemple est présenté en colonne de droite avec la partie en deux coups (w_0, w_1, w_2) où $w_0 = 110101$, $w_1 = 001101$ et $w_2 = 010001$.

ÉTAPE 1 : Nous organisons les mots de W sous forme d'un tableau de n colonnes et $m+1$ lignes en positionnant le mot w_i dans la ligne i. Les lignes sont numérotées de bas en haut.

w_2	0	1	0	0	0	1
w_1	0	0	1	1	0	1
w_0	1	1	0	1	0	1

ÉTAPE 2 : Pour tout indice de ligne i compris entre 0 et $m-1$, pour tout indice de colonne j compris entre 1 et n, quand les symboles $[w_i]_j$ et $[w_{i+1}]_j$ sont égaux, nous effaçons le symbole $[w_{i+1}]_j$ inscrit en position $(i+1, j)$, de sorte qu'il ne reste que le mot w_0 dans la ligne 0 et les 3 symboles qui ont été modifiés dans les autres lignes.

	1	0	0		
0	0	1			
1	1	0	1	0	1

ÉTAPE 3 : Pour tout indice i compris entre 0 et $m-1$, lorsqu'un coup vers la gauche a été joué entre le mot w_i et le mot w_{i+1}, nous ajoutons la flèche $\leftarrow$ en indice des symboles de la ligne $i+1$; lorsqu'un coup vers la droite a été joué, nous ajoutons la flèche $\rightarrow$ en indice.

	$1_{\leftarrow}$	$0_{\leftarrow}$	$0_{\leftarrow}$		
$0_{\rightarrow}$	$0_{\rightarrow}$	$1_{\rightarrow}$			
1	1	0	1	0	1

ÉTAPE 4 : Nous numérotons dans chaque ligne différente de la ligne 0 les trois symboles restants par les marques I, II et III en exposant, en allant de gauche à droite.

	$1^{\text{I}}_{\leftarrow}$	$0^{\text{II}}_{\leftarrow}$	$0^{\text{III}}_{\leftarrow}$		
$0^{\text{I}}_{\rightarrow}$	$0^{\text{II}}_{\rightarrow}$	$1^{\text{III}}_{\rightarrow}$			
1	1	0	1	0	1

ÉTAPE 5 : Nous abaissons l'ensemble des symboles, colonne par colonne, pour les regrouper dans les alvéoles les plus bas.

	$1^{\text{I}}_{\leftarrow}$	$0^{\text{II}}_{\leftarrow}$			
$0^{\text{I}}_{\rightarrow}$	$0^{\text{II}}_{\rightarrow}$	$1^{\text{III}}_{\rightarrow}$	$0^{\text{III}}_{\leftarrow}$		
1	1	0	1	0	1

ÉTAPE 6 : Si x est un symbole marqué par l'exposant I ou II et est initialement issu de la ligne i et si y est son successeur dans le mot w_i, nous adjoignons à x le numéro de ligne dans lequel y se trouve après abaissement.

	$(1^{\text{I}}_{\leftarrow}, 2)$	$(0^{\text{II}}_{\leftarrow}, 1)$			
$(0^{\text{I}}_{\rightarrow}, 1)$	$(0^{\text{II}}_{\rightarrow}, 1)$	$1^{\text{III}}_{\rightarrow}$	$0^{\text{III}}_{\leftarrow}$		
1	1	0	1	0	1

La trace de la partie W est la suite des colonnes du tableau obtenue par la procédure ainsi décrite.

☐ 25 – Soit T la trace d'une partie. Décrire une construction qui produit une partie de trace T à partir de la trace T. (Il est suggéré de raisonner par récurrence.)

☐ 26 – Montrer que l'on peut majorer le nombre de lignes non vides dans la trace d'une partie par une constante indépendante de la dimension n du tablier ou de la longueur de la partie m. En déduire qu'une colonne de la trace d'une partie ne peut prendre qu'un nombre fini de valeurs distinctes.

Nous appelons Δ l'ensemble, fini, des valeurs colonnes que peuvent prendre les colonnes d'une trace de partie. Nous appelons $\mathcal{T}$ le langage des mots sur Δ qui représentent la trace d'une partie.

☐ 27 – Donner une caractérisation de l'ensemble des facteurs de longueur 2 de mots du langage $\mathcal{T} \subseteq \Delta^*$.

☐ 28 – Montrer qu'il existe un automate local qui reconnaît exactement le langage $\mathcal{T} \subseteq \Delta^*$.

☐ 29 – Montrer qu'il existe un automate fini qui reconnaît le langage $\mathcal{R} \subseteq \Sigma^*$ des mots de longueur quelconque de motifs résolubles.

Dans la question qui suit, le terme *complexité* désigne un ordre de grandeur asymptotique de la complexité en temps et dans le pire des cas.

☐ 30 – On s'intéresse au problème de déterminer si un motif sur le tablier unidimensionnel est résoluble ou non. Montrer que l'on peut déduire de l'automate construit à la question 29 un algorithme de complexité optimale pour résoudre ce problème.

Les résultats démontrés dans la section 2 ont été établis par B. Ravikumar pour un tablier rectangulaire de dimension $n \times n_0$ où n_0 est un entier fixé et n est un entier pouvant varier.

Fin de l'épreuve