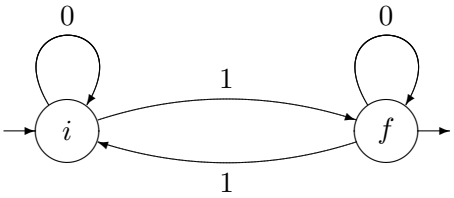
Version A

Exercice 1

Soit l'alphabet $B = \{0, 1\}$.

1) a) Soit l'ensemble L des mots sur B contenant un nombre impair d'occurrences de 1 et l'automate $\mathcal{A}$ défini par :



L'état i est initial.
L'état f est final.

Montrer que $\mathcal{A}$ reconnaît le langage L.

b) Montrer que le langage M constitué des mots sur $\{0, 1\}$ contenant un nombre pair de 0 et un nombre impair de 1 est reconnaissable.

2) On souhaite maintenant concevoir un circuit logique qui renvoie le nombre modulo 2 de 0 présents dans le mot passé en entrée au circuit.

On pourra utiliser les portes logiques élémentaires *et*, *ou*, *non* et *xor* (ou exclusif), que l'on représente simplement par :

Porte Non — Porte Et — Porte Ou — Porte Xor

a) Commençons par un mot de longueur 1 : $u = a$, avec $a \in \{0, 1\}$.

Construire un circuit $\mathcal{P}_0$ à une seule entrée a et une sortie s qui vaut 0 si u contient un nombre pair de 0 et 1 sinon.

b) Supposons conçu un circuit $\mathcal{P}_n$ qui prend en entrée un mot u de longueur 2^n et possède une sortie s qui vaut 0 si u contient un nombre pair de 0 et 1 sinon.

Construire, à l'aide de $\mathcal{P}_n$, un circuit $\mathcal{P}_{n+1}$ qui résout le même problème pour un mot de longueur 2^{n+1} en entrée.

c) Déterminer, lorsque n tend vers $+\infty$, un équivalent du nombre de portes logiques élémentaires utilisées dans $\mathcal{P}_n$.

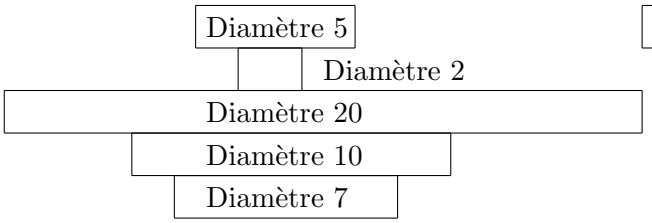
3) Soit N l'ensemble des mots u sur $\{0, 1\}$ tels que :

$$|u|_0 = 2\,|u|_1$$

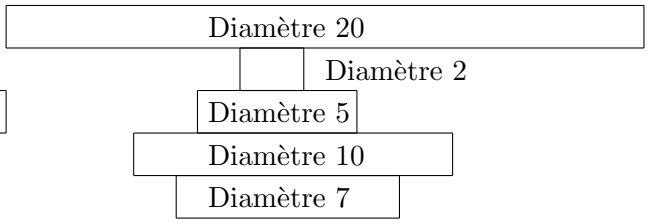
où $|u|_0$ et $|u|_1$ désignent respectivement le nombre d'occurrences de 0 et de 1 dans u.
Ce langage est-il reconnaissable ?

Exercice 2

On dispose d'une pile de rondelles de diamètres variés, que l'on souhaite ranger par ordre décroissant de taille, la plus large devant être en bas de la pile. Malheureusement, on ne peut manipuler la pile que « par morceaux » : la seule opération autorisée consiste à prendre la partie de la pile surmontant une certaine rondelle et à retourner d'un bloc tout le paquet.
Par exemple, si une pile contient des rondelles de diamètres 7, 10, 20, 2 et 5 et que l'on retourne le bloc commençant à la rondelle de diamètre 20, on obtient la pile de diamètres successifs 7, 10, 5, 2 et 20.



Pile au départ



Pile après retournement

Nous modéliserons la pile de rondelles par un tableau (ou vecteur) d d'entiers, indicé de 0 à $n-1$, où n est le nombre de rondelles, la case numéro i contenant le diamètre en millimètres de la rondelle i. La case numéro 0 correspond au bas de la pile.
En Caml comme en Pascal, nous supposerons la constante n définie.
En Caml, d est de type `int vect`, et en Pascal de type `Pile` défini par :

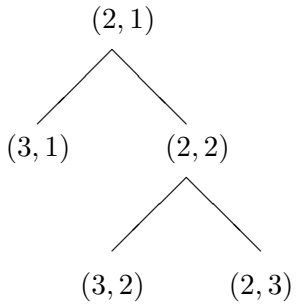
```
Pile = ARRAY[0..n-1] OF INTEGER
```

1) Écrire :
   - en Caml une fonction `retourne : int vect -> int -> unit` telle que `retourne d i` modifie le vecteur `d` pour « retourner » la partie supérieure de la pile à partir de la rondelle d'indice i (incluse) ;
   - en Pascal une procédure `retourne(VAR d : Pile ; i : INTEGER)` telle que `retourne(d,i)` modifie le tableau `d` pour « retourner » la partie supérieure de la pile à partir de la rondelle d'indice i (incluse).
2) Quel est, en nombre d'affectations dans `d`, le coût de ce retournement ?
3) Comment faire pour mettre en bas de pile la rondelle la plus large ?
4) Écrire
   - en Caml une fonction `place : int vect -> int -> unit` telle que `place d i` modifie le vecteur `d` pour mettre à l'indice i la rondelle la plus large de la partie de la pile surmontant la rondelle d'indice i (incluse) ;
   - en Pascal une procédure `place(VAR d : Pile ; i : INTEGER)` telle que `place(d,i)` modifie le tableau `d` pour mettre à l'indice i la rondelle la plus large de la partie de la pile surmontant la rondelle d'indice i (incluse) ;
5) En déduire, selon le langage choisi, une fonction ou une procédure qui trie `d`.
6) Quel est le coût au pire de cette méthode, en nombre de retournements d'un morceau de pile ?

Version B

Exercice 1

Alice et Bob jouent à un jeu dans lequel chacun d'entre eux a une chance sur deux de remporter chaque partie. On convient que le gagnant est le premier qui a remporté trois parties. À chaque instant, nous représenterons la situation par un couple (a, b), où a (resp. b) est le nombre de parties remportées par Alice (resp. Bob).
Supposons par exemple qu'Alice ait déjà remporté deux parties et Bob une seule : le score actuel est donc $(2, 1)$. Après le prochain coup, il peut être $(3, 1)$, auquel cas Alice a gagné et le jeu est fini, ou $(2, 2)$, et le jeu continue. On représente ceci par l'arbre des scores possibles :



Dans cet arbre, la racine, étiquetée $(2, 1)$, se trouve à la profondeur 0. La feuille étiquetée $(3, 1)$ est la seule feuille à la profondeur 1, et il existe à la profondeur 2 deux feuilles, étiquetées $(3, 2)$ et $(2, 3)$.
On remarque que, si $f(p)$ désigne le nombre de feuilles à la profondeur p, cet arbre vérifie :

$$\sum_{k=0}^{2} f(k)2^{-k} = 1.$$

1) a) Dessiner l'arbre que l'on obtient de la même façon à partir du score $(1, 0)$ (i.e. lorsqu'Alice a gagné une partie et Bob aucune).

   b) Vérifier que cet arbre satisfait encore la relation $\sum_{k=0}^{4} f(k)2^{-k} = 1$.

2) Montrer que, pour tout arbre binaire, de hauteur quelconque h, on a :

$$\sum_{k=0}^{h} f(k)2^{-k} = 1 \text{ où } f(k) \text{ désigne le nombre de feuilles à la profondeur } k.$$

Exercice 2

On souhaite gérer de façon aussi automatique que possible la création d'un colloscope pour une classe de la filière MP-MP*.
On suppose les étudiants répartis en un nombre pair $2n$ de groupes (quitte à avoir créé des groupes vides ou partiellement vides, par exemple avec deux étudiants, en cas de besoin), avec $n \geqslant 1$. Chaque groupe est repéré par son numéro, compris entre 0 et $2n - 1$.
Chaque groupe doit avoir chaque semaine une interrogation de mathématiques, et une interrogation de physique-chimie ou de langue en alternance une semaine sur deux. La difficulté est que la classe est bilingue : p groupes sont constitués d'anglicistes, et $2n - p$ de germanistes. On supposera que les groupes d'anglicistes sont ceux numérotés de 0 à $p - 1$, les groupes de germanistes étant numérotés de p à $2n - 1$.
On a donc recruté $2n$ interrogateurs de mathématiques, n interrogateurs de physique-chimie, q interrogateurs d'anglais et r d'allemand, où $q = \left\lceil \frac{p}{2} \right\rceil$ et $r = \left\lceil \frac{2n-p}{2} \right\rceil$ (la notation $\lceil x \rceil$ désigne la partie entière supérieure de x, i.e. le plus petit entier supérieur à x). Les interrogateurs sont repérés par un numéro compris entre 0 et respectivement $2n-1$ ou $n-1$ pour les mathématiques ou la physique, 0 et $q - 1$ pour l'anglais, q et $q + r - 1$ pour l'allemand.

Le colloscope lui-même est constitué de trois matrices m, p et ℓ telles que le terme $m_{i,j}$ (resp. $p_{i,j}$ ou $\ell_{i,j}$) de m (resp. p ou ℓ) indique le numéro de l'interrogateur de mathématiques (resp. physique-chimie ou langue) que le groupe i doit rencontrer durant la semaine j, avec $0 \leqslant j \leqslant s-1$ où s est le nombre de semaines de l'année. Lorsque la case devrait être vide (une semaine par exemple sans interrogation de langue), on lui donnera la valeur -1.

En Caml comme en Pascal, nous supposerons les constantes n, p, q, r, s définies.
En Caml, m, p et ℓ seront de type `int vect vect`. On rappelle qu'on crée une matrice de taille $a \times b$ dont tous les termes sont initialisés à la valeur c par `make_matrix a b c`. On accède au terme $m_{i,j}$ par `m.(i).(j)`.

En Pascal, on définit le type

```
Matrice = ARRAY[0..2*n-1, 0..s-1] OF INTEGER
```

m, p et ℓ sont du type `Matrice`. On rappelle qu'on accède au terme $m_{i,j}$ par `m[i, j]`.

On suppose pour commencer qu'il n'existe aucune incompatibilité horaire entre les différents interrogateurs.

1) On veut que le groupe 0 passe en mathématiques d'abord avec l'interrogateur 0, puis le 1 puis le 2 ..., tandis que le groupe 1 passera avec l'interrogateur 1 puis avec le 2 ...

   a) Quelle doit être l'expression de $m_{i,j}$ en fonction de i, j et n ?

   b) Écrire :
   - en Caml une fonction `maths : int -> int -> int vect vect` qui prend en arguments n et s et renvoie la matrice m correctement remplie ;
   - en Pascal une procédure `maths(VAR m : Matrice)` qui remplit correctement la matrice m.

2) Justifier de même que l'on utilise pour p le terme général suivant, sachant que l'on souhaite que le groupe 0 passe avec l'interrogateur 0, puis 1 ... :

$$p_{i,j} = \begin{cases} \dfrac{i+j}{2} \pmod{n} \text{ si } i \text{ et } j \text{ sont tous deux pairs ou impairs} \\ -1 \text{ sinon} \end{cases}$$

   et écrire, selon le langage choisi, une fonction ou une procédure `physique` qui assure un remplissage correct de la matrice p.

3) On supposera désormais que la matrice ℓ pour les langues a été remplie.

   Il y a en général des incompatibilités entre les horaires des différents interrogateurs. Il faut donc rechercher les couplages qui poseront problème après ce premier remplissage. Nous nous intéresserons aux incompatibilités entre interrogateurs de mathématiques et de langues.

   a) Décrire (en français ou en pseudo-langage clair) un algorithme prenant en arguments les numéros a d'un interrogateur de mathématiques et b d'un interrogateur de langues et renvoyant, s'il en existe, un couple d'indices (i, j) tel que :

$$m_{i,j} = a \text{ et } \ell_{i,j} = b.$$

   S'il n'existe pas de tel couple, on renverra $(-1, -1)$.

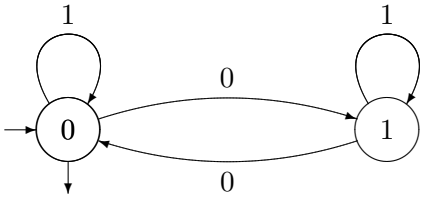
   b) Préciser le coût au pire de l'algorithme ci-dessus en nombre d'accès à un élément de matrice.

Corrigé, version A

Exercice 1

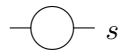
1) a) Pour un mot de longueur 1 : $u = a \in \{0, 1\}$:
– si $a = 0$, a contient un nombre pair de 1. Or l'automate s'arrête sur 0 ;
– si $a = 1$, a contient un nombre impair de 1. Or l'automate s'arrête sur 1.
Hypothèse de récurrence : pour tout mot $u = a_1 \ldots a_k$ de longueur k, l'automate s'arrête sur 0 (resp. sur 1) après avoir lu u lorsque celui-ci contient un nombre pair (resp. impair) de 1.
Prenons un mot $u = a_1 \ldots a_{k+1}$ de longueur $k + 1$. On peut appliquer l'hypothèse de récurrence au mot $v = a_1 \ldots a_k$.
S'il y a un nombre pair de 1 dans v, on est sur 0 après l'avoir lu, et quand on lit a_{k+1} :
– soit $a_{k+1} = 0$: on reste sur 0 et il y a bien un nombre pair de 1 dans u ;
– soit $a_{k+1} = 1$: on passe sur 1 et il y a bien un nombre impair de 1 dans u.
S'il y a un nombre impair de 1 dans v, on est sur 1 après l'avoir lu, et quand on lit a_{k+1} :
– soit $a_{k+1} = 0$: on reste sur 1 et il y a bien un nombre impair de 1 dans u ;
– soit $a_{k+1} = 1$: on revient sur 0 et il y a bien un nombre pair de 1 dans u.
Dans tous les cas, l'hypothèse de récurrence est vérifiée.
Ainsi, le mot u est reconnu si et seulement si on s'arrête sur 1, i.e. si et seulement si u contient un nombre impair de 1. $\boxed{\mathcal{A} \text{ reconnaît } L}$.

b) De la même façon, l'automate :



reconnaît le langage L' constitué des mots qui contiennent un nombre pair de 0. Alors $\boxed{M = L \cap L' \text{ est reconnaissable.}}$

2) a) On a : $s = \neg a$.



b) Soit u un mot de longueur 2^{n+1} que l'on coupe au milieu en deux mots de longueurs 2^n notés v et w : $u = vw$.
On prend deux exemplaires du circuit $\mathcal{P}_n$ auxquels on envoie v et w. Notons s_v et s_w leurs sorties respectives.
Alors si s est la sortie de $\mathcal{P}_{n+1}$, on doit avoir :

$$s = (s_v \oplus s_w)$$

où $\oplus$ désigne l'opérateur `xor`.

c) Notons p_n le nombre de portes utilisées dans $\mathcal{P}_n$. On a alors :

$$p_0 = 1 \text{ et } p_{n+1} = 2p_n + 1$$

d'où par une récurrence immédiate $\boxed{p_n = \sum_{k=0}^{n} 2^k = 2^{n+1} - 1 \underset{n \to +\infty}{\sim} 2^{n+1}}$.

3) Supposons N reconnaissable. D'après le lemme de l'étoile, il existe un entier $n \geqslant 1$ tel que, quel que soit le mot $w \in N$ de longueur supérieure ou égale à n, w admet une factorisation de la forme $w = u.x.v$ où $u.x$ est de longueur au plus n, x est non vide et $u.x^*.v$ est contenu dans N.
Prenons le mot $w = 1^n 0^{2n}$. C'est bien un mot de N. Dans la factorisation précédente, x doit être de la forme 1^q avec $q \geqslant 1$, donc $ux^2v = 1^{n+q}0^{2n}$ n'est plus un mot de N : il y a contradiction.
Ainsi $\boxed{N \text{ n'est pas reconnaissable}}$.

Exercice 2

1) – En Caml : il faut échanger les termes d'indices $i+j$ et $n-1-j$ tant que $i+j < n-1-j$ donc pour j variant de 0 à $\left\lfloor \frac{n-i-1}{2} \right\rfloor$. Écrivons d'abord une fonction auxiliaire qui échange les termes d'indices i et j puis la fonction demandée :

```
let echange d i j = let tampon=d.(i) in
  d.(i) <- d.(j) ;
  d.(j) <- tampon ;;

let retourne d i = for j=0 to (n-i-1)/2 do echange d (i+j) (n-1-j) done ;;
```

– En Pascal, le principe est le même :

```
PROCEDURE echange(VAR d : Pile ; i, j : INTEGER);
VAR tampon : INTEGER ;
    BEGIN
        tampon := d[i] ;
        d[i] := d[j] ;
        d[j] := tampon
    END;

PROCEDURE retourne(VAR d : Pile ; i : INTEGER);
VAR j : INTEGER ;
    BEGIN
         FOR i := 0 TO (n-i-1)/2 DO echange(d, i+j, n-1-j) ;
    END ;
```

2) On fait deux affectations pour chaque appel à `echange`, et celle-ci est appelée $\left\lfloor \frac{n-i+1}{2} \right\rfloor$ fois, d'où un coût $\boxed{2\left\lfloor \frac{n-i+1}{2} \right\rfloor}$.

3) On trouve un indice i_0 tel que $d(i_0)$ soit maximum (il peut y en avoir plusieurs). On retourne alors la partie de pile au-dessus de i_0, ce qui la met tout en haut du tas, puis on retourne à nouveau toute la pile.

4) On applique le principe précédent, mais en faisant comme si la place i était le bas de la pile.
– En Caml, version sans références : on écrit une fonction auxiliaire qui recherche i_0 :

```
let place d i =
   let rec maxi j i0 = if j = n then i0 else
        if d.(j) > d.(i0) then maxi (j+1) j else maxi (j+1) i0
   in retourne d (maxi (i+1) i) ;
   retourne d i;;
```

En Caml, version avec références :

```
let place d i = let i0=ref i in
  for j=(i+1) to (n-1) do if d.(j) > d.(!i0) then i0:=j done;
  retourne d !i0 ;
  retourne d i;;
```

– En Pascal :

```
PROCEDURE place(VAR d : Pile ; i : INTEGER);
VAR i0, j : INTEGER ;
    BEGIN
         i0:=i ;
         FOR j := i+1 TO n-1 DO
              IF d[j] > d[i0] THEN i0:=j ;
         retourne(d, i0) ;
         retourne(d, i)
    END ;
```

5) Pour trier, il ne reste plus qu'à utiliser `place` pour i variant de 0 à $n-2$.

```
let tri d = for i=0 to n-2 do place d i done ;;
```

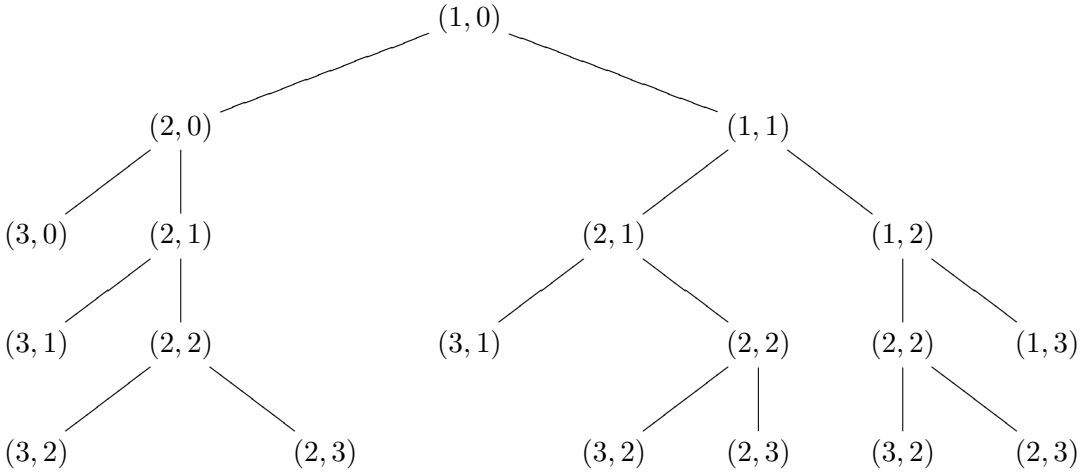
```
PROCEDURE tri(VAR d : Pile);
VAR i : INTEGER ;
    BEGIN
        FOR i := 0 TO n-2 DO place(d, i)
    END ;
```

6) Chaque appel à `place` fait deux retournements, soit un coût de $2(n-1)$ retournements.

Corrigé, version B

Exercice 1

1) a)



b) Alors $f(0) = f(1) = 0$, $f(2) = 1$, $f(3) = 3$, $f(4) = 6$ donc $\boxed{\sum_{k=0}^{4} f(k)2^{-k} = \frac{1}{4} + \frac{3}{8} + \frac{6}{16} = 1}$.

2) On travaille par récurrence sur h. C'est évidemment vrai pour $h = 0$ (arbre réduit à sa racine).

Supposons le résultat vrai pour tout arbre de hauteur au plus n, et prenons un arbre A de hauteur $n+1$. Notons g et d ses sous-arbres gauche et droit, $g(k)$ et $d(k)$ le nombre de feuilles à la profondeur k dans g et d.

Une feuille à la profondeur k dans g ou d se trouve à la profondeur $k+1$ dans A. Ainsi, en notant $f(k)$ le nombre de feuilles de A à la profondeur k :

$$f(0) = 0 \text{ et pour } k \geqslant 1, f(k) = g(k-1) + d(k-1).$$

Alors $\sum_{k=0}^{n+1} f(k)2^{-k} = \sum_{k=1}^{n+1} g(k-1)2^{-k} + \sum_{k=1}^{n+1} d(k-1)2^{-k} = \sum_{k=0}^{n} g(k)2^{-k-1} + \sum_{k=0}^{n} d(k)2^{-k-1}$.

g et d sont de hauteur au plus n, ce qui permet de leur appliquer l'hypothèse de récurrence :

$$\sum_{k=0}^{n} g(k)2^{-k} = \sum_{k=0}^{n} d(k)2^{-k} = 1$$

donc $\sum_{k=0}^{n+1} f(k)2^{-k} == \frac{1}{2} + \frac{1}{2} = 1$ et l'hypothèse de récurrence est vérifiée.

Ainsi, pour tout arbre binaire, de hauteur quelconque h : $\boxed{\sum_{k=0}^{h} f(k)2^{-k} = 1.}$

Exercice 2

1) a) On prend $\boxed{m_{i,j} = (i+j) \pmod{(2n)}}$.

b) – En Caml :

```
let maths n s = let m=make_matrix (2*n) s (-1) in
   for i=0 to 2*n-1 do
      for j = 0 to s-1 do
          m.(i).(j) <- (i+j) mod (2*n)
      done;
   done;
m;;
```

– En Pascal :

```
PROCEDURE maths(VAR m:Matrice) ;
VAR i, j : INTEGER ;
BEGIN
   FOR i:=0 TO 2*n-1 DO
      FOR j:=0 TO s-1 DO
          m[i,j] := (i+j) MOD (2*n)
END;
```

2) Les groupes de numéros pairs (resp. impairs) ont bien colle une semaine sur deux : les semaines paires (resp. impaires).

À chaque fois que i ou j augmente de deux unités (i.e. pour le groupe suivant ou la quinzaine suivante), le numéro du colleur augmente bien de 1.

Enfin, le modulo n permet de faire tourner le tout avec une périodicité de n colleurs.

– En Caml :

```
let physique n s = let p=make_matrix (2*n) s (-1) in
   for i=0 to 2*n-1 do
      for j = 0 to s-1 do
         if (i+j) mod 2 = 0 then p.(i).(j) <- ((i+j)/2) mod n
      done;
   done;
p;;
```

– En Pascal :

```
PROCEDURE physique(VAR p:Matrice) ;
VAR i, j : INTEGER ;
BEGIN
   FOR i:=0 TO 2*n-1 DO
      FOR j:=0 TO s-1 DO
          IF (i+j) MOD 2 = 0 THEN p[i,j] := ((i+j)/2) MOD n
          ELSE p[i, j] := -1
END;
```

3) a) On peut faire deux boucles emboîtées pour i variant de 0 à $2n-1$ et j variant de 0 à $s-1$ qui, pour chaque couple (i,j), lisent $m_{i,j}$ et $\ell_{i,j}$ et renvoient le couple (i,j) lorsque cela coïncide avec a et b. Mais cela donne un coût de l'ordre de ns alors qu'on peut le faire seulement en $O(n)$ par la méthode ci-dessous.

On remarque que $m_{i,j} = (i+j) \pmod{(2n)}$ donc on sait que $m_{i,j}$ est égal à a exactement sur la « diagonale » définie par la relation :

$$j = a - i \pmod{(2n)}.$$

Une boucle double est ainsi inutile : il suffit de parcourir cette diagonale. L'algorithme est donc le suivant :

– Initialiser i à 0

– Prendre $j = a - i \pmod{(2 * n)}$

– Si $\ell_{i,j} = b$, renvoyer le couple (i, j)
– Sinon, si $i < 2n$, recommencer avec $i + 1$ et si $i = 2n$, renvoyer $(-1, -1)$.

b) Cf. réponse ci-dessus.