

# Lumière

Épreuve pratique d'algorithmique et de programmation

Concours commun des Écoles normales supérieures

Durée de l'épreuve : 3 heures 30 minutes

Juin/Juillet 2024

**ATTENTION !**

N'oubliez en aucun cas de recopier votre  $u_0$   
à l'emplacement prévu sur votre fiche réponse

## Important.

Il vous a été donné un numéro  $u_0$  qui servira d'entrée à vos programmes. Les réponses attendues sont généralement courtes et doivent être données sur la fiche réponse fournie à la fin du sujet. À la fin du sujet, vous trouverez en fait deux fiches réponses. La première est un exemple des réponses attendues pour un  $\tilde{u}_0$  particulier (précisé sur cette même fiche et que nous notons avec un tilde pour éviter toute confusion !). Cette fiche est destinée à vous aider à vérifier le résultat de vos programmes en les testant avec  $\tilde{u}_0$  au lieu de  $u_0$ . Vous indiquerez vos réponses (correspondant à votre  $u_0$ ) sur la seconde et vous la remettrez à l'examineur à la fin de l'épreuve.

En ce qui concerne la partie orale de l'examen, lorsque la description d'un algorithme est demandée, vous devez présenter son fonctionnement de façon schématique, courte et précise. Vous ne devez en aucun cas recopier le code de vos procédures !

Quand on demande la complexité en temps ou en mémoire d'un algorithme en fonction d'un paramètre  $n$ , on demande l'ordre de grandeur en fonction du paramètre, par exemple :  $O(n^2)$ ,  $O(n \log n)$ ,...

Il est recommandé de commencer par lancer vos programmes sur de petites valeurs des paramètres et de **tester vos programmes sur des petits exemples que vous aurez résolus préalablement à la main ou bien à l'aide de la fiche réponse type fournie en annexe**. Enfin, il est recommandé de lire l'intégralité du sujet avant de commencer afin d'effectuer les bons choix de structures de données dès le début.



Les parties 1, 2 et 4 doivent être traitées en OCAML, et la partie 3 en C. Ces consignes de langage seront répétées au début des parties concernées. Elles doivent impérativement être suivies.

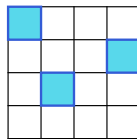
Pour la partie 3, un fichier `base.c` est fourni. Il contient quelques fonctions prédéfinies dont vous pouvez vous servir si vous le souhaitez. Il est fortement conseillé de faire une copie de ce fichier, pour ne pas risquer de l'effacer par erreur.

La partie 1 présente le problème étudié, ainsi que la génération des données qui seront utilisées dans les parties 2 et 4, et doit donc être traitée avant ces dernières. Les parties 2, 3 et 4 sont indépendantes.

## 1 Préliminaires

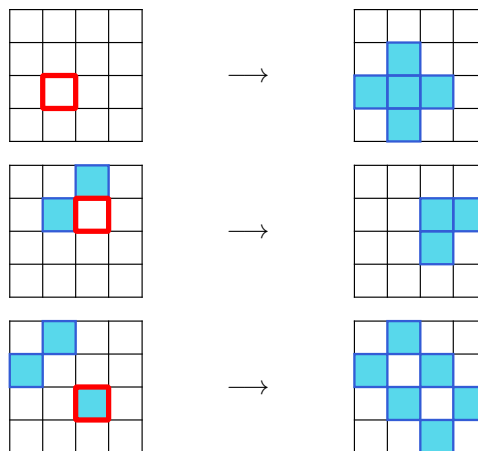
### 1.1 Un problème d'éclairage

On s'intéresse à un puzzle qui se joue sur une grille carrée de taille  $n \times n$ , pour un entier  $n \geq 1$ . Chaque case de la grille est une lampe, qui a deux états possibles : elle peut être allumée ou bien éteinte. On peut voir ci-dessous l'exemple d'une grille  $4 \times 4$ , dans laquelle trois cases sont allumées.

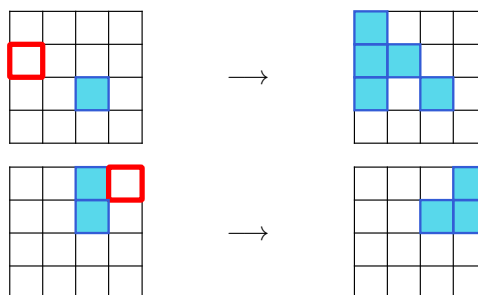


Une instance du puzzle est une grille donnée, dont certaines cases sont allumées et d'autres éteintes. Le but du jeu est d'allumer les lampes de façon à reproduire cette grille, en partant d'une grille initialement complètement éteinte. On appellera souvent dans la suite cette grille à atteindre la *cible*.

Pour cela, la seule action autorisée est d'appuyer sur les cases de la grille. Appuyer sur une case a pour effet d'inverser son état, ainsi que celui des quatre cases directement adjacentes, au dessus, en dessous, à gauche, et à droite : une case éteinte s'éclaire, une case allumée s'éteint. Dans les trois exemples ci-dessous, si l'on appuie sur la case marquée dans la grille de gauche, on obtient la grille de droite.



Appuyer sur une case modifie ainsi l'état d'au plus cinq cases, disposées en forme de croix. Si la case sur laquelle on appuie est située au bord de la grille, cette croix sera tronquée, comme on peut le voir dans les deux exemples suivants.



Résoudre le puzzle consiste à trouver une séquence de cases sur lesquelles appuyer pour reproduire la grille cible. Notons qu'il n'y a *a priori* en général ni existence ni unicité de la solution.

On remarque rapidement qu'appuyer plusieurs fois sur la même case est inutile, et que l'ordre dans lequel les cases sont choisies n'a pas d'importance. Plutôt qu'une séquence ordonnée, on recherchera par conséquent un ensemble de cases qui permettront d'obtenir la grille voulue.

Par exemple, si la grille cible est celle ci-dessous à gauche, l'ensemble des cases marquées sur la grille de droite est une solution : appuyer une fois sur chacune de ces cases, dans n'importe quel ordre, produira la grille voulue.

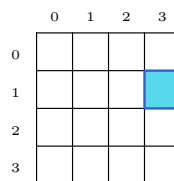


## 1.2 Notations

On rappelle que, pour tous  $a, b \in \mathbb{Z}$  tels que  $b \neq 0$ ,  $a \bmod b$  désigne le reste de la division euclidienne de  $a$  par  $b$ , c'est-à-dire le plus petit entier naturel  $r$  tel qu'il existe un entier  $q$  vérifiant  $a = b \times q + r$ . Par ailleurs, pour un entier naturel  $n$ , on note  $\llbracket 0, n \rrbracket$  l'ensemble des entiers jusqu'à  $n$  :  $\{0, 1, 2, \dots, n-1, n\}$ .

Dans une grille  $G$  de taille  $n \times n$ , on identifiera chaque case avec sa position, c'est-à-dire le couple d'entiers  $(i, j) \in \llbracket 0, n-1 \rrbracket^2$  formé de sa ligne et sa colonne, en commençant en haut à gauche de  $G$ .

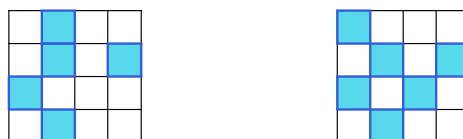
Par exemple, la case  $(1, 3)$  est allumée dans la grille ci-dessous.



On note  $\mathcal{A}(G)$  l'ensemble des cases allumées de  $G$ . Pour deux grilles  $G, G'$  de même taille, la *distance* de  $G$  à  $G'$ , notée  $d(G, G')$ , est définie comme le nombre de cases n'ayant pas le même état dans les deux grilles :

$$d(G, G') = \left| (\mathcal{A}(G) \setminus \mathcal{A}(G')) \cup (\mathcal{A}(G') \setminus \mathcal{A}(G)) \right|$$

Par exemple, les deux grilles ci-dessous sont à distance 3.



Pour un entier  $n \geq 1$  et un ensemble de cases  $C \subset \llbracket 0, n-1 \rrbracket^2$ , on appellera l'image de  $C$ , notée  $\mathcal{F}(C)$ , la grille obtenue en appuyant une fois sur chaque case de  $C$ , en partant d'une grille vide.

Pour tout entier  $n \geq 1$ , on considère l'ordre lexicographique sur les cases de  $\llbracket 0, n-1 \rrbracket^2$  :  $(i, j) \leq (i', j')$  lorsque  $i < i'$ , ou  $i = i' \wedge j \leq j'$ .

### 1.3 Génération de données

Les trois questions qui suivent doivent être traitées en OCAML.

Étant donné  $u_0$ , on définit par récurrence :

$$\forall i \in \mathbb{N}. u_{i+1} = (569 \times u_i) \bmod 2\,424\,259$$

**Question 1** Écrire un programme qui calcule la suite  $u$ , et en donner les valeurs suivantes :

- a)**  $u_1 \bmod 10\,000$       **b)**  $u_{128} \bmod 10\,000$       **c)**  $u_{2024} \bmod 10\,000$       **d)**  $u_{123\,456} \bmod 10\,000$

**Indication.** Il pourra être judicieux, afin d'obtenir de meilleures performances dans les questions suivantes, de précalculer les valeurs de  $u_n$  jusqu'à un  $n$  assez grand : on pourra les calculer au début du programme, et les mémoriser dans un tableau, pour ne pas devoir refaire le calcul à chaque fois.

Pour tous entiers naturels  $n, m, p$ , avec  $n > 0$  et  $m \leq p$ , on notera  $G_{n,m,p}$  la grille de taille  $n \times n$  dont les cases qui sont allumées sont celles aux positions  $(i, j)$  telles que  $(u_{n^2+89m+71p+ni+j} \bmod p) < m$ .

**Question 2** Écrire un programme qui calcule  $G_{n,m,p}$ . Donner, pour les entrées suivantes, la somme :

$$\left( \sum_{(i,j) \in \mathcal{A}(G_{n,m,p})} i + j \right) \bmod 10\,000.$$

- a)**  $n = 3, m = 1, p = 4$       **b)**  $n = 10, m = 3, p = 4$       **c)**  $n = 100, m = 2, p = 5$

**Question à développer pendant l'oral 1** Décrire la structure de données utilisée pour représenter les grilles. Évaluer sa complexité en espace, ainsi que la complexité en temps des opérations consistant à appuyer sur une case et à parcourir la liste des cases allumées.

**Question 3** Écrire un programme qui calcule la distance  $d$ . Donner les distances suivantes.

- a)**  $d(G_{3,1,4}, G_{3,1,2})$       **b)**  $d(G_{10,3,4}, G_{10,1,2})$       **c)**  $d(G_{100,2,5}, G_{100,3,7})$

## 2 Résolution du puzzle

Cette partie doit être traitée en OCAML.

Cette partie est consacrée à la résolution du puzzle décrit dans la partie précédente.

### 2.1 Recherche exhaustive

On se propose, dans un premier temps, d'adopter une approche exhaustive, consistant à essayer un par un tous les ensembles de cases possibles, jusqu'à trouver une solution. Plus précisément, puisqu'il n'existe *a priori* pas forcément de solution, on recherchera dans cette question un ensemble de cases  $C$  qui permet de se rapprocher le plus possible de la grille cible  $G$ , c'est-à-dire tel que  $d(G, \mathcal{F}(C))$  est minimal. Si plusieurs ensembles de cases peuvent convenir, on en renverra un qui rend maximale la valeur

$$\left( \sum_{(i,j) \in C} i + j \right) \bmod 10\,000.$$

**Question 4** Écrire un programme qui effectue la recherche exhaustive décrite ci-dessus. Pour chacune des grilles suivantes, calculer pour l'ensemble de cases  $C$  obtenu la valeur

$$\left( \sum_{(i,j) \in C} i + j \right) \bmod 10\,000.$$

a)  $G_{3,1,2}$

b)  $G_{4,1,4}$

c)  $G_{4,3,4}$

**Question à développer pendant l'oral 2** Décrire le fonctionnement de votre programme. Évaluer sa complexité en temps.

## 2.2 Algorithme glouton

Pour obtenir un programme plus rapide, on propose à présent de recourir à un algorithme glouton. Il s'agit de choisir successivement à chaque étape une case non encore utilisée qui nous rapproche le plus possible de la cible – ou nous en éloigne le moins. À une étape donnée, si  $C$  est l'ensemble des cases déjà sélectionnées, on choisira donc une case  $c \notin C$ , qui rend  $d(G, \mathcal{F}(C \cup \{c\}))$  aussi petit que possible, pour l'ajouter à  $C$ . Lorsque l'on a le choix, on prendra toujours la case  $c$  la plus grande pour l'ordre lexicographique. On s'arrête lorsque toutes les cases ont été sélectionnées.

Au cours de l'algorithme,  $C$  se voit donc progressivement augmenté case par case, jusqu'à toutes les contenir. À la fin, on examine les valeurs successives de l'ensemble  $C$  au cours de l'exécution. On renvoie celle qui avait permis d'obtenir la distance  $d(G, \mathcal{F}(C))$  la plus petite. Si l'on a le choix, on renverra la première telle valeur de  $C$ , c'est-à-dire celle qui contient le moins de cases.

**Question 5** Écrire un programme qui applique l'algorithme glouton décrit ci-dessus. Pour chacune des grilles suivantes, calculer l'ensemble de cases  $C$  obtenu, et donner la valeur

$$\left( \sum_{(i,j) \in C} i + j \right) \bmod 10\,000.$$

a)  $G_{3,3,4}$

b)  $G_{20,2,5}$

c)  $G_{40,5,7}$

**Question à développer pendant l'oral 3** Décrire le fonctionnement de votre programme. Évaluer sa complexité en temps et en espace.

**Question à développer pendant l'oral 4** L'algorithme glouton calcule-t-il toujours une solution, si elle existe ? Si oui, justifier, si non, donner un contre-exemple.

## 2.3 Algèbre linéaire

Ce puzzle peut être résolu efficacement en appliquant des techniques d'algèbre linéaire, permettant de déterminer si une solution existe, et le cas échéant de la calculer.

Pour cela, on modélise le puzzle par un système d'équations linéaires sur le corps  $\mathbb{Z}/2\mathbb{Z}$ .

Considérons une grille cible  $G$  de taille  $n \times n$ . On posera  $N = n^2$ .  $G$  peut être vue comme un vecteur colonne  $V_G = (g_k)_{0 \leq k \leq N-1}$  dans  $(\mathbb{Z}/2\mathbb{Z})^N$ , dont la  $(ni + j)$ -ième composante vaut 1 si la case  $(i, j)$  de  $G$  est allumée, et 0 sinon. Autrement dit :

$$\forall (i, j) \in \llbracket 0, n-1 \rrbracket^2. g_{ni+j} = 1 \Leftrightarrow (i, j) \in \mathcal{A}(G).$$

Par exemple, la grille ci-dessous à gauche sera représentée par le vecteur à droite.

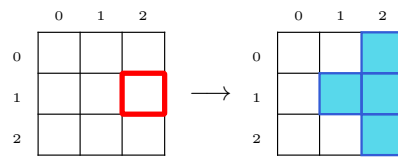
$$G = \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & \text{bleu} & \text{blanc} & \text{blanc} \\ 1 & \text{blanc} & \text{blanc} & \text{bleu} \\ 2 & \text{bleu} & \text{bleu} & \text{blanc} \end{array} \quad V_G = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

De la même manière, on peut voir un ensemble de cases  $C$  comme le vecteur colonne  $V_C = (c_k)_{0 \leq k \leq N-1}$  dont la  $(ni + j)$ -ième composante vaut 1 lorsque la case  $(i, j)$  est dans  $C$  :

$$\forall (i, j) \in \llbracket 0, n-1 \rrbracket^2. c_{ni+j} = 1 \Leftrightarrow (i, j) \in C.$$

On construit également une matrice de taille  $N \times N$ ,  $M = (m_{k,l})_{0 \leq k, l \leq N-1}$ , telle que pour tous entiers naturels  $i, j, i', j'$  inférieurs à  $n-1$ ,  $m_{ni+j, ni'+j'}$  vaut 1 si et seulement si l'état de la case  $(i, j)$  dans  $G$  s'inverse lorsque l'on appuie sur la case  $(i', j')$ .

Par exemple pour  $n = 3$ ,  $M$  est la matrice  $9 \times 9$  suivante :

$$M = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & \mathbf{0} & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$


Comme on le voit à droite, la case  $(0, 2)$  change d'état lorsque l'on appuie sur  $(1, 2)$ , et par conséquent  $m_{2,5} = 1$ . En revanche, la case  $(2, 0)$  ne change pas d'état, et par conséquent  $m_{6,5} = 0$ .

Résoudre le puzzle revient à résoudre le système linéaire de  $N$  équations suivant, où  $X$  est un vecteur colonne de  $N$  inconnues :

$$M \cdot X = V_G.$$

En effet, pour tout ensemble de cases  $C$ , on peut montrer (et on admettra dans la suite) que

$$M \cdot V_C = V_G \Leftrightarrow \mathcal{F}(C) = G.$$

**Question à développer pendant l'oral 5** Justifier le résultat précédent : pourquoi les solutions du puzzle et du système linéaire sont-elles les mêmes ? On n'attend pas une démonstration détaillée, mais une explication informelle du lien entre les deux.

Il ne reste donc plus qu'à résoudre ce système linéaire. Ceci peut être fait grâce à la méthode bien connue du pivot de Gauss.

**Question 6** Écrire un programme qui utilise la méthode du pivot de Gauss pour résoudre le puzzle. Pour chacune des grilles G suivantes, calculer une solution C du puzzle, et donner la somme

$$\left( \sum_{(i,j) \in C} i + j \right) \bmod 10\,000.$$

a) G<sub>6,4,7</sub>

b) G<sub>31,4,7</sub>

c) G<sub>43,4,7</sub>

d) G<sub>52,4,7</sub>

**Indication.** Pour les valeurs numériques demandées, le puzzle a toujours une et une seule solution.

**Question à développer pendant l'oral 6**

Expliquer le fonctionnement de votre programme. Évaluer sa complexité en temps.

### 3 Matrices en C

Cette partie doit être traitée en C.

À la fin de la partie 2, on a été amené à représenter le problème par un système d'équations linéaires, dont la résolution permettait de trouver une solution au puzzle.

La matrice associée à ce système linéaire avait une structure particulière : ses coefficients étaient majoritairement nuls, et les coefficients non nuls étaient ramassés autour de la diagonale. On peut le voir par exemple dans la matrice  $9 \times 9$  donnée en 2.3.

Les matrices de ce type, que l'on appelle parfois *matrices bandes*, possèdent des propriétés intéressantes en termes algorithmiques : certains algorithmes peuvent tirer parti de leur structure, pour être plus efficaces que dans le cas général.

On étudiera, dans cette partie, la représentation de matrices bandes en C. On considèrera des matrices dont les coefficients sont 0 ou 1.

#### 3.1 Données de test

Pour le test numérique des programmes de cette partie, on utilisera de nouveau des données générées à partir de la suite  $u$ . On définit pour tous entiers naturels  $n, m, p$ , avec  $n \geq 1$  et  $m \leq p$ , la matrice symétrique  $A_{n,m,p} = (a_{i,j})_{0 \leq i,j \leq n-1}$  de taille  $n \times n$ , telle que

$$\forall (i, j) \in \llbracket 0, n-1 \rrbracket^2. a_{i,j} = \begin{cases} 1 & \text{si } i \leq j \text{ et } (u_{n^2+89m+71p+ni+j} \bmod p) < m; \\ 0 & \text{si } i \leq j \text{ et } (u_{n^2+89m+71p+ni+j} \bmod p) \geq m; \\ a_{j,i} & \text{si } i > j. \end{cases}$$

**Question 7** Écrire un programme qui calcule la matrice  $A_{n,m,p}$ . Pour les valeurs de  $n, m, p$  suivantes, si  $(a_{i,j})_{i,j}$  désignent les coefficients de  $A_{n,m,p}$ , donner la somme

$$\left( \sum_{0 \leq i,j \leq n-1} (ni + j) \times a_{i,j} \right) \bmod 10\,000.$$

a)  $n=3, m=1, p=2$

b)  $n=20, m=3, p=4$

c)  $n=100, m=3, p=4$

d)  $n=1\,000, m=1, p=1\,000$



### 3.2 Largeur de bande

Soit  $M = (m_{i,j})_{0 \leq i,j \leq n-1}$  une matrice carrée de taille  $n \times n$  pour un entier  $n \geq 1$ . On appelle *largeur de bande* de  $M$  le plus petit entier  $\ell$  tel que

$$\forall i, j \in \llbracket 0, n-1 \rrbracket. |i - j| > \ell \implies m_{i,j} = 0.$$

Une matrice de largeur de bande  $\ell$  a donc la forme suivante :

$$\begin{pmatrix} * & \dots & * & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ * & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & * \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & * & \dots & * \end{pmatrix}$$

$\ell$   $\ell$

Les seuls coefficients potentiellement non nuls de la matrice se trouvent dans la bande délimitée par \*, de largeur  $\ell$  de part et d'autre de la diagonale. En particulier, les matrices diagonales sont celles de largeur de bande nulle.

**Question 8** Écrire un programme qui calcule la largeur de bande d'une matrice carrée. Donner la largeur de bande des matrices suivantes.

a)  $A_{10,1,10}$

b)  $A_{100,1,100}$

c)  $A_{200,1,1\ 000}$

Un intérêt de manipuler des matrices avec une faible largeur de bande est la possibilité de les stocker plus efficacement, de façon à occuper moins d'espace. En effet, puisque l'on sait que les coefficients en dehors de la bande sont nuls, il suffit de stocker ceux de la bande.

**Question à développer pendant l'oral 7**

- Décrire comment ce stockage optimisé d'une matrice pourrait être réalisé en C.
- Évaluer l'espace occupé par une matrice stockée sous cette forme, en fonction notamment de sa largeur de bande.
- Quelle serait la complexité en temps de l'opération consistant à mettre une matrice stockée comme un tableau  $n \times n$  sous cette forme ? Et celle de l'accès à un élément de la matrice ?

Il n'est pas demandé de réellement programmer cette façon de stocker les matrices.

### 3.3 Réduction de la largeur de bande

Étant donné une matrice, on peut tenter de réordonner ses lignes et ses colonnes de façon à réduire sa largeur de bande. Cela laisse les solutions du système linéaire associé inchangées (à l'ordre des inconnues près), tout en permettant d'optimiser plus le stockage de la matrice.

Dans le cas d'une matrice symétrique, une méthode qui donne souvent d'assez bons résultats consiste à utiliser pour cela une variante du parcours en largeur.

Soit  $M = (m_{i,j})_{0 \leq i,j \leq n-1}$  une matrice symétrique de taille  $n \times n$ . On voit  $M$  comme la matrice d'adjacence d'un graphe  $\mathcal{G}$  non orienté : son ensemble de sommets est  $\mathcal{S} = \llbracket 0, n-1 \rrbracket$ , et une arête relie  $i$  et  $j$  lorsque  $m_{i,j} = 1$ . On autorise une arête à relier un sommet à lui-même. On rappelle que le degré  $\deg(i)$  d'un sommet  $i$  est le nombre d'arêtes qui y sont reliées. On considère sur  $\mathcal{S}$  l'ordre suivant :  $i \prec j$  si  $\deg(i) < \deg(j)$ , ou si  $\deg(i) = \deg(j) \wedge i < j$ .

L'algorithme consiste, intuitivement, à effectuer un parcours en largeur de  $\mathcal{G}$ , dans lequel à chaque fois que l'on a le choix de l'ordre dans lequel traiter plusieurs sommets, on les considère suivant l'ordre  $\prec$  croissant.

```

F := file vide
Vus := ∅
tant que ∃s ∈ S \ Vus
  s := sommet de S \ Vus minimal pour ≺
  ajouter s à Vus
  enfiler s dans F
  tant que F ≠ file vide
    s0 := défiler F
    V := voisins de s0 qui ne sont pas dans Vus
    pour chaque v ∈ V dans l'ordre croissant pour ≺
      ajouter v à Vus
      enfiler v dans F
renvoyer l'ordre d'ajout des sommets à Vus

```

**Algorithme 1** : Parcours en largeur

À l'issue de l'algorithme, chaque sommet a été ajouté à l'ensemble **Vus** exactement une fois. L'ordre dans lequel les sommets ont été ajoutés à **Vus** fournit une permutation  $\sigma$  :  $\sigma(0)$  est le sommet ajouté en premier,  $\sigma(1)$  le second, *etc.*

On renumérote les sommets de  $\mathcal{G}$  suivant  $\sigma$  pour obtenir un graphe  $\mathcal{G}'$  : ses sommets sont toujours  $\mathcal{S}$ , et une arête relie maintenant  $i$  et  $j$  lorsque  $\sigma(i)$  et  $\sigma(j)$  étaient reliés dans  $\mathcal{G}$ .

On note  $R(M)$  la matrice d'adjacence de  $\mathcal{G}'$  : si  $(r_{i,j})_{0 \leq i,j \leq n-1}$  désignent ses coefficients,  $r_{i,j}$  vaut 1 lorsque  $i$  et  $j$  sont reliés par une arête dans  $\mathcal{G}'$ , et 0 sinon.

Cette matrice  $R(M)$  est en fait la matrice  $M$  à laquelle on a appliqué la permutation  $\sigma^{-1}$  à la fois sur les lignes et les colonnes, et a souvent en pratique une plus faible largeur de bande que  $M$ .

**Question 9** Écrire un programme qui calcule  $R(M)$  à partir de  $M$ . Pour les matrices  $M$  suivantes, si l'on note  $R(M) = (r_{i,j})_{0 \leq i,j \leq n-1}$ , donner la somme

$$\left( \sum_{0 \leq i,j \leq n-1} (ni + j) \times r_{i,j} \right) \bmod 10\,000.$$

**a)**  $A_{10,1,10}$

**b)**  $A_{100,1,100}$

**c)**  $A_{1\,000,1,1\,000}$

**d)**  $A_{1\,000,1,10}$

**Indication.** Le fichier `base.c` fourni contient une fonction `trie_tableau`, dont le comportement est expliqué en commentaire, qui peut être utile pour cette question.

**Question à développer pendant l'oral 8** Expliquer le fonctionnement de votre programme. Évaluer sa complexité en temps.

## 4 Éclairage sous contrainte

Cette partie doit être traitée en OCAML.

Cette partie est consacrée à une variante du puzzle, dont l'objectif reste le même, à savoir trouver sur quelles cases appuyer pour reproduire une grille cible donnée. On impose une contrainte supplémentaire : il est maintenant obligatoire de sélectionner exactement une case sur chaque ligne de la grille. Il ne s'agit pas d'un cas particulier du puzzle de départ : il se peut que pour une grille donnée, le puzzle non contraint ait une solution, sans pour autant que le puzzle contraint en ait une.

On s'intéressera dans cette partie à déterminer l'existence d'une solution, plutôt qu'à la recherche d'une solution particulière.

Plus précisément, étant donné une grille cible  $G$  de taille  $n \times n$ , on souhaite calculer pour chaque entier  $d \in \llbracket 0, n^2 \rrbracket$  le booléen  $E_G(d)$ , qui est vrai lorsqu'il existe un ensemble de cases qui respecte la contrainte (une case par ligne), et dont l'image est à distance  $d$  de  $G$  :

$$E_G(d) \iff \exists C \text{ respectant la contrainte. } d(G, \mathcal{F}(C)) = d.$$

#### 4.1 Recherche exhaustive

On propose, pour commencer, de résoudre cette variante du problème par une approche exhaustive. On souhaite donc tester un par un tous les choix possibles d'une case (et une seule) par ligne, pour déterminer quelles distances  $d$  peuvent être atteintes.

**Question 10** *Écrire un programme qui effectue cette recherche exhaustive. Donner, pour les entrées suivantes, la somme (mod 10 000) des valeurs de  $d$  telles que  $E_G(d)$  est vrai :*

$$\left( \sum_{d \in \llbracket 0, n^2 \rrbracket \text{ tel que } E_G(d)} d \right) \text{ mod } 10\,000.$$

a)  $G_{3,1,2}$

b)  $G_{5,3,7}$

c)  $G_{6,3,7}$

**Question à développer pendant l'oral 9** *Évaluer la complexité en temps de votre programme.*

#### 4.2 Calcul efficace

La recherche exhaustive n'est pas très efficace, et ne permet pas de calculer le résultat souhaité pour de grandes tailles de grille.

**Question 11** *Écrire un programme qui calcule efficacement la fonction  $E$ . Donner pour les entrées suivantes, comme précédemment, la somme mod 10 000 des valeurs de  $d$  telles que  $E_G(d)$  est vrai :*

$$\left( \sum_{d \in \llbracket 0, n^2 \rrbracket \text{ tel que } E_G(d)} d \right) \text{ mod } 10\,000.$$

a)  $G_{12,3,7}$

b)  $G_{16,5,7}$

c)  $G_{20,3,7}$

**Indication.** On pourra utiliser des techniques de programmation dynamique.

**Question à développer pendant l'oral 10** *Décrire le fonctionnement de votre programme. Évaluer sa complexité en temps et en espace.*





## Fiche réponse type : Lumière

$\widetilde{u}_0$  : 1 571

### Question 1

a) 3 899

b) 617

c) 9 837

d) 394

### Question 2

a) 5

b) 698

c) 8 745

### Question 3

a) 1

b) 50

c) 4 890

### Question 4

a) 10

b) 36

c) 39

### Question 5

a) 12

b) 1 447

c) 2 630

### Question 6

a) 82

b) 4 347

c) 8 387

d) 7 660

### Question 7

a) 20

b) 5 016

c) 79

d) 555

### Question 8

a) 9

b) 77

c) 163

**Question 9**

a) 968

b) 6 693

c) 6 905

d) 6 584

**Question 10**

a) 26

b) 225

c) 425

**Question 11**

a) 4 623

b) 5 939

c) 4 991



## Fiche réponse : Lumière

Nom, prénom, u<sub>0</sub> : .....

### Question 1

- a)
- b)
- c)
- d)

### Question 2

- a)
- b)
- c)

### Question 3

- a)
- b)
- c)

### Question 4

- a)
- b)
- c)

### Question 5

- a)
- b)
- c)

### Question 6

- a)
- b)
- c)
- d)

### Question 7

- a)
- b)
- c)
- d)

### Question 8

- a)
- b)

c)

**Question 9**

a)

b)

c)

d)

**Question 10**

a)

b)

c)

**Question 11**

a)

b)

c)

