Calculatrices interdites. Pensez à numéroter vos feuilles.
Sur votre copie, les questions doivent apparaître dans l'ordre de l'énoncé.

Dans tout le sujet, `n` $\in \mathbb{N}^*$ est un entier strictement positif. Toutes les complexités devront être exprimées en fonction de `n`. On pourra utiliser sans restriction la fonction `copy` ci-contre qui renvoie une copie d'une liste de listes donnée en entrée.

```
def copy(G):
    """
    G: list[list[int]]
    Returns: list[list[int]]
    """
    return [L[:] for L in G]
```

1 Introduction

Le taquin est un jeu de réflexion composé d'un cadre contenant `n` lignes et `n` colonnes. Parmi les $\texttt{n}^2$ cases de ce cadre, une case est vide et les autres contiennent des plaques numérotées de `1` à $\texttt{n}^2 - 1$. Les figures 1 à 5 présentent des grilles possibles pour le jeu du taquin dans le cas où `n = 4`.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

FIGURE 1

5	11	1	2
14	10	4	7
3	13		12
9	8	6	15

FIGURE 2

5	11	1	2
14	10	4	7
3		13	12
9	8	6	15

FIGURE 3

5	11	1	2
14	10	4	7
3	8	13	12
9		6	15

FIGURE 4

5	11	1	2
14	10	4	7
3	8	13	12
9	6		15

FIGURE 5

Le but du joueur est de déplacer les plaques de telle manière que :

- La case vide soit en bas à droite.
- Les numéros des plaques soient dans l'ordre croissant lorsqu'on les lit ligne par ligne.

Par exemple, dans le cas `n = 4`, l'objectif est d'obtenir la grille de la figure 1.

À chaque tour de jeu, les plaques peuvent bouger en glissant les unes par rapport aux autres dans le cadre. Ainsi, pour la grille de la figure 2, il y a quatre mouvements possibles : faire glisser la plaque 4, 6, 13 ou 12 vers la case vide. Si le joueur choisit de faire glisser la case 13, il obtient la grille de la figure 3 où il y a de nouveau quatre mouvements possibles. Pour passer de la figure 3 à la figure 4, il faut faire glisser la case 8 vers le haut. De même, pour passer de la figure 4 à la figure 5, il faut faire glisser la case 6 vers la gauche. Ainsi, à chaque étape, il y a deux, trois ou quatre mouvements possibles en fonction de la position de la case vide (deux mouvements pour la figure 1, trois pour les figures 4 et 5, et quatre pour les figures 2 et 3).

2 Représentation en Python

Étant donnée une grille de taquin avec `n` lignes et `n` colonnes, chaque case est repérée par un couple d'entiers `(i, j)` $\in [\![\texttt{0}, \texttt{n - 1}]\!]^2$ appelés les ***coordonnées*** de la case. L'indice `i = 0` correspond à la ligne du haut, `i = n - 1` à la ligne du bas, `j = 0` à la colonne de gauche et `j = n - 1` à la colonne de droite. En machine, une grille est représentée par une liste de listes d'entiers `G` telle que pour tout `(i, j)` $\in [\![\texttt{0}, \texttt{n - 1}]\!]^2$:

$$\texttt{G[i][j]} = \begin{cases} \texttt{0} & \text{si la case de coordonnées (i,j) est vide.} \\ \texttt{x} & \text{si la case de coordonnées (i,j) contient la plaque numéro x.} \end{cases}$$

Par exemple, les grilles des figures 1, 2 et 5 correspondent respectivement à `GFig1`, `GFig2` et `GFig5`.

```
GFig1 = [[1,2,3,4], [5,6,7,8], [9,10,11,12], [13,14,15,0]]
GFig2 = [[5,11,1,2], [14,10,4,7], [3,13,0,12], [9,8,6,15]]
GFig5 = [[5,11,1,2], [14,10,4,7], [3,8,13,12], [9,6,0,15]]
```

1. Dans cette question, on note `G` une liste de listes d'entiers et `n` le nombre de sous-listes de `G`.
   (a) Écrire une fonction de signature `test1(G: list[list[int]]) -> bool` qui renvoie `True` si toutes les sous-listes de `G` sont de taille `n` et `False` dans le cas contraire.
   (b) Écrire une fonction de signature `test2(G: list[list[int]]) -> bool` qui renvoie `True` si tous les entiers présents dans `G` appartiennent à $[\![\texttt{0}; \texttt{n}^2 - \texttt{1}]\!]$ et `False` dans le cas contraire. Attention : on ne suppose pas que `test1(G)` vaut `True`.

Jusqu'à la fin du sujet, on pourra supposer sans le vérifier que `test1(G)` et `test2(G)` s'évaluent `True`. On dira que `G` est ***bien formée*** si tous les entiers de $[\![\texttt{0}; \texttt{n}^2 - \texttt{1}]\!]$ sont présents dans `G`.

2. À l'aide de boucles `while` et sans utiliser de boucle `for`, écrire une fonction de signature

```
coord(G: list[list[int]], x: int) -> [(int,int) ou NoneType]
```

qui renvoie le couple `(i, j)` tel que `G[i][j]` vaut `x`, ou renvoie `None` si `x` est absent de `G`.

3. Afin de tester si `G` est bien formée, on définit la fonction ci-contre. Donner la signature et la complexité de la fonction `tousPresents`. Pour la complexité, on attend une justification, et la réponse devra être de la forme « le temps d'exécution est de l'ordre de `f(n)` » avec `f` une fonction bien choisie.

```
def tousPresents(G):
    """tousPresents(G: list[list[int]]) -> bool"""
    n = len(G)
    for x in range(n*n):
        if coord(G, x) == None:
            return False
    return True
```

La complexité de la fonction `tousPresents` n'étant pas optimale, on cherche à l'améliorer.

4. Expliquer comment tester si `G` est bien formée avec une complexité quadratique en `n`. Votre réponse doit être une description en français, pas un code Python.

Jusqu'à la fin du sujet, on pourra supposer sans le vérifier que `G` est bien formée.

3 Mouvements au taquin

À chaque tour de jeu, le joueur fait glisser l'une des plaques vers la case vide. Il s'agit donc d'un mouvement vers le haut, vers le bas, vers la gauche ou bien vers la droite. On représentera ces quatre possibilités par une chaîne de caractères `M` $\in$ `{"H", "B", "G", "D"}` (pour Haut, Bas, Gauche, Droite). Par exemple, pour passer de la figure 2 à la figure 5, en passant par les figures 3 et 4, on effectue les mouvements `"D"`, `"H"`, puis `"G"`.

5. (a) Supposons avoir accès aux coordonnées `(i0, j0)` de la case vide de la grille. Écrire une fonction de complexité constante `estLicite(i0: int, j0: int, n: int, M: str) -> bool` qui renvoie `True` si le mouvement `M` est possible et `False` sinon. Votre fonction renverra également `False` dans le cas où `M` $\notin$ `{"H", "B", "G", "D"}`. Par exemple :

M	"H"	"B"	"G"	"D"
i0 = 3, j0 = 3, n = 4 (Figure 1)	False	True	False	True
i0 = 2, j0 = 2, n = 4 (Figure 2)	True	True	True	True
i0 = 3, j0 = 2, n = 4 (Figure 5)	False	True	True	True

(b) Écrire une fonction de signature `mvt(G1: list[list[int]], M: str) -> list[list[int]]` qui prend en entrée une grille de taquin `G1` et renvoie la grille `G2` obtenue après avoir effectué le mouvement `M`. Votre fonction déclenchera une erreur si la fonction de la question précédente indique que le mouvement n'est pas possible. Votre fonction devra commencer par la ligne « `G2 = copy(G1)` ». Par exemple :

`mvt(GFig2, "D")` est la grille de la figure 3. et `mvt(GFig5, "D")` est la grille de la figure 4.

(c) Donner en la justifiant la complexité de la fonction `mvt` en fonction de `n`.

Au taquin, le but est d'obtenir une grille particulière appelée ***grille finale*** et notée `GF` dans la suite. Pour générer cette grille, on écrit deux fonctions :

```
1  def makeGF1(n):
2      GF = []
3      k = 1
4      for _ in range(n):
5          GF.append([])
6          for _ in range(n):
7              GF[-1].append(k)
8              k += 1
9      GF[-1][-1] = 0
10     return GF
```

```
1  def makeGF2(n):
2      GF = f(n)
3      for i in range(n):
4          for j in range(n):
5              GF[i][j] = g(i,j,n)
6      return GF
```

6. (a) Lors d'un appel à la fonction `makeGF1` avec l'argument `n = 3` :

(i) Quelle est la valeur de la variable `GF` à la fin de chacun des trois tours de la boucle `for` située ligne 4 ?

(ii) Quelle est la liste renvoyée ?

(b) On souhaite faire en sorte que la fonction `makeGF2` renvoie la même liste que la fonction `makeGF1`. Définir les fonctions `f` et `g` appelées lignes 2 et 5 de `makeGF2`. Le code de la fonction `f` devra faire une ligne (sans compter la ligne avec le `def`) et contenir uniquement des compréhensions de liste. La fonction `g` devra s'exécuter en temps constant.

7. (a) Écrire une fonction de signature `egales(L: list[list[int]], M: list[list[int]]) -> bool` qui indique si les deux listes en entrée sont égales. En d'autres termes, votre fonction doit renvoyer le booléen `L == M`, mais sans appliquer l'opérateur `==` entre deux listes. Les listes de listes d'entiers `L` et `M` sont ici quelconques.

   (b) Soit « `LM: list[str]` » une liste de mouvements (c'est à dire une liste dont chaque élément est `"H"`, `"B"`, `"G"` ou `"D"`). Écrire une fonction de signature

   `verif(G1: list[list[int]], G2: list[list[int]], LM: list[str]) -> bool`

   qui indique si la liste de mouvements `LM` permet de passer de la grille `G1` à la grille `G2`. Par exemple :

   `verif(GFig2, GFig5, ["D", "H", "G"])` vaut `True`

4 Configurations solubles

Certaines grilles de taquin ne peuvent pas être résolues (dans le sens où aucune suite de mouvements ne permet d'obtenir la grille finale définie dans la partie 3). Par exemple, si dans la figure 1, on échange les plaques numérotées 1 et 2 alors il n'est pas possible de retrouver la grille de la figure 1. Pour tester si une grille peut être résolue, on appelle `L(G)` la liste contenant les entiers de la grille lorsque celle-ci est lue ligne par ligne. Par exemple pour la figure 2, on obtient :

`L(G) = [5, 11, 1, 2, 14, 10, 4, 7, 3, 13, 0, 12, 9, 8, 6, 15]`

Étant donnée une liste `G`, on appelle ***nombre caractéristique de G*** l'entier $c = d + i_0 + j_0$ où :

- d est le nombre de couples (k_1, k_2) tels que $k_1 < k_2$ et $L(G)[k_1] > L(G)[k_2]$.
- i_0 et j_0 sont les deux entiers tels que $G[i_0][j_0] = 0$.

Théorème 1 (admis)**.** *Soient* `G` *et* G' *deux grilles et* c, c' *leurs nombres caractéristiques alors :*

$c \equiv c' \mod 2$. $\quad\Leftrightarrow\quad$ *Il est possible de passer de* `G` *à* G' *par une suite de mouvements.*

8. (a) Écrire une fonction `nbCarac` qui prend en entrée une liste de listes d'entiers `G` et renvoie son nombre caractéristique.

   (b) Quelle est la complexité de `nbCarac` en fonction de `n` ? Justifier.

9. À l'aide du théorème 1 :

   (a) Montrer que la grille de la figure 6 ne peut pas être résolue.

   (b) Écrire une fonction `testParite` qui prend en entrée deux grilles `G1`, `G2` et renvoie un booléen indiquant s'il est possible de passer de l'une à l'autre par une suite de mouvements.

5 Résolution

Dans cette partie, on présente une méthode de résolution pour le jeu du taquin. Soit G_0 une grille et $k \in \mathbb{N}$ un entier. Dans une premier temps, on va générer toutes les grilles atteignables à partir G_0 en utilisant `k` mouvements. On note L_k la liste contenant tous les couples de la forme `(G, LM)` avec :

- `LM` une liste de taille `k` dont chaque élément est `"H"`, `"B"`, `"G"` ou `"D"`. Dans un soucis d'optimisation, on interdit à deux mouvements successifs de s'annuler. Formellement, pour tout $i \in [\![0; k-2]\!]$:

$$(LM[i], LM[i+1]) \notin \Big\{("H","B");("B","H");("G","D");("D","G")\Big\}.$$

- `G` la grille obtenue à partir de G_0 en appliquant la suite des mouvements contenus dans `LM`.

10. Supposons que la liste L_k ait déjà été générée. Écrire une fonction `etape` qui prend en entrée L_k et renvoie L_{k+1}.

Une stratégie envisageable pour résoudre le problème du taquin serait de partir de la grille initiale et de générer les liste L_0, L_1, L_2, ... jusqu'à ce que l'une d'entre elles contienne la grille finale. Malheureusement, cette méthode est beaucoup trop lente pour pouvoir être utilisée en pratique. On va donc décomposer la résolution en plusieurs petites étapes ; et pour chaque étape, la suite des mouvements à effectuer sera déterminée grâce à la fonction `etape`.

On aura également besoin d'une procédure auxiliaire permettant de placer la case vide à côté d'une plaque dont le numéro `x` est donné. Afin de ne pas bouger des plaques placées précédemment, on procédera en deux étapes :

- La case vide est d'abord déplacée dans le coin inférieur droit de la grille en utilisant uniquement les mouvements `"H"` et `"G"`.
- La case vide est ensuite déplacée sur une case adjacente à la plaque `x` en utilisant uniquement des mouvements `"B"` et `"D"`. Pour cela, on distinguera deux cas :
  - Si la plaque `x` se trouve sur la dernière ligne, alors on place la case vide sur la case se trouvant à droite de `x`.
  - Sinon, on place la case vide sur la case se trouvant en dessous de `x`.

11. Écrire une fonction de signature

```
deplVide(G: list[list[int]], x: int) -> (list[list[int]], list[str])
```

qui déplace la case vide à côté de la plaque numérotée par `x` en suivant la procédure décrite ci-dessus. Votre fonction renverra la grille obtenue ainsi que la liste des mouvements à effectuer. La complexité devra être de l'ordre de n^3.

2	1	3	4
5	6	7	8
9	10	11	12
13	14	15	

FIGURE 6

1	2	.	3
.	.	.	4
.	.	.	.
.	.	.	.

FIGURE 7

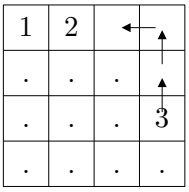

FIGURE 8

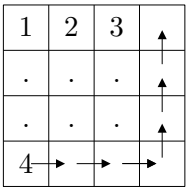

FIGURE 9

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	18	20	22	24
17	19	21	23	

FIGURE 10

Dans la suite, on manipulera une liste `LC` appelée ***liste de contraintes***. Chaque élément d'une liste de contraintes est un triplet d'entiers `(i,j,x)` signifiant qu'on souhaite placer la plaque numéro `x` sur la case de coordonnées `(i, j)`. Par exemple, la liste de contraintes `[(0,0,1), (0,1,2), (0,3,3), (1,3,4)]` signifie qu'on souhaite obtenir une grille comme dans la figure 7 (une case avec un point peut être vide ou bien contenir une plaque quelconque).

Soit `(i, j, x)` trois entiers et `LC` une liste de contraintes. On souhaite déplacer la plaque numéro `x` vers la case de coordonnées `(i, j)` tout respectant les contraintes de `LC`. Pour cela, on transfert la case vide à côté de `x` (voir question 11), puis on bouge la plaque `x` de proche en proche jusqu'à ce qu'elle ait atteint sa position finale. Soient (i_x, j_x) les coordonnées initiales de la plaque `x`. On suppose que $i_x \geqslant i$ (cette condition sera toujours vérifiée lors de la résolution). On distingue alors deux cas :

- Si $j_x > j$, on commence par déplacer la plaque sur la ligne `i`, puis sur la colonne `j`. La figure 8 est un exemple où `i = 0`, `j = 2`, `x = 3`, $i_x = 2$ et $j_x = 3$. Notez que sur cette figure, une flèche ne représente pas un seul mouvement, mais une suite de mouvements à déterminer grâce à la fonction `etape`.
- Si $j_x \leqslant j$, on commence par déplacer la plaque sur la colonne `j`, puis sur la ligne `i`. La figure 9 est un exemple où `i = 0`, `j = 3`, `x = 4`, $i_x = 3$ et $j_x = 0$.

12. Écrire une fonction `nvContrainte` qui prend en entrée une grille `G`, une liste de contraintes `LC`, ainsi que trois entiers `i`, `j`, `x`, et place la plaque `x` sur la case de coordonnées `(i, j)` en suivant la procédure décrite ci-dessus. La fonction renverra la grille obtenue ainsi que la liste des mouvements à effectuer.

Il ne reste plus qu'à placer les plaques les unes après les autres dans la grille. Pour cela, on commence par remplir les `n - 2` premières lignes dans l'ordre, puis les deux dernières lignes colonne par colonne. Par exemple pour `n = 5`, la figure 10 indique l'ordre à suivre. Notez que contrairement aux autres figures, les nombres sur la figure 10 ne représentent pas les numéros des plaques, mais l'ordre dans lequel il faut les placer.

13. Écrire une fonction de signature `resolution(G: list[list[int]]) -> (list[str] ou NoneType)` qui renvoie la liste des mouvements à effectuer pour passer de `G` à la grille finale (voir question 6). Cette fonction doit renvoyer `None` dans le cas où il n'y a pas de solution (voir partie 4).