1 Préliminaires

1.1 Représentation d'ensembles d'entiers en Python

Soit $A \subset \mathbb{N}$ un ensemble d'entiers. En Python, on décide de représenter A par une liste triée sans doublon. Par exemple, l'ensemble $A = \{0, 5, 8\}$ est représenté par la liste `[0, 5, 8]`. En revanche, A n'est pas représenté par `[5, 0, 8]` car cette liste n'est pas triée et n'est pas non plus représentée par `[0, 5, 5, 8]` car cette liste contient des doublons.

1. (a) Écrire une fonction `estTriee(A: list[int]) -> bool` qui renvoie `True` si `A` est triée, et `False` sinon. Voir le tableau ci-dessous pour des exemples.
   (b) Quelle est la complexité de votre fonction ? Justifier.
2. (a) Compléter le code suivant pour obtenir une fonction

   `sansDoublV1(A: list[int]) -> bool`

   qui renvoie `True` si `A` ne contient pas de doublon, et `False` sinon. Voir le tableau ci-dessous pour des exemples.

```
def sansDoublV1(A):
    for i in range(...):
        for j in range(...):
            if ... :
                return ...
    return ...
```

   (b) Quelle est la complexité de votre fonction ? Justifier.
3. Écrire une fonction `sansDoublV2(A: list[int]) -> bool` qui renvoie `True` si `A` ne contient pas de doublon, et `False` sinon. Votre fonction doit donc renvoyer le même résultat que `sansDoublV1`, mais son temps d'exécution doit être en $\mathcal{O}(n)$ où n est la taille de `A`.

A	[]	[5]	[1, 7]	[0, 5, 8]	[0, 5, 5, 8]	[5, 0, 8]	[0, 4, 8, 10, 0]
estTriee(A)	True	True	True	True	True	False	False
sansDoublV1(A)	True	True	True	True	False	True	False
sansDoublV2(A)	True	True	True	True	False	True	False

Jusqu'à la fin du sujet, le type Python « `ens` » désignera une liste d'entiers triée sans doublon. Par exemple, si une question vous demande d'écrire une fonction « `f(A: ens) -> ens` » cela signifie que :

→ Vous pouvez supposer sans le vérifier que `A` est une liste d'entiers triée sans doublon.

→ Vous devez faire en sorte que votre fonction renvoie une liste d'entiers triée sans doublon.

1.2 Représentation de listes d'ensembles d'entiers en Python

À partir de maintenant, $n \in \mathbb{N}^*$ et $m \in \mathbb{N}^*$ sont deux entiers strictement positifs et $A_0, \ldots, A_{m-1}$ sont des ensembles d'entiers. En Python, les A_i sont représentés par une liste de listes d'entiers « `L: list[ens]` ». Par exemple avec :

$$n = 5 \qquad m = 4 \qquad A_0 = \{0, 4\} \qquad A_1 = \{-1, 2\} \qquad A_2 = \{\} \qquad A_3 = \{-4, 0, 1, 2, 3\}$$

on obtient la liste `[[0,4], [-1,2], [], [-4,0,1,2,3]]`.

4. Écrire une fonction `test(n: int, L: list[ens]) -> bool` qui renvoie `True` si tous les entiers présents dans `L` sont compris entre `0` et `n-1`, et `False` sinon. Voir le tableau ci-dessous pour des exemples.
5. Écrire une fonction `maxi(L: list[ens]) -> (int ou NoneType)` qui renvoie le maximum des entiers présents dans `L`. Si `L` ne contient aucun entier, votre fonction renverra `None`. Voir le tableau ci-dessous pour des exemples.

`L`	`[]`	`[[], []]`	`[[0,4], [-1,2], [], [-4,0,1,2,3]]`	`[[0,1,2,3], [2], [0,1,3]]`
`test(4, L)`	`True`	`True`	`False` (à cause de `-4`, `-1` et `4`)	`True`
`maxi(L)`	`None`	`None`	4	3

1.3 Le problème de la couverture exacte

Le but de ce sujet est d'étudier le ***problème de la couverture exacte***. Soit $n \in \mathbb{N}^*$ et $m \in \mathbb{N}^*$ deux entiers strictement positifs. Étant donnés des ensembles $A_0, \ldots, A_{m-1}$ inclus dans $[\![0\,; n-1]\!]$, le problème de la couverture exacte consiste à sélectionner certains des A_j pour que chaque entier $i \in [\![0\,; n-1]\!]$ apparaisse dans exactement un des A_j sélectionnés. Par exemple avec $n = 7$, $m = 9$ et :

$$A_0 = \{3\}, \quad A_1 = \{0,1\} \quad A_2 = \{2,6\} \quad A_3 = \{0,3,5\} \quad A_4 = \{0,3,4,5,6\}$$
$$A_5 = \{0,2,3\} \quad A_6 = \{1,3,4\} \quad A_7 = \{2,4,5,6\} \quad A_8 = \{2,5\}$$

L'unique solution du problème de la couverture exacte est $S = \{0,1,7\}$. En effet, chaque entier $i \in [\![0\,; 6]\!]$ apparaît dans un et un seul des ensembles A_0, A_1, A_7.

6. Donner sans justification les deux solutions au problème de la couverture exacte dans le cas où $n = 8$, $m = 6$ et :

$$A_0 = \{1,7\} \quad A_1 = \{4,5,6\} \quad A_2 = \{1,2\} \quad A_3 = \{0,3\} \quad A_4 = \{7\} \quad A_5 = \{2\}$$

7. Écrire une fonction `estSol(n: int, L: list[ens], S: ens) -> bool` qui renvoie `True` si `S` est une solution du problème de la couverture exacte, et `False` sinon. Ici :
   - La liste `L` contient les ensembles $A_0, A_1, \ldots A_{m-1}$. Pour l'exemple ci-dessus, on a :

```
L = [[3], [0,1], [2,6], [0,3,5], [0,3,4,5,6],
     [0,2,3], [1,3,4], [2,4,5,6], [2,5]]
```

   - La liste `S` contient les indices j des ensembles A_j sélectionnés. Pour l'exemple ci-dessus, on a `S = [0,1,7]`.

   On pourra supposer sans le vérifier que tous les entiers présents dans `L` sont compris entre `0` et `n-1`. À des fins d'optimisation, chaque entier de `L` devra être vu au plus une fois.

2 Génération de tests aléatoires

L'objectif de cette partie est de générer des listes « `L: list[ens]` » à donner en entrée de la fonction `estSol` de la question 7. On va d'abord créer une liste « `A: ens` » dans laquelle chaque entier $i \in [\![0\,; n-1]\!]$ apparaît avec une certaine probabilité $p \in [\,0\,;1\,]$. Pour cela, on adopte la stratégie suivante :

Pseudo-code 1
Entrées: $n \in \mathbb{N}^*$ et $p \in [\,0\,;1\,]$
Sortie: $A \subset [\![\,0\,;n-1\,]\!]$

Initialement, A est vide.
pour chaque $i \in [\![\,0\,;n-1\,]\!]$ **faire**
    On ajoute i dans A avec probabilité p.
fin pour
Renvoyer A.

En particulier, si $p = 0$ (resp. $p = 1$), alors cette procédure renvoie nécessairement l'ensemble vide (resp. l'ensemble $[\![\,0\,;n-1\,]\!]$).

En Python, on aura besoin de la fonction « `random() -> float` » qui renvoie un flottant choisi aléatoirement dans l'intervalle `[ 0 ; 1 ]`. Cette fonction appartient au module `random` (notez que la fonction et le module portent le même nom).

8. Quelle ligne doit-on écrire en Python pour pouvoir utiliser la fonction `random` ? On veut par exemple que le code suivant s'exécute sans erreur :

```
x = random.random()
y = random.random()
print(x+y)
```

À partir de maintenant, on suppose que la ligne évoquée à la question 8 a été exécutée. Parmi toutes les fonctions du module `random`, on s'autorise uniquement à utiliser la fonction `random`.

9. Écrire une fonction `ensAlea(n: int, p: float) -> ens` qui renvoie l'ensemble A comme décrit dans le pseudo-code 1.

On souhaite construire une liste « `L: list[ens]` » de manière aléatoire.

Pseudo-code 2
Entrées: $n \in \mathbb{N}^*$ et $m \in \mathbb{N}^*$.
Sortie: une liste contenant des ensembles $A_0, A_1, \dots A_{m-1}$ tous inclus dans $[\![\,0\,;n-1\,]\!]$.

pour chaque $j \in [\![\,0\,;m-1\,]\!]$ **faire**
    On choisit un flottant $p \in [\,0.1\,;0.5\,]$ aléatoire.
    À l'aide du pseudo-code 1, on construit un ensemble A_j non vide.
fin pour
Renvoyer la liste contenant $A_0, A_1, \dots A_{m-1}$.

10. Écrire une fonction `listeEnsAlea(n: int, m: int) -> list[ens]` qui renvoie une liste d'ensembles comme décrit dans le pseudo-code 2. Attention : à chaque étape on doit avoir $p \in [\,0.1\,;0.5\,]$ et A_j doit être non vide.

3 Le problème des dominos

3.1 Formulation du problème des dominos

Dans cette partie, on s'intéresse au "problème des dominos" qui peut s'exprimer comme un cas particulier du problème de la couverture exacte.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

FIGURE 1

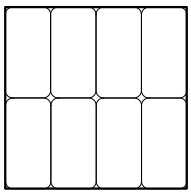

FIGURE 2

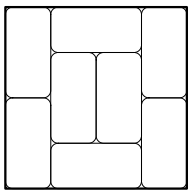

FIGURE 3

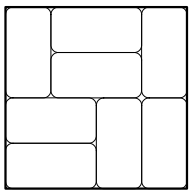

FIGURE 4

Soit $k \in \mathbb{N}^*$ un entier, on s'intéresse à une grille composée de k lignes et k colonnes. Par exemple pour $k = 4$, la grille contient 16 cases (voir la figure 1). Notre objectif est de recouvrir toutes les cases de la grille avec des dominos, sachant que les dominos ne peuvent pas se chevaucher et que chaque domino occupe deux cases adjacentes. Par exemple, pour paver une grille de côté $k = 4$ avec 8 dominos, plusieurs solutions existent (voir les figures 2, 3, 4).

11. Dessiner toutes les solutions du problème des dominos pour $k = 2$ (utilisez les carreaux de votre feuille et une règle pour faire des dessins propres).
12. Quel est le nombre de solutions au problème des dominos lorsque k est impair ? Justifier.

On souhaite déterminer le nombre de solutions dans le cas où $k = 4$. Pour cela, on numérote les 16 cases de la grille de 0 à 15 comme dans la figure 1. On appelle "solution de type 1" une solution dans laquelle la case 0 est recouverte par un domino vertical (qui recouvre donc les cases 0 et 4 comme dans la figure 5) et "solution de type 2" une solution où ce domino est horizontal (il recouvre donc les cases 0 et 1 comme dans la figure 6).

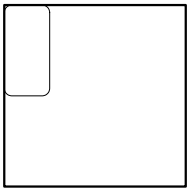

FIGURE 5

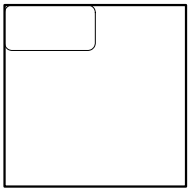

FIGURE 6

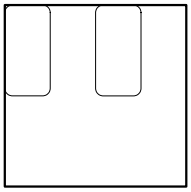

FIGURE 7

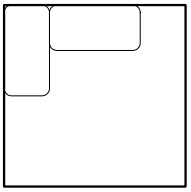

FIGURE 8

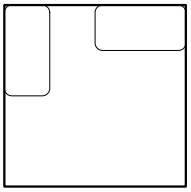

FIGURE 9

Pour déterminer le nombre de solutions de type 1, on s'intéresse à la case numéro 2. Cette case peut être recouverte de trois façons différentes (voir les figures 7, 8, 9).

13. (a) Dessiner les cinq façons de compléter le pavage de la figure 7.
    (b) Quel est le nombre de façons de compléter le pavage de la figure 8 ? Justifiez succinctement votre réponse.
    (c) Même question avec la figure 9.
14. En déduire le nombre de solutions au problème des dominos lorsque $k = 4$.

3.2 Traduction du problème des dominos

Comme dans la partie précédente, $k \in \mathbb{N}^*$ désigne le nombre de lignes et de colonnes dans la grille. Pour traduire le problème des dominos en un problème de couverture exacte, on note $n = k^2$ le nombre de cases dans la grille et $m \in \mathbb{N}^*$ le nombre de positions possibles pour un domino. Les cases de la grille sont alors numérotées de 0 à $n-1$ de la gauche vers la droite et du haut vers le bas comme dans la figure 1. Chaque position possible pour un domino est représentée par un ensemble de cardinal 2 contenant les numéros des deux cases recouvertes par le domino. Par exemple, la position du domino de la figure 10 correspond à l'ensemble $\{5, 6\}$ et celle du domino de la figure 12 à l'ensemble $\{1, 5\}$.

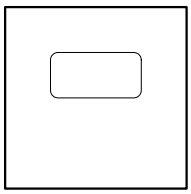

FIGURE 10

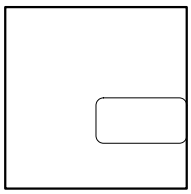

FIGURE 11

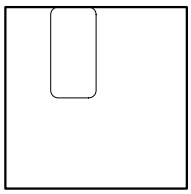

FIGURE 12

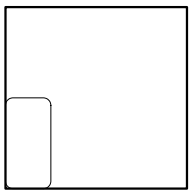

FIGURE 13

15. (a) Déterminer les ensembles correspondant aux dominos des figures 11 et 13.
    (b) Déterminer en fonction de k la valeur de m (c'est à dire le nombre de placements possibles pour un domino sur une grille de côté k). Justifier votre réponse.
16. Écrire une fonction `posDom(k: int) -> list[ens]` qui renvoie une liste de longueur m contenant toutes les positions possibles pour un domino sur une grille de côté `k`.

Résoudre le problème des dominos sur une grille de côté $k \in \mathbb{N}^*$ est alors équivalent à résoudre le problème de la couverture exacte avec $n = k^2$ et $A_0, \ldots, A_{m-1}$ les ensembles de la liste renvoyée par la fonction `posDom`.

4 Résolution du problème de la couverture exacte

Les notations utilisées dans cette partie sont les mêmes que dans la partie 1.3.

4.1 Méthode 1

Une solution au problème de la couverture exacte sera représentée par une liste « `B: list[bool]` » de longueur m telle que `B[j]` vaut `True` lorsque l'ensemble A_j fait partie de la solution, et `False` sinon. Pour résoudre le problème, on va tester toutes les possibilités. Il s'agit donc dans un premier temps de générer toutes les listes de taille m contenant des `True` et des `False`.

17. Écrire une fonction `generer(m: int) -> list[list[bool]]` qui renvoie une liste contenant une et une seule fois chaque liste de taille `m` contenant des `True` et des `False`. On pourra supposer sans le vérifier que `m` $\in \mathbb{N}^*$. Par exemple, pour `m = 3`, on obtient (l'ordre des sous-listes n'a pas d'importance) :

```
[[False, False, False], [True, False, False],
 [False, True, False], [True, True, False], [False, False, True],
 [True, False, True], [False, True, True], [True, True, True]]
```

18. Écrire une fonction `estSolBis(n: int, L: list[ens], B: list[bool]) -> bool` qui indique si `B` correspond à une solution au problème de la couverture exacte. On pourra supposer sans le vérifier que les listes `L` et `B` sont toutes les deux de taille m.
19. En déduire une fonction `toutesSolV1(n: int, L: list[ens]) -> list[ens]` qui renvoie la liste de toutes les solutions au problème de la couverture exacte. Pour l'exemple de la partie 1.3 où la seule solution était $S = \{0, 1, 7\}$, la fonction doit renvoyer `[[0,1,7]]`.

4.2 Méthode 2

Dans cette partie, on suppose que les ensembles $A_0, \ldots, A_{m-1}$ sont tous non vides. Dans le but d'améliorer le temps de résolution du problème de la couverture exacte, on s'intéresse à la procédure suivante :

Pseudo-code 3
Entrées: $n \in \mathbb{N}^*$ et des ensembles $A_0, \dots, A_{m-1}$ inclus dans $[\![0\,;n-1]\!]$.
Sortie: Une solution au problème de la couverture exacte.

Initialement, S est vide.
pour chaque $i \in [\![0\,;n-1]\!]$ **faire**
  si $\forall j \in S, i \notin A_j$ **alors**
    Soit $J = \left\{ j_1 \in [\![0\,;m-1]\!] : i \in A_{j_1} \text{ et } (\forall j_2 \in S, A_{j_1} \cap A_{j_2} = \varnothing) \right\}$
    si $J = \varnothing$ **alors**
      On considère que l'exécution est un échec.
    fin si
    Choisir un indice $j \in J$.
    Ajouter j dans S.
  fin si
fin pour
Renvoyer S.

Notons que cette procédure a plusieurs exécutions possibles car, à chaque étape, le choix de l'indice $j \in J$ est arbitraire. On peut même se convaincre que toute solution au problème de la couverture exacte s'obtient à partir d'une exécution de cette procédure avec un choix judicieux des indices $j \in J$.

20. En s'inspirant du pseudo-code 3, écrire une fonction

```
toutesSolV2(n: int, L: list[ens]) -> list[ens]
```

qui renvoie la liste de toutes les solutions au problème de la couverture exacte. Notez que contrairement au pseudo-code 3, votre fonction doit renvoyer toutes les solutions au problème. On essaiera d'obtenir la fonction la plus efficace possible et on expliquera succinctement la procédure utilisée.

21. Écrire une fonction `nbSolDom(k: int) -> int` qui renvoie le nombre de solutions au problème des dominos étudié dans la partie 3.