DS4 - INFORMATIQUE
TYPE CCINP

SNAKES AND LADDERS

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Notes : les importations des modules suivantes sont réalisées tout le long du sujet :

```
from collections import deque
import matplotlib.pyplot as plt
import random as r
import math as m
import networkx as nx
import numpy as np
```

Ce sujet utilise la syntaxe des annotations pour préciser le type des arguments et du résultat des fonctions à écrire. Ainsi

```
def maFonction(n:int, X:[float], c:str, u) -> (int, np.ndarray):
```

signifie que la fonction `maFonction` prend quatre arguments, le premier (`n`) est un entier, le deuxième (`X`) une liste de nombres à virgule flottante, le troisième (`c`) une chaine de caractères et le type du dernier (`u`) n'est pas précisé. Cette fonction renvoie un couple dont le premier élément est un entier et le deuxième un tableau `numpy`. **Il n'est pas demandé aux candidats de recopier les entêtes avec annotations telles qu'elles sont fournies dans ce sujet, ils peuvent utiliser des entêtes classiques.**

Il est **fortement recommandé** d'utiliser les fonctions mises en place précédemment pour répondre à certaines questions. Toutes les fonctions et structures de données demandées sont supposées viables pour la suite des questions. Il n'est donc **pas du tout nécessaire pour vous de les avoir codées pour pouvoir les utiliser**.

Il suit du paragraphe précédent que la quasi-totalité des **questions** sont **indépendantes** (les seules exceptions étant les questions portant sur la complexité temporelle des fonctions).

Une **Annexe** est présente à la fin pour vous aider (documentations, complexités, définitions, etc).

1. Présentation du jeu

Le jeu *serpents et échelles* est un jeu de société où on espère monter les échelles en évitant de trébucher sur les serpents. Il provient d'Inde et est utilisé pour illustrer l'influence des vices et des vertus sur une vie.

Le plateau

- Le plateau comporte 100 cases numérotées de 1 à 100 en boustrophédon[1] : le 1 est en bas à gauche et le 100 est en haut à gauche ;
- des serpents et échelles sont présents sur le plateau : les serpents font descendre un joueur de sa tête à sa queue, les échelles font monter un joueur du bas de l'échelle vers le haut.

1. à la manière du bœuf traçant des sillons, avec alternance gauche-droite et droite-gauche

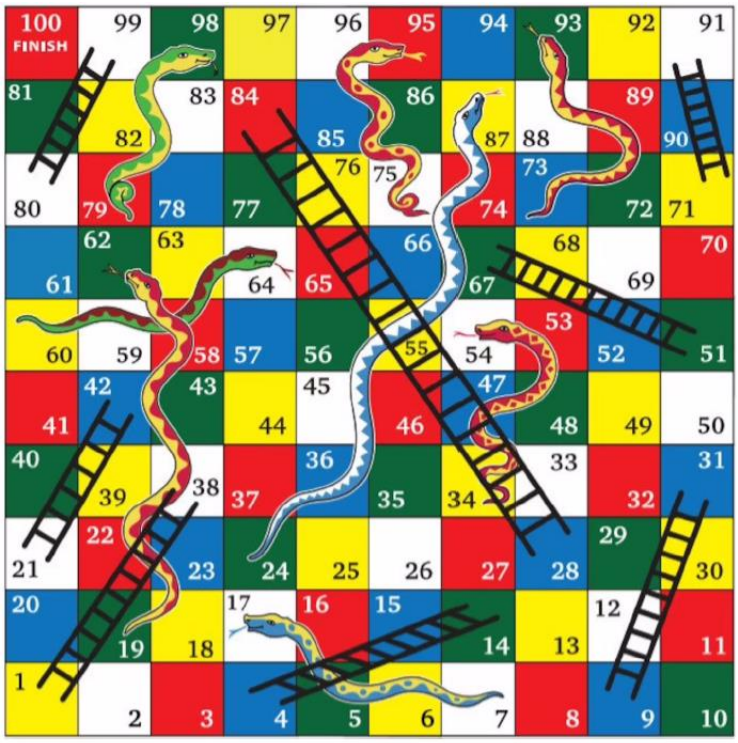

FIGURE 1 – Exemple d'un plateau de serpents et échelles

Déroulement

- Chaque joueur a un pion sur le plateau. Plusieurs pions peuvent être sur une même case. Les joueurs lancent un dé à tour de rôle et ils avancent du nombre de cases marqués sur le dé. S'ils atterrissent sur un bas d'échelle ou une tête de serpent, ils vont directement à l'autre bout ;
- les joueurs commencent sur une case 0 hors du plateau : la première case où mettre leur pion correspond donc au premier lancer de dé ;
- le premier joueur à arriver sur la case 100 a gagné ;
- il existe 3 variantes quand la somme de la case actuelle et du dé dépasse 100 :
  - ◇ le rebond : on recule d'autant de cases qu'on dépasse ;
  - ◇ l'immobilisme : on n'avance pas du tout si on dépasse :
  - ◇ la fin rapide : on va à la case 100 quoi qu'il arrive.

On utilisera les notations suivantes pour les complexités : N_{cases}, le nombre de cases du plateau (100), et N_{SeE} la somme du nombre de serpents et du nombre d'échelle (16 dans notre exemple). Ces variables ne sont pas déclarées dans le script.

2. Simulation du jeu

Question 1 Écrire une fonction `lancerDe() -> int` qui renvoie un nombre entier compris entre 1 et 6 en utilisant une fonction du module `random`. Vous pourrez vous aider des documentations en annexe.

Les serpents et les échelles seront, dans un premier temps, représentés par une liste de 2 listes nommée `LSeE` telle que, pour un élément `i` donné, `LSeE[0][i]` représente le numéro de la case de départ (tête du serpent ou base de l'échelle) et `LSeE[1][i]` représente le numéro de la case d'arrivée (queue du serpent ou haut de l'échelle).

Avec l'exemple de la FIGURE 1, on a :

```
LSeE = [[ 1,  4,  9, 17, 21, 28, 51, 54, 62, 64, 71, 80, 87, 93, 95, 98],
        [38, 14, 31,  7, 42, 84, 67, 34, 19, 60, 91, 99, 24, 73, 75, 79]]
```

On suppose que cette variable est accessible depuis toutes les futures fonctions, c'est-à-dire qu'elle est déjà déclarée dans le script.

Question 2 Écrire une fonction `caseFuture(case: int) -> int` qui renvoie le numéro de la case où va se retrouver le joueur en atterrissant sur la case numérotée `case`. Par exemple, `caseFuture(5)` doit renvoyer `5` (c'est un numéro de case stable), `caseFuture(1)` doit renvoyer `38` (à cause du pied de l'échelle) et `caseFuture(17)` doit renvoyer `7` (à cause de la tête du serpent).

Question 3 Quelle est la complexité de cette dernière ?

Les serpents et les échelles sont maintenant, et pour toute la suite, représentés par un dictionnaire `dSeE` telle que, pour une case de départ numérotée `i`, `dSeE[i]` donne le numéro de la case d'arrivée.

Avec l'exemple de la FIGURE 1, on a :

```
dSeE = {  1: 38,  4: 14,  9: 31, 17:  7, 21: 42, 28: 84, 51: 67, 54: 34,
         62: 19, 64: 60, 71: 91, 80: 99, 87: 24, 93: 73, 95: 75, 98: 79}
```

On suppose que cette variable est accessible depuis toutes les futures fonctions, c'est-à-dire qu'elle est déjà déclarée dans le script.

Question 4 Réécrire la fonction `caseFuture(case: int) -> int` avec le dictionnaire au lieu de la liste de listes.

Question 5 Quelle est la complexité de cette dernière ?

Question 6 Écrire une fonction : `avanceCase(case: int, de: int, choix: str) -> int` qui renvoie la case d'arrivée lorsqu'on part de la case `case` et qu'on a comme résultat au lancer du dé la valeur `de`. La variable `choix` est une chaine de caractère correspondant à la stratégie de fin différente : `"r"` pour le rebond, `"i"` pour l'immobilisme et `"q"` pour une fin rapide.

Question 7 Écrire une fonction `partie(choix: str) -> [int]` qui lance une partie à un joueur et renvoie la liste successive des cases visitées sur le plateau. Elle commencera donc forcément par `0` et finira forcément par `100`. Le choix du mode de fin est en argument, de façon similaire à la question précédente.

3. Plus court chemin

On souhaite, dans cette partie, utiliser un algorithme glouton pour trouver la partie la plus courte.

Question 8 Écrire une fonction `casesAccessibles(case: int) -> [int]` qui renvoie la liste des 6 cases accessibles pour la case donnée en entrée. Vous utiliserez la fonction `avanceCase` de la question 6. La liste renvoyée `cases` doit avoir le codage suivant : `case[i]` doit correspondre à la case d'arrivée avec le résultat de dé `i+1` (donc la liste retournée doit toujours avoir une longueur de 6). On prendra l'option de fin rapide.

Question 9 Écrire une fonction `meilleurChoix(case: int) -> int` qui renvoie la meilleure case accessible depuis `case`. Il est interdit d'utiliser la fonction `max` dans cette question.

L'algorithme glouton consistera à choisir la valeur du dé permettant de maximiser son déplacement à chaque coup.

Question 10 Écrire une fonction `partieGloutonne() -> [int]` qui renvoie la liste des cases par lesquelles passe le pion dans l'algorithme glouton.

Cette dernière fonction nous renvoie `[0, 38, 44, 50, 67, 91, 97, 100]`.

Question 11 Construire un exemple de plateau, contenant par exemple 2 échelles et pas de serpent, pour lequel notre algorithme ne trouve pas le chemin le plus court en nombre de coups : vous préciserez le résultat de l'algorithme glouton et un exemple d'une partie strictement plus rapide.

4. Etude du graphe correspondant

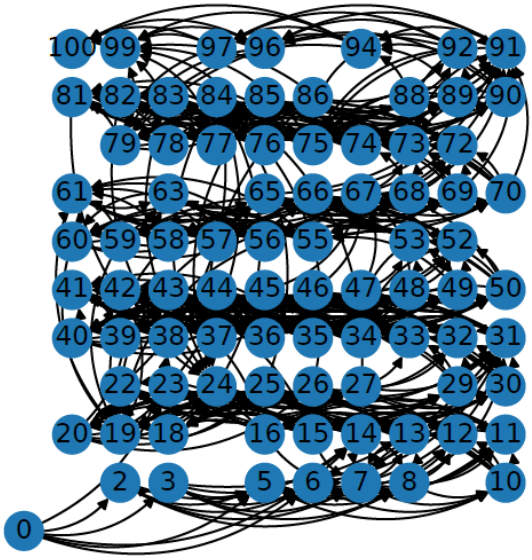

FIGURE 2 – Graphe correspondant au plateau de la FIGURE 1

Question 12 Écrire une fonction `eliminationDoublon(L: [int]) -> [int]` qui renvoie une liste ayant exactement les mêmes éléments, mais sans répétition. Vous n'utiliserez ni dictionnaire, ni tri.

Question 13 Prouver que cette fonction est de complexité quadratique en la longueur de la liste dans le pire des cas.

Question 14 Écrire une fonction `eliminationDoublon(L: [int]) -> [int]` ayant une complexité en $O\left(len(L)\log(len(L))\right)$ en utilisant un tri sur `L`.

Question 15 Proposer une fonction `eliminationDoublon(L: [int]) -> [int]` ayant une complexité linéaire en la longueur de la liste. Vous pourrez vous aider d'un dictionnaire.

On souhaite créer un graphe à partir du jeu : chaque case stable est un sommet et il existe un arc de la case `case1` à la case `case2` si un lancer de dé permet d'aller de `case1` à `case2`. On notera que la case `100` n'a pas de successeurs (puisque le jeu s'arrête). De plus, une case `0` sera utilisée pour prendre en compte le départ. Une représentation de ce graphe est donnée sur la FIGURE 2.

Question 16 En vous aidant des fonctions `casesAccessibles` et `eliminationDoublon`, construire un graphe orienté `G` sous la forme d'un dictionnaire d'adjacence (dictionnaire de listes).

Question 17 Pourquoi a-t-on besoin d'éliminer des doublons ? Prenez un exemple pour illustrer.

Question 18 Quel est le nombre de sommets de ce graphe en fonction de N_{cases} et N_{seE}.

Afin de résoudre le problème du parcours à nombre de coups minimum, on a mis au point une fonction qui trouve si un chemin existe entre un sommet de départ et un sommet d'arrivée :

```
1  def chemin(G, depart, arrivee):
2      predecesseur = {}
3      file = deque([depart])
4      predecesseur[depart] = None
5      while file:
6          sommet = file.popleft()
7          for successeurDeSommet in G[sommet]:
8              if successeurDeSommet not in predecesseur.keys():
9                  file.append(successeurDeSommet)
10                 predecesseur[successeurDeSommet] = sommet
11     return arrivee in predecesseur
```

Question 19 Sur quel type de parcours est basée la fonction `chemin` ? Justifiez cette réponse. Précisez l'intérêt d'utiliser ce parcours ?

Pour l'instant, la fonction renvoie un booléen, mais on souhaiterait qu'elle renvoie l'ensemble des cases par lesquelles on est passé pour passer de `depart` à `arrivee` avec, comme premier élément de la liste retournée `depart` et comme dernier élément de la liste retournée `arrivee`. La fonction renverra une liste vide si aucun chemin n'a été trouvé.

Question 20 Remplacer la ligne 11 par un ensemble de lignes permettant de répondre à cette exigence.

Question 21 Écrire une fonction `partieOptimale() -> [int]` qui renvoie une liste de case à longueur minimum partant de 0 et arrivant à 100 en vous aidant de la fonction mise en place précédemment.

Cette dernière fonction nous renvoie `[0, 38, 39, 45, 67, 91, 94, 100]`. Pour ce plateau, elle ne nous donne pas une partie plus courte que l'algorithme glouton, et un chemin qui n'est pas fondamentalement différent.

5. Statistiques

On souhaite mettre en place des outils pour s'assurer que le jeu vérifie des standards quant à la longueur d'une partie, en particulier en fonction de la stratégie de fin de partie.

Le code pour compiler la longueur des parties est donné ici :

```
1 longR, longI, longQ = [], [], []
2 for i in range(5000):
3     longR.append(len(partie('r'))-1)
4     longI.append(len(partie('i'))-1)
5     longQ.append(len(partie('q'))-1)
```

Question 22 Expliquer en quoi la soustraction par 1 est nécessaire pour connaitre la longueur de la partie (la longueur d'une partie étant défini par le nombre de lancers de dé).

On va vouloir s'intéresser à différentes statistiques sur ces longueurs de parties, comme la moyenne, l'écart-type ou encore la médiane. L'argument d'entrée pour les 4 questions suivantes est une liste d'entiers (`longR`, `longI` ou `longQ`).

Question 23 Écrire une fonction `moyenne(L: [int]) -> float`.

Question 24 Écrire une fonction `variance(L: [int]) -> float`. On souhaite une complexité linéaire.

Question 25 En déduire une fonction `ecartType(L: [int]) -> float`.

La fonction `mediane(L: [int]) -> int`, qui prend en argument `L` et renvoie la valeur médiane du tableau de valeurs `L` est définie ci-dessous. La recherche de la médiane est basée sur un algorithme de tri.

```
1  def mediane(L):
2      n = len(L)
3      for i in range(n):
4          cle = L[i]
5          j = i
6          while 0 < j and cle < L[j-1]:
7              # ligne à compléter
8              # ligne à compléter
9          L[j] = cle
10     return L[n//2]
```

Question 26 Choisir une des 4 propositions données pour compléter les 2 lignes manquantes.

Propositions :

1.
```
L[j-1] = L[j]
j = j-1
```

2.
```
L[j] = L[j-1]
j = j-1
```

3.
```
L[j+1] = L[j]
j = j+1
```

4.
```
L[j] = L[j+1]
j = j+1
```

Question 27 Donner le nom, puis la complexité de l'algorithme de tri employé, dans le meilleur des cas (liste déjà triée) et le pire des cas (liste triée à l'envers).

6. Plateau de jeu

On souhaite faire une représentation schématique du plateau (voir FIGURE 3) avec le code suivant :

```
for case in range(1, 101): # pour les cases 1 à 100
    i, j = position(case)
    plt.text(i, j, str(case),
        horizontalalignment='center', verticalalignment='center')

for caseD, caseA in dSeE.items():
    iD, jD = position(caseD)
    iA, jA = position(caseA)
    if caseA > caseD:
        couleur = 'b'
    else:
        couleur = 'r'
    plt.plot(iA, jA, '^', color=couleur)
    plt.plot(iD, jD, 'o', color=couleur)
    plt.plot([iA, iD], [jA, jD], color=couleur)
plt.axis("equal")
plt.show()
```

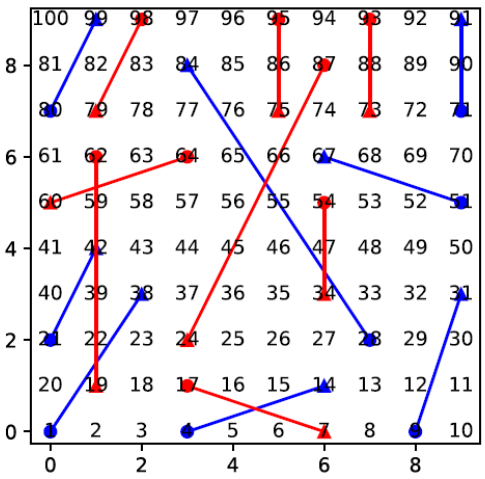

FIGURE 3 – Représentation schématisée du plateau de la FIGURE 1

Question 28 Écrire une fonction `position(case: int) -> (int, int)` qui renvoie les coordonnées de la case numérotée `case` sous las forme d'un couple d'entier. On doit avoir `position(1)` qui renvoie `(0, 0)` (coin en base à gauche) et `position(91)` qui renvoie `(9, 9)` (coin en haut à droite).

Question 29 En commentant la ligne 6, dites à quoi correspondent les variables `caseD` et `caseA` par rapport au dictionnaire `dSeE`.

Question 30 Commenter le rôle des lignes 9 à 12 du code fourni.

7. Annexes

Utilisation du module `random`

On vous donne les docstrings correspondant à deux fonctions du module `random` :

```
randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.

choice(seq) method of random.Random instance
    Choose a random element from a non-empty sequence.
```

Complexité des opérations sur les listes et dictionnaires

Principales opérations sur les listes `n`, longueur de la liste `L`, `k`, un indice valide en négatif (`1` à `n`).

Opération	Moyen
Longueur (`len(L)`)	$O(1)$
Accès en lecture d'un élément	$O(1)$
Accès en écriture d'un élément	$O(1)$
Copie (`L.copy()` ou `L[:]`)	$O(n)$
Ajout (`L.append(elt)` ou `L+=[elt]`)	$O(1)$
Extension (`L1.extend(L2)` ou `L1+=L2`)	$O(n_2)$
Concaténation (`L1 + L2`)	$O(n_1+n_2)$
Test de présence (`elt in L`)	$O(n)$
Désempiler dernier (`L.pop()`)	$O(1)$
Désempiler autre (`L.pop(-k)`)	$O(k)$
Maximum ou minimum (`max(L)` et `min(L)`)	$O(n)$
Tri (`L.sort()` ou `sorted(L)`)	$O(n\log(n))$

Principales opérations sur les dictionnaires `n`, longueur du dictionnaire `d`, `k`, une clé du dictionnaire.

Opération	Moyen
Longueur (`len(d)`)	$O(1)$
Accès en lecture d'un élément (`x = d[k]`)	$O(1)$
Accès en écriture d'un élément (`d[k] = x`)	$O(1)$
Copie (`d.copy()`)	$O(n)$
Ajout (`d[k] = x` la première fois)	$O(1)$
Test de présence (`k in d`)	$O(1)$
Retrait d'un élément (`del d[k]` ou `d.pop(k)`)	$O(1)$

Utilisation du module `matplotlib`

`plt.text(x, y, s)` permet de placer la chaine de caractères `s` aux coordonnées `(x, y)`. Des arguments optionnels permettent la méthode de placement horizontal (`horizontalalignment` qui peut prendre les valeurs `'center'`, `'right'` ou `'left'`) et vertical (`verticalalignment` qui peut prendre les valeurs `'center'`, `'top'`, `'bottom'`, `'baseline'`, `'center_baseline'`).

`plt.plot(x, y, '^')` permet de placer un point aux coordonnées `(x, y)` sous la forme d'un triangle vers le haut. Beaucoup d'autres existent, en plus du triangle vers le haut, dont (liste non-exhaustive) : `'v'`, `'<'` et `'>'` pour des triangles orientés différemment ; `'.'` et `'o'` pour des disques plus ou moins gros ; `'s'`, `'p'` et `'h'` pour des figures régulières à 4, 5 ou 6 côtés.

`plt.plot(Sx, Sy)` permet de tracer une courbe en trait plein en reliant les points dans l'ordre des 2 séquences données en entrées (le point de coordonnées `(Sx[i], Sy[i])` est lié au point de coordonnées `(Sx[i+1], Sy[i+1])`.

Un argument de couleur peut être utilisé avec `plt.plot` : des raccourcis existent pour les couleurs le plus fréquentes : `'b'` pour bleu, `'r'` pour rouge, `'g'` pour vert, `'c'` pour cyan, `'m'` pour magenta, `'y'` pour jaune, `'k'` pour noir et `'w'` pour blanc.

`plt.axis('equal')` permet de contraindre un ratio de 1 entre l'échelle en abscisses et en ordonnées (repère orthonormé).

`plt.show()` crée la figure en cours dans une nouvelle fenêtre.

Rappel de statistique : variance et écart-type

La variance des n nombres $x_1, ..., x_n$. est la moyenne des carrés des écarts à la moyenne, c'est-à-dire :

$$\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^2 = \frac{1}{n}\sum_{i=1}^{n}x_i^2 - \overline{x}^2 \quad \text{avec} \quad \overline{x} = \frac{1}{n}\sum_{i=1}^{n}x_i$$

L'écart-type est la racine carré de la variance.