EXERCICE 1 (6 points)

Cet exercice porte sur les listes de listes ainsi que la programmation orientée objet.

Le solitaire bulgare est un jeu de cartes dans lequel on forme des piles de cartes dont seul le nombre de cartes est important. On dispose au départ les cartes sur une seule pile. À chaque tour, on prend une carte de chaque pile et on forme une nouvelle pile à l'aide de ces cartes. Quand une pile a été vidée, on considère qu'elle disparait.

L'ensemble des tailles des piles est appelé l'état et on peut le représenter par une liste triée d'entiers dans l'ordre croissant.

Par exemple, si les piles contiennent respectivement 4, 6 et 10 cartes, elles seront représentées par la liste `[4, 6, 10]`. Pour passer à l'étape d'après, on pioche la carte au sommet de chacune de ces piles. On pioche ainsi trois cartes et les piles sont de tailles respectives 3, 5 et 9. On ajoute les trois cartes sur une nouvelle pile et on obtient alors quatre piles de tailles 3, 3, 5 et 9, représentées par la liste `[3, 3, 5, 9]`.

On pourra noter `[4, 6, 10] → [3, 3, 5, 9]` cet enchainement d'état.

Si le jeu de départ comporte 15 cartes, on commence avec une unique pile représentée par `[15]`. On ne peut piocher qu'une carte, ce qui amène aux deux piles `[1, 14]`. En continuant, on pioche deux cartes, se faisant la pile de taille 1 est vidée alors qu'on rajoute une pile de taille 2. La situation est alors `[2, 13]`. On a donc l'enchainement `[15] → [1, 14] → [2,13]`.

Si on continue, la partie se déroule ainsi :

```
[15] → [1, 14] → [2, 13] → [1, 2, 12] → [1, 3, 11] →
[2, 3, 10] → [1, 2, 3, 9] → [1, 2, 4, 8] → [1, 3, 4, 7] →
[2, 3, 4, 6] → [1, 2, 3, 4, 5] → [1, 2, 3, 4, 5] etc.
```

On constate que l'état `[1, 2, 3, 4, 5]` se réduit sur lui-même : on pioche cinq cartes, on vide ainsi la première pile et on se ramène à des piles de taille 1, 2, 3 et 4 auxquelles on rajoute cette nouvelle pile de cinq cartes. On retombe alors sur `[1, 2, 3, 4, 5]`. On dit que l'état `[1, 2, 3, 4, 5]` est stable.

Un deuxième exemple à la main

1. Vérifier qu'avec 10 cartes, la partie se termine de nouveau avec un état stable.

Par ailleurs, il est également possible que la partie se termine par un cycle comme la partie commençant avec 12 cartes :

```
[12] → [1, 11] → [2, 10] → [1, 2, 9] → [1, 3, 8] → [2, 3, 7] →
[1, 2, 3, 6] → [1, 2, 4, 5] → [1, 3, 4, 4] → [2, 3, 3, 4] →
[1, 2, 2, 3, 4] → [1, 1, 2, 3, 5] → [1, 2, 4, 5] → [1, 3, 4, 4] etc.
```

Avec 12 cartes, la partie aboutit au bout de 12 itérations par un cycle : les piles suivantes sont les mêmes que celles que l'on avait calculées à partir de la 8e itération.

Un troisième exemple à la main

2. Vérifier qu'avec 9 cartes, la partie se termine avec un cycle.

On souhaite maintenant écrire un programme permettant de savoir si le solitaire bulgare se termine par un état stable ou par un cycle.

Une classe pour la partie

```
1 class Partie:
2     def __init__(self, nb_cartes):
3         """
4             self.courant est l'état actuel
5             self.precedents est la liste des états précédents
6         """
7         self.courant = [nb_cartes] # état initial
8         self.precedents = []
9
10    def suivante(self):
11        """
12            place l'état courant dans la liste des états
                précédents
13            calcule le nouvel état courant et le remplace
14        """
15        self.precedents.append(...)
16        valeurs = self.courant
17        suiv = [len(valeurs)]
18        for v in valeurs:
19            if v > 1:
20                suiv.append(v-1)
21        suiv.sort() # tri des valeurs dans l'ordre croissant
22        self.courant = ...
23
24    def est_stable(self):
25        ...
26
27    def est_cyclique(self):
28        """
29            renvoie True si l'état courant a déjà été rencontré
30        """
31        return self.courant in self.precedents
```

3. Compléter les lignes 15 et 22 de la méthode appelée `suivante` qui permet de passer d'un état de la partie à l'état suivant.

4. Compléter la méthode `est_stable` qui renvoie `True` si l'état courant est le même que le dernier état précédent et `False` sinon.

5. Écrire une fonction `partie_stable` prenant en paramètre `nb_cartes` et qui renvoie `True` si la partie finie par un état stable et `False` si la partie finie par un état cyclique.
   On pourra suivre l'algorithme suivant : créer une instance de partie avec le bon nombre de cartes puis tant que la partie n'est ni stable ni cyclique, passer à l'état suivant.

6. Proposer des tests pour vérifier si votre fonction `partie_stable` semble correcte.

7. Dans la méthode `suivante`, on trie une liste qui est déjà triée sauf éventuellement la valeur `len(valeurs)`. Donner, en le justifiant, un tri classique qui serait plus efficace dans ce cas.