Les six parties qui suivent sont indépendantes.

I Graphes et cycles

I.1 Définition et représentation

▸ **Question 1.** *Définir proprement la structure de graphe, et ce qu'elle permet de modéliser (exemples pouvant être donnés en classe).*

▸ **Question 2.** *Par quelles structures de données peut-on représenter un graphe ? Quels sont leurs avantages/inconvénients ?*

I.2 Recherche de cycle

Dans ce qui suit, on considère des graphes non-orientés.

▸ **Question 3.** *Définir proprement le problème de détection de cycle et le motiver.*

▸ **Question 4.** *Proposer un algorithme pour la détection de cycle adapté à chacun des structures de données proposées antérieurement, et donner leur complexité.*

▸ **Question 5.** *Implémenter celle reposant sur un parcours de graphe. Vous vous baserez sur le squelette déjà implémenté.*

II Complexité

II.1 "Que veut dire $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$?"

Une élève vous demande, suite à un article qu'elle a lu mais pas tout à fait compris, de lui expliquer ce que signifie la *question du millénaire* $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$.

▸ **Question 1.** *Définir proprement la classe de complexité P, et introduire la classe de complexité NP avec moins de formalisme. Répondre alors à la question de l'élève.*

▸ **Question 2.** *Montrer que tout problème de P est dans NP.*

II.2 Le 3-coloriage de graphe

Un problème notoirement NP-complet est le 3-coloriage de graphe.

▸ **Question 3.** *Poser le problème.*

▸ **Question 4.** *Montrer que ce problème est dans NP.*

▸ **Question 5.** *Donner un algorithme linéaire pour le 3-coloriage de graphe pour une classe de graphe de votre choix. Vient-on de prouver que le problème de 3-coloriage est dans P ?*

III Étranges expériences

▸ **Question 1.** *Exécutez une à une les expériences suivantes, et donnez une explication pour chacune d'elle. Vous devrez probablement développer rapidement des concepts.*

```
import time

def experience1():
    N = 10000

    A = [[1 for _ in range(N)] for _ in range(N)]
    start = time.time()
    for i in range(N):
        for j in range(N):
            A[i][j] += 1
    stop = time.time()
    print("Temps V1: ", stop - start)

    A = [[1 for _ in range(N)] for _ in range(N)]
    start = time.time()
    for i in range(N):
        for j in range(N):
            A[j][i] += 1
    stop = time.time()
    print("Temps V2: ", stop - start)

def experience2():
    def grapheVide(n):
        return [[]] * n

    g = grapheVide(6)
    g[0].append(1)

    print("(g[5]==[]):", g[5]==[])

def experience3():
    un_dixieme    = 1/10
    trois_dixieme = 3/10

    print("un_dixieme:", un_dixieme)
    print("trois_dixieme:", trois_dixieme)
    print("(3 * un_dixieme == trois_dixieme):", 3 * un_dixieme == trois_dixieme)

def experience4():
    strange_number = 1e+308 # Rappel: "e+X" est une syntaxe pour "*10^X"
    print("(sn != sn * 2 / 2):", strange_number != strange_number * 2 / 2)

def experience5():
    strange_number = 1e+308
    print("(10^1000 < 2 * sn):", 10 ** 1000 < 2 * strange_number)
```

IV Pile et file

▸ **Question 1.** *Définir la structure de pile et celle de file.*

▸ **Question 2.** *Montrer qu'il est possible de réaliser une file à l'aide de deux piles. Donner la complexité des opérations de base sur la file* dans le pire cas. *Est-elle une mesure pertinente ? si non, proposer une meilleure mesure.*

▸ **Question 3.** *Implémenter la file en complétant le code fourni.*

```
from queue import LifoQueue

# myStack = LifoQueue()       creates a queue
# myStack.put(x)              put x in myStack
# myStack.get_nowait()        return the first element, or an exception
# myStack.empty()             return True/False if myStack is empty/non-empty

class FifoQueue:
    def __init__(self):
        self.in_stack = LifoQueue()
        self.out_stack = LifoQueue()
    def empty(self):
        # Compléter ici
    def put(self, x):
        # Ici aussi
    def get_nowait(self):
        if self.empty():
            raise ValueError("You cannot get an element from an empty FifoQueue")
        else:
        # Ici aussi
```

▸ **Question 4.** *Définir la structure d'arbre, et quelques quantités intéressantes sur les arbres.*

▸ **Question 5.** *Implémenter* `parcoursArabe` *après avoir donné un algorithme pour.*

```
class Tree:
    def __init__(self, x, leftTree = None, rightTree = None):
        ...

exampleTree = Tree(1, Tree(3, Tree(7), Tree(6)), Tree(2, Tree(5), Tree(4)))

def parcoursArabe(tree):
    '''
        parcoursArabe(exampleTree) doit renvoyer [1,2,3,4,5,6,7].
    '''
    def parcoursArabeAux(file, visites):
        if file.empty():
            return visites
        else:
            # Compléter ici
            return parcoursArabeAux(file, visites)
    # Ici aussi
    return parcoursArabeAux(f, [])
```

V Listes simplement chainées et fonction d'ordre supérieur

V.1 Listes simplement chaînées

▸ **Question 1.** *Définir les listes simplement chainées et montrer à quoi correspondent, en mémoire, les opérations de base.*

▸ **Question 2.** *Connaissez-vous d'autres types de liste ? Quels avantages ont-ils par rapport aux listes simplement chaînées ?*

▸ **Question 3.** *Implémentez une classe linkedList, en vous appuyant sur le code déjà écrit. Vous vérifierez qu'elle fonctionne bien comme voulu. On rappelle que* `__repr__` *est la fonction utilisée pour l'affichage dans la console (c'est la* `string` *qui est affichée).*

```
class linkedList:
    def __init__(self, l = []):
        if l == []:
            # À compléter
        else:
            # Ici aussi
    def __repr__(self):
        def getList(self):
            # Remplacer les XXX
            if self.XXX == None:
                return []
            else:
                return [self.XXX] + getList(self.XXX)
        return str(getList(self))
    def append(self, ll):
        # À compléter
```

V.2 Fonctions d'ordre supérieur

▸ **Question 4.** *Rappelez brièvement l'algorithme de tri fusion. Quelle est sa complexité (donnez en particulier la relation qui vous permet de la déduire)?*

▸ **Question 5.** *Utiliser la paramètre* `key` *de la fonction* `sorted` *pour produire, à partir d'une permutation de* $[\![0, 19]\!]$*, les listes demandées.*

```
# The sorted function has a (often hidden) parameter key. This is a function.
# The array a is then sorted according to the values array [key(x) for x in a].

import random
dummyFunction = lambda x: 1
liste = random.sample(list(range(20)), 20) # Random permutation of [0..20]

print("Liste: ", liste)
print("Liste triée par ordre croissant: ", sorted(liste, key=dummyFunction)
print("Liste triée par somme des chiffres", sorted(liste, key=dummyFunction))
print("Liste triée par nombre de 1 dans l'écriture binaire: ", sorted(liste, key=dummyFunction))
print("[20, 19, ..., 1, 0]: ", sorted(liste, key=dummyFunction))
print("[0, 19, 1, 18, 2, 17, ..., 9, 10]: ", sorted(liste, key=dummyFunction))
print("[0, 2, 4, ..., 1, 3, ..., 19] ", sorted(liste, key=dummyFunction))
```

VI Sécurité et calcul numérique

VI.1 Sécurité des communications

▸ **Question 1.** *Faire un exposé sommaire autour de la sécurité des communications. Vous donnerez la différence entre la cryptographie symmétrique et l'asymétrique, définirez les deux grandes familles de constructions asymétriques et discuterez leurs notions de sécurité. Vous êtes encouragé à proposer des activités débranchées, des exemples d'application, etc.*

VI.2 Calcul de numérique

▸ **Question 2.** *On souhaite estimer* $\int_0^1 f(x)$. *Donnez des algorithmes déterministes et un algorithme probabiliste (dit de Monte-Carlo) pour cela. On supposera que* f *est continue sur l'intervalle et donc, en particulier, bornée.*

▸ **Question 3.** *On souhaite estimer précisement* x *tel que* $f(x) = 0$. *Donnez deux algorithmes classiques répondant à cette tâche. Vous pourrez restreindre la classe des fonctions* f *sur lesquelles ces algorithmes peuvent s'appliquer.*