Devoir surveillé n°3 - 2h - Jeudi 11 décembre 2025

La calculatrice n'est pas autorisée.

Consignes : *Vous devez répondre directement sur le* ***Document Réponse****, soit à l'emplacement prévu pour la réponse lorsque celle-ci implique une rédaction, soit en complétant les différents programmes en langage Python.*
Important : vous pouvez utiliser les fonctions des questions précédentes, même si vous ne les avez pas toutes implémentées.

Exercice 1 : .

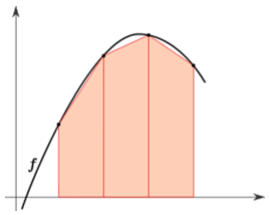
Pour cet exercice, vous pouvez vous référer à l'**annexe 1**, qui contient un rappel des instructions Python dont vous pourriez avoir besoin.
On cherche à calculer une valeur approchée de l'intégrale d'une fonction donnée par des points dont les coordonnées sont placées dans un fichier nommé `integrale.csv`.
Le fichier `integrale.csv` contient une quinzaine de lignes selon le modèle suivant :

$$0.0;1.00988282142$$
$$0.1;1.07221264497$$

Chaque ligne contient une chaine de caractères correspondants à deux valeurs flottantes séparées par un point-virgule, représentant respectivement l'abscisse et l'ordonnée d'un point. Les points sont donnés par abscisses croissantes.

1. Ecrire une suite d'instructions permettant d'ouvrir le fichier en lecture, et de construire la liste `LX` des abscisses et la liste `LY` des ordonnées contenues dans ce fichier. Le type des éléments de `LX` et de `LY` doit être flottant.
2. Ecrire une suite d'instructions permettant de représenter les points, reliés entre eux, sur une figure.

Les points précédents sont situés sur la courbe représentative d'une fonction f. On souhaite déterminer une valeur approchée de l'intégrale I de cette fonction sur le segment où elle est définie.
Pour cela on va utiliser la méthode des trapèzes, consistant à ajoutant les aires de tous les trapèzes délimités par l'axe des abscisses et le segment reliant deux points consécutifs.



3. Ecrire une fonction `trapeze`, d'arguments deux listes `liste_abs` et `liste_ord` de même longueur (correspondant aux abscisses et aux ordonnées des points successifs), qui renvoie l'aire obtenue en appliquant la méthode des trapèzes.
4. Quelle instruction permet alors de calculer une valeur approchée de notre intégrale I ?

Exercice 2 (*Coloriage de graphe*) : .

La coloration d'une carte de pays consiste à attribuer une couleur à chacun des pays de manière à ce que deux pays voisins soient de couleurs différentes.
Étudier ces techniques de coloration revient de façon plus abstraite à travailler sur des graphes.
Le champ d'applications de la coloration de graphes est très vaste et couvre des domaines aussi variés que le problème de l'attribution de fréquences dans les télécommunications, la conception de puces électroniques ou l'allocation de registres en compilation.
Soulignons que tous les graphes considérés dans ce sujet sont non-orientés.
Quelques rappels de syntaxe Python figurent dans l'**annexe 2**.

Introduction sur un exemple

On cherche à colorer une carte de pays avec comme seule contrainte que deux pays ayant une frontière commune ne peuvent pas être de la même couleur.
Comme les pays, les couleurs sont numérotées à partir de zéro.
À titre d'exemple, on considèrera la carte suivante (figure 1), comportant 8 pays numérotés de 0 à 7 ,

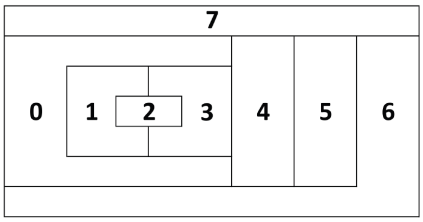


Figure 1 - Exemple d'une carte de pays

que l'on représentera par le graphe G_{ex} donné en figure 2 :

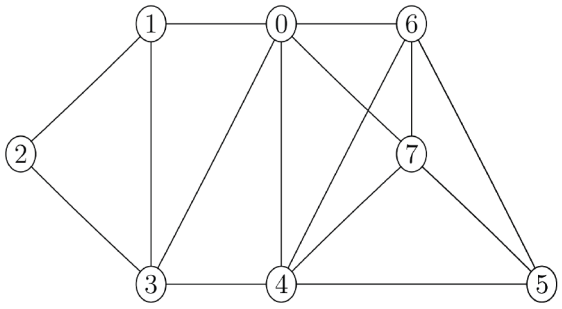


Figure 2 - Graphe G_{ex} associé à la carte de pays de la **figure 1**

Ainsi, sur le graphe de la figure 2, deux pays sont voisins si et seulement si les sommets correspondants sont reliés par une arête.

Q1. Un graphe peut être représenté par une matrice d'adjacence.
Le **DR** contient la matrice d'adjacence pré-remplie du graphe G_{ex}.
La compléter et expliquer le processus de construction.

Q2. Une autre manière de représenter en mémoire un graphe est d'utiliser une liste d'adjacence.
La liste d'adjacence d'un graphe (constitué de sommets numérotés de 0 à $n-1$) est une liste `LA`, telle que `LA[i]` est la liste des sommets voisins du sommet `i`. Par convention, les voisins énumérés dans `LA[i]` seront listés dans l'ordre croissant.
Donner la liste d'adjacence du graphe G_{ex} présenté en exemple.

Q3. Donner un avantage et un inconvénient d'une représentation par matrice d'adjacence.
Donner un avantage et un inconvénient d'une représentation par liste d'adjacence.

Q4. On rappelle que le degré d'un sommet s d'un graphe G est le nombre de voisins du sommet s, c'est-à-dire le nombre de sommets reliés à s.
Le **DR** fournit le tableau des degrés des différents sommets du graphe G_{ex}.
Le compléter.

Tester si une coloration est valide

Q5. Écrire une fonction `voisins` avec trois arguments, deux nombres entiers distincts `i` et `j` et une liste d'adjacence `LA` représentant un graphe, qui renvoie `True` si les sommets numérotés `i` et `j` sont reliés par une arête et `False` sinon.

On considère une carte de pays, représentée par un graphe `G` pour laquelle on a une proposition de coloration donnée par une liste `C`. Ainsi, `C[i]` donne le numéro de la couleur attribuée au sommet `i`. On souhaite déterminer si la coloration proposée est valide (deux sommets du graphe reliés par une arête ne peuvent pas être de la même couleur).

Q6. Écrire la fonction `coloration_valide` avec pour arguments une liste d'adjacence `LA` et une liste de couleurs `C`, qui renvoie `True` si la coloration est valide, `False` sinon.

Q7. Pour un graphe comportant `n` sommets, quelle est la complexité temporelle dans le pire des cas de la fonction `coloration_valide` ?

Un algorithme intuitif de coloration

L'attribution des couleurs à chaque sommet est caractérisée par une liste `C` où `C[i]` est la couleur attribuée au sommet i ; `C[i]` vaut -1 si la couleur n'est pas encore attribuée. La liste `C` ne contient que des -1 au départ et ses valeurs sont modifiées progressivement au fur et à mesure que les couleurs sont attribuées.

Q8. On suppose (dans cette question seulement) que `n` est une constante déjà définie. Écrire la ou les instruction(s) permettant de créer une liste initiale `C` composée de `n` éléments valant -1 .

Q9. Compléter la fonction `colore_sommet` ayant trois arguments, la liste `C` des couleurs attribuées, le numéro `s` du sommet à colorer et la liste d'adjacence `LA` caractérisant le graphe.
Cette fonction ne renvoie rien mais modifie la liste `C` en donnant à `C[s]` la plus petite couleur possible, en fonction des couleurs des sommets voisins qui sont déjà colorés.
Par exemple, pour le graphe G_{ex}, prenons `C=[0,1,-1,-1,-1,-1,-1,-1]`. Les sommets 0 et 1 ont donc déjà été colorés avec les couleurs `C[0]=0` et `C[1]=1`.
L'appel `colore_sommet(C,LA)` modifie la liste `C` en `C=[0,1,0,-1,-1,-1,-1,-1]`. Cela veut dire que le sommet 2 , adjacent au sommet 1 mais pas au sommet 0 , a reçu la même couleur que le sommet 0 .

Q10. À l'aide de **Q9**, écrire une fonction `colorer1` avec pour argument une liste `LA` caractérisant un graphe, qui crée et renvoie la liste `C` des numéros des couleurs attribuées en colorant les sommets un par un par ordre croissant de leurs numéros.
Par exemple, l'application de la fonction `colorer1` au graphe G_{ex} renverra la liste de couleurs `[0,1,0,2,1,0,2,3]`.

Q11. L'ordre de coloration imposé à la question précédente est arbitraire. On souhaite maintenant colorer le graphe en traitant les sommets selon un ordre arbitraire donné en argument.
Écrire une fonction `colorer2` analogue à `colorer1` et avec un argument supplémentaire, une liste ordre fixant l'ordre de coloration des pays.
Par exemple `colorer2([0,2,4,6,1,3,5,7], LA)` colorera le graphe G_{ex} en commençant d'abord par le sommet 0 , puis en continuant par les sommets $2, 4, 6 \ldots$

Q12. Donner la liste des couleurs renvoyée par `colorer2` pour colorer le graphe G_{ex} donné en exemple en prenant `ordre=[7,6,5,4,3,2,1,0]`.
Combien de couleurs ont-elles été utilisées ?

La méthode que nous venons de décrire est rapide et fonctionne plutôt bien. Cependant, si on cherche à utiliser le nombre minimum de couleurs, l'efficacité de l'algorithme proposé ci-dessus dépend en grande partie de l'ordre dans lequel on choisit de colorer les sommets du graphe.
L'objectif des sous-parties suivantes est d'affiner la stratégie pour mieux choisir cet ordre de coloration.

Variante de Welsh-Powell

Une alternative est donnée par la variante de Welsh-Powell. L'idée est de parcourir l'ensemble des sommets du graphe par ordre décroissant de leurs degrés.
Comme le degré d'un sommet est un entier positif, il est possible d'écrire un algorithme de tri efficace (dit par répartition).

Q13. Écrire une fonction `degre` avec pour argument la liste d'adjacence `LA` d'un graphe quelconque, qui renvoie la liste des degrés des sommets du graphe.
Par exemple, pour un graphe de liste d'adjacence `LA=[[1,2],[0,2,3],[0,1,3],[1,2,4],[3]]`, la fonction `degre` renverra la liste `[2,3,3,3,1]`.

Q14. Écrire une fonction `init` avec pour argument un entier n, qui renvoie une liste de listes `R` de taille n, telle que `R[i]` soit une liste vide.
Par exemple, `init(3)` renverra `[[ ], [ ], [ ]]`.

Q15. Écrire une fonction `ranger` avec pour argument une liste d'adjacence `LA`, qui renvoie une liste `R` de même taille que `LA`, telle que `R[i]` soit la liste des sommets de degré i.
Ainsi, pour l'exemple de la question **Q13**, l'appel `ranger(LA)` renverra
la liste `[[ ],[4],[0],[1,2,3],[ ]]`.

Q16. Écrire une fonction `renverse` avec pour argument une liste `L` , qui crée et renvoie une nouvelle liste obtenue en lisant `L` dans l'ordre inverse.
Par exemple, `renverse([1,2,3,4])` renverra `[4,3,2,1]`.

Q17. Écrire une fonction `trier_sommets` avec pour argument une liste d'adjacence `LA`, qui renvoie la liste des sommets triés dans l'ordre décroissant de leur degré. Par exemple, pour un graphe de liste d'adjacence `LA=[[1,2],[0,2,3],[0,1,3],[1,2,4],[3]]`, la fonction `trier_sommets` renverra la liste de sommets `[1,2,3,0,4]`.

Q18. Pour un graphe à n sommets, quelle est la complexité temporelle de la fonction `trier_sommets` dans le pire des cas ?

Q19. Écrire la fonction `colorer3` avec pour argument une liste d'adjacence `LA`, qui crée et renvoie une liste de couleurs `C`, telle que `C[i]` soit la couleur à attribuer au sommet numéro `i`, les sommets étant colorés dans l'ordre décroissant de leur degré.
Quelle est la complexité de `colorer3` dans le pire des cas pour un graphe à n sommets ?

Q20. Pour le graphe G_{ex} de la figure 2, donner la liste `C` des couleurs renvoyée par la fonction `colorer3` .

Annexe 1

Ouverture d'un fichier en lecture	f=open('nom_fichier','r')
Fermeture d'un fichier	f.close()
Récupération des lignes d'un fichier On obtient une liste qui contient des chaines de caractères. La liste obtenue a autant d'élément qu'il y avait de lignes dans la fichier	liste1=f.readlines()
Eliminer le \n en fin d'une chaine de caractères	chaine.strip()
Découper une chaîne de caractères selon un caractère passé en argument On obtient une liste qui contient les blocs de caractères qui étaient séparés	liste2=chaine.split(',')
Tracer une courbe qui relie des points dont les abscisses et ordonnées sont fournies en arguments	import matplotlib.pyplot as plt plt.plot(liste_abs,liste_ord)

Annexe 2

Test d'appartenance	`>>> 2 in [1,2,3,4]`
	`True`
	`>>> 2 in [1,3,4]`
	`False`

Définir une liste	`>>> L=[1,2,3]`
	`>>> L[0]`
	`1`

Ajouter un élément à la fin d'une liste	`>>> L=[1,2,3]`
	`>>> L`
	`[1,2,3]`
	`>>> L.append(5)`
	`>>> L`
	`[1,2,3,5]`

Ajouter tous les éléments d'une liste `L1` à la fin d'une liste `L`	`>>> L=[6,8,7]`
	`>>> L1=[-1,-2]`
	`>>> L.extend(L1)`
	`>>> L`
	`[6,8,7,-1,-2]`