Numéro d'anonymat :

Université de Bordeaux
Département Licence, Informatique, **4TPU114U/4TPU115U**

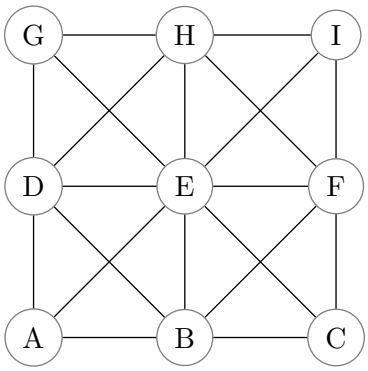
Examen du 18 janvier 2022
11h30-13h00

Le sujet comporte 6 exercices et 9 pages, dont une page d'annexe

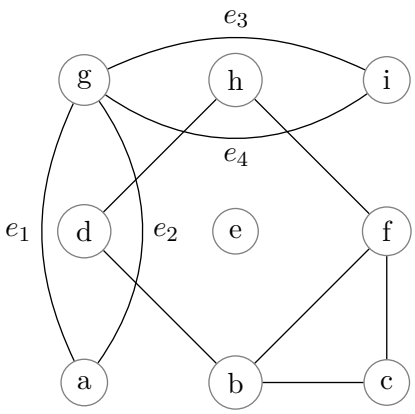
Aucun document n'est autorisé – Toutes les fonctions de manipulation d'images et de graphes disponibles sont rappelées en annexe page 9. Vous pouvez détacher cette annexe pour plus de facilité.

Exercice 1 Les questions de cet exercice portent toutes sur ces trois graphes : G_0, G_1 et G_2

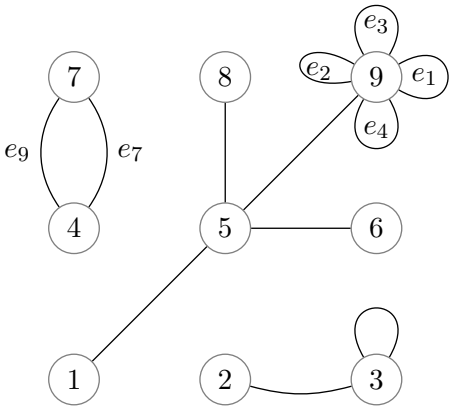
Graphe G_0



Graphe G_1



Graphe G_2



Remplir le tableau ci-dessous.

	G_0	G_1	G_2
Est-il connexe ? (si non, justifier)			
Donner un sommet de degré maximal et préciser son degré.			
Donner un cycle élémentaire de longueur maximale. Vous pouvez donner une simple liste de sommets, sauf s'il y a ambiguité.			

Exercice 2 On considère les deux énoncés :

$$\forall x\ \exists y \quad x * x = y \tag{1}$$
$$\forall x\ \exists y \quad x = y * y \tag{2}$$

On interprète le symbole "=" comme l'égalité et le symbole "$*$" comme le produit des nombres. On considère plusieurs "univers" numériques dans lesquels x, y prennent leurs valeurs.

1. On se place dans l'univers des entiers naturels ($\mathbb{N}$).
   L'énoncé (1) est-il vrai ou faux ?
   L'énoncé (2) est-il vrai ou faux ?

2. On se place dans l'univers des entiers relatifs ($\mathbb{Z}$).
   L'énoncé (1) est-il vrai ou faux ?
   L'énoncé (2) est-il vrai ou faux ?

3. On se place dans l'univers des nombres réels positifs ou nuls ($\mathbb{R}^+$).
   L'énoncé (1) est-il vrai ou faux ?
   L'énoncé (2) est-il vrai ou faux ?

4. Choisissez un des deux énoncés et un univers où il est vrai. Démontrez votre affirmation.

5. Choisissez un des deux énoncés et un univers où il est faux. Démontrez votre affirmation.

Exercice 3 Soit le code ci-dessous :

```
def mystere (x, L:list) -> int:
    i = 0
    lg = len(L)
    while i < lg and L[i] < x:
        i = i + 1
    return i
```

Que retournent les appels suivants :

`mystere(8, [2,3,6,8,10])`

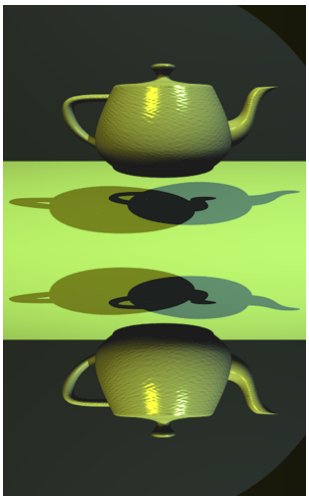
`mystere(9, [2,3,6,8,10])`

`mystere(12, [2,3,6,8,10])`

`mystere(1, [2,3,6,8,10])`

En supposant que la liste `L` est une liste de nombres **triée par ordre croissant** et **sans doublon**, que retourne l'appel `mystere(x, L)` en fonction du nombre `x` et de la liste `L` ?

Exercice 4 On souhaite copier une image au dessus de son "image miroir" dans une autre image (de taille double) comme dans l'exemple ci-dessous où l'image de départ est la célèbre théière représentée dans le fichier `"teapot.png"`.



1. Pour cela, vous devez écrire une fonction `miroir(img1: image, img2: image)` qui copie deux fois dans `img2` l'image `img1` : une fois à l'endroit en haut, et une fois à l'envers en bas. Les paramètres de la fonction seront supposés être deux images `img1` et `img2` telles que `img2` a la même largeur que `img1` et une hauteur égale au double de celle de `img1` (on n'aura donc pas à vérifier si `img2` est correctement dimensionnée).

2. Écrire une suite d'instructions permettant d'obtenir l'image donnée en exemple à partir de l'image stockée dans le fichier `"teapot.png"`.

Exercice 5 Un *stable* est un ensemble de sommets d'un graphe simple (le graphe ayant au moins un sommet) tel que les sommets dans cet ensemble sont deux à deux non voisins. Pour rappel, un graphe est dit simple s'il n'a ni boucle ni arête multiple.

Par exemple, pour le graphe G_0 de l'exercice 1, l'ensemble $\{A, C, I, G\}$ est un stable, tandis que l'ensemble $\{A, B, I, G\}$ n'en est pas un car les sommets A et B sont voisins.

L'objectif de cet exercice est d'écrire une fonction, `stable`, qui, à partir d'un graphe simple, construit et retourne une liste de sommets qui soit un stable pour ce graphe. Un algorithme utilisant le coloriage des sommets est proposé en question 4. Les questions 1 à 3 portent sur des fonctions auxiliaires.

Écrire le code des fonctions suivantes.

1. `colorierEnBlanc(G:graphe)` colorie tous les sommets du graphe `G` avec la couleur blanche.

2. `voisinsTousCouleurBlanc(s:sommet) -> bool` teste si les voisins du sommet `s` sont tous de couleur blanche, c'est à dire retourne `True` si c'est le cas ou si `s` n'a pas de voisins et retourne `False` si `s` a au moins un voisin d'une autre couleur que blanc.

3. `sommetEtVoisinsTousCouleurBlanc(G:graphe) -> sommet` retourne un sommet du graphe `G` tel que ce sommet ainsi que tous ses voisins sont de couleur blanche, ou bien retourne `None` s'il n'existe pas de tel sommet.
   Pour écrire cette fonction, il vous est demandé de faire appel à la fonction `voisinsTousCouleurBlanc`.

4. `stable(G:graphe, c:str) -> list` retourne une liste de sommets du graphe `G` telle que cette liste de sommets soit un stable. Cette fonction a également pour effet de colorier tous les sommets constituant ce stable avec la couleur `c` (qui doit nécessairement ne pas être la couleur blanche).
   L'algorithme à mettre en oeuvre dans la fonction `stable` est le suivant :
   (a) initialisation : colorier tous les sommets du graphe `G` avec la couleur blanche ; choisir aléatoirement un sommet (`s`) de G ; colorier le sommet `s` avec la couleur `c` ; initialiser la liste résultat, `L`, en y mettant le sommet `s`.

(b) tant qu'il existe un sommet `v` de couleur blanche et dont tous les voisins sont de couleur blanche, colorier le sommet `v` avec la couleur `c` et ajouter `v` à la liste résultat `L`.

Pour choisir aléatoirement un sommet du graphe, vous pouvez faire appel à la fonction `elementAleatoireListe` (voir en annexe).

Pour écrire la fonction `stable`, vous devez faire appel aux fonctions écrites précédemment, selon les besoins.

Exercice 6 Soit $L = [x_0, x_1, \ldots, x_i, \ldots, x_{n-1}]$ une liste d'entiers naturels.

1. Montrer que, s'il existe un graphe G à n sommets $s_0, s_1, \ldots, s_i, \ldots, s_{n-1}$ tel que pour tout $i \in [0, n-1]$, le degré du sommet s_i est x_i, alors $\sum_{i=0}^{n-1} x_i = x_0 + x_1 + \ldots + x_{n-1}$ est un nombre pair.

2. On considère le cas particulier où $L = [1, 0, 1, 0, 1, 1]$. Dessiner un graphe à 6 sommets ayant cette liste de degrés.

3. On suppose que chacun des nombres x_i vaut soit 0, soit 1. Montrer que, si $\sum_{i=0}^{n-1} x_i = x_0 + x_1 + \ldots + x_{n-1}$ est un nombre pair, alors on peut construire un graphe simple, à n sommets, qui a la liste de degrés L.

4. On considère maintenant le cas général où les x_i sont des entiers positifs ou nuls. Pour chaque $i \in [0, n-1]$, on suppose que $x_i = 2 \cdot d_i + r_i$, où chaque d_i est un entier naturel et $r_i \in \{0, 1\}$. Soit G_0 un graphe simple qui a pour liste de degrés $[r_0, r_1, \ldots, r_{n-1}]$. Montrer que, en ajoutant à G_0 un ensemble d'arêtes judicieusement choisi, on peut construire un graphe G_1, qui a la liste de degrés L.

FIN

Annexe détachable

Annexe - Vous pouvez détacher cette annexe pour plus de facilité

Voici un rappel des principales fonctions disponibles pour manipuler les images et les graphes (à vous de voir celles qui sont utiles pour ce devoir).

Ci-dessous, `img` est une image	
`ouvrirImage(nom:str) -> image`	Ouvre le fichier *nom* et retourne l'image contenue dedans (par exemple `img = ouvrirImage("teapot.png")`).
`nouvelleImage(large:int, haut:int) -> image`	Retourne une image de taille $large \times haut$, initialement noire.
`afficherImage(img:image)`	Affiche l'image *img*.
`L = largeurImage(img)` `H = hauteurImage(img)`	Récupère la largeur de *img*. Récupère la hauteur de *img*.
`colorierPixel(img:image, x:int, y:int, (r,g,b))`	Peint le pixel (x, y) dans l'image *img* de la couleur (r, g, b)
`(r,g,b) = couleurPixel(img, x, y)`	Retourne la couleur du pixel (x, y) dans l'image *img*

L'argument `G` est un graphe	
`listeSommets(G:graphe) -> list`	retourne la *liste* des *sommets* de `G`
`nbSommets(G:graphe) -> int`	retourne le *nombre* de *sommets* de `G`
`sommetNom(G:graphe, etiquette:str) -> sommet`	retourne le *sommet* de `G` désigné par son *nom* (`etiquette`). Exemple : `s = sommetNom(Europe, 'Italie')`

L'argument `s` est un sommet	
`listeVoisins(s:sommet) -> list`	retourne la *liste* des *voisins* de `s`
`degre(s:sommet) -> int`	retourne le *degré* de `s`
`nomSommet(s:sommet) -> str`	retourne le *nom* (étiquette) de `s`
`colorierSommet(s:sommet, c:str)`	colorie `s` avec la couleur `c`. Exemples de couleurs : `'red'`, `'green'`, `'blue'`, `'white'`, `'cyan'`, `'yellow'`
`couleurSommet(s:sommet) -> str`	retourne la *couleur* de `s`
`marquerSommet(s:sommet)` `demarquerSommet(s:sommet)`	marque le sommet `s` démarque le sommet `s`
`estMarqueSommet(s:sommet) -> bool`	retourne `True` si `s` est marqué, `False` sinon

L'argument `u` est une liste	
`elementAleatoireListe (u:list)`	retourne un élément choisi aléatoirement dans la liste `u` si celle-ci est non vide. Si la liste `u` est vide la fonction retourne une erreur `IndexError`. Exemple : `elementAleatoireListe (listeSommets(G))` où `G` contient un graphe.

Remarque : les couleurs possibles des sommets font partie d'une liste prédéfinie ; par exemple, "white", "black", "red", "green", "blue", "yellow", ...