Numéro d'anonymat :

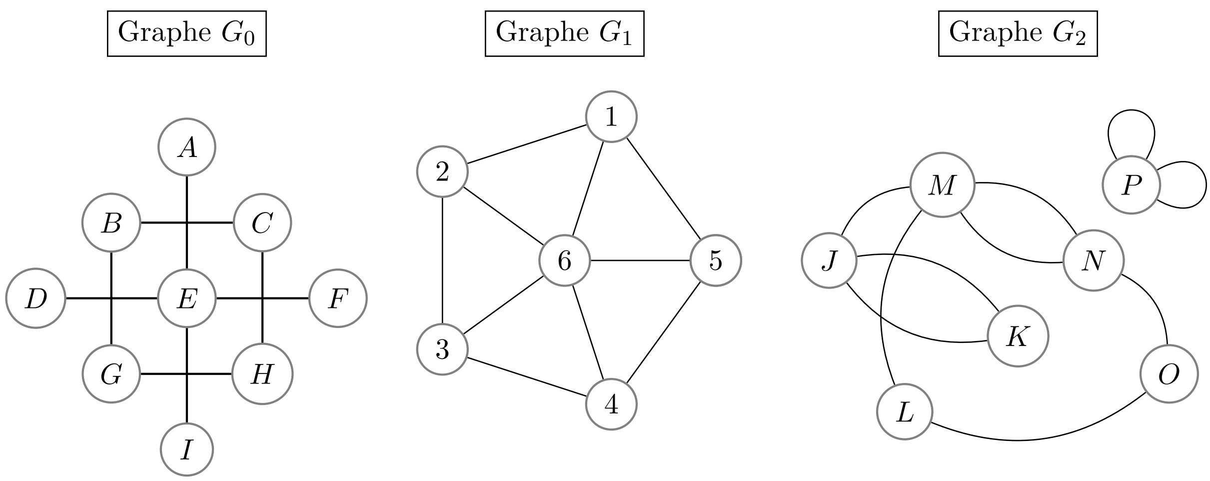
Université de Bordeaux
Département Licence, **4TPM101U**

Examen du 14 janvier 2019
9h00-10h30

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.



Remplir le tableau ci-dessous.

	G_0	G_1	G_2
Donner les ensembles de sommets correspondant aux composantes connexes.			
Donner un sommet de degré maximal et préciser son degré.			
Donner un cycle élémentaire de longueur maximale.			

Exercice 2 Dans les deux questions suivantes, on supposera que l'appel à : `elementAleatoireListe(range(1,7))` retourne successivement les entiers 1, 6, 4, 5, 6, 6, 3, 6, 2, 5, 5, 6,... (c'est à dire que la fonction retourne la valeur 1 lors du premier appel, puis 6 au deuxième appel, puis 4 au troisième, etc.)

On considère tout d'abord la fonction `mystere1` suivante :

```
def mystere1(n):
    test = 0
    cpt = 0
    while test < n:
        e = elementAleatoireListe(range(1,7))
        if e == 6 :
            test = test + 1
        cpt = cpt + 1
    return cpt
```

1. Simuler l'exécution de `mystere1(2)` en complétant le tableau suivant :

test	**0**	0	0																						
cpt		**0**	0																						
e			**1**																						

Que vaut `mystere1(2)` ?

On considère maintenant la fonction `mystere2` suivante :

```
def mystere2(n):
    test = 0
    cpt = 0
    while test < n:
        e = elementAleatoireListe(range(1,7))
        if e == 6 :
            test = test + 1
        else:
            test = 0
        cpt = cpt + 1
    return cpt
```

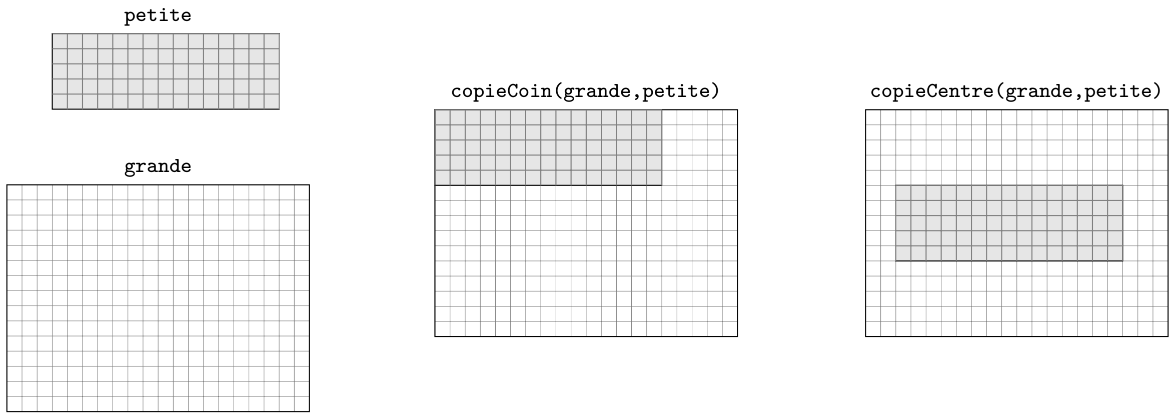
2. Simuler l'exécution de `mystere2(2)` en complétant le tableau suivant :

test	**0**	0	0																						
cpt		**0**	0																						
e			**1**																						

Que vaut `mystere2(2)` ?

3. De façon générale donner la signification (le sens) de la valeur retournée par les appels à `mystere1(k)` et `mystere2(k)` pour un entier positif `k`.

Exercice 3 On souhaite incruster une image dans une plus grande image selon deux manières : dans le coin supérieur gauche ou bien de façon centrée.



1. Écrire une fonction `copieCoin(grande,petite)` qui copie l'image `petite` dans le coin supérieur gauche de l'image `grande`. On pourra supposer que largeur et hauteur de l'image `petite` sont inférieures ou égales à celles de l'image `grande`.

2. Écrire une fonction `copieCentre(grande,petite)` qui incruste l'image `petite` au centre de l'image `grande` (à l'arrondi près). On pourra supposer que largeur et hauteur de l'image `petite` sont inférieures ou égales à celles de l'image `grande`.

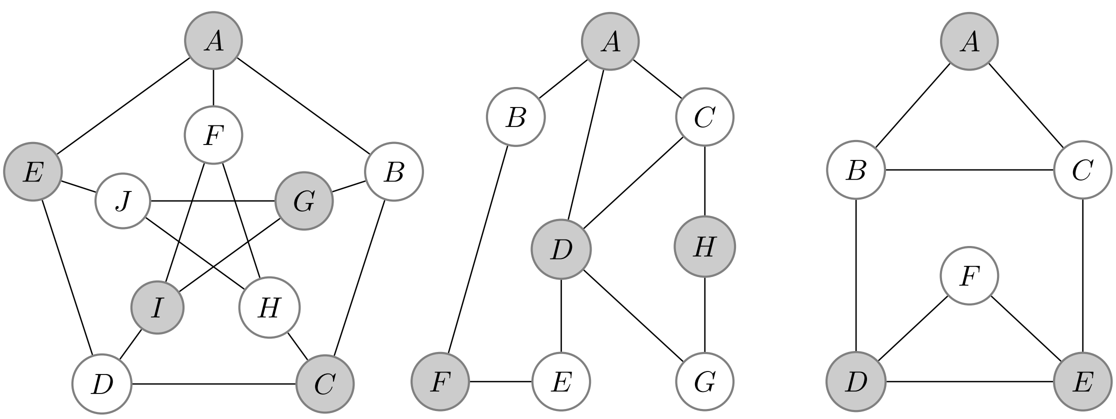
Exercice 4 L'objectif de cet exercice est de déterminer si un graphe possède la propriété $\mathcal{P}$ suivante :

Propriété $\mathcal{P}$: pour toute arête, au moins l'une de ses extrémités est marquée.

Autrement dit, si un sommet n'est pas marqué, alors **tous** ses voisins doivent être marqués.

1. Dans les figures ci-dessous, les sommets marqués sont les sommets grisés. Quels sont les graphes qui ne satisfont pas la propriété $\mathcal{P}$ (justifier) ?

Graphe G_3 Graphe G_4 Graphe G_5



2. Écrire une fonction `tousMarques(s)` qui renvoie `True` si tous les voisins du sommet `s` sont marqués, `False` sinon.

3. Écrire une fonction `aretesCouvertes(G)` qui renvoie `True` si le graphe G satisfait la propriété $\mathcal{P}$, `False` sinon.

Exercice 5 Pour un sommet donné, on cherche à déterminer le nombre de ses voisins différents. Pour ce faire, nous allons utiliser la technique de marquage des sommets afin de ne pas compter plusieurs fois un sommet relié par plusieurs arêtes au sommet donné. On pourra utiliser sans la redéfinir la fonction `demarquerVoisins(s)` qui démarque tous les voisins du sommet donné en paramètre.

1. Écrire une fonction `nbVoisinsDifferents(s)` qui compte et renvoie le nombre de sommets, différents de `s`, reliés à `s` par au moins une arête. On ne doit donc pas compter deux fois un même sommet, tout comme on ne doit pas compter `s` comme un voisin dans le cas où il possède une ou plusieurs boucles.

2. Un sommet est dit *simplement universel* s'il n'a pas de boucle et s'il est relié par exactement une arête à chaque autre sommet du graphe. En utilisant la fonction `nbVoisinsDifferents`, écrire la fonction `estSimplementUniversel(s,G)` qui retourne `True` si le sommet donné en paramètre est simplement universel et `False` autrement.

Exercice 6 – *Suppression d'une arête et connexité.*

Soit G un graphe simple, soit $\mathcal{C}$ un cycle élémentaire dans G, soit e une arête de ce cycle. Soient x et y deux sommets distincts de G (pas forcément compris dans le cycle élémentaire $\mathcal{C}$). On souhaite montrer la propriété suivante :

Propriété 1 : s'il existe une chaîne élémentaire dans G reliant x et y qui passe par l'arête e, alors il existe une autre chaîne (pas forcément élémentaire) reliant x et y mais qui évite l'arête e.

1. Dessiner un exemple de graphe simple qui illustre cette propriété 1. Sur votre exemple, vous penserez à désigner le sommet x, le sommet y, l'arête e ainsi que la chaîne élémentaire qui relie x à y et qui emprunte l'arête e.

2. Démontrer la propriété 1.

3. Soit G un graphe simple, soit $\mathcal{C}$ un cycle élémentaire dans G, soit e une arête de ce cycle. Notons $G' = G \setminus e$ le graphe obtenu à partir de G en supprimant l'arête e (mais en gardant ses extrémités). En supposant que la propriété 1 est vraie et démontrée, cochez parmi les propositions suivantes celle qui est vraie :

   ➪ soient x et y deux sommets distincts de G. S'il existe une chaîne reliant x et y dans G, alors :

   - ☐ il existe une chaîne reliant x et y dans G', uniquement si G est un graphe connexe.
   - ☐ il existe une chaîne reliant x et y dans G'.
   - ☐ il n'existe pas de chaîne reliant x et y dans G'.
   - ☐ il peut dans certains cas exister et dans d'autres cas ne pas exister de chaîne reliant x et y dans G'.
   - ☐ Aucune des 4 affirmations précédentes n'est vraie.

4. En déduire et cochez l'affirmation vraie parmi les proposition suivantes :
   ➪ si l'on supprime une arête (toujours en gardant ses extrémités) d'un cycle élémentaire dans un graphe simple connexe, alors :

   - ☐ le graphe obtenu n'est plus connexe.
   - ☐ le graphe obtenu est toujours connexe.
   - ☐ cela peut dans certains cas détruire la connexité et dans d'autres cas la conserver.
   - ☐ Aucune des 3 affirmations précédentes n'est vraie.

FIN.

Annexe : 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).

L'argument `img` est une image	
`ouvrirImage(nom)`	Ouvre le fichier *nom* et retourne l'image contenue dedans (par exemple `ouvrirImage("teapot.png")`).
`nouvelleImage(largeur,hauteur)`	Retourne une image de taille $largeur \times hauteur$, initialement noire.
`ecrireImage(img, nom)`	Sauvegarde l'image *img* dans le fichier *nom*.
`largeurImage(img)` `hauteurImage(img)`	Récupère la largeur de *img*. Récupère la hauteur de *img*.
`colorierPixel(img, x, y, (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)`	retourne la *liste* des *sommets* de `G`
`nbSommets(G)`	retourne le *nombre* de *sommets* de `G`
`sommetNom(G,etiquette)`	retourne le *sommet* de `G` désigné par son *nom* (`etiquette`). Exemple : `sommetNom(Europe, 'Italie')`

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