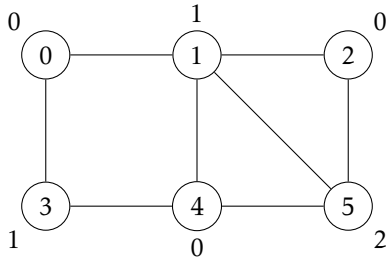
Corrigé : coloriage de graphes

Partie I. Définitions

Question 1.

```
G1 = [[1, 3], [0, 2, 4, 5], [1, 5], [0, 4], [1, 3, 5], [1, 2, 4]]
```

Question 2.



Question 3. Les sommets 1, 4 et 5 sont tous reliés entre eux donc doivent prendre 3 couleurs distinctes. Il ne peut donc exister de 2-coloration.

Question 4.

```
def valide(G, C):
    n = len(G)
    for i in range(n):
        for j in G[i]:
            if C[i] == C[j]:
                return False
    return True
```

Partie II. Degré

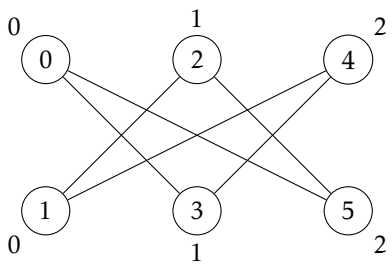
Question 5.

```
def deg(G, s):
    return len(G[s])
```

Question 6.

```
def degMax(G):
    n = len(G)
    dmax = 0
    for s in range(n):
        dmax = max(deg(G, s), dmax)
    return dmax
```

Question 7. Lorsqu'on applique cet algorithme au graphe G_2 on obtient la 3-coloration suivante :



Mais il existe une 2-coloration, qui consiste à colorer d'une première couleur les sommets 0, 2, et 4 et d'une seconde couleur les sommets 1, 3, et 5.

Question 8.

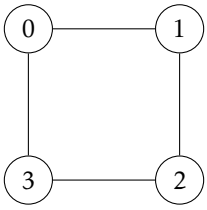
```
1  def deltaColor(G):
2      n = len(G)
3      C = [None for _ in range(n)]
4      for i in range(n):
5          CouleurPossible = [True for _ in range(n)]
6          for j in G[i]:
7              if C[j] is not None:
8                  CouleurPossible[C[j]] = False
9          c = 0
10         while not CouleurPossible[c]:
11             c += 1
12         C[i] = c
13     return C
```

Les lignes 5 à 8 passent en revue les couleurs des voisins du sommet i et les marquent comme `False` dans le tableau `CouleurPossible`. Les lignes 9 à 12 déterminent la première des couleurs possibles et attribuent à i cette couleur.

Partie III. Clique

Question 9. Une clique de taille k nécessite k couleurs distinctes pour être coloriée, donc un graphe contenant une clique de taille k ne pourra être colorié à l'aide de $k-1$ couleurs.

Question 10. Le graphe suivant est 2-coloriable (on colorie d'une même couleur les sommets 0 et 2 et d'une autre couleur les sommets 1 et 3) et il ne contient pas de clique d'ordre 3. La réciproque de la question précédente est donc fausse.



Question 11.

```
def clique(G, S):
    for i in S:
        for j in S:
            if i != j and j not in G[i]:
                return False
    return True
```

Question 12.

```
def cliqueMax(G, S):
    for j in range(len(G)):
        if j not in S and clique(G, S + [j]):
            return False
    return True
```

Partie IV. Graphe biparti

Question 13.

```
1  def biparti(G):
2      n = len(G)
3      Couleur = [None for _ in range(n)]
4      aTraiter = [0]
5      c = 0
6      Couleur[0] = c
7      while aTraiter != []:
8          i = aTraiter.pop()
9          c = 1 - Couleur[i]
10         for j in G[i]:
11             if Couleur[j] is None:
12                 aTraiter.append(j)
13                 Couleur[j] = c
14             elif Couleur[j] != c:
15                 return False
16     return True
```

Notons que la liste `Couleur` joue le rôle de la liste `dejaVu` du cours : la valeur `None` signifie que le sommet n'a pas encore été vu. Une fois le sommet j découvert pour la première fois (ligne 11) il se voit attribuer la couleur opposée du sommet i qui l'a découvert (lignes 9 et 13). S'il est rencontré de nouveau et qu'il est de la même couleur que le sommet i (ligne 14), le graphe n'est pas biparti.