DS 4 Corrigé : Coloriage de graphe 

Définitions ¶

In [2]:
```
# 1.
G1 = [[1, 3], [0, 2, 4, 5], [1, 5], [0, 4], [3, 1, 5], [4, 1, 2]]
```

Q2. Exemple de 3-coloriage possible :

Sommet	0	1	2	3	4
Couleur	0	1	0	1	0

Q3. Les 3 sommets 1, 2, 5 sont tous reliés entre eux donc doivent être de couleurs différentes. D'où la nécessité d'avoir au moins 3 couleurs.

Q4. Il suffit de colorier chaque sommet avec une couleur différente.

In [32]:
```
# Q5
def valid(G, C):
    for u in range(len(G)):
        for v in G[u]:
            if C[u] == C[v]:
                return False
    return True

not valid(G1, [0, 0, 1, 2, 3, 4]) and valid(G1, [3, 0, 1, 0, 3, 4])
```

Out[32]: True

Degré

In [33]:
```python
# 1.
def deg(G, v):
    return len(G[v])
```

In [34]:
```python
# 2.
def deg_max(G):
    maxi = 0
    for v in range(len(G)):
        d = deg(G, v)
        if d > maxi:
            maxi = d
    return maxi

deg_max(G1)
```

Out[34]: 3

In [37]:
```python
# 3.
def delta_color(G):
    n = len(G)
    colors = [-1]*n # les couleurs de chaque sommet
    for u in range(n):
        colors_u = [False]*n # colors_u[c] est True ssi la couleur c est utilisée parmis les voisins de u
        for v in G[u]:
            if colors[v] != -1:
                colors_u[colors[v]] = True
        for i in range(n):
            if colors_u[i] == False:
                colors[u] = i # on donne la couleur i à u
                break
    return colors

delta_color(G1)
```

Out[37]: [0, 1, 0, 1, 0, 2]

Clique

Q1. Tous les sommets d'une clique doivent être de couleur différente.

In [24]:
```python
# 2.
def is_clique(G, V):
    for u in V:
        for v in V:
            if u != v and G[u][v] == 0:
```

```
                return False
    return True
```

2-coloration par parcours en profondeur

In [29]:
```
# 1.
def color2(G):
    C = [-1]*len(G)
    def aux(v, c): # parcours en profondeur sur v, en lui donnant la couleur c
        if C[v] == 1 - c:
            return False
        if C[v] == c:
            return True
        C[v] = c
        for w in G[v]:
            if not aux(w, 1 - c):
                return False
        return True
    if not aux(0, 0):
        return False
    return C

color2(G1)
```

Out[29]: False

In [30]:
```
# 2.
def color2(G):
    C = [-1]*len(G)
    def aux(v, c): # parcours en profondeur sur v, en lui donnant la couleur c
        if C[v] == 1 - c:
            return False
        if C[v] == c:
            return True
        C[v] = c
        for w in G[v]:
            if not aux(w, 1 - c):
                return False
        return True
    for i in range(len(G)):
        if not aux(i, 0):
            return False
    return C

color2(G1)
```

Out[30]: False

Comptage du nombre de couleurs

In [43]:
```python
# 1.
def ncolor1(C):
    L = []
    for c in C:
        if c not in L:
            L.append(c)
    return len(L)

ncolor1([1, 4, 0, 4, 1])
```

Out[43]: 3

Q2. Complexité : O(n^2) car chaque `not in L` est en O(n).

In [44]:
```python
# 3.
def ncolor2(C):
    C.sort()
    n = 1
    for i in range(len(C) - 1):
        if C[i] != C[i + 1]:
            n += 1
    return n

ncolor2([1, 4, 0, 4, 1])
```

Out[44]: 3

In [45]:
```python
# 4.
def ncolor3(C):
    B = [False]*len(C)
    for c in C:
        B[c] = True
    n = 0
    for i in range(len(B)):
        if B[i]:
            n += 1
    return n

ncolor3([1, 4, 0, 4, 1])
```

Out[45]: 3