

# DM d'informatique

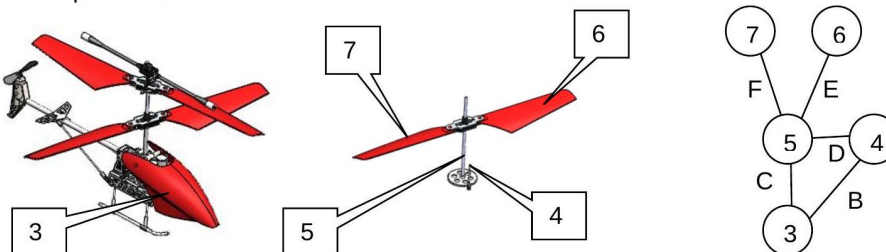
## CORRECTION

### 1 Graphe de liaison

On s'intéresse au graphe de liaison du mécanisme de transmission d'une des pales d'un hélicoptère téléguidé :

Les solides qui seront les sommets du graphe sont :

- 3 : le châssis
- 4 : pignon moteur
- 5 : rotor
- 7 : pale gauche
- 6 : pale droite



1. De quel type est ce graphe (simple, avec cycle, arbre, complet, connexe) ? Justifier votre réponse.

Graphe simple (une seule liaison entre 2 sommets) connexe (un chemin simple est possible entre chacun des sommets) avec cycle (3-4-5-3) donc pas un arbre.

Pas complet car 7, par exemple, n'est pas en liaison avec tous les autres sommets.

### 2 Implémentation d'un graphe non orienté

#### 2.1 Listes des sommets et des arêtes

Un graphe  $G(Ls, La)$  peut être défini par la liste de ses sommets  $Ls$  et de ses arêtes  $La$ .

2. Affecter la liste des sommets  $Ls$  et la liste des arêtes  $La$  du graphe de liaisons (on utilisera le type adapté aux éléments).

```
Ls=[3,4,5,6,7]
La=['B','C','D','E','F']
```

3. Affecter la liste des arêtes sous la forme d'une liste de tuples ('B' par exemple est l'arête (3,4) qui rejoint 3 et 4; on peut indifféremment définir cette arête non orientée par le tuple (4,3)).

```
La=[(3,4),(3,5),(4,5),(5,6),(5,7)]
```

## 2.2 Liste adjacente du graphe

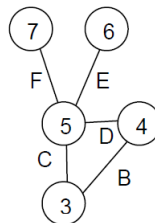
4. Affecter la liste d'adjacence du graphe  $G(Ls,La)$  au dictionnaire  $d\_G$  (les sommets sont les clés et les valeurs sont les listes des sommets accessibles depuis la clé).

```
dG={3: [4, 5] , 4 : [3,5], 5 : [3,4,6,7], 6 :[5], 7 :[5]}
```

## 2.3 Matrice d'adjacence

La matrice d'adjacence du graphe définit pour chaque ligne des sommets, les arêtes à destination des autres sommets positionnés sur les colonnes :

- 0 si pas de relations,
- 1 si la relation existe.



5. Écrire l'affectation de la matrice d'adjacence du graphe  $G(Ls, La)$  sous la forme d'une liste de listes que l'on nommera  $MG$ .

```
MG=[ [0, 1, 1, 0, 0],
      [1, 0, 1, 0, 0],
      [1, 1, 0, 1, 1],
      [0, 0, 1, 0, 0],
      [0, 0, 1, 0, 0]]
```

6. Quelle est la propriété caractéristique de la matrice adjacente de ce graphe non pondéré ? La justifier.

La matrice adjacente MG est **symétrique**. En effet, le graphe est non orienté.

### 3 Propriétés d'un graphe non orienté

7. Écrire l'instruction qui permet d'obtenir l'ordre de ce graphe selon chacune des 3 implémentations du graphe définies dans la partie précédente. Indiquer en commentaire la valeur de cet ordre.

```
print(len(Ls)) # renvoie l'ordre egal a 5
print(len(dG))
print(len(MG))
```

8. Écrire une fonction *deg\_d()* qui admet en argument un sommet *s* et un graphe défini par un dictionnaire *d* et qui renvoie le degré du sommet *s* dans le graphe.

```
def deg_d(s,d):
    return len(d[s])
```

9. Écrire une fonction *deg\_ext(graphe)* qui renvoie :
- le degré maximum et le degré minimum de l'ensemble des sommets du graphe défini par un dictionnaire.
  - les sommets correspondants.

```
def deg_ext(graphe):
    min=10**6
    max=0
    for sommet in graphe:
        if len(graphe[sommet])>max:
            max=len(graphe[sommet])
            sommet_max=sommet
        if len(graphe[sommet])<min:
            min=len(graphe[sommet])
            sommet_min=sommet
    return(min, sommet_min, max, sommet_max)
```

10. Définir une fonction *deg\_M()* qui admet un sommet *s* (entier) et un graphe défini par une matrice d'adjacence *M* et qui renvoie le degré du sommet *s* dans le graphe (on utilisera l'algorithme de la somme des éléments d'une ligne).

```

def deg_M(s,M):
    n=len(M)
    deg=0
    for j in range(n):
        deg=deg+M[s-3][j]
    return deg

```

11. En déduire l'affectation à la variable **d3** du degré du sommet 3 à l'aide de la fonction **deg\_M()**. Donner la valeur de ce degré dans un commentaire.

```

d3 = deg_M(3,MG) # degre 2
print(d3)

```

12. Quelles sont les complexités temporelles des algorithmes de calcul du degré selon que le graphe est défini par une liste d'adjacence (avec dictionnaire) ou par une matrice d'adjacence ?  
 $o(1)$  pour un dictionnaire (une extraction)  
 $o(n)$  pour une liste d'adjacence (où  $n$  est le nombre de sommets du graphe)

## 4 Parcours d'un graphe en largeur

Le parcours en largeur d'un graphe fait appel à une file : le premier sommet découvert sera exploré.

```

Algorithme parcours en largeur (graphe, deb)
''graphe : dictionnaire du graphe
deb : clé de départ du parcours
La fonction renvoie la liste explores des sommets
atteignables depuis deb''
file ← creer_file()
enfiler deb dans file
explores ← liste vide
Tant Que file n'est pas vide Répéter
    nœud_courant ← defiler(file)
    ajouter nœud_courant à explores
    Pour chacun des voisins v de nœud_courant Faire
        Si v n'appartient ni à explores ni à file Alors
            enfiler v dans file
        Fin Si
    Fin Pour
Fin Tant Que
Renvoyer explores
Fin algorithme

```

13. Écrire en python la fonction **parcours()** dont le pseudo code est proposé (on utilisera les fonctions de file défiler(p), enfiler(p,v), creer\_file() et est\_vide(p)). Le graphe est supposé être un dictionnaire.

```
def parcours (graphe, deb):  
    file = creer_file()  
    enfiler(file, deb)  
    explores = [ ]  
    while not est_vide(file) :  
        noeud_courant = depiler(file)  
        explores.append( noeud_courant)  
        for v in graphe[noeud_courant] :  
            if (v not in explores) and (v not in file):  
                enfiler(file,v)  
    return explores
```