

**Exercice 1 :**

1. ordre : 8
2.  $d_D = 4$
3. Le graphe est connexe.
4. Parcours en profondeur (Les nœuds déjà dans la pile n'ont pas été ajoutés à nouveau) :

nœud en cours	état de la pile	chemin parcouru
D	G,A,B	
B	G,A,E	D
E	G,A,F	D,B
F	G,A,C	D,B,E
C	G,A,H	D,B,E,F
H	G,A	D,B,E,F,C
A	G	D,B,E,F,C,H
G		D,B,E,F,C,H,A
		D,B,E,F,C,H,A,G

5. Parcours en largeur du graphe (Les nœuds déjà dans la pile n'ont pas été ajoutés à nouveau) :

nœud en cours	état de la file	chemin parcouru
D	G,A,B	
B	E,G,A	D
A	C,E,G	D,B
G	C,E	D,B,A
E	F,C	D,B,A,G
C	H,F	D,B,A,G,E
F	H	D,B,A,G,E,C
H		D,B,A,G,E,C,F
		D,B,A,G,E,C,F,H

**Exercice 2 :**

1. Cela revient à construire un graphe de 7 sommets dont chaque sommet correspond à une équipe. Comme chaque équipe doit en rencontrer 5 autres, alors chaque sommet serait de degré 5. Cependant la somme des degrés doit être paire (égale au double du nombre d'arêtes). Cette somme étant égale à 35, le tournoi n'est pas réalisable.
2. La somme des degrés est égale à 28. Le tournoi est réalisable. Il faut alors construire le graphe.

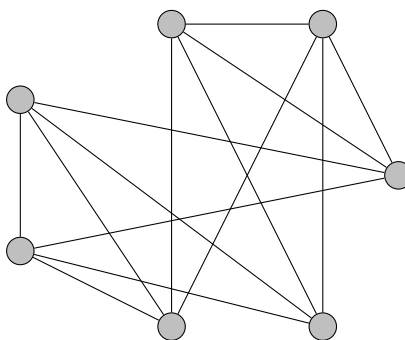


FIGURE 1 – Tournoi de handball

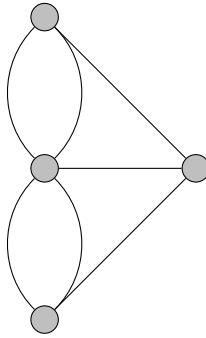
**Exercice 3 :**

FIGURE 2 – Les sept ponts de Königsberg

Tous les sommets sont de degrés impairs.

**Exercice 4 :**

## 1. DFS récursif

```

1 def DFS_rec(graphe: Graphe, sommet: str, visites: list = [])->list:
2     if not(sommet in visites):
3         visites.append(sommet)
4         for voisin in graphe.get_adjacents(sommet):
5             DFS_rec(graphe, voisin, visites)
6     return visites

```

## 2. Test

3. La méthode est tout aussi efficace : elle ne parcourt les arêtes qu'une seule fois.

## 4. Connexité

```

1 def est_connexe(graphe: Graphe)->bool:
2     sommets = graphe.get_sommets()
3     return len(sommets) == len(DFS_rec(graphe, sommets[0]))

```

## 5. DFS récursif avec un dictionnaire

```

1 def DFS_rec_dico(graphe: Graphe, sommet: str, origine: str = None,
2     visites: dict = {})->dict:
3     if not(sommet in visites):
4         visites[sommet] = origine
5         for voisin in graphe.get_adjacents(sommet):
6             DFS_rec_dico(graphe, voisin, sommet, visites)
7     return visites

```

## 6. Chemin

```

1 def chemin(graphe: Graphe, depart: str, arrivee: str)->list:
2     # parcours en profondeur
3     parcours = DFS_rec_dico(g, "D")
4
5     # si arrivee n'est pas atteignable
6     if arrivee not in parcours:
7         return None
8
9     # un chemin

```

```
10     chemin = [arrivee]
11     en_cours = arrivee
12     while not(en_cours == depart):
13         # ajouter l'origine de en_cours
14         origine = parcours[en_cours]
15         chemin.append(origine)
16         en_cours = origine
17     # le chemin a été construit à l'envers
18     chemin.reverse()
19     return chemin
```

### Exercice 5 :

#### 1. BFS avec un dictionnaire

```
1 def BFS_dico(graphe: Graphe, sommet: str)->dict:
2     visites = {sommet: None}
3     voisins = {sommet}
4     prochains = set()
5     while len(voisins) > 0:
6         en_cours = voisins.pop()
7         """
8         get_adjacents renvoie un ensemble
9         """
10        for v in graphe.get_adjacents(en_cours):
11            if v not in visites:
12                # garde son origine
13                visites[v] = en_cours
14                # on l'ajoute aux prochains à visiter
15                prochains.add(v)
16
17        """
18        si on a épuisé tous les voisins on prend
19        les prochains
20        """
21        if len(voisins) == 0:
22            voisins, prochains = prochains, set()
23
24    return visites
```

#### 2. Cette fois il s'agit du plus court chemin.

```
1 def chemin(graphe: Graphe, depart: str, arrivee: str)->list:
2     # parcours en largeur
3     parcours = BFS_dico(g, "D")
4
5     # si arrivee n'est pas atteignable
6     if arrivee not in parcours:
7         return None
8
9     # un chemin
10    chemin = [arrivee]
11    en_cours = arrivee
```

```
12 while not(en_cours == depart):
13     # ajouter l'origine de en_cours
14     origine = parcours[en_cours]
15     chemin.append(origine)
16     en_cours = origine
17 # le chemin a été construit à l'envers
18 chemin.reverse()
19 return chemin
```