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 pair (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.

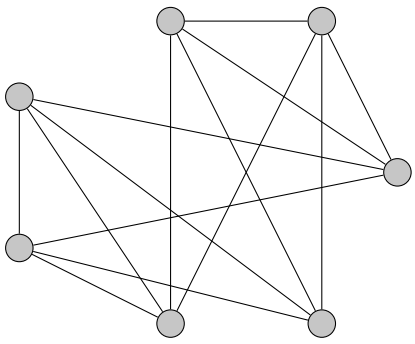

FIGURE 1 – Tournoi de handball

Christophe Viroulaud

Exercice 3 :

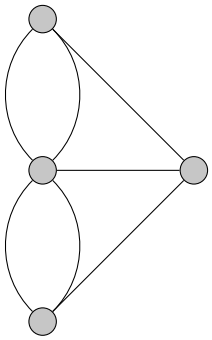

FIGURE 2 – Les sept ponts de Königsberg

Tous les sommets sont de degrés impairs.

Exercice 4 :

1. DFS récursif

```
def DFS_rec(graphe: Graphe, sommet: str, visites: list = [])->list:
    if not(sommet in visites):
        visites.append(sommet)
        for voisin in graphe.get_adjacents(sommet):
            DFS_rec(graphe, voisin, visites)
    return visites
```

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

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

5. DFS récursif avec un dictionnaire

```
def DFS_rec_dico(graphe: Graphe, sommet: str, origine: str = None,
   visites: dict = {})->dict:
    if not(sommet in visites):
        visites[sommet] = origine
        for voisin in graphe.get_adjacents(sommet):
            DFS_rec_dico(graphe, voisin, sommet, visites)
    return visites
```

6. Chemin

```
def chemin(graphe: Graphe, depart: str, arrivee: str)->list:
    # parcours en profondeur
    parcours = DFS_rec_dico(graphe, depart)

    # si arrivee n'est pas atteignable
    if arrivee not in parcours:
        return None

    # un chemin
```


Christophe Viroulaud

```
    chemin = [arrivee]
    en_cours = arrivee
    while not(en_cours == depart):
        # ajouter l'origine de en_cours
        origine = parcours[en_cours]
        chemin.append(origine)
        en_cours = origine
    # le chemin a été construit àl'envers
    chemin.reverse()
    return chemin
```

Exercice 5 :

1. BFS avec un dictionnaire

```
def BFS_dico(graphe: Graphe, sommet: str)->dict:
    visites = {sommet: None}
    voisins = {sommet}
    prochains = set()
    while len(voisins) > 0:
        en_cours = voisins.pop()
        """
        get_adjacents renvoie un ensemble
        """
        for v in graphe.get_adjacents(en_cours):
            if v not in visites:
                # garde son origine
                visites[v] = en_cours
                # on l'ajoute aux prochains àvisiter
                prochains.add(v)

        """
        si on a épuisé tous les voisins on prend
        les prochains
        """
        if len(voisins) == 0:
            voisins, prochains = prochains, set()

    return visites
```

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

```
def chemin(graphe: Graphe, depart: str, arrivee: str)->list:
    # parcours en largeur
    parcours = BFS_dico(g, "D")

    # si arrivee n'est pas atteignable
    if arrivee not in parcours:
        return None

    # un chemin
    chemin = [arrivee]
    en_cours = arrivee
```


Christophe Viroulaud

```
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 àl'envers
18      chemin.reverse()
19      return chemin
```


Christophe Viroulaud