Exercice 1 :

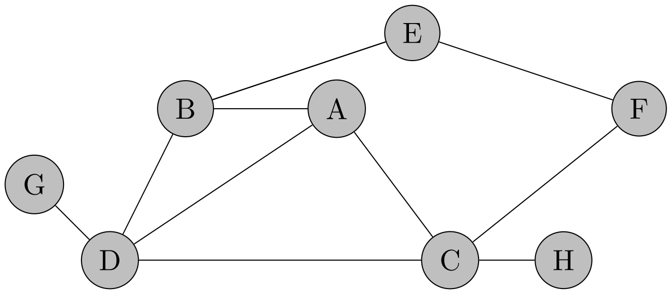

Figure 1 – Graphe à parcourir

Pour les deux parcours il faudra détailler l'évolution de la structure utilisée (pile ou file) et le chemin final parcouru.

1. Quel est l'ordre du graphe 1 ?
2. Quel est le degré du sommet D ?
3. Ce graphe est-il connexe ?
4. Effectuer à la main un parcours en profondeur du graphe 1.
5. Effectuer à la main un parcours en largeur du graphe 1.

Exercice 2 : La ville de Königsberg (aujourd'hui Kaliningrad) est construite autour de deux îles situées sur le Pregel et reliées entre elles par un pont.

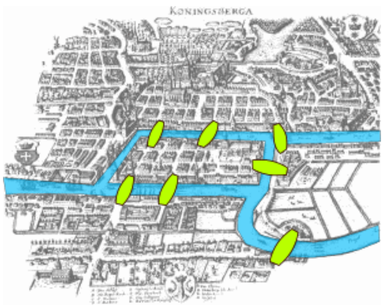

Figure 2 – Les sept ponts de Königsberg

Le problème, énoncé et résolu par Euler au XVIII° siècle, consiste à déterminer s'il existe une promenade permettant en partant d'un point, de revenir à ce même point en ayant traversé une et une seule fois chaque pont.

1. Modéliser la situation par un graphe.
2. Tenter de réaliser la promenade « à la main ».

Ce problème est à l'origine de *la théorie des graphes*. C'est donc Euler qui commença à théoriser des problèmes mathématiques par cette méthode. Un vocabulaire spécifique a été crée en hommage.

Définition :

— une chaîne eulérienne est une chaîne passant une et une seule fois par toutes les arêtes du graphe.

— un cycle eulérien est une chaîne eulérienne dont le sommet de départ et le sommet d'arrivée sont identiques.

Théorème :

— Un graphe connexe possède **une chaîne eulérienne** si et seulement si ses sommets sont tous de degré pair sauf au plus deux.

— Un graphe connexe possède **un cycle eulérien** si et seulement si tous ses sommets sont de degré pair.

Christophe Viroulaud

3. Vérifier si le théorème est vérifié dans le cas des pont de Königsberg.
4. Peut-on créer une chaîne eulérienne pour le graphe 1 ? Et un cycle eulérien ?

Exercice 3 : Le fonctionnement du parcours en profondeur peut être décrit de la manière suivante :

— Choisir un nœud.

— S'il n'est pas déjà visité :

  — le marquer *vu*,

  — pour chaque voisin, effectuer la même démarche.

Nous reconnaissons ici une démarche récursive.

1. Écrire la fonction **DFS_rec(graphe : Graphe, sommet : str, visites : list=list()) → list** qui renvoie la liste des nœuds atteignables depuis *sommet*. Nous utiliserons la classe *Graphe* du cours (sans faire appel aux méthodes de parcours déjà élaborées).
2. Tester la fonction sur le graphe 1.
3. Comparer l'efficacité de cette fonction avec la méthode *DFS* implémentée dans le cours.
4. Écrire alors la fonction **est_connexe(graphe : Graphe) → bool** qui renvoie *True* si le graphe est connexe.
5. Écrire la fonction **DFS_rec_dico(graphe : Graphe, sommet : str, origine : str = None, visites : dict={}) → dict** qui renvoie un dictionnaire des nœuds atteignables depuis *sommet*. Le dictionnaire associera chaque sommet à son origine (sommet depuis lequel on l'a atteint).
6. Écrire la fonction **chemin(graphe : Graphe, depart : str, arrivee : str) → list** qui renvoie un chemin entre *depart* et *arrivee*. Cette fonction utilisera *DFS_rec_dico*. Il est à noter que le chemin obtenu n'est pas nécessairement le plus court.

Exercice 4 : Il n'est pas obligatoire d'utiliser une file pour réaliser un parcours en largeur. Nous pouvons par exemple nous servir de deux ensembles :

— **voisins :** qui contiendra les sommets voisins du sommet d'origine,

— **prochains :** qui contiendra les sommets à une distance $n + 1$ du sommet d'origine et qui seront visités après ceux de l'ensemble *voisins*.

Quand l'ensemble *voisins* est vide, il suffit de le remplir avec les sommets de *prochains*.

1. Écrire la fonction **BFS_dico(graphe : Graphe, origine : str) → dict** qui associe chaque sommet au sommet depuis lequel on l'a atteint lors du parcours en profondeur.
2. Écrire la fonction **chemin(graphe : Graphe, depart : str, arrivee : str) → list** qui renvoie un chemin entre *depart* et *arrivee*. Cette fonction utilisera *BFS_dico*.

Christophe Viroulaud