Exercice 1 : La ville de Königsberg (aujourd'hui Kaliningrad) est construite autour de deux îles situées sur le Pregel et reliées entre elles par un pont.

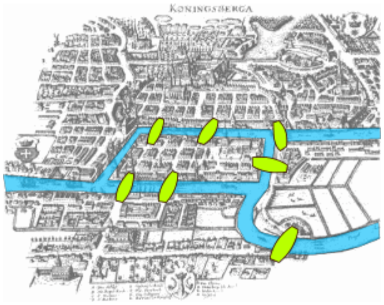


Figure 1 – Les sept ponts de Königsberg

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.

Rappel

— 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.

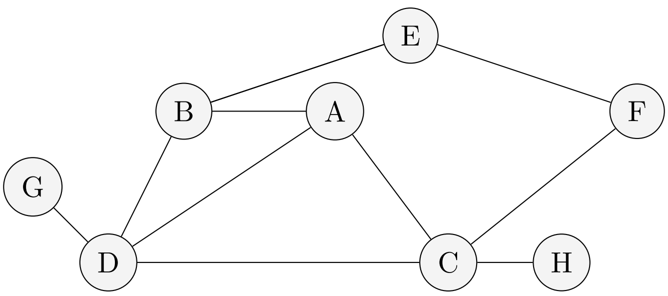
Théorème

— 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.

3. Vérifier si le théorème est vrai dans le cas des ponts de Königsberg.

Exercice 2 :



1. Donner l'ordre du graphe.
2. Donner le degré du sommet D.

Christophe Viroulaud

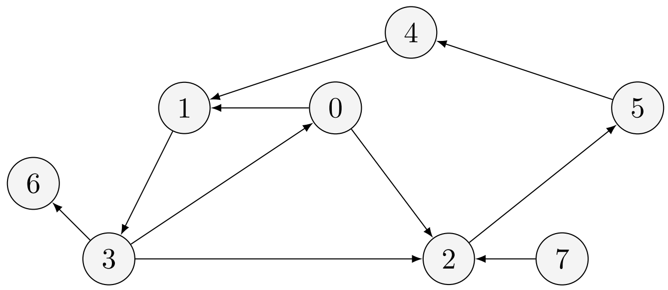
3. Construire le dictionnaire d'adjacence du graphe.
4. Reprendre l'implémentation du parcours en profondeur vu en classe (code 1) et remplacer le tableau **`visites`** par un dictionnaire associant chaque sommet à un booléen.

```python
def profondeur(graphe: dict, noeud: str, visites: list) -> None:
    if noeud not in visites:
        print(noeud, end=" ")
        visites.append(noeud)
        for voisin in graphe[noeud]:
            profondeur(graphe, voisin, visites)
```

Code 1 – Parcours en profondeur vu en cours

5. Écrire la fonction **`get_indice(sommet: str)`** $\rightarrow$ **`int`** qui renvoie l'indice associé à chaque sommet. Par exemple, la fonction renverra **`0`** pour le sommet **`A`**.
6. Reprendre alors l'implémentation (code 1) en remplaçant **`visites`** par un tableau de booléens.

Exercice 3 :



1. Construire la liste d'adjacence des successeurs du graphe.
2. On dispose de la fonction **`parcours`**. Un sommet :
   - **`BLANC:`** n'a pas encore été atteint,
   - **`GRIS:`** est en cours de visite,
   - **`NOIR:`** a terminé son parcours.

   Le tableau **`visites`** associe à chaque sommet, son état **`coul`** et son prédécesseur **`pred`**.

```python
BLANC, GRIS, NOIR = 0, 1, 2

def parcours(graphe: list) -> list:
    visites = [{"coul": BLANC, "pred": None} for i in range(len(graphe))]
    for i in range(len(graphe)):
        if visites[i]["coul"] == BLANC:
            dfs(graphe, i, visites)
    return visites
```

   Écrire la fonction **`dfs(graphe: list, sommet: int, visites: list)`** $\rightarrow$ **`None`** qui effectue récursivement le parcours en profondeur du **`sommet`**. La fonction utilisera le principe des trois couleurs et associera le prédécesseur de chaque voisin.