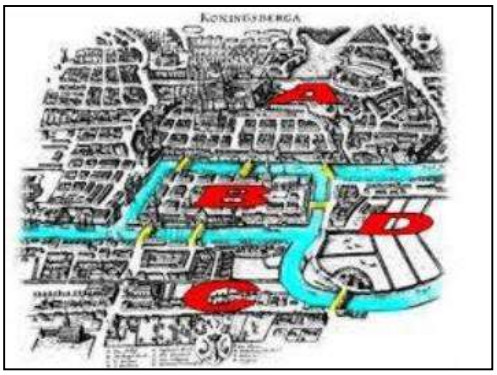
CHAPITRE 4 : INTRODUCTION À LA THÉORIE DES GRAPHES

I. Origines ?

Le problème des sept ponts de Königsberg Résolu par Leonhard Euler en 1736 est à l'origine de la théorie des graphes. Ce problème consiste à déterminer s'il existe ou non une promenade dans les rues de Königsberg permettant, à partir d'un point de départ au choix, de passer une et une seule fois par chaque pont, et de revenir à son point de départ, étant entendu qu'on ne peut traverser le Pregel qu'en passant sur les ponts.



II. Définitions :

- **Graphe :** Un graphe G est défini par G=(V,U), ou V est un ensemble de sommets et U l'ensemble d'arcs(ou arêtes) ;
- **Un arc (ou arête)** : est un couple de sommets, donc, un élément du produit cartésien VxV
- **Graphe orienté et Graphe non orienté :** Si les arêtes ne sont pas orientées, on parle d'un graphe non orienté. Dans le cas contraire on parle d'un graphe orienté et les arêtes sont appelées aussi les arcs.
- **Graphe Pondéré :** Un graphe pondéré est défini par le triplet (***V,U,C***) où : ***V*** est l'ensemble des sommets, ***U*** est l'ensemble des arrêtes (ou arcs), et ***C*** est la fonction de coût de ***U*** dans ***IR***. Par convention C_u représente le coût ou le poids de l'arc (ou de l'arête) u.
- **Graphe Connexe :** Un graphe connexe est un graphe dont tout couple de sommets peut être relie par une chaine de longueur n>=1

III. Vocabulaire de la théorie des graphes :

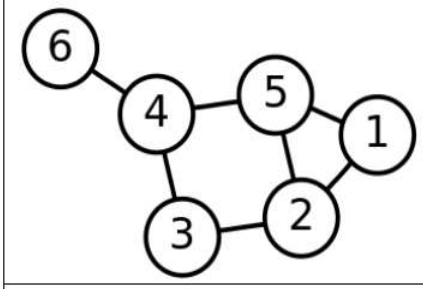
- ***Ordre du Graphe :*** le nombre de sommet du Graphe
- ***Degré d'un sommet :*** nombre d'arêtes reliées à ce sommet
- ***Adjacences:*** Deux arcs sont dits adjacents s'ils ont une extrémité en commun. Et deux sommets sont dits adjacents si un arc les relie.
- ***Boucle :*** est un arc qui part d'un sommet vers le même sommet
- ***Chaîne :*** Une chaine de longueur n est une suite de n arêtes qui relient un sommet i à un autre j ou à lui même.
- ***Cycle :*** Un cycle est une chaine qui permet de partir d'un sommet et revenir à ce sommet en parcourant une et une seule fois les autres sommets.
- ***Distance*** entre deux sommets i et j : est la longueur de la chaine la plus courte qui les relie
- ***Chemin :*** c'est une chaine bien orientée
- ***Circuit :*** est un cycle "bien orienté", à la fois cycle et chemin.
- ***Chaine eulérienne:*** une chaine est dite eulérienne est une chaine comportant exactement une fois toutes les arêtes du graphe.
- ***Cycle eulérien :*** si le sommet de départ d'une chaine eulérienne et celui d'arrivé on parle de cycle eulérienne
- ***Graphe eulérien :*** Un graphe admettant une chaine eulérienne est dit Graphe eulérien

- **Cycle hamiltonien :** c'est un cycle passant une seule fois par tous les **sommets** d'un graphe et revenant au sommet de départ**.**
- **Un graphe simple** : est un graphe sans boucle tel qu'entre deux sommets différents, il y ait au plus un arc.
  - ✓ *Dans un graphe simple, on pourra donc noter un arc à l'aide de ses extrémités initiale et finale.*
  - ✓ *Dans un graphe simple, on peut parler de l'ensemble des arcs.*

IV. Représentation informatique d'un graphe :

Un graphe peut être implémenté de différentes manières selon le langage utilisé. En Python on peut représenter un graphe à l'aide d'un dictionnaire ou à l'aide d'une matrice d'adjacence.

Exemple :

<table>
<tr>
<td> 
</td>
<td>

```
Graphe={
    1:[2,5],
    2:[1,3,5],
    3:[2,4],
    4:[3,5,6],
    5:[1,2,4],
    6:[4],
}
```

</td>
<td>

```
Graphe=[
[0,1,0,0,1,0],
[1,0,1,0,1,0],
[0,1,0,1,0,0],
[0,0,1,0,1,1],
[1,1,0,1,0,0],
[0,0,0,1,0,0]
]
```

</td>
</tr>
<tr>
<td>Graphe</td>
<td>dictionnaire</td>
<td>Matrice d'adjacence</td>
</tr>
</table>

Propriétés de la matrice d'adjacence :

- La matrice est symétrique si le graphe n'est pas orienté.
- La somme des nombres d'une même ligne (ou d'une même colonne) donne le degré du sommet correspondant.
- La diagonale ne contient que des zéros.
- Les termes a_{ij} de la matrice A^n donnent le nombre de chaînes de longueur n reliant i à j.

V. PILE et FILE :

Les notions de pile et de file sont deux structures de données abstraites importantes en informatique.
On limite ci-dessous la présentation de ces notions aux besoins des parcours de graphes envisagés ci-après.
Pile: Structure LIFO (last in, first out)

La structure de pile est celle d'une pile d'assiettes :

- Pour ranger les assiettes, on les empile les unes sur les autres.
- Lorsqu'on veut utiliser une assiette, c'est l'assiette qui a été empilée en dernier qui est utilisée.

FILE (queue): Structure FIFO (first in, first out).

La structure de file est celle d'une file d'attente à un guichet :

- Les nouvelles personnes qui arrivent se rangent à la fin de la file d'attente.
- La personne servie est celle qui est arrivée en premier dans la file.

VI. Parcours de graphe :

Par la suite nous utilisons la représentation en utilisant la matrice d'adjacence d'un graphe

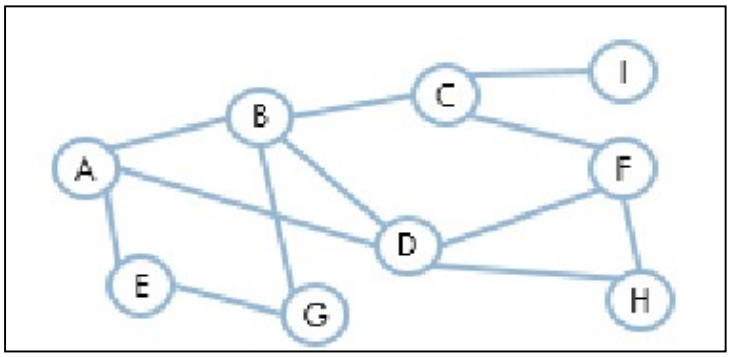
VI-1. Le parcours en profondeur d'abord DFS (Depht First Search)

Algorithme :

- Initialement tous les nœuds sont marqués " non visités".
- Choisir un nœud v de départ et le marquer " visité".
- Chaque nœud adjacent à v, non visité, est à son tour visité en utilisant DFS récursivement.
- Une fois tous les nœuds accessibles à partir de v ont été visités, la recherche de v (DFS(v)) est complète.
- Si certains nœuds du graphe restent "non visités", sélectionner un comme nouveau nœud de départ et répéter le processus jusqu'à ce que tous les nœuds soient visités.

Exemple :

Soit le graphe suivant :



Le parcours en profondeur de ce graphe : A, B, C, F, D, H, I, G, E

En python :

```
def parcoursProfondeur(G):
    test=[0]*len(G)
    for i in range(len(G)):
        if test[i]==0:
            parcoursDFSpile(G,i,test)
```

```
def succNonVisite(G,s,test):
    n=len(G[s])
    L=[]
    for i in range(n):
        if G[s][i]==1 and test[i]==0:
            L.append(i)
    return L
```

On peut réaliser le parcours en profondeur en utilisant la récursivité ou en utilisant les Piles. Ci-après les deux variantes :

```python
def parcoursDFSpile(G,s,test):
    lifo=[s]
    while lifo:
        noeud=lifo.pop()
        if test[noeud]==0:
            print(noeud,end=' ')
            test[noeud]=1
            succ=succNonVisite(G,noeud,test)
            succ.reverse()
            if succ:
                for x in succ:
                    lifo.append(x)
```

```python
def DFSRecursif(G,s,test):
    test[s]=1
    print(s,end=' ')
    for x in succNonVisite(G,s,test):
        if test[x]==0:
            DFSRecursif(G,x,test)
```

parcoursProfondeur(g) : fonction qui lance le parcours, elle peut faire appel soit à la fonction *parcoursDFSpile* soit à la fonction *parcoursDFSRecursif* selon le parcours souhaité
succNonVisite(g,s,test): retourne la liste des successeurs non visités de s

VI-2. Le parcours en largeur d'abord :BFS (Breadh First Search)

On procède par niveau en considérant d'abord tous les sommets à une distance donnée, avant de traiter ceux du niveau suivant.

Exemple : Le parcours en largeur du graphe précèdent visite les sommets dans l'ordre suivant : A, B, D, E, C, G, F, H, I

En python :

```python
def BFSFiles(G,s,test):
    fifo=[s]
    while fifo:
        x=fifo.pop(0)
        if test[x]==0:
            test[x]=1
            print(x,end=' ')
            succX=succNonVisite(G,x,test)
            if succX:
                fifo+=succX
```

```python
def ParcoursEnlargeur(G):
    test=[0]*len(G)
    for i in range(len(G)):
        if test[i]==0:
            BFSFiles(G,i,test)
```