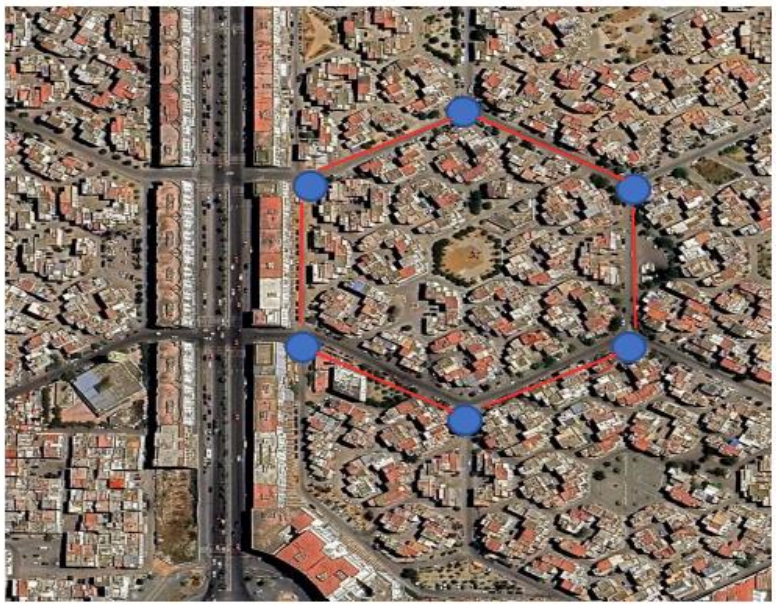
Université Abdelmalek Essaâdi
Faculté des sciences et techniques de Tanger

LST : Analytique des données
Module : Structure des données avancées et théories des graphes

Rapport du Projet



Théorie de Graphe

IMPLEMENTATION EN LANGAGE C

Réalisé par :
Hanaa Belaid
Firdaous Fatmi
Amina Lkhoyaali
Safae Boukhardal

Encadré par : Pr BAIDA Ouafae

Année universitaire : 2023-2024

Table de matière

I. Introduction :

L'objectif de la théorie des graphes est de résoudre des problèmes en simplifiant leur représentation à une série de sommets et d'arêtes reliant ces sommets. La résolution du problème peut alors être traitée de façon logique en opérant des algorithmes particuliers cherchant à optimiser le graphe, ou à déterminer le plus court chemin, ou juste à caractériser la structure du graphe…

Ainsi que théorie des graphes et les algorithmes qui lui sont liées est un des outils privilégiés de modélisation et de résolution des problèmes dans un grand nombre de domaines allant de la science fondamentale aux applications technologiques concrètes. Ça devenue un domaine de recherche actif, avec des applications dans de nombreux domaines, notamment les mathématiques, les sciences informatiques, l'ingénierie, l'économie et les sciences sociales.

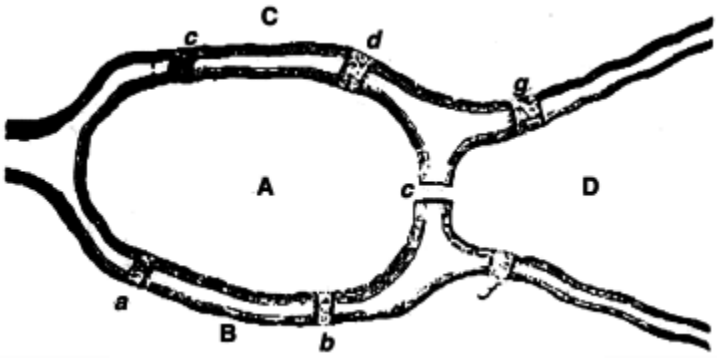
Avant-propos

Dans ce rapport, nous allons faire une implémentation d'une partie des algorithmes de théorie des graphes et ceci en utilisant le langage C comme langage de programmation.

Noter que les implémentations sont faites en langage C sous forme d'un projet Visual Studio Code.

II. Historique :

La théorie des graphes en tant que discipline mathématique distincte a commencé à se développer au XVIIIe siècle. Le mathématicien suisse Leonhard Euler a résolu le problème des ponts de Königsberg, qui est un problème de théorie des graphes. Ce problème a conduit à la découverte du théorème d'Euler, qui est un résultat fondamental de la théorie des graphes.



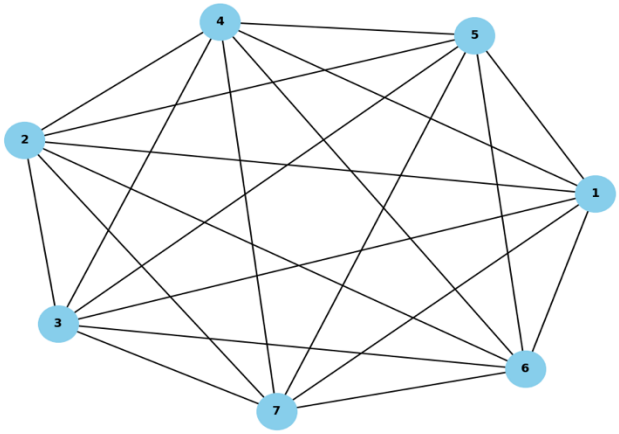
Au XIXe siècle, les mathématiciens allemands Carl Friedrich Gauss et Peter Gustav Lejeune Dirichlet ont étudié les propriétés des graphes planaires. Ils ont découvert plusieurs résultats importants, notamment le théorème de Gauss-Bonnet, qui est un résultat en géométrie différentielle.

Au XXe siècle, la théorie des graphes a connu un développement rapide. Les mathématiciens ont découvert de nombreux résultats importants, notamment le théorème de Ramsey, le théorème de König, et théorème de Havel-hakimi qui offre une méthode efficace pour vérifier et construire des graphes réalisables à partir degré de séquence

III. *Présentation Graphique et sur machine des structures des données*

1. Présentation Graphique :

La présentation graphique d'un graphe est une représentation visuelle des sommets et des arêtes du graphe. Elle permet de visualiser la structure du graphe et de comprendre les relations entre les sommets.



2. Présentation sur machine des structures des données

La modélisation de graphes sur machine s'avère indispensable pour résoudre une multitude de problèmes. Pour ce faire, plusieurs structures de données existent :

- ✓ Matrice d'adjacence : Simple à implémenter, efficace pour tester l'existence d'une arête et pour parcourir le graphe en profondeur ou en largeur.
- ✓ Matrice d'incidence : Permet de détecter facilement les boucles et les arêtes multiples.
- ✓ Listes d'adjacence : Adaptées aux graphes clairsemés, utilisation efficace de la mémoire, flexibles pour les modifications du graphe.
- ✓ Les tableaux LS et PS : Efficaces pour représenter des graphes orientés, permettent de stocker des informations supplémentaires sur les arêtes.

Chaque représentation présente des avantages et des inconvénients, et leurs utilisations dépendent de problème à résoudre.

Nous avons choisi de représenter notre graphe à l'aide d'une matrice d'adjacence. Pour ce faire, nous avons créé un code qui permet de générer la matrice d'adjacence de n'importe quel graphe.

- Formule mathématique

Pour un graphe non oriente $G = (X, Y)$

Mij =1 si $(xa, xb) \in Y$ $sinon$ Mij =0

- **Code:**

matrice.c > ...

```c
#include <stdio.h>

#define MAX_SOMMET 7

void initmatrice(int adjacencyMatrix[MAX_SOMMET][MAX_SOMMET]) {
    for (int i = 0; i < MAX_SOMMET; i++) {
        for (int j = 0; j < MAX_SOMMET; j++) {
            adjacencyMatrix[i][j] = 0;
        }
    }
}

void remplir(int adjacencyMatrix[MAX_SOMMET][MAX_SOMMET], int point1, int point2) {
    adjacencyMatrix[point1][point2] = 1;
    adjacencyMatrix[point2][point1] = 1;
}

void affiche(int adjacencyMatrix[MAX_SOMMET][MAX_SOMMET]) {
    for (int i = 0; i < MAX_SOMMET; i++) {
        for (int j = 0; j < MAX_SOMMET; j++) {
            printf("%d ", adjacencyMatrix[i][j]);
        }
        printf("\n");
    }
}
```

```c
int main() {
    int matrice[MAX_SOMMET][MAX_SOMMET];
    initmatrice(matrice);

    int numEdges;

    // Saisie du nombre d'arêtes
    printf("Entrez le nombre d'aretes : ");
    scanf("%d", &numEdges);

    // Saisie des relations entre les arêtes
    printf("Entrez les relations entre les aretes (format : sommet1 sommet2) :\n");
    for (int i = 0; i < numEdges; i++) {
        int point1, point2;
        printf("Relation %d : ", i + 1);
        scanf("%d %d", &point1, &point2);
        remplir(matrice, point1 - 1, point2 - 1);
    }

    // Affichage de la matrice d'adjacence
    printf("\nVoici la Matrice d'adjacence de votre graphe :\n");
    affiche(matrice);

    return 0;
}
```

Les relations de graphe :

Output :

```
Entrez le nombre d'aretes : 21
Entrez les relations entre les aretes (format : sommet1 sommet2) :
Relation 1 : 1 2
Relation 2 : 1 3
Relation 3 : 1 4
Relation 4 : 1 5
Relation 5 : 1 6
Relation 6 : 1 7
Relation 7 : 2 3
Relation 8 : 2 4
Relation 9 : 2 5
Relation 10 : 2 6
Relation 11 : 2 7
Relation 12 : 3 4
Relation 13 : 3 5
Relation 14 : 3 6
Relation 15 : 3 7
Relation 16 : 4 5
Relation 17 : 4 6
Relation 18 : 4 7
Relation 19 : 5 6
Relation 20 : 5 7
Relation 21 : 6 7

Voici la Matrice d'adjacence de votre graphe :
0 1 1 1 1 1 1
1 0 1 1 1 1 1
1 1 0 1 1 1 1
1 1 1 0 1 1 1
1 1 1 1 0 1 1
1 1 1 1 1 0 1
1 1 1 1 1 1 0
PS C:\Users\HP ELITEBOOK\OneDrive\Bureau\structure\projet>
```

IV. *Présentation des algorithmes associée au problème :*

Il existe plusieurs algorithmes associés à la détection de cycles et au parcours alternatif dans les graphes :

- Algorithmes de détection de cycles :

Algorithme de Tarjan: Cet algorithme utilise une pile pour suivre les nœuds visités. Si, lors du parcours, on tombe sur un nœud déjà visité qui n'est pas sur la pile, alors on a détecté un cycle.

Algorithme de Kosaraju : Cet algorithme divise le graphe en deux ensembles, puis utilise un parcours en profondeur pour détecter les cycles dans chacun des ensembles.

- Algorithmes de parcours alternatifs:

Algorithme de Plus Court Chemin (Dijkstra) : Trouve le chemin le plus court entre deux nœuds en attribuant des poids aux arêtes. Peut être utilisé pour trouver un parcours alternatif en considérant des chemins autres que le plus court.

Algorithme de Bellman-Ford : Un autre algorithme pour trouver le plus court chemin, capable de gérer des poids d'arêtes négatifs. Utile pour trouver des parcours alternatifs en tenant compte des poids d'arêtes négatifs.

1. Algorithme de Havel-Hakimi :

En théorie des graphes, l'algorithme de Havel-Hakimi est un algorithme résolvant le problème de la réalisation d'un graphe, c'est-à-dire, étant donné une liste d'entiers positifs ou nuls, déterminer s'il existe un graphe simple dont les degrés sont exactement cette liste. Si oui, la liste est dite graphique. Cette construction est basée sur un algorithme récursif

a. Théorème (Havel-Hakimi) :

Soient $n > 1$ et $d1 \geq \cdots \geq dn$ des entiers naturels. Alors, on a l'équivalence suivante. $(d1, \ldots, dn)$ est graphique $\Leftrightarrow$ $(d2 - 1, \ldots, d(d1+1) - 1, d(d1+2), \ldots, dn)$ est graphique

b. Exemple :

D'après le théorème d'Havel et Hakimi, on a les équivalences suivantes.

(4, 3, 3, 3, 3) graphique $\Leftrightarrow$ (2, 2, 2, 2) graphique

$\Leftrightarrow$ (1, 1, 2) graphique

$\Leftrightarrow$ (2, 1, 1) graphique

$\Leftrightarrow$ (0, 0) graphique

$\Leftrightarrow$ (0) graphique.

c. Implémentation en Langages C :

```c
int graphExists(int a[], int n) {
    while (1) {
        // Tri décroissant de la séquence de degrés (algorithme de tri à bulles).
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (a[j] < a[j + 1]) {
                    int temp = a[j];
                    a[j] = a[j + 1];
                    a[j + 1] = temp;
                }
            }
        }
        // Condition de sortie : si le premier et le dernier élément sont zéro, la séquence est valide.
        if (a[0] == 0 && a[n - 1] == 0) {
            return 1; // La séquence est valide (True).
        }
        // Récupère le degré du premier élément de la séquence.
        int v = a[0];
        for (int i = 0; i < n - 1; i++) {
            a[i] = a[i + 1];
        }
        n--;
        // Si le degré est supérieur au nombre d'éléments restants, la séquence est invalide.
        if (v > n) {
            return 0; // La séquence est invalide (False).
        }

        // Réduit les degrés des premiers v éléments.
        for (int i = 0; i < v; i++) {
            a[i] -= 1;

            // Si un degré devient négatif, la séquence est invalide.
            if (a[i] < 0) {
                return 0; // La séquence est invalide (False).
            }
        }
    }
}
```

2. Algorithme de Euler :

On appelle chaîne eulérienne d'un graphe G une chaîne passant une et une seule fois par chacune des arêtes de G. Un graphe ne possédant que des chaînes eulériennes est semi-eulérien. On appelle cycle eulérien d'un graphe G un cycle passant une et une seule fois par chacune des arêtes de G. Un graphe est dit eulérien s'il possède un cycle eulérien. Plus simplement, on peut dire qu'un graphe est eulérien (ou semi-eulérien) s'il est possible de dessiner le graphe sans lever le crayon et sans passer deux fois sur la même arête.

a. *Théorèmes d'Euler :*

Un graphe connexe admet une chaîne eulérienne si et seulement si le nombre de sommets de degré impair est 0 ou 2.

Un graphe connexe admet un cycle eulérien si et seulement si le nombre de sommets de degré impair est 0 (tous les sommets ont un degré pair).

- Cas d'un cycle eulérien :

Le graphe est connexe et tous les sommets ont un degré pair.

1ère Étape :

On choisit arbitrairement un sommet du graphe et on crée un cycle contenant des arêtes du graphe au plus une fois. Si toutes les arêtes du graphe sont contenues une et une seule fois dans le cycle alors on a terminé car on a un cycle eulérien. Sinon on passe à la deuxième étape.

2ème Etape :

On choisit un sommet du cycle précédent pour lequel il est possible de créer une chaîne fermée (cycle) d'origine et d'extrémité ce sommet et contenant des arêtes du graphe au plus une fois et n'étant pas contenues dans le cycle précédent. En regroupant les deux cycles, on obtient un nouveau cycle (de longueur la somme des deux longueurs des cycles précédents). Si toutes les arêtes sont contenues une et une seule fois dans le nouveau cycle alors le nouveau cycle est eulérien. Sinon on recommence la deuxième étape avec le nouveau cycle.

- Cas d'une chaîne eulérienne :

Le graphe est connexe et deux sommets (et deux seulement) ont un degré impair.

1ère Etape :

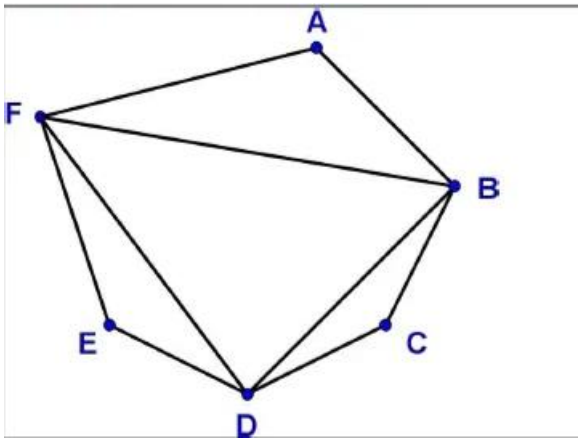
On choisit une chaîne, contenant des arêtes du graphe au plus une fois, reliant les deux sommets de degré impair. Si toutes les arêtes du graphe sont contenues dans la chaîne alors la chaîne est eulérienne. Sinon on passe à la deuxième étape.

2ème Etape :

On choisit un sommet de la chaîne précédente pour lequel il est possible de créer une chaîne fermée (cycle)d'origine et d'extrémité ce sommet choisi et contenant des arêtes du graphe au plus une fois et n'étant pas contenues dans la chaîne précédente. En regroupant les deux chaînes, on obtient une nouvelle chaîne (de longueur la somme des deux longueurs des chaînes précédentes). Si toutes les arêtes du graphe sont contenues une et une seule fois dans la nouvelle chaîne alors la chaîne est eulérienne. Sinon on recommence la deuxième étape avec la nouvelle chaîne.

b. Exemple :

On considère le graphe suivant :



Les sommets A, C et E ont pour degré 2.

Les sommets B, D et F ont pour degré 4.

Il existe donc un cycle eulérien.

On choisit le cycle initial : ABFA. En B on considère le cycle BCDB, on obtient ABCDBFA.

En F on considère le cycle FDEF on obtient le cycle eulérien **ABCDBFDEFA.**

c. *Implémentation en Langages C :*

```c
void hasEulerianPath(int vertices, int adj_matrix[vertices][vertices]) {
    int odd_degrees = 0;

    // Vérifier si le graphe est connexe
    for (int i = 0; i < vertices; i++) {
        int sum = 0;
        for (int j = 0; j < vertices; j++) {
            sum += adj_matrix[i][j];
        }
        if (sum == 0) {
            printf("Le graphe n'est pas connexe, il n'a ni cycle  eulerien ni chaine eulerienne\n");
            return;
        }
    }

    // Compter le nombre de sommets de degré impair
    for (int i = 0; i < vertices; i++) {
        int sum = 0;
        for (int j = 0; j < vertices; j++) {
            sum += adj_matrix[i][j];
        }
        if (sum % 2 != 0) {
            odd_degrees++;
        }
    }

    // Si le nombre de sommets de degré impair est 0 ou 2, le graphe a un cycle  eulérien
    if (odd_degrees == 0) {
        printf("Le graphe a un cycle eulerien\n");
    }
    // Si le nombre de sommets de degré impair est 2, le graphe a une chaîne eulérienne
    else if (odd_degrees == 2) {
        printf("Le graphe a une chaine eulerienne\n");
    }
    // Sinon, le graphe n'a ni cycle  eulérien ni chaîne eulérienne
    else {
        printf("Le graphe n'a ni cycle eulerien ni chaine eulerienne\n");
    }
}
```

V. Exemples d'application des Algorithmes

1. Algorithme de Havel-Hakimi:

- **En biologie :**

-l'algorithme de Havel-Hakimi peut être utilisé pour étudier les réseaux alimentaires. En effet, les espèces d'un réseau alimentaire peuvent être représentées par des sommets d'un graphe, et les relations de prédation entre les espèces peuvent être représentées par des arêtes. La somme des degrés des sommets d'un graphe correspond au nombre de relations de prédation dans le réseau alimentaire. L'algorithme de Havel-Hakimi permet donc de déterminer si un réseau alimentaire est possible,

- **En informatique :**

- l'algorithme de Havel-Hakimi peut être appliqué pour étudier les propriétés des réseaux informatiques. Les nœuds d'un réseau peuvent être représentés par des sommets d'un graphe, et les liens entre les nœuds peuvent être représentés par des arêtes. La somme des degrés des sommets d'un graphe correspond au nombre de liens dans le réseau. L'algorithme de Havel-Hakimi peut être utilisé pour déterminer si un tel réseau est possible, c'est-à-dire s'il peut être représenté par un graphe simple.

- **En économie:**

L'algorithme de Havel-Hakimi peut être appliqué pour étudier les réseaux d'interactions entre les agents économiques. Dans ce contexte, chaque agent économique peut être représenté par un sommet (ou nœud) dans un graphe, et les relations entre les agents sont représentées par des arêtes. La somme des degrés des sommets dans ce graphe correspond au nombre total de relations entre les agents.

2. Algorithme d'Euler :

- **En cartographie :**

-L'algorithme d'Euler peut être utilisé pour déterminer si un itinéraire touristique peut être effectué sans revenir sur ses pas. Par exemple, si on souhaite faire un circuit touristique autour d'un lac, on peut représenter le lac par un graphe planaire, et les routes qui entourent le lac par les arêtes du graphe. Si tous les sommets du graphe ont un nombre pair de degrés, alors il est possible de faire un circuit touristique autour du lac sans revenir sur ses pas.

- **En ingénierie :**

- L'algorithme d'Euler peut être utilisé pour déterminer si un réseau de tuyauterie peut être parcouru sans débrancher les tuyaux. Par exemple, si on souhaite installer un réseau de tuyauterie dans une maison, on peut représenter le réseau par un graphe planaire, et les tuyaux par les arêtes du graphe. Si tous les sommets du graphe ont un nombre pair de degrés, alors il est possible d'installer le réseau de tuyauterie sans débrancher les tuyaux.

- **En circuit électrique :**

- L'algorithme d'Euler vérifie si chaque connexion a un point d'origine et de destination, évitant ainsi de débrancher les fils. On représente le circuit sous forme de graphe, où les composants sont des nœuds et les connexions sont des arêtes. Si tous les composants ont un nombre pair de connexions, l'algorithme permet de trouver un chemin parcourant chaque fil une seule fois. On commence à n'importe quel composant et on suit les connexions jusqu'au retour au point de départ, assurant ainsi une conception optimale et un parcours efficace du circuit électrique.

✓ ***Exemples d'application d'Algorithme de Havel-Hakimi et Euler :***

➢ **les circuits électriques:**

-En utilisant les deux algorithmes ensemble, il est possible de résoudre des problèmes plus complexes. Par exemple, on peut utiliser l'algorithme de Havel-Hakimi pour déterminer si un circuit électrique est possible, puis utiliser l'algorithme d'Euler pour déterminer si le circuit peut être parcouru sans débrancher les fils.

➢ **réseaux de transport routier:**

- Utiliser l'algorithme de Havel-Hakimi pour vérifier la possibilité physique de construire des routes entre les intersections avec le bon nombre de connexions. Si le réseau routier est possible utiliser l'algorithme d'Euler pour déterminer s'il est possible de parcourir toutes les routes du réseau sans emprunter une route plusieurs fois. Cela peut être crucial pour la planification du trafic et l'efficacité du réseau routier.

➢ **Reseaux sociaux en ligne:**

-Utiliser l'algorithme de Havel-Hakimi pour vérifier la possibilité d'établir des relations d'amitié entre les utilisateurs avec le bon nombre de connexions. Si le réseau social est possible utiliser l'algorithme d'Euler pour déterminer s'il existe un chemin permettant de parcourir toutes les relations d'amitié sans répétition. Cela peut être utile pour identifier des utilisateurs qui peuvent être atteints via des relations d'amitié

VI. Présentation de base de données

Notre base de données de livraison mise en place a destinée et à gérer le processus de livraison de commandes pour plusieurs sociétés. Chaque société est associée à cinq rues possibles comme itinéraires, et chaque commande est livrée en utilisant l'un de ces itinéraires. La structure de la base de données explique comment elle prend en charge la planification des livraisons pour chaque société.

- **ID-commande :** (Clé primaire) identifiant unique de la commande
- **Commande :** Nombre de colis inclus dans la commande.
- **Société :** nom de la société associée à la commande [MO, GC, VA, TX, CA, AL].
- **Rue (1,2,3,4,5) :** représentant les cinq rues possibles comme itinéraires.
- **GC :** la société résponsable de livraison car il a 0 commande

- **AL,MO,TX,VA,GA,NS:** les sociétés qui ont reçu les commandes et chaque société a un nombre différents des commandes

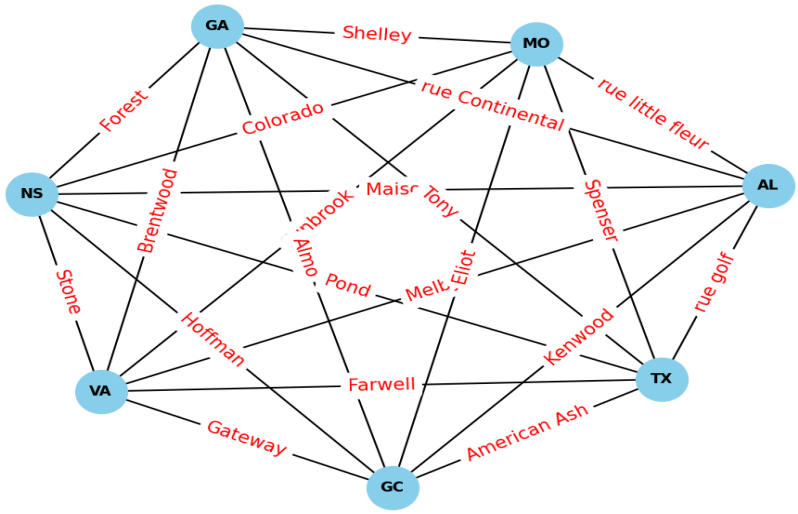
Id-commande	*Commande*	*Société*	*Rue1*	*Rue 2*	*Rue 3*	*Rue 4*	*Rue 5*	*Rue 6*
1	0	GC	Kenwood	Eliot	AmericanAsh	Gateway	Almo	Hoffman
2	6	AL	Kenwood	Little fleur	Golf	Melby	Continental	Maison
3	2	MO	Eliot	Little fleur	Spenser	Sunbrook	Shelley	Colorado
4	8	TX	AmericanAsh	Golf	Spenser	Farwell	Tony	Pond
5	5	VA	Gateway	Melby	Sunbrook	Farwell	Brentwood	Stone
6	1	GA	Almo	Continental	Shelley	Tony	Brentwood	Forest
7	2	NS	Hoffman	Maison	Colorado	Pond	Stone	Forest

Objectif de la base de données :

- La base de données est conçue pour aider un livreur à planifier et à suivre les livraisons de colis à différentes sociétés.
- Chaque commande est associée à une société, à un ensemble de rues potentielles et à un nombre spécifique de colis
- La base de données est utilisée pour optimiser les itinéraires de livraison et améliorer l'efficacité du processus de livraison pour le livreur

Pour la source de la base de données, générée avec le site web https://mockaroo.com
Nous lui avons fourni le sujet des données et les caractéristiques dont nous avions besoin, puis il nous a fourni l'ensemble de données sous forme d'un fichier Excel.

1. Présentation graphique de base de données :



Nous avons créé un code en Python pour présenter graphiquement notre base de données. Voici le code :

graphe.py

```python
import networkx as nx
import matplotlib.pyplot as plt

def creer_graphe(sommets, relations):
    G = nx.Graph()

    # Ajouter les sommets au graphe
    G.add_nodes_from(sommets)

    # Ajouter les relations entre les sommets avec un attribut 'nom'
    for relation in relations:
        G.add_edge(relation[0], relation[1], nom=relation[2])  # ajout de l'attribut 'nom' à l'arête

    return G

def dessiner_graphe(G):
    pos = nx.spring_layout(G)  # disposition des nœuds avec l'algorithme de Fruchterman-Reingold

    # Extraire les noms des relations pour les afficher sur le graphe
    edge_labels = {(u, v): d['nom'] for u, v, d in G.edges(data=True)}

    nx.draw(G, pos, with_labels=True, font_weight='bold', node_size=700, node_color='skyblue', font_size=8, font_color='black')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='red')  # Ajout des noms des relations
    plt.show()

# Exemple d'utilisation avec des noms de sommets personnalisés
sommets = ['AL', 'MO', 'TX', 'GA', 'GC', 'VA','NS']
relations = [('AL', 'MO', 'rue little fleur'), ('AL', 'TX', 'rue golf'), ('AL', 'GA', 'rue Continental'),
             ('AL', 'GC', 'Kenwood'), ('AL', 'VA', 'Melby'),('AL', 'NS', 'Maison'), ('MO', 'TX', 'Spenser'),
             ('MO', 'GA', 'Shelley'), ('MO', 'GC', 'Eliot'), ('MO', 'VA', 'Sunbrook'),('MO', 'NS', 'Colorado'),
             ('TX', 'GA', 'Tony'), ('TX', 'GC', 'American Ash'), ('TX', 'VA', 'Farwell'),('TX', 'NS', 'Pond'),
             ('GA', 'GC', 'Almo'), ('GA', 'VA', 'Brentwood'),('GA', 'NS', 'Forest'), ('GC', 'VA', 'Gateway'),('GC', 'NS', 'Hoffman'),('VA', 'NS', 'Stone')]

graphe = creer_graphe(sommets, relations)
dessiner_graphe(graphe)
```

VII. Teste des algorithmes sur la base de données

1. Algorithme Havel-Hakimi

Nous avons choisi de représenter notre graphe de base de données à l'aide d'une matrice d'adjacence.

La matrice d'adjacence du graphe se présente comme dessous :

$$\begin{matrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{matrix}$$

Alors degré de séquence 6,6,6,6,6,6,6.

Triez-la par ordre décroissant : 6, 6, 6, 6, 6, 6,6. (déjà triée)

Soustrayez 1 du premier élément et retirez les prochains 4 éléments : Nouvelle séquence : 5, 5, 5, 5,5,5.

Répétez le processus :

Soustrayez 1 du premier élément et retirez les prochains 5 éléments :

Nouvelle séquence : 4,4,4,4,4.

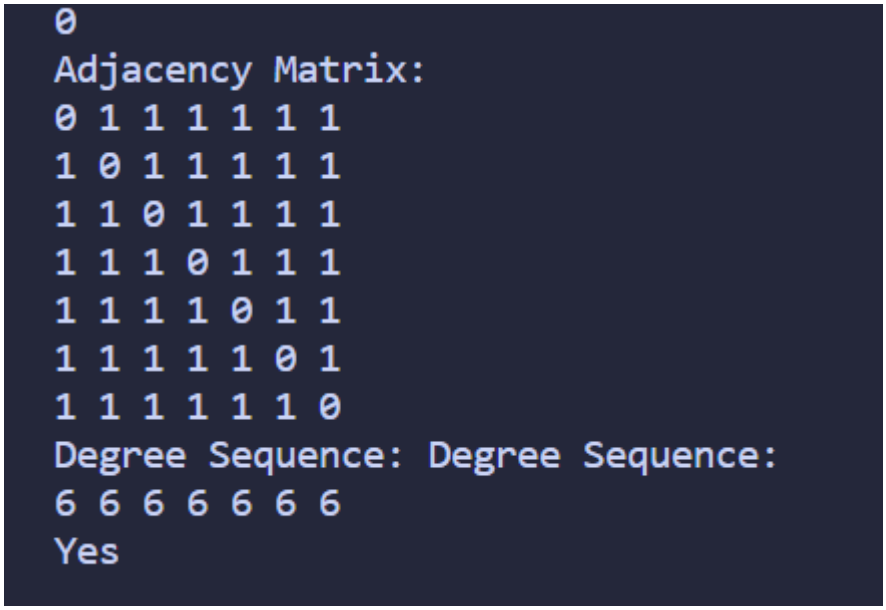
En Répétant le processus…

Soustrayez 1 de l'unique élément restant :

Nouvelle séquence : 1.

Soustrayez 1 de l'unique élément restant (s'il y en avait un), mais dans ce cas, la séquence est vide. Alors il existe un graphe simple.

Output :



2. Algorithme de Euler :

Nous avons choisi de représenter notre graphe de base de données à l'aide d'une matrice d'adjacence.

Nous avons créé un code en C pour vérifier si le graphe notre base de données est un circuit eulérien ou chaine eulérienne.

Code :

algo.c > hasEulerianPath(int, int [vertices][vertices])

```c
#include <stdio.h>

// Fonction pour initialiser le graphe avec le nombre de sommets donné
void initializeGraph(int vertices, int adj_matrix[vertices][vertices]) {
    printf("Veuillez entrer les valeurs de la matrice adjacente du graphe :\n");
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            printf("Veuillez entrer la valeur de adj_matrix[%d][%d] : ", i, j);
            scanf("%d", &adj_matrix[i][j]);
        }
    }
}
```

```c
void hasEulerianPath(int vertices, int adj_matrix[vertices][vertices]) {
    int odd_degrees = 0;

    // Vérifier si le graphe est connexe
    for (int i = 0; i < vertices; i++) {
        int sum = 0;
        for (int j = 0; j < vertices; j++) {
            sum += adj_matrix[i][j];
        }
        if (sum == 0) {
            printf("Le graphe n'est pas connexe, il n'a ni cycle  eulerien ni chaine eulerienne\n");
            return;
        }
    }

    // Compter le nombre de sommets de degré impair
    for (int i = 0; i < vertices; i++) {
        int sum = 0;
        for (int j = 0; j < vertices; j++) {
            sum += adj_matrix[i][j];
        }
        if (sum % 2 != 0) {
            odd_degrees++;
        }
    }

    // Si le nombre de sommets de degré impair est 0 ou 2, le graphe a un cycle  eulérien
    if (odd_degrees == 0) {
        printf("Le graphe a un cycle eulerien\n");
    }
    // Si le nombre de sommets de degré impair est 2, le graphe a une chaîne eulérienne
    else if (odd_degrees == 2) {
        printf("Le graphe a une chaine eulerienne\n");
    }
    // Sinon, le graphe n'a ni cycle  eulérien ni chaîne eulérienne
    else {
        printf("Le graphe n'a ni cycle eulerien ni chaine eulerienne\n");
    }
}
```

0 ⚠ 0 0

```c
int main() {
    int vertices;
    printf("Veuillez entrer le nombre de sommets du graphe : ");
    scanf("%d", &vertices);

    int adj_matrix[vertices][vertices];
    initializeGraph(vertices, adj_matrix);
    hasEulerianPath(vertices, adj_matrix);

    return 0;
}
```

Output :

```
Veuillez entrer la valeur de adj_matrix[6][0] : 1
Veuillez entrer la valeur de adj_matrix[6][1] : 1
Veuillez entrer la valeur de adj_matrix[6][2] : 1
Veuillez entrer la valeur de adj_matrix[6][3] : 1
Veuillez entrer la valeur de adj_matrix[6][4] : 1
Veuillez entrer la valeur de adj_matrix[6][5] : 1
Veuillez entrer la valeur de adj_matrix[6][6] : 0
Le graphe a un cycle eulerien
PS C:\Users\HP\Desktop\pr_algo_VF>
```

VIII. Interprétation des résultats

1. Havel-Hakimi

D'après les résultats

- Cela signifie qu'il est possible de construire un graphe simple où chaque sommet a un degré correspondant à cette séquence. En d'autres termes, on peut créer un graphe avec 7 sommets, où chaque sommet est connecté à exactement 6 autres sommets.
  Autrement dit, l'ajustement des rues pour les livreurs en fonction des degrés associés aux sociétés (Rues) est possible.

2. Euler

D’après l’algorithme d’Euler on constate quand on le graphe est un cycle eulérien. C’est-à-dire un cycle qui passe par chaque arête exactement une fois. Pour qu'un graphe ait un cycle eulérien, il doit être connexe (tous les sommets sont reliés) et chaque sommet doit avoir un nombre pair d'arêtes adjacentes. Cela se produit lorsque tous les sommets ont un degré pair, ce qui garantit l'existence d'un chemin alternatif dans le graphe.

IX. Conclusion

Le théorème de Havel-Hakimi est un résultat fondamental en théorie des graphes qui fournit une condition nécessaire et suffisante pour qu'une suite d'entiers soit la suite de degrés d'un graphe simple.

Un graphe est considéré comme eulérien s'il peut être dessiné sans lever le crayon et sans traverser deux fois la même arête. Le théorème d'Euler indiqué que pour qu'un graphe forme un cycle eulérien, il doit être connexe et tous les sommets doivent avoir un degré pair. Si le graphe est connexe et a exactement deux sommets de degré impair, il contient une chaîne eulérienne, un chemin qui traverse chaque arête une fois, mais qui peut ne pas revenir au point de départ.

X. Références

https://www.geeksforgeeks.org/find-if-a-degree-sequence-can-form-a-simple-graph-havel-hakimi-algorithm/

https://www.lecluse.fr/nsi/NSI_T/algo/graphes/

https://webusers.imj-prg.fr/~benjamin.girard/Havel_et_Hakimi.pdf

https://fr.scribd.com/document/691586821/3-chaines-euleriennes-cycles-euleriens