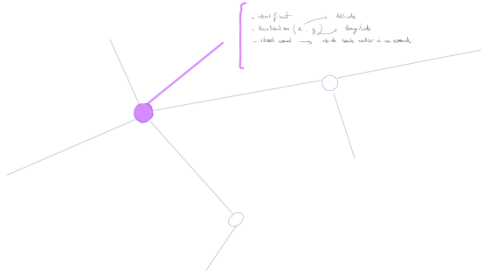


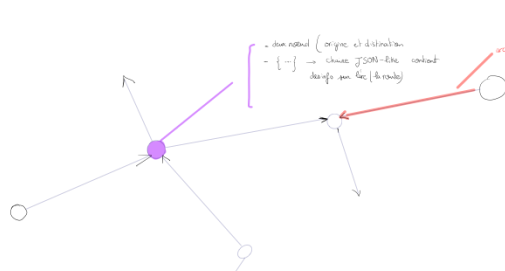
# Projet: Carte Routière

## 1. Présentation du Projet :

Le fichier : `nodes.csv`  
Les info sur les nœuds



Le fichier : `arcs.csv`  
Les info sur les arcs



Objectif :

- Stocker les données des fichiers `nodes.csv` et `arcs.csv` dans des structures de données adaptées.
- Développer des algorithmes pour exploiter ces données, comme parcourir le graphe, trouver des chemins, etc.

Attribut	Explication	Exemple
<b>osmid</b>	Identifiant unique de l'arc (route).	7985436
<b>oneway</b>	Indique si la route est à sens unique. True = sens unique, False = double sens.	True → Route à sens unique. False → Double sens.
<b>reversed</b>	Indique le sens de la route quand oneway = True.	False → Sens origine → destination. True → Sens destination → origine.
<b>lanes</b>	Nombre de voies de la route.	2 → La route possède 2 voies.
<b>name</b>	Le nom de la route.	"Avenue du Moulin de Notre-Dame"
<b>highway</b>	Type de route (autoroute, résidentiel, départementale, etc.).	residential → Route résidentielle.
<b>length</b>	Longueur de la route en mètres.	45.056 mètres.
<b>geometry</b>	Coordonnées qui définissent la trajectoire de la route.	<LINESTRING (4.812 43.932, 4.812 43.933)>
<b>maxspeed</b>	Vitesse maximale autorisée sur la route en km/h.	50 km/h
<b>speed_kph</b>	Vitesse moyenne estimée si maxspeed est manquant.	32.7 km/h
<b>travel_time</b>	Temps estimé pour parcourir la route en secondes, basé sur la vitesse.	5.0 → 5 secondes pour parcourir la route.

L'objectif de ce projet :

- **Stocker les données** des fichiers `nodes.csv` et `arcs.csv` dans des **structures de données adaptées**.
- **Développer des algorithmes** pour exploiter ces données, comme parcourir le graphe, trouver des chemins, etc.

Les structures sont **partiellement définies** (classes de base), et il faut **les compléter** pour répondre aux différentes questions posées dans le projet.

## 2. Présentation des taches :

### Structures de données

#### Tache 1: Structure et organisation des arcs:

L'objectif est de :

1. Créer une classe arc qui représente une route reliant deux nœuds dans le graphe.
2. Définir une structure de données pour E (ensemble des arcs) permettant de :
  - Ajouter des arcs.
  - Rechercher rapidement un arc à partir de son identifiant (id).
  - Afficher le contenu de l'ensemble des arcs.

#### Étapes pour réaliser la tâche

##### Comprendre la classe arc

La classe arc représente un arc (ou route) dans le graphe et contient les informations suivantes :

- id : identifiant de la route. Si la route n'a pas de nom, utiliser le couple (origine, destination) comme nom.
- o (origine) : identifiant du nœud de départ.
- d (destination) : identifiant du nœud d'arrivée.
- Autres attributs : comme length (longueur), oneway (sens unique), etc.
- Méthodes importantes :
  - Constructeur pour initialiser les arcs.
  - Méthode affiche pour afficher les informations d'un arc.

##### Choisir une structure de données pour E

L'ensemble E doit permettre :

1. Ajout rapide d'un arc.
2. Recherche rapide d'un arc à partir de son identifiant.
3. Affichage de tous les arcs.

Options possibles :

1. Table de hachage (avec STL unordered\_map) :
  - Avantages :
    - Permet un accès rapide aux arcs grâce à l'identifiant id.
    - Complexité de recherche :  $O(1)$  en moyenne.

- Inconvénients :
  - Pas d'ordre dans les données stockées.
2. Liste chaînée (avec STL list) :
- Avantages :
  - Simple à implémenter pour un petit nombre d'arcs.
  - Inconvénients :
  - La recherche d'un arc est lente : Complexité  $O(n)$  (il faut parcourir toute la liste).
3. Arbre binaire de recherche (BST) :
- Avantages :
  - Les arcs sont stockés en ordre croissant (selon l'identifiant).
  - Inconvénients :
  - Recherche et insertion : Complexité  $O(\log n)$  en moyenne.

### **Justification du choix**

Le choix de la structure dépend de l'objectif :

- Si on privilégie la rapidité de recherche → Table de hachage est le meilleur choix.
- Si on souhaite trier les arcs par id → Utiliser un arbre binaire de recherche.
- Si on travaille sur un petit ensemble → Une liste chaînée peut suffire.

### **Ajouter des arcs dans la structure E**

- Lire les arcs depuis le fichier arcs.csv.
- Pour chaque ligne :

Créer un objet arc avec les informations extraites (id, origine, destination, etc.).

Vérifier si l'id existe déjà dans la structure :

- Si oui → Ajouter un suffixe (name2, name3, etc.) pour éviter les doublons.
- Si non → Ajouter l'arc directement.

Stocker l'objet arc dans la structure choisie (ex : table de hachage).

### **Rechercher un arc dans E**

- Utiliser l'identifiant id pour accéder directement à l'arc (si table de hachage).
- Si une liste chaînée est utilisée, parcourir la liste jusqu'à trouver l'arc avec le bon id.

### **Afficher les arcs de E**

- Parcourir la structure (table de hachage, liste ou arbre).
- Appeler la méthode affiche() de chaque objet arc pour afficher ses informations.

## **Tache 2: Structure et organisation des nœuds**

L'objectif est de :

1. Créer une classe noeud qui représente un nœud du graphe.
2. Définir une structure de données pour les arcs entrants et sortants d'un nœud.
3. Définir un ensemble V de nœuds permettant de :
  - Ajouter des nœuds.
  - Rechercher rapidement un nœud à partir de son identifiant (id).
  - Afficher le contenu de l'ensemble des nœuds.

### **Étapes pour réaliser la tâche:**

#### **Comprendre la classe noeud**

La classe noeud représente un point dans le graphe et contient les informations suivantes :

- id : identifiant unique du nœud.
- arcs sortants : liste des arcs où ce nœud est l'origine.
- arcs entrants : liste des arcs où ce nœud est la destination.

Méthodes importantes :

- Constructeur pour initialiser les nœuds.
- Méthode affiche pour afficher les informations du nœud.

#### **Choisir une structure de données pour les arcs entrants et sortants**

Chaque nœud possède deux listes :

1. Arcs sortants : les arcs dont le nœud est l'origine.
2. Arcs entrants : les arcs dont le nœud est la destination.

Options possibles :

1. Liste chaînée (STL list) :
  - Avantages : Simple à manipuler pour ajouter des arcs.
  - Inconvénients : La recherche d'un arc est lente ( $O(n)$ ).
2. Table de hachage (STL unordered\_map ou unordered\_set) :
  - Avantages : Accès rapide aux arcs via leur identifiant ( $O(1)$ ).
  - Inconvénients : Moins flexible si l'ordre des arcs est important.

Justification :

- Pour gérer un grand nombre d'arcs, la table de hachage est préférable pour un accès rapide.
- Pour un petit nombre d'arcs, une liste chaînée suffit.

#### **Choisir une structure pour l'ensemble V des nœuds**

L'ensemble V doit permettre :

1. Ajout rapide d'un nœud.
2. Recherche rapide d'un nœud à partir de son identifiant.
3. Affichage de tous les nœuds.

Options possibles :

1. Table de hachage (STL unordered\_map) :
  - Avantages :
    - Accès direct aux nœuds grâce à leur identifiant (id).
    - Complexité d'accès :  $O(1)$  en moyenne.
  - Inconvénients : Pas de tri des nœuds.
2. Arbre binaire de recherche (BST) :
  - Avantages : Maintient les nœuds triés par leur id.
  - Inconvénients : Recherche et ajout plus lents ( $O(\log n)$ ).
3. Liste chaînée :
  - Avantages : Simple à mettre en place pour un petit graphe.
  - Inconvénients : Recherche d'un nœud lente ( $O(n)$ ).

Justification :

- Si on a besoin d'une recherche rapide, la table de hachage est le meilleur choix.
- Pour un graphe de petite taille, une liste chaînée peut suffire.

### **Ajouter des nœuds dans la structure V**

1. Lire les nœuds depuis le fichier nodes.csv.
2. Pour chaque ligne :
  - Extraire les informations du nœud (id, coordonnées, etc.).
  - Créer un objet noeud avec ces informations.
  - Ajouter cet objet dans l'ensemble V.

### **Mettre à jour les arcs entrants et sortants**

1. Lire les arcs depuis le fichier arcs.csv.
2. Pour chaque arc :
  - Ajouter l'arc dans la liste des arcs sortants du nœud d'origine.
  - Ajouter l'arc dans la liste des arcs entrants du nœud de destination.

### **Rechercher un nœud dans V**

1. Utiliser l'identifiant id comme clé pour accéder au nœud (si table de hachage).

2. Si une liste chaînée est utilisée, parcourir la liste pour trouver le nœud correspondant.

### **Afficher les nœuds de V**

1. Parcourir la structure (table de hachage, arbre ou liste chaînée).
2. Appeler la méthode affiche() de chaque objet noeud pour afficher ses informations.

## **Tache 3 : Implémentation de la classe graphe**

L'objectif est de :

1. Créer une classe graphe qui représente l'ensemble du graphe avec :
  - V : l'ensemble des nœuds (lesnoeuds).
  - E : l'ensemble des arcs (lesarcs).
  - nombredenoeuds et nombredarcs : compteurs pour les nœuds et arcs.
2. Ajouter des méthodes permettant de :
  - Lire les arcs depuis arcs.csv et les ajouter dans lesarcs.
  - Lire les nœuds depuis nodes.csv et les ajouter dans lesnoeuds.
  - Mettre à jour les arcs sortants et arcs entrants pour chaque nœud (liste d'incidence).

### **Étapes pour réaliser la tâche**

#### **Comprendre la classe graphe**

La classe graphe représente un graphe avec :

- Attributs :
  - V lesnoeuds : ensemble des nœuds du graphe.
  - E lesarcs : ensemble des arcs du graphe.
  - nombredenoeuds : nombre total de nœuds.
  - nombredarcs : nombre total d'arcs.
- Méthodes à ajouter :
  - int lecture\_arcs(nomfichier) : Lit le fichier arcs.csv pour insérer les arcs dans lesarcs.
  - int lecture\_noeuds(nomfichier) : Lit le fichier nodes.csv pour insérer les nœuds dans lesnoeuds.
  - void liste\_incidence() : Met à jour les arcs sortants et entrants pour chaque nœud.

#### **Ajouter la méthode int lecture\_arcs(nomfichier)**

Objectif : Lire le fichier arcs.csv ligne par ligne et insérer chaque arc dans l'ensemble lesarcs.

Étapes à suivre :

1. Ouvrir le fichier arcs.csv.
2. Lire chaque ligne :
  - Extraire les informations de l'arc : id, origine (o), destination (d) et autres attributs (length, oneway, etc.).
  - Créer un objet arc avec ces informations.
3. Ajouter l'arc dans la structure lesarcs (par exemple, une table de hachage).
4. Incrémenter le compteur nombredarcs.
5. Retourner le nombre total d'arcs lus.

### **Ajouter la méthode int lecture\_noeuds(nomfichier)**

Objectif : Lire le fichier nodes.csv ligne par ligne et insérer chaque nœud dans l'ensemble lesnoeuds.

Étapes à suivre :

1. Ouvrir le fichier nodes.csv.
2. Lire chaque ligne :
  - Extraire les informations du nœud : id, latitude, longitude, etc.
  - Initialiser les arcs sortants et arcs entrants à NULL ou à une structure vide.
  - Créer un objet noeud avec ces informations.
3. Ajouter le nœud dans la structure lesnoeuds (ex : table de hachage).
4. Incrémenter le compteur nombredenoeuds.
5. Retourner le nombre total de nœuds lus.

### **Ajouter la méthode void liste\_incidence()**

Objectif : Mettre à jour les arcs entrants et arcs sortants pour chaque nœud dans lesnoeuds en utilisant les arcs de lesarcs.

Étapes à suivre :

1. Parcourir tous les arcs dans lesarcs.
2. Pour chaque arc :
  - Identifier le nœud d'origine (o) et le nœud de destination (d).
  - Ajouter l'arc dans la liste des arcs sortants du nœud d'origine.
  - Ajouter l'arc dans la liste des arcs entrants du nœud de destination.
3. Répéter cette opération pour tous les arcs afin de compléter les listes d'incidence.

### **Construire le graphe via le constructeur**

Objectif : Initialiser l'ensemble des nœuds et arcs en appelant les méthodes lecture\_arcs et lecture\_noeuds.

Étapes :

1. Dans le constructeur graphe(fichierarcs, fichiernoeuds) :
  - Appeler la méthode lecture\_noeuds() pour lire et insérer les nœuds dans lesnoeuds.
  - Appeler la méthode lecture\_arcs() pour lire et insérer les arcs dans lesarcs.
2. Appeler la méthode liste\_incidence() pour mettre à jour les arcs entrants et sortants pour chaque nœud.

### **Ajouter la méthode void affiche()**

Objectif : Afficher les informations du graphe, notamment :

- Le nombre total de nœuds et d'arcs.
- Les informations de chaque nœud (id, arcs entrants, arcs sortants).
- Les informations de chaque arc.

Étapes :

1. Afficher le nombre total de nœuds et d'arcs.
2. Parcourir lesnoeuds pour afficher chaque nœud.
3. Parcourir lesarcs pour afficher chaque arc.

## **Algorithmes de parcours**

### **Tâche 4 : Calcul du degré des nœuds**

L'objectif est de :

1. Définir une méthode void degre(int n) dans la classe graphe.
2. Calculer le degré (nombre total d'arcs entrants et sortants) de chaque nœud.
3. Afficher les n premiers nœuds ayant le plus fort degré.

- **Étapes pour réaliser la tâche**

- **Définir le degré d'un nœud**

- Le degré d'un nœud est calculé comme la somme :
  - Du nombre d'arcs entrants.
  - Du nombre d'arcs sortants.
- 2. Parcourir tous les nœuds dans lesnoeuds



- Pour chaque nœud, calculer son degré en :
  - Comptant les éléments dans sa liste d'arcs entrants.
  - Comptant les éléments dans sa liste d'arcs sortants.
- 3. Trier les nœuds par degré décroissant
  - Utiliser une structure adaptée (ex : tableau ou liste) pour trier les nœuds selon leur degré.
- 4. Afficher les n premiers nœuds
  - Parcourir les n nœuds ayant le plus fort degré et afficher leurs informations.

## **Tâche 5 : Vérifier l'existence d'un chemin entre deux nœuds**

L'objectif est de :

1. Ajouter une méthode `int chemin(string o, string d)` dans la classe `graphe`.
2. Utiliser un parcours en profondeur (DFS) pour vérifier s'il existe un chemin entre deux nœuds.

### **Étapes pour réaliser la tâche**

- **Ajout de l'attribut visite dans la classe noeud**
  - Cet attribut indique si un nœud a déjà été visité.
- **Initialiser le parcours**
  - Créer une pile pour stocker les nœuds à visiter.
  - Mettre tous les nœuds dans un état non visité.
  - Ajouter le nœud d'origine (o) dans la pile.
- **Effectuer un parcours en profondeur (DFS)**
  - Tant que la pile n'est pas vide :
    1. Dépiler un nœud s.
    2. Si s n'a pas encore été visité :
      - Le marquer comme visité.
      - Ajouter tous ses arcs sortants dans la pile.
- **Vérifier l'arrivée au nœud destination**
  - Si le nœud destination (d) est visité pendant le parcours, un chemin existe.
- **Retourner le résultat**
  - Si un chemin existe, retourner le nombre d'arcs parcourus.
  - Sinon, retourner 0.

## Tâche 6 : Trouver le plus court chemin entre deux nœuds

L'objectif est de :

1. Ajouter une méthode `int pluscourtchemin(string o, string d)`.
2. Utiliser un parcours en largeur (BFS) pour trouver le chemin ayant le plus petit nombre d'arcs.

### Étapes pour réaliser la tâche

- **Initialiser le parcours**
  - Créer une file pour stocker les nœuds à visiter, accompagnés de leur distance depuis le nœud d'origine.
  - Mettre tous les nœuds dans un état non visité.
  - Ajouter le nœud d'origine (o) dans la file avec une distance de 0.
- **Effectuer un parcours en largeur (BFS)**
  - Tant que la file n'est pas vide :
    1. Dépiler un nœud s avec sa distance actuelle.
    2. Si s est la destination (d), retourner la distance actuelle.
    3. Sinon, ajouter tous ses arcs sortants dans la file avec une distance incrémentée.
- **Retourner le résultat**
  - Si le nœud destination est atteint, retourner la distance minimale.
  - Sinon, retourner 0.

## Tâche 7 : Trouver un itinéraire entre deux rues

L'objectif est de :

1. Ajouter une méthode `int itineraire(string origine, string destination)`.
2. Permettre de trouver le chemin le plus court entre deux rues (au lieu des nœuds).

Étapes pour réaliser la tâche

1. Trouver les nœuds correspondants aux noms des rues
  - Parcourir les arcs pour associer chaque nom de rue (name) à un arc.

- Identifier les nœuds d'origine (o) et de destination (d) correspondant aux rues données.
- 2. Appliquer la méthode pluscourtchemin
  - Une fois les nœuds trouvés, utiliser la méthode pluscourtchemin(o, d) pour trouver le chemin le plus court.
- 3. Retourner et afficher le résultat
  - Retourner la distance en nombre d'arcs.
  - Afficher les noms des rues empruntées.