Rapport de Projet – Livrable 2 Application de Navigation dans le Métro de Paris

Auteurs : Le Guen Germain Lauret Raphaël Marseille Baptiste

<https://github.com/baptistemrsl/1er-Livrable-Projet-PSI-MRSEILLE-LAURET-LE-GUEN>

Pseudo GitHub : GermainLeGuen Raphiloulou974 baptistemrsl

Date : 04/04/2025

# 1. Introduction

Dans le cadre de ce projet, nous avons conçu une application console en C# permettant de modéliser le métro de Paris sous forme de graphe orienté pondéré, d’afficher les stations avec leurs coordonnées géographiques, et de calculer le plus court chemin entre deux stations en utilisant différents algorithmes classiques (Dijkstra, Bellman-Ford, Floyd-Warshall).  
  
Nous avons aussi construit une interface qui répond aux attentes des modules du projet.

# 2. Prérequis & Connexion

- Connexion BDD : PSI.sql -> LivinParis

- Identifiants BDD : Serveur : localhost ; Port 3306 ; ID : root ; mdp : root.

- Exécution : Il est nécessaire d’avoir .NET installé ainsi que la bibliothèque EPPlus pour lire les fichiers Excel. Le fichier 'MetroParis.xlsx' doit être placé à la racine du projet.

# 3. Structure du Projet

Le projet est composé des fichiers suivants :  
- Graphe.cs : implémente un graphe orienté pondéré.  
- Noeud.cs : représente un nœud du graphe.  
- Station.cs : contient les propriétés d’une station de métro.  
- Construction\_Metro.cs / MetroGraphBuilder.cs : construisent le graphe à partir du fichier Excel.  
- Chemin.cs : contient les algorithmes de plus court chemin.  
- Program.cs : exécute l'application en console.  
- Interface.cs : contient la logique de l’interface utilisateur console.

-ClientModule.cs : l’outil doit pouvoir permettre d’entrer, supprimer ou modifier un nouveau Client depuis la console ou depuis un fichier Il faut à tout moment pouvoir afficher l’ensemble des Clients selon plusieurs critères.

-CuisinierModule.cs : de même un cuisinier peut être inséré, modifié ou supprimé depuis la Console ou depuis un fichier.

-CommandeModule.cs : Une commande ne peut être réalisée que si le client existe dans la base ou sinon il faut le créer. Une commande doit mettre en jeu un parcours entre une adresse de départ et une adresse d’arrivée et établir le chemin le plus court pour la date déterminée.

-StatistiqueModule.cs : affiche des moyennes et des listes si demandé.

-Database.cs : connecte le projet C# au Sql.

-RequetesSimples.cs : permet de faire des requêtes sur les classes.

-Program.cs : main de l’interface.

# 4. Fonctionnalités

- Chargement dynamique du réseau de métro à partir d’un fichier Excel.  
- Affichage des stations (avec nom, ligne, latitude et longitude).  
- Vérification de la connexité du graphe.  
- Choix interactif de l’algorithme de chemin le plus court.  
- Comparaison des performances (temps d’exécution) des trois algorithmes.  
- Interface utilisateur console intuitive avec menus interactifs.

# 5. Création de l’Interface

L’interface a été développée en C# en mode console, avec une attention particulière portée à l’expérience utilisateur malgré la simplicité du terminal. Elle est structurée autour de menus dynamiques permettant à l’utilisateur de :  
  
- Sélectionner deux stations parmi la liste disponible (avec validation de saisie).  
- Choisir l’algorithme à utiliser pour le calcul du plus court chemin (Dijkstra, Bellman-Ford ou Floyd-Warshall).  
- Visualiser les informations relatives au chemin trouvé : distance, temps d’exécution, et liste des stations parcourues.  
- Relancer une recherche ou quitter l’application via un menu principal réaffiché à la fin de chaque exécution.  
  
Chaque étape est accompagnée de messages explicites, et des contrôles sont mis en place pour éviter les erreurs de saisie (noms de stations invalides, algorithmes inexistants, etc.).  
  
Ce système de navigation console servira de base pour une potentielle évolution vers une interface graphique, avec des composants visuels plus riches (carte interactive, clics sur stations, animations).

# 6. Analyse des Algorithmes de Plus Court Chemin

- Dijkstra  
Principe : Calcule les plus courts chemins à partir d’une source, en utilisant une file de priorité.  
Avantages : Rapide, optimal avec poids positifs.  
Inconvénients : Ne fonctionne pas avec des poids négatifs.  
Complexité : O((V + E) log V)  
  
- Bellman-Ford  
Principe : Applique la relaxation des arêtes V-1 fois.  
Avantages : Supporte les poids négatifs.  
Inconvénients : Plus lent que Dijkstra.  
Complexité : O(V × E)  
  
- Floyd-Warshall  
Principe : Algorithme tout-vers-tout basé sur la programmation dynamique.  
Avantages : Donne tous les plus courts chemins entre toutes les paires.  
Inconvénients : Très coûteux en mémoire et temps.  
Complexité : O(V³)

Résultats de tests :  
Pour des trajets standards entre deux stations, les trois algorithmes donnent les mêmes distances (vérification de cohérence). Floyd-Warshall est plus rapide à l’exécution une fois pré-calculé, mais prend un temps initial élevé. Bellman-Ford est plus lent que Dijkstra dans tous les cas.

# 7. Conclusion

Ce projet nous a permis d’appliquer les structures de données avancées et les algorithmes de graphe pour résoudre un problème concret de planification d’itinéraire. Les performances et la modularité du système facilitent de nombreuses évolutions, y compris vers une application web ou mobile.