

Rapport

- **préciser le login/password du compte administrateur BDD si nécessaire à l'exécution de votre application**

login : root

password : &Mot2passe

- **votre lien Github et votre pseudo**

lien : <https://github.com/Tibo88/Etape1>

pseudo : Elliott748 et Tibo88

- **vos analyses des algorithmes de plus court chemin :**

Les trois algorithmes de plus court chemin – Dijkstra, Bellman-Ford et Floyd-Warshall – ont des performances différentes en fonction du type de graphe et du contexte d'utilisation.

Dijkstra est le plus rapide lorsqu'il s'agit de trouver le chemin le plus court entre deux sommets, à condition que tous les poids soient positifs. Il utilise une file de priorité et sa complexité est $O((V+E)\log V)$, ce qui le rend efficace sur les grands graphes clairsemés. Cependant, il ne fonctionne pas si des poids négatifs sont présents.

Bellman-Ford, quant à lui, est plus lent avec une complexité de $O(VE)$, car il parcourt toutes les arêtes du graphe $V-1$ fois. Son avantage principal est qu'il peut gérer les poids négatifs et détecter les cycles absorbants (négatifs), ce que Dijkstra ne peut pas faire. Il est donc utile si l'on doit prendre en compte des coûts négatifs dans le réseau.

Floyd-Warshall est différent des deux autres car il ne se contente pas de trouver le plus court chemin entre deux sommets, mais il calcule tous les plus courts chemins entre toutes les paires de sommets. Sa complexité est $O(V^3)$, ce qui le rend impraticable pour les très grands graphes, bien qu'il soit adapté aux petits graphes denses.

Dans le cas de notre projet, le graphe n'a que des poids positifs, Dijkstra est le plus pertinent car il est rapide et efficace (néanmoins, nous avons des problèmes pour le mettre en place. Il fonctionne sur certaines lignes mais n'arrive pas à changer de ligne). Si des poids négatifs existent, alors Bellman-Ford sera nécessaire. Enfin, si l'objectif est de comparer tous les chemins possibles entre plusieurs points, Floyd-Warshall peut être utilisé, bien que son coût en calcul soit élevé.

- **prompts visualisation du graphe :**

```

graphe.AfficherListe();
Console.WriteLine();
Console.WriteLine("Exécution de Floyd-Warshall");
graphe.FloydWarshall();
graphe.AfficherCheminPlusCourt(80, 148);
Console.WriteLine();

Console.WriteLine("Exécution de Dijkstra");
List<int> cheminDijkstra = graphe.Dijkstra(80, 148);
graphe.AfficherChemin(cheminDijkstra);
Console.WriteLine();

Console.WriteLine("Exécution de Bellman-Ford");
List<int> cheminBellmanFord = graphe.BellmanFord(80, 148);
graphe.AfficherChemin(cheminBellmanFord);

// MenuPrincipal();
}
} je te renvoie mon code, j'ai besoin d'afficher le graphe donc
afficher le plan du métro parisien, peux tu me compléter mon code
pour faire ça ? Chaque noeud à des coordonnées (longitude
latitude) et il y a une méthode permettant de calculer la distance
entre les noeuds grâce à la formule de Haversine, si jamais t'en as
besoin. Peux tu aussi faire en sorte que l'image générée s'affiche
quand j'exécute le code. ESSAYE D'UTILISER SKIASHARP

```

Pour afficher un graphe sous forme de plan de métro parisien avec les stations et les connexions entre elles en utilisant SkiaSharp, vous pouvez générer une image représentant le plan du métro. Voici comment vous pouvez procéder :

- 1. Calcul des coordonnées pour dessiner les stations : Utilisez les coordonnées (latitude et longitude) pour calculer la position de chaque station dans l'image.**
- 2. Affichage des stations : Chaque station sera dessinée sous forme de point ou d'icône.**
- 3. Affichage des liens : Reliez les stations adjacentes par des lignes.**

Nous allons générer une image avec SkiaSharp et afficher cette image. L'idée sera de créer une représentation simplifiée d'un plan du métro en fonction des données du graphe.

Voici la méthode pour générer et afficher l'image du plan du métro :

- 1. Installer SkiaSharp :**

Si ce n'est pas déjà fait, vous devez installer SkiaSharp via NuGet :

mathematica

Copier

Install-Package SkiaSharp

2. Implémentation pour générer le graphe :

csharp

Copier

```
using SkiaSharp;
using System;
using System.Collections.Generic;
using System.Linq;

namespace Etape2
{
    public class MetroPlan
    {
        public static void GenererPlanDuMetro(Graph<int> graphe,
string nomFichier)
        {
            // Taille de l'image
            int largeurImage = 800;
            int hauteurImage = 600;

            // Créer une image vide
            using (var surface = SKSurface.Create(new
SKImageInfo(largeurImage, hauteurImage)))
            {
                var canvas = surface.Canvas;
                canvas.Clear(SKColors.White); // Fond blanc

                // Calculer l'échelle pour convertir les coordonnées
de la station en pixels
                double minLongitude = graphe.Noeuds.Values.Min(n =>
n.Longitude);
                double maxLongitude = graphe.Noeuds.Values.Max(n =>
n.Longitude);
                double minLatitude = graphe.Noeuds.Values.Min(n =>
n.Latitude);
                double maxLatitude = graphe.Noeuds.Values.Max(n =>
n.Latitude);
```

```

        double echelleLongitude = largeurImage /
(maxLongitude - minLongitude);
        double echelleLatitude = hauteurImage / (maxLatitude
- minLatitude);

        // Dessiner les liens entre les stations
        foreach (var noeud in graphe.Noeuds.Values)
        {
            int x1 = (int)((noeud.Longitude - minLongitude)
* echelleLongitude);
            int y1 = (int)((noeud.Latitude - minLatitude) *
echelleLatitude);

            // Dessiner les voisins
            foreach (var lien in noeud.Liens)
            {
                Noeud<int> voisin = lien.Destination;
                int x2 = (int)((voisin.Longitude -
minLongitude) * echelleLongitude);
                int y2 = (int)((voisin.Latitude -
minLatitude) * echelleLatitude);

                // Dessiner une ligne entre les stations
                var paint = new SKPaint
                {
                    Color = SKColors.Black,
                    StrokeWidth = 2,
                    IsAntialias = true
                };

                canvas.DrawLine(x1, y1, x2, y2, paint);
            }
        }

        // Dessiner les stations
        foreach (var noeud in graphe.Noeuds.Values)
        {
            int x = (int)((noeud.Longitude - minLongitude) *
echelleLongitude);

```



```

using System;

class Program
{
    static void Main()
    {
        Graph<int> graphe = new Graph<int>();
        graphe.ChargerDepuisFichier("Noeuds2.txt", "Arcs2.txt");
        graphe.AjouterLiensManquants();

        // Générer le plan du métro
        string nomFichier = "plan_metro_parisien.png";
        MetroPlan.GenererPlanDuMetro(graphe, nomFichier);

        // Ouvrir l'image générée (optionnel)
        System.Diagnostics.Process.Start(nomFichier);
    }
}

```

Explications :

1. Calcul des positions des stations :
 - Nous utilisons les coordonnées (latitude et longitude) de chaque station pour la placer sur un plan. Les coordonnées sont transformées en pixels sur l'image en fonction de l'échelle.
2. Dessiner les stations :
 - Chaque station est dessinée sous forme de cercle avec son nom à côté.
3. Dessiner les liens :
 - Les liens entre les stations (les connexions) sont dessinés sous forme de lignes.
4. Générer l'image :
 - L'image est générée à l'aide de SkiaSharp et est enregistrée dans un fichier PNG.
5. Affichage de l'image :

- **Après avoir généré l'image, nous pouvons l'ouvrir avec le programme par défaut pour afficher le plan du métro.**

Résumé :

- **Ce code génère une image représentant le plan du métro en fonction des stations et de leurs connexions.**
- **Les stations sont placées sur l'image selon leurs coordonnées géographiques, et les connexions entre elles sont dessinées sous forme de lignes.**
- **L'image générée est enregistrée sous le nom "plan_metro_parisien.png" et peut être ouverte automatiquement après l'exécution du programme.**

Vous pouvez ajuster les tailles, couleurs et positions pour améliorer la lisibilité et la précision de l'affichage selon vos besoins.