

Métro Boulot Dodo

Projet Mathématiques pour l'Informatique

Antoine BANHA
antoine.banha@efrei.net

Yanis BOUMEDINE
yanis.boumedine@efrei.net

Mehdi AJROUD
mehdi.ajroud@efrei.net

Tanguy DOMERGUE
tanguy.domergue@efrei.net



Introduction du projet

Dans le cadre de notre formation à l'EFREI, nous avons eu à réaliser en groupe de 4 un projet répondant à des problématiques de théorie des graphes. Le projet de ce module à en effet pour objectif de tirer profit de la théorie des graphes pour proposer une solution optimale à un problème commun : **la résolution du plus court chemin**, appliqué au métro parisien. Nous déterminerons également l'arbre couvrant de poids minimal. Ainsi, nous proposerons un outil sous forme d'application web permettant à un utilisateur de sélectionner deux stations de métro différentes, et de tracer le trajet en listant les lignes de métro et les correspondances à emprunter pour arriver à destination ainsi que l'estimation du temps du trajet.

Logos Node, Vue & Pico

Ce qui a été réalisé



Nous avons réalisé une application 2-tiers pour répondre aux attentes du projet. Ainsi, le back-end de notre application propose une **API** permettant d'interfacer les algorithmes mis en place, qui est donc utilisé par le front-end pour compléter l'**interface utilisateur**.

Notre application offre deux fonctionnalités principales pour l'utilisateur :

- L'établissement et le tracé du plus court chemin entre deux stations de métro à l'aide de l'algorithme de **Dijkstra**
- L'affichage de l'arbre couvrant de poids minimal par l'intermédiaire de l'algorithme de **Kruskal**

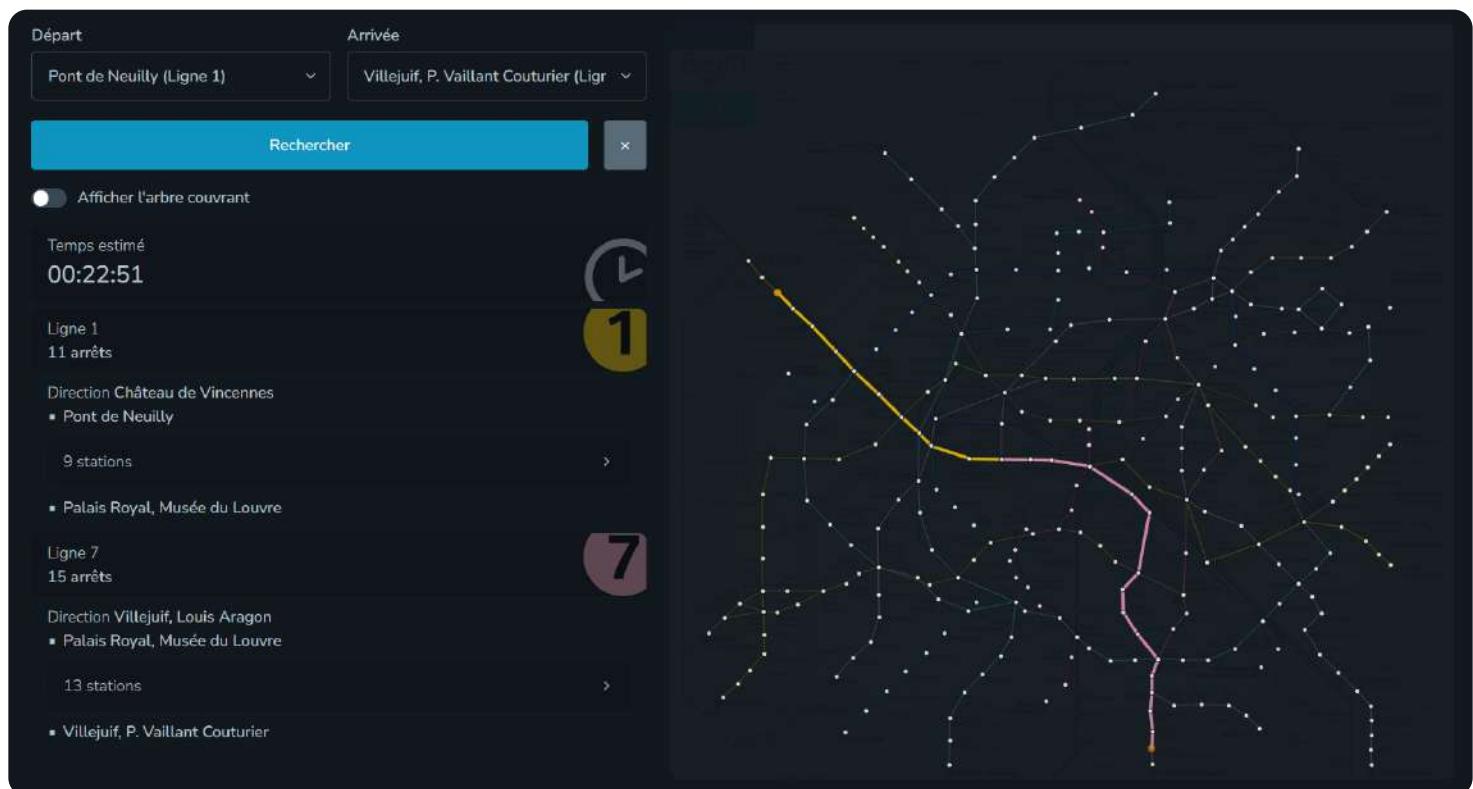
Le Front-end utilise des technologies web classiques: un mélange de HTML, CSS et de JavaScript. Pour faciliter la conception, nous avons opté pour un framework JavaScript léger appelé **Vue.js**. Celui-ci nous a permis de rendre notre interface dynamique sans avoir à toucher directement au DOM.



Introduction du projet

Nous avons également opté pour un framework CSS appelé **PicoCSS**. Il nous a permis de concevoir très rapidement et simplement une interface moderne et supportant un dark theme contrôlé par le système d'exploitation/navigateur.

Pour afficher et tracer les itinéraires directement sur la carte du client web, nous avons utilisé les **Scalable Vector Graphics (SVG)**. Nous avons longuement hésité avec Canvas mais par son manque d'interactivité (pour le clic sur les points de la carte et les animations de tracé), nous avons finalement décidé d'opter pour les SVGs, ce qui nous offre également un gain important en flexibilité. Cependant, le serveur utilise Canvas en mode serveur pour tracer Kruskal ainsi que le plan initial.



Interface utilisateur

Introduction du projet

Le back-end quant à lui a été conçu avec **Node.js** (TypeScript) ainsi que le framework Express. La librairie node-canvas a également été utilisée pour faire les affichages du plan.

Au lancement du serveur, on lit les fichiers de données avec la librairie standard **fs** de Node.js. On se charge ensuite d'extraire les données et on les formatent en objets typés pour les rendre facilement exploitables plus tard pour les différentes fonctionnalités de l'application.

On va ensuite exposer des points de terminaison REST (ou endpoints) pour que le front-end puisse communiquer avec le back-end, accessible via la route **“/api/<endpoint>”**. Certains points de terminaisons renvoient des données JSON (endpoints stations et path), alors que d'autres renvoient des images générées par la librairie Canvas (endpoints canvas et kruskal).

Pour répondre aux attentes du projet plus efficacement l'architecture de l'application est dite **monolithique**, c'est-à-dire que le front-end et le back-end sont réunis dans une seule application.

Dans une démarche de réalisme, le projet se compile en **image Docker** afin de permettre son déploiement dans n'importe quel environnement. De plus, dans l'hypothèse où l'on isolerait à 100% le back-end, on pourrait facilement s'adapter à un changement d'ordre de grandeur de la demande (on parle de scalabilité).

The screenshot shows a GitHub repository page for 'Projet de Maths Info - EFREI LS1'. The repository has 1 branch and 0 tags. The commit history shows 36 commits over the last month, with the most recent being 'JiveOff Kruskal terminé' by 'Update docker-image.yml' 7 days ago. Other commits include 'Ajout des icones et dijkstra' and 'Kruskal terminé'. The repository has 0 stars, 1 watching, and 0 forks. It also lists packages and no packages published.

Notre outil de collaboration: GitHub



Structures et algorithmes

Nous avons utilisé **4** structures de données.

Station

Arrêt donné tel quel par le dataset fourni pour le projet. Chaque ligne commençant par V est traité pour remplir une structure de ce type. Le numéro, le nom et la ligne sont des éléments d'unicité. Les coordonnées et les *AdjacentStation* sont également stockés pour l'affichage ainsi que pour Dijkstra.

Les doublons de stations dans le dataset correspondent aux correspondances avec les autres lignes.

Attribut	Type	Description
num	number	Identifiant unique incrémenté fourni par le dataset
nom	string	Nom de la station traitée
ligne	string	Ligne correspondante (String car 7b, 3b)
terminus	boolean	Si la station est un terminus de ligne
branchement	number	Numéro de branchement (ligne fourchées)
posX	number	Position X de la station dans le plan
posY	number	Position Y de la station dans le plan
adjacentStations	AdjacentStation[]	Tableau contenant les stations adjacentes. Ce champ permet des relations cycliques non-supportées par JSON.



Structures et algorithmes

AdjacentStation

Station adjacente à une autre station. Le temps est une propriété de cet objet d'union.

Attribut	Type	Description
station	Station	Station adjacente
time	number	Temps associé pour aller jusqu'à la station

Gare

Agrégation de stations sur le nom de station. Cette structure existe pour rendre le Kruskal plus visuel sur l'interface utilisateur.

Attribut	Type	Description
nom	string	Nom de la gare
lignes	string[]	Liste des lignes desservies
posX	number	Position X de la gare dans le plan
posY	number	Position Y de la gare dans le plan



Structures et algorithmes

Intergare

Cette structure rejoint deux gares avec le temps de parcours.

Il est bon de rappeler que cette structure n'est pas utilisée pour les correspondances dans la même gare.

Attribut	Type	Description
gare_1	Gare	Gare n°1
gare_2	Gare	Gare n°2
time	number	Temps entre les 2 gares
ligne	string	Ligne que le lien emprunte

Algorithmes

readAndParseData (Back-end)

Une procédure du programme qui a pour objectif de traiter les chaînes de caractère des fichiers en structure de données. Dans un premier temps, on lit le document “Metro.txt”. de celui-ci on extrait l'ensemble des stations afin de les parser et les mettre dans la structure de données. Puis on ajoute les gares en vérifiant bien leur unicité dans la structure de données. Dans un second temps, on pose des points dans le fichier pospoints.txt, pour placer les gares. Pour être compris, les espaces sont transformés en “@” dans le fichier .Txt. C'est cette fonction qui enregistre les gares, stations et intergares en mémoire.



Structures et algorithmes

Dijkstra (Back-end)

Cette fonction permet de déterminer le plus court chemin entre une station et une autre. L'algorithme doit être capable de retenir les stations qu'il a visité, celles qu'il n'a pas encore visité et les stations adjacentes. On part de la station qu'a choisie l'utilisateur comme point de départ. On récupère les stations adjacentes qui n'ont pas encore été visitée. L'algorithme compare le temps que prend un parcours entre une station et celle présente dans la liste des stations adjacente non visitée et prend le parcours le plus court. Pour terminer, la station est ajoutée dans la liste des stations visitées.

Kruskal (Back-end)

Cette fonction permet de construire un arbre couvrant de poids minimum en utilisant la structure de données Union Find. Cet algorithme commence par copier profondément les différentes gares et intergares pour éviter de toucher aux données en mémoire. Les intergares sont ensuite ordonnés par ordre croissant. On initialise ensuite notre tableau d'intergares final ainsi que la classe d'ACPM. Nous créons un set pour chaque gare pour ainsi refaire une itération sur toutes nos intergares. Dans cette itération, nous venons retrouver les 2 gares de l'intergare pour ensuite vérifier que ce ne sont pas les mêmes avant de les ajouter dans le tableau final et les union entre elles. Cet algorithme s'exécute en moyenne en 4ms.

createMapCanvas (Front-end)

Cette fonction permet d'afficher le plan et d'afficher les gares sur le plan. Les gares sont représentées sous forme de points cliquables. Les lignes entre les gares et les gares elle-même sont dessinées en se basant sur l'intergare passer en argument.



Lancement du projet

En ligne

Vous pouvez accéder au projet via cette URL:

<https://mathsinfo.efrei.jiveoff.fr>

Installation via Docker

Vous pouvez télécharger et lancer le projet en une seule commande peu importe l'environnement tant que celui supporte **Docker**:

```
> docker run -p 3000:3000 jiveoff/projet-mathsinfo
```

Une fois lancé, le projet sera disponible sur votre machine local sur:

<http://localhost:3000>

Installation via GitHub & NPM

Prérequis: **Node.js v16** (vérifiez avec `node -v` sur le terminal)

Il est nécessaire d'exécuter les commandes suivantes pour télécharger, installer et lancer le projet.

```
> git clone https://github.com/JiveOff/projet-mathsinfo.git  
> cd projet-mathsinfo && npm install  
> npm start
```

Le succès de cette méthode dépend de votre configuration et de votre système. Il est très recommandé d'utiliser Docker si vous ne vous en sortez pas.



Conclusion

Ce projet a été très enrichissant pour l'ensemble du groupe, car nous avons pu utiliser nos connaissances acquises dans le module de Mathématiques pour l'Informatique pour proposer une solution à un problème concret qui nous est familier. Nous avons également voulu donner une dimension réaliste au projet, en orientant la conception de l'application pour permettre un déploiement simple et robuste, que nous pensons avoir réussi.

D'un point de vue technique, l'un des membres de notre groupe était excellent avec les technologies du développement web, ce qui nous a permis d'avoir un tel résultat : une application performante et pratique. Malgré des difficultés pour certains membres du groupe à comprendre ce qui était réalisé, nous pensons tous avoir progressé et monté en compétence.

Dans une perspective d'amélioration de l'application, nous avons quelques idées. On pourrait d'abord imaginer utiliser une API en temps réel mise à disposition par la RATP pour pouvoir avoir des temps de trajets plus réalistes et donc une application plus fiable. Dans la même dimension, on pourrait imaginer utiliser une carte du métro plus complète et récente permettant d'incorporer les RER et les futures nouvelles lignes de métro.

