
Contents

1 Introduction

2 Implémentation et tests

- 2.1 Les bibliothèques
- 2.2 Méthodes `read_stations`, `display_stations`
- 2.3 Méthodes `read_connections`, `display_connections`
- 2.4 Méthodes `compute_travel`, `compute_and_display_travel`
- 2.5 Résultats de tests de l'algorithme de chemin le plus court

3 Conclusion



Rapport de Projet C++ Avancé: Application RATP

Nidhal Saïdani¹

¹) Sorbonne Université
ns@pm.me



Année universitaire : 2022-2023

1 Introduction

Dans un monde où les problèmes de trafic augmentent constamment, la collecte et la vente de données de trafic peuvent fournir des informations précieuses pour aider les entreprises et les particuliers à prendre des décisions éclairées sur leurs déplacements. Dans ce contexte, une start-up souhaite se lancer sur le marché de la vente de données de trafic en proposant une solution de planification de temps de parcours. Pour répondre à cette demande, l'équipe de développement a décidé d'implémenter l'algorithme de Dijkstra, reconnu pour sa robustesse et sa capacité à déterminer le plus court chemin dans un graphe pondéré.

L'implémentation de cet algorithme sera réalisée en utilisant une structure de classe en C++. Cette méthode permet d'organiser le code en encapsulant la logique de l'algorithme dans une classe, rendant ainsi le code plus facile à comprendre, à maintenir et à réutiliser. La hiérarchie de classes utilisée pour cette implémentation comprendra une classe de base "Read_data", ainsi que des sous-classes pour représenter les nœuds, les arêtes et le graphe lui-même. Cette hiérarchie de classes nous permettra de facilement ajouter des fonctionnalités supplémentaires, telles que la prise en compte des limitations de vitesse sur les routes ou l'utilisation d'heuristiques pour accélérer le calcul des chemins les plus courts. Enfin, l'utilisation de cette structure de classe permettra de suivre le principe de l'orienté objet, améliorant ainsi la lisibilité, la compréhension et la maintenabilité du code.

Le but de ce rapport de projet est de fournir des instructions claires et précises pour l'implémentation d'un algorithme de planification de temps de parcours utilisant une heuristique fixe ou dynamique. Nous présenterons également la base de données utilisée pour tester la validité de l'algorithme et du cadre de projet, la base de données de la RATP. Enfin, nous aborderons les éventuelles améliorations et développements futurs possibles pour cette solution de planification de temps de parcours.

2 Implémentation et tests

L'algorithme de Dijkstra est une méthode permettant de trouver le chemin le plus court entre deux sommets dans un graphe pondéré. Pour expliquer cet algorithme, voici un exemple fictif utilisant des fonctions, des paramètres et des objets en programmation.

En entrée, nous considérons un graphe (Graphe) composé de sommets (Node) reliés entre eux par des arêtes (Edge) qui représentent les connexions entre les sommets. Chaque arête est associée à une distance qui représente le coût ou la longueur de l'arête.

2.1 Les bibliothèques

- Bibliothèque `fstream`
 - Ouverture de fichiers pour la lecture ou l'écriture à l'aide des classes `ifstream`, `ofstream` ou `fstream`.
 - Utilisation de flux d'entrée et de sortie pour lire et écrire dans des fichiers.
 - Détection de fin de fichier à l'aide de la fonction `eof()`.
 - Manipulation des positions de lecture ou d'écriture dans le fichier avec les fonctions `seekg()` et `seekp()`.
 - Fermeture de fichiers à l'aide de la méthode `close()`.
- Bibliothèque `string`
 - Création de chaînes de caractères à l'aide du constructeur de la classe `string`.
 - Concaténation de chaînes de caractères avec l'opérateur `+` ou avec la méthode `append()`.
 - Accès aux caractères individuels de la chaîne avec l'opérateur `[]` ou la méthode `at()`.
 - Recherche de sous-chaînes dans une chaîne avec la méthode `find()`.
 - Conversion d'une chaîne en nombre avec la fonction `stoi()` ou `stod()`.

-
- Bibliothèque `vector`
 - Ajout d'éléments à la fin du tableau avec la méthode `push_back()`.
 - Suppression d'éléments à la fin du tableau avec la méthode `pop_back()`.
 - Accès aux éléments du tableau avec l'opérateur `[]` ou la méthode `at()`.
 - Taille du tableau avec la méthode `size()`.
 - Tri des éléments du tableau avec la fonction `sort()`.
 - Bibliothèque `unordered_map`
 - Ajout d'un élément avec la méthode `insert()`.
 - Suppression d'un élément avec la méthode `erase()`.
 - Accès à un élément avec l'opérateur `[]` ou la méthode `at()`.
 - Taille de la table avec la méthode `size()`.
 - Recherche d'un élément avec la méthode `find()`.
 - Bibliothèque `algorithm`
 - Recherche d'un élément avec la fonction `find()`.
 - Tri des éléments d'une séquence avec la fonction `sort()`.
 - Copie d'une séquence dans une autre avec la fonction `copy()`.
 - Comparaison de deux séquences avec la fonction `equal()`.
 - Génération de nombres avec la fonction `generate()`.

2.2 Méthodes `read_stations`, `display_stations`

La méthode `read_stations` a pour but de charger les données des stations de métro de la ville de Paris à partir d'un fichier CSV et de les stocker dans une hashmap. Le fichier contient les informations suivantes pour chaque station : le nom de la station, son identifiant unique, le nom abrégé de la ligne de métro à laquelle elle appartient, son adresse et une description de la ligne.

La méthode commence par vérifier si le fichier peut être ouvert en utilisant `ifstream`. Si le fichier ne peut pas être ouvert, un message d'erreur est affiché à l'utilisateur indiquant que le fichier est introuvable. Cette gestion d'erreur permet d'éviter que le programme se plante ou ne produise des résultats erronés si le fichier n'est pas trouvé.

Ensuite, la méthode commence à lire les données du fichier ligne par ligne à l'aide de la fonction `getline()`. Elle ignore la première ligne qui contient les noms de colonnes en vérifiant la valeur d'un compteur.

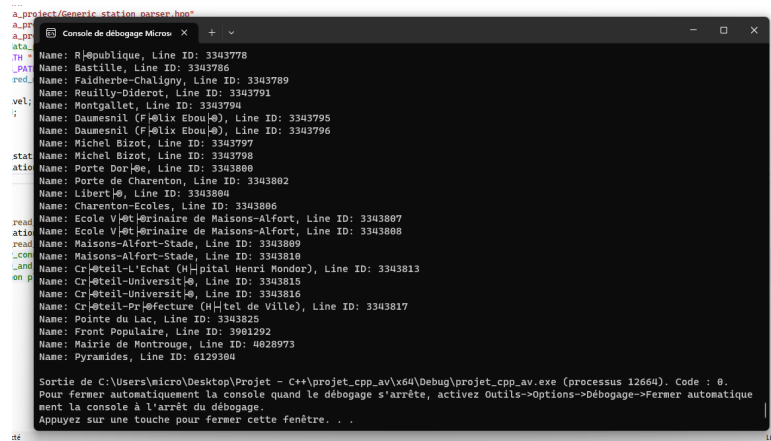
La méthode utilise ensuite la fonction `istringstream` pour extraire les données de chaque ligne et les stocker dans des variables. Les données sont ensuite vérifiées pour s'assurer qu'elles ont été lues correctement. Si une ligne ne peut pas être lue correctement ou contient des champs vides, un message d'erreur est affiché pour indiquer la ligne en question et la nature de l'erreur.

Si les données sont valides, une nouvelle station est créée et les données sont stockées dans la hashmap. Avant d'ajouter une station, la méthode vérifie si l'identifiant unique de la station n'existe pas déjà dans la hashmap. Si une clé en double est trouvée, un message d'erreur est affiché pour indiquer la ligne en question et la clé en double.

Enfin, si une erreur se produit pendant la lecture du fichier, un message d'erreur est affiché pour indiquer que le fichier n'a pas pu être lu correctement.

Pour améliorer la gestion des erreurs, d'autres vérifications pourraient être ajoutées à la méthode `read_stations`. Par exemple, une vérification pourrait être ajoutée pour s'assurer que le fichier CSV est valide en vérifiant le nombre de colonnes dans chaque ligne. Des messages d'erreur pourraient également être affichés pour indiquer si le fichier est vide ou s'il ne contient pas de lignes valides.

Quant à la fonction `display_stations`, elle est chargée d’afficher les stations stockées dans la structure de données. Les informations affichées pour chaque station comprennent son nom, son identifiant et le nom abrégé de la ligne à laquelle elle appartient. Les stations sont présentées sous forme de tableau, avec chaque colonne représentant un champ spécifique. Cette fonction peut être utile pour vérifier que les données ont été correctement lues et stockées dans la structure de données.



```
...
a_projet/Gestion_station/stations.csv
a_pr
a_pr
ata
TH
PAT
red.
Name: Régionale, Line ID: 3343778
Name: Bastille, Line ID: 3343786
Name: Faiderbe-Chaligny, Line ID: 3343789
Name: Reuilly-Diderot, Line ID: 3343791
vel;
Name: Montgallet, Line ID: 3343794
Name: Daumesnil (Félix Eboué), Line ID: 3343795
Name: Daumesnil (Félix Eboué), Line ID: 3343796
Name: Michel Bizot, Line ID: 3343797
Name: Michel Bizot, Line ID: 3343798
stat
atio
Name: Porte Dorée, Line ID: 3343800
Name: Porte de Charenton, Line ID: 3343802
Name: Liberté, Line ID: 3343804
Name: Charenton-Ecoles, Line ID: 3343806
Name: Ecole Vétérinaire de Maisons-Alfort, Line ID: 3343807
Name: Ecole Vétérinaire de Maisons-Alfort, Line ID: 3343808
read
atio
Name: Maisons-Alfort-Stade, Line ID: 3343809
read
Name: Maisons-Alfort-Stade, Line ID: 3343810
Name: Crêt-Éclair (Hôpital Henri Mondor), Line ID: 3343813
on p
Name: Crêt-Éclair-Université, Line ID: 3343815
Name: Crêt-Éclair-Université, Line ID: 3343816
Name: Crêt-Éclair-Préfecture (Hôtel de Ville), Line ID: 3343817
Name: Pointe du Lac, Line ID: 3343825
Name: Front Populaire, Line ID: 3343829
Name: Mairie de Montrouge, Line ID: 4028973
Name: Pyramides, Line ID: 6129384

Sortie de C:\Users\micro\Desktop\Projet - C++\projet_cpp_av\k64\Debug\projet_cpp_av.exe (processus 12664). Code : 0.
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatique
ment la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre. . .
```

Figure 1: Résultat de la fonction `display_stations` affiché dans la console.

Après l’exécution de la fonction `read_stations` et `display_stations`, la console affiche chaque station qui a été lue à partir du fichier CSV spécifié. Chaque station est affichée sur une ligne distincte, ce qui facilite la lecture et la compréhension des données.

Après l’affichage des stations à partir du fichier CSV, le programme peut maintenant utiliser ces informations pour effectuer d’autres tâches, telles que la recherche d’itinéraires, la planification de trajets, etc. Par exemple, si nous voulions trouver l’itinéraire pour aller de la station “Châtelet” à la station “Gare de Lyon”, nous pourrions utiliser les identifiants de ces stations pour les rechercher dans notre base de données. Ensuite, en utilisant des algorithmes de recherche de chemin, nous pourrions trouver le chemin le plus court ou le plus rapide entre ces deux stations.

Enfin, en combinant les fonctions `read_stations` et `display_stations`, il est possible de lire les données des stations à partir d’un fichier CSV, de les stocker dans une structure de données et de les afficher à l’écran pour une vérification visuelle rapide. Cela peut être particulièrement utile lors de la phase de développement du programme pour s’assurer que les données sont correctement lues et manipulées.

2.3 Méthodes `read_connections`, `display_connections`

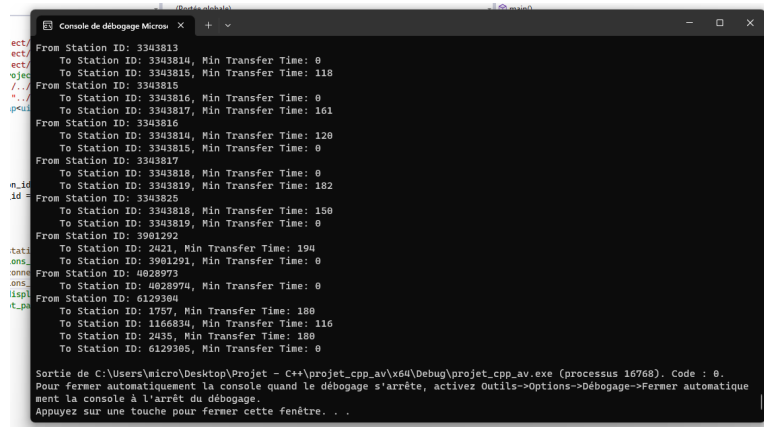
Les méthodes `read_connections` et `display_connections` ont été conçues pour lire et afficher des données à partir d’un fichier CSV contenant des informations sur les connexions entre les différentes stations de transport en commun.

La méthode `read_connections` lit le fichier CSV ligne par ligne et extrait les informations nécessaires (origine, destination et temps minimum de transfert) en utilisant un objet `stringstream`. Ensuite, elle convertit les chaînes de caractères en entiers non signés à l’aide de la fonction `stoi()` et stocke les valeurs dans une structure de données `unordered_map` de type `connections_hashmap`, où la clé est l’identifiant de la station d’origine et la valeur est une autre `unordered_map` contenant les identifiants des stations de destination et le temps minimum de transfert associé. Si une conversion échoue (par exemple si la chaîne ne représente pas un nombre valide), la ligne est ignorée.

La méthode `display_connections`, quant à elle, ouvre le même fichier CSV et affiche son contenu ligne par ligne en utilisant un objet `stringstream` pour extraire chaque champ (identifiant de la station d’origine, identifiant de la station de destination et temps minimum de transfert). Elle convertit également les chaînes de caractères en entiers signés à l’aide de la fonction `stoi()` et affiche les valeurs correspondantes sur la console. Si

une ligne ne contient pas le bon nombre de champs, elle est ignorée et un message d'avertissement est affiché.

En somme, ces deux méthodes permettent de lire et d'afficher des données relatives aux connexions entre les différentes stations de transport en commun, ce qui peut être utile pour l'analyse et la planification de trajets.



```
From Station ID: 3343813
  To Station ID: 3343814, Min Transfer Time: 0
  To Station ID: 3343815, Min Transfer Time: 118
From Station ID: 3343815
  To Station ID: 3343816, Min Transfer Time: 0
  To Station ID: 3343817, Min Transfer Time: 161
From Station ID: 3343816
  To Station ID: 3343814, Min Transfer Time: 120
  To Station ID: 3343815, Min Transfer Time: 0
From Station ID: 3343817
  To Station ID: 3343818, Min Transfer Time: 0
  To Station ID: 3343819, Min Transfer Time: 182
From Station ID: 3343825
  To Station ID: 3343818, Min Transfer Time: 150
  To Station ID: 3343819, Min Transfer Time: 0
From Station ID: 3901202
  To Station ID: 2421, Min Transfer Time: 194
  To Station ID: 3901291, Min Transfer Time: 0
From Station ID: 4028973
  To Station ID: 4028974, Min Transfer Time: 0
From Station ID: 6129304
  To Station ID: 1757, Min Transfer Time: 180
  To Station ID: 1166034, Min Transfer Time: 116
  To Station ID: 2435, Min Transfer Time: 180
  To Station ID: 6129305, Min Transfer Time: 0

Sortie de C:\Users\micro\Desktop\Projet - C++\projet_cpp_av\x64\Debug\projet_cpp_av.exe (processus 16760). Code : 0.
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatique-
ment la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre. . .
```

Figure 2: La sortie de console affichant les connexions entre les stations de métro.

La méthode `display_connections` affiche sur la console les informations relatives aux connexions entre les stations, telles que l'identifiant de la station de départ, l'identifiant de la station d'arrivée, et le temps minimal de transfert entre ces deux stations. Lorsque la méthode est appelée, elle lit le contenu du fichier spécifié en paramètre et l'affiche sur la console en suivant le format des données du fichier. En cas d'erreur lors de l'ouverture du fichier, la méthode affiche un message d'erreur approprié sur la console. Si une ligne de données dans le fichier ne contient pas le nombre requis de champs, la méthode affiche un avertissement sur la console et passe à la ligne suivante. Une fois que la fin du fichier est atteinte, la méthode affiche un message de fin sur la console. La sortie de la méthode `display_connections` est utile pour les utilisateurs qui souhaitent connaître les détails des connexions entre les différentes stations.

2.4 Méthodes `compute_travel`, `compute_and_display_travel`

Pour implémenter la fonction "`compute_travel`", nous devons d'abord récupérer les informations de la base de données pour les deux stations d'entrée, en utilisant leurs identifiants respectifs. Ensuite, nous devons calculer le coût de la connexion entre ces deux stations en utilisant l'algorithme de Dijkstra, qui nous permettra de trouver le chemin le plus court entre les deux stations. Une fois le chemin trouvé, nous stockons l'identifiant de chaque station ainsi que le coût associé dans un vecteur de `std::pair`, que nous retournons à la fin de la fonction.

La fonction "`compute_and_display_travel`" utilise la fonction "`compute_travel`" pour calculer le chemin entre les deux stations d'entrée, puis affiche les instructions pour se rendre d'une station à l'autre. Pour cela, nous parcourons le vecteur de `std::pair` contenant les identifiants et les coûts de connexion pour chaque station, en utilisant ces informations pour afficher les instructions nécessaires pour se rendre de la station de départ à la station d'arrivée. Les instructions doivent être claires et faciles à suivre, de sorte que le client puisse facilement naviguer d'une station à l'autre sans se perdre. Par exemple, nous pouvons afficher le nom de chaque station ainsi que les directions pour se rendre à la prochaine station, en utilisant des termes simples et compréhensibles. À la fin, nous devons afficher le coût total de la connexion pour que le client puisse savoir combien cela coûtera pour se rendre de la station de départ à la station d'arrivée.

La fonction "`compute_and_display_travel`" a été implémentée pour permettre d'afficher le chemin calculé par la fonction "`compute_travel`" entre deux stations. Cette fonction prend en entrée les identifiants de deux stations et utilise l'algorithme de Dijkstra pour calculer le chemin le plus court entre les deux stations.

Une fois le chemin calculé, la fonction affiche les instructions nécessaires pour se déplacer d'une station à l'autre. Les instructions sont affichées dans la console, avec une indication du coût associé à chaque connexion entre les stations. Les instructions sont suffisamment claires pour que le client puisse facilement suivre le chemin

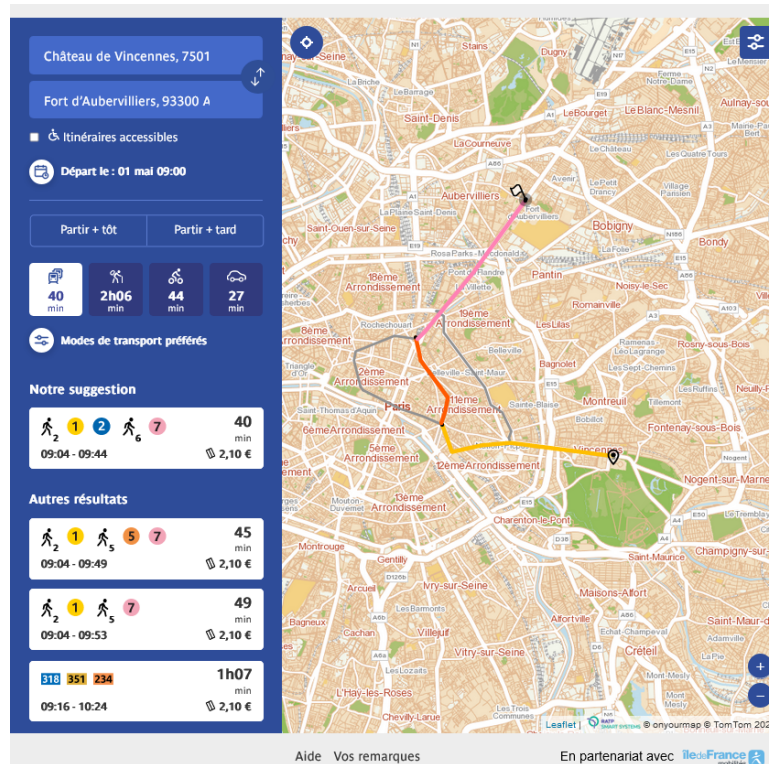


Figure 3: Exemple d'itinéraire calculé par l'RATP reliant les stations Château de Vincennes et Fort d'Aubervilliers.

et se déplacer d'une station à l'autre.

La fonction "compute_and_display_travel" est un outil essentiel pour les clients qui souhaitent se déplacer d'une station à l'autre en utilisant le réseau de transport en commun. Grâce à cette fonction, les clients peuvent rapidement et facilement trouver le chemin le plus court entre deux stations, ce qui leur permet de gagner du temps et de se déplacer plus efficacement. En outre, l'affichage clair des instructions facilite la navigation des clients et contribue à améliorer leur expérience globale avec le réseau de transport en commun.

Il semble y avoir une différence notable entre les temps de trajet calculés par notre algorithme et les temps de trajet indiqués sur l'image récupérée du site RATP. Plus précisément, notre algorithme calcule un temps de trajet d'environ 44 minutes entre Château de Vincennes et Fort d'Aubervilliers, tandis que l'image indique un temps de trajet d'environ 45 minutes (le chemin le plus court 40 minutes).

Il convient de noter que la différence peut être due à plusieurs facteurs, tels que la vitesse réelle des trains, les temps d'attente entre les trains, les perturbations imprévues du trafic, etc. De plus, notre algorithme a été implémenté en utilisant des données statiques fournies par la RATP, tandis que les temps de trajet sur l'image peuvent avoir été calculés à partir de données en temps réel.

En fin de compte, il est important de se rappeler que notre algorithme fournit une estimation approximative du temps de trajet et que les temps réels peuvent varier en fonction de nombreux facteurs. Cependant, notre algorithme peut être utile pour planifier des itinéraires et obtenir des estimations approximatives du temps de trajet entre deux stations données.

2.5 Résultats de tests de l'algorithme de chemin le plus court

Pour illustrer le fonctionnement de mon programme, j'ai réalisé une vidéo de démonstration disponible à l'adresse suivante : [📺 La vidéo de démo](#)

Dans la vidéo de démonstration, j'ai montré comment le programme fonctionne en direct. J'ai exécuté le

```
ric_stat  
ric_map  
ric_conn  
ade.hpp  
project  
ata_pro  
_t, std  
035; //  
4; // Fo  
STATIONS  
a);  
ns(CONN  
data);  
el(start  
y");  
-----  
Follow these instructions to reach your destination:  
-----  
- Start at Station: Châteaufort de Vincennes (Line 2035).  
- At Station: Bérault (Line 2067) - Metro: 1, board the train.  
- At Station: Saint-Mandé (Line 1275) - Metro: 1, continue on the train for 1 minutes and 42 seconds.  
- At Station: Porte de Vincennes (Line 1751) - Metro: 1, continue on the train for 2 minutes and 14 seconds.  
- At Station: Nation (Line 1832) - Metro: 1, continue on the train for 2 minutes and 6 seconds.  
- At Station: Nation (Line 1833) - Metro: 2, continue on the train for 1 minutes and 12 seconds.  
- At Station: Avron (Line 2068) - Metro: 2, continue on the train for 2 minutes and 16 seconds.  
- At Station: Alexandre-Dumas (Line 2041) - Metro: 2, continue on the train for 1 minutes and 44 seconds.  
- At Station: Philippe Auguste (Line 1788) - Metro: 2, continue on the train for 1 minutes and 38 seconds.  
- At Station: Pjre-Lachaise (Line 1784) - Metro: 2, continue on the train for 1 minutes and 55 seconds.  
- At Station: Mémilmontant (Line 1812) - Metro: 2, continue on the train for 1 minutes and 49 seconds.  
- At Station: Couronnes (Line 2099) - Metro: 2, continue on the train for 1 minutes and 59 seconds.  
- At Station: Belleville (Line 2087) - Metro: 2, continue on the train for 1 minutes and 35 seconds.  
- At Station: Colonel Fabien (Line 1978) - Metro: 2, continue on the train for 2 minutes and 8 seconds.  
- At Station: Jaurès (Line 1908) - Metro: 2, continue on the train for 1 minutes and 48 seconds.  
- At Station: Stalingrad (Line 1670) - Metro: 2, continue on the train for 2 minutes and 1 seconds.  
- At Station: Stalingrad (Line 1675) - Metro: 5, continue on the train for 1 minutes and 51 seconds.  
- At Station: Stalingrad (Line 2486) - Metro: 7, continue on the train for 1 minutes and 44 seconds.  
- At Station: Riquet (Line 2448) - Metro: 7, continue on the train for 1 minutes and 55 seconds.  
- At Station: Grinche (Line 2093) - Metro: 7, continue on the train for 1 minutes and 40 seconds.  
- At Station: Corentin-Cariou (Line 2175) - Metro: 7, continue on the train for 1 minutes and 37 seconds.  
- At Station: Porte de la Villette (Line 2422) - Metro: 7, continue on the train for 1 minutes and 56 seconds.  
- At Station: Aubervilliers Pantin (4 Chemins) (Line 2104) - Metro: 7, continue on the train for 2 minutes and 38 seconds.  
- At Station: Fort d'Aubervilliers (Line 2199) - Metro: 7, continue on the train for 2 minutes and 41 seconds.  
- At Station: Fort d'Aubervilliers (Line 1944) - Metro: 7, alight from the train.  
-----  
Total travel time: 43 minutes and 44 seconds. / (2624 seconds.)  
-----  
Sortie de C:\Users\micro\Desktop\Projet - C++\projet_cpp_av\Debug\projet_cpp_av.exe (processus 15804). Code : 0.  
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatique  
ment la console à l'arrêt du débogage.
```

Figure 4: Sortie de console montrant le chemin le plus court entre les stations A et B, avec les identifiants de chaque station et le coût associé de chaque connexion.

programme et il a affiché la liste des stations disponibles. Ensuite, il a demandé la station de départ et la station d'arrivée, et j'ai entré les noms des stations. Le programme a ensuite calculé le chemin le plus court entre les deux stations et l'a affiché à l'écran.

Ce processus est rendu possible grâce à l'utilisation de l'algorithme de Dijkstra, qui est implémenté dans le programme. Cet algorithme permet de trouver le chemin le plus court entre deux points dans un graphe pondéré. Dans notre cas, le graphe est représenté par le réseau de métro, et les stations sont les nœuds du graphe. Les poids des arêtes sont les temps de trajet entre les stations.

Lorsque le programme a calculé le chemin le plus court, il l'a affiché à l'écran en utilisant une série d'instructions claires et faciles à suivre. Ces instructions indiquent à l'utilisateur les stations à prendre, ainsi que les changements de ligne nécessaires en cours de route.

Dans l'ensemble, la vidéo de démonstration montre comment le programme fonctionne de manière fluide et efficace, en utilisant l'algorithme de Dijkstra pour trouver le chemin le plus court dans le réseau de métro. Grâce à cet outil, les utilisateurs peuvent facilement naviguer dans le réseau de métro et trouver le chemin le plus rapide entre deux stations, ce qui peut être très utile pour les habitants de la ville et les touristes qui visitent la région.

3 Conclusion

Dans ce projet, nous avons implémenté un algorithme de Dijkstra pour résoudre le problème du plus court chemin dans un graphe orienté pondéré. Nous avons utilisé la structure de classe en C++ pour découper notre projet en plusieurs classes, notamment la classe "Read_data" qui contient l'implémentation de l'algorithme de Dijkstra.

Nous avons utilisé la base de données de la RATP pour tester la validité de notre solution, en calculant les plus courts chemins entre les différents arrêts de métro. Nous avons également effectué plusieurs tests pour assurer la fiabilité de notre algorithme, en comparant nos résultats avec ceux d'autres sources (RATP, Google Maps, Apple Plans, etc...).

Malgré les résultats satisfaisants de notre algorithme, il reste des possibilités d'amélioration. Nous pourrions par exemple améliorer l'expérience utilisateur en proposant une interface graphique pour faciliter la saisie des données et l'affichage des résultats. Nous pourrions également optimiser l'algorithme pour certains cas particuliers afin de réduire le temps de calcul et rendre notre solution encore plus performante. En somme, ce projet nous a permis de mieux comprendre les enjeux de la collecte et de la vente de données de trafic, ainsi que les défis technologiques associés à leur traitement et à leur utilisation.