Projet Traitement de données
1A ENSAI 2023

Johann Faouzi

1 Avant-propos

Le *Projet Traitement de données* s'adresse aux étudiant•e•s en 1ère année à l'ENSAI. Chaque équipe-projet sera constitué de 3 à 4 étudiant•e•s pour travailler sur ce projet. Ce projet permet avant tout d'approfondir et de mettre en pratique les connaissances acquises en informatique en général et plus particulièrement en Python. Vous devrez mettre en avant le côté *programmation orientée objet* du langage Python. Votre code devra être bien structuré et documenté.

Pour chaque équipe, quatre séances encadrées sont consacrées à l'avancement et au suivi du projet. Attention, pour chacune de ces séances de 3 heures, un encadrant devra s'occuper de plusieurs équipes en même temps. Une part du travail lors de ces séances se fera donc en autonomie. Des créneaux dans votre emploi du temps vous sont réservés pour vous permettre d'avancer en autonomie en dehors des séances encadrées.

La création d'un programme informatique est souvent structurée en 4 phases :

1. **Spécification des besoins** : analyser et reformuler le problème posé afin de rédiger le **cahier des charges**.
2. **Conception globale** : définir l'architecture globale du programme.
3. **Conception détaillée** : décrire la structure interne des composants utilisés dans la conception globale.
4. **Implémentation** : transcrire la conception détaillée dans un langage de programmation, **Python** ici.

Néanmoins, il est tout à fait possible modifier la conception de votre programme informatique en cours de route si vous vous êtes rendus compte plus tard que l'implémentation pose problème avec la conception actuelle.

En coordination avec l'encadrant, vous devrez organiser votre projet en sous-tâches réparties chronologiquement et/ou par sous-groupe. Chaque séance débute par un point d'avancement de chacune des sous-tâches et, si nécessaire, par une mise à jour des tâches. Lors de la première séance encadrée, vous validerez avec votre encadrant les spécifications de votre programme, sa conception et l'organisation prévisionnelle des tâches. Vous devez donc venir à cette séance avec un (pseudo-)cahier des charges illustrant le fruit de votre réflexion concernant ces sujets. Le cahier des charges résultant de cette première séance sera soumis sur Moodle au plus tard le **3 avril 2023** à 23h59. **Attention :** ce cahier des charges est important pour bien commencer la première séance de travail encadrée. Il n'est pas demandé un cahier des charges parfait, mais plutôt suffisamment de pistes de réflexion sur qui utilisera votre programme et comment, quelles sont les fonctionnalités de votre programme, ou comment une exécution se déroule.

Vous trouverez sur Moodle les données que vous devez traiter. Le rapport doit être rédigé en LaTeX(un modèle a été déposé sur Moodle, vous pouvez l'utiliser librement). La date limite pour rendre le rapport et le code est fixée au **24 mai 2023** à 23h59. La date limite pour le suivi personnel est le **22 mai 2023** à 23h59.

Les soutenances sont programmées du **31 mai au 2 juin 2023**. Une présentation avec démonstration est demandée. Vous aurez 30 minutes : 20 minutes de présentation et démonstration, 10 minutes de questions. Les attendus pour le cahier des charges, le rapport, le code et la soutenance sont détaillés dans le tableau 1.

Vous êtes libres d'organiser votre travail au sein de votre équipe. Cependant, voici quelques conseils :

Objet	Attendu / Contenu
Cahier des charges ($\leqslant 5$ pages) **Pour la 1ère séance**	• Rédigez de façon claire et propre. • Décrivez l'architecture / l'arborescence de votre projet : À quoi correspond chaque dossier / chaque fichier ? Quelles fonctionnalités sont implémentées dans chaque fichier ? • Décrivez les fonctionnalités que vous coderez.
Suivi personnel ($\leqslant 2$ pages) **Pour la 4ème séance**	• Rôle dans l'équipe. • Tâches réalisées. **Un par étudiant** • Tâches en cours de réalisation.
Code	• Votre programme doit fonctionner. • Votre code doit être cohérent avec ce que vous décrivez dans le rapport. • La quantité et la qualité des fonctionnalités sont importantes dans l'évaluation. • Le code doit être découpé (au moins en fichiers, et éventuellement en dossiers si vous avez plusieurs fichiers pour une même partie de votre projet). • Le code doit être documenté. • Le code pour l'algorithme de Dijkstra doit être testé.
Rapport ($\leqslant 25$ pages)	• Rédigez de façon claire et propre. Afin de rédiger un document LaTeX, vous pouvez utiliser le logiciel TeXstudio ou des outils en ligne tel que `overleaf.com`. • N'oubliez pas que votre rapport doit être compréhensible par une personne ne connaissant pas ni le projet en général, ni votre projet en particulier. Prenez donc du recul dans l'écriture du rapport en introduisant les informations nécessaires pour que vos choix et l'aspect plus "technique" soient compréhensibles pour une personne extérieure. • Énoncez les fonctionnalité du cahier des charges qui ont été implémentées. Dans le cas contraire, expliquer pourquoi. • Les éventuelles images doivent être propres et lisibles. • L'architecture de votre rapport doit être cohérente avec l'architecture de votre code.
Soutenance	• Une démonstration de votre application. • Vous n'avez pas que 20 minutes de présentation. Ne passez pas trop de temps sur la présentation du problème (le jury aura lu votre rapport), présentez surtout votre travail (votre programme informatique) avec une prise de recul (vos choix et leurs justifications). • Les pistes d'amélioration (architecture, gestion de projet, fonctionnalités, documentation, réalisme des hypothèses, etc.).

TABLEAU 1 – Les attendus du projet.

— Pour le développement collaboratif de code, rien de mieux qu'un système de gestion de version, qui permettra de garder un historique des changements permettant de revenir en arrière si besoin, et une trace de l'activité de chacun. Il vous est conseillé d'utiliser Git pour cela. Consultez la page Moodle pour les détails. Vous pourrez également utiliser Git pour versionner votre rapport. Néanmoins, cet aspect ne sera pas pris en compte dans l'évaluation.

— Un espace de discussion est disponible sur Moodle. En particulier, consultez cet espace lorsque vous avez des questions, et formulez vos questions dans cet espace afin que les réponses soient partagées entre tous.

2 Données

Les données que vous utiliserez sont des données publiques de la SNCF. Le sujet général est le problème du plus court chemin, c'est-à-dire trouver un chemin dont la "distance" (au sens général, pas uniquement la distance euclidienne) est la plus courte, appliqué au transport ferroviaire en France. Trois jeux de données sur ce thème sont mis à disposition. Le premier concerne les tarifs minimaux et maximum des trains à grande vitesse (TGV). Le deuxième concerne les tarifs des trains express régionaux (TER). Le troisième concerne les informations sur les gares de voyageurs SNCF. Ici, le terme "distance" correspond donc au prix d'un trajet.

2.1 Tarifs TGV INOUI et OUIGO

Source Ce jeu de données est disponible sur le site `https://ressources.data.sncf.com/explore/dataset/tarifs-tgv-inoui-ouigo/`. Les prix plein tarif des trains TGV INOUI et OUIGO en vigueur au 1er mai 2022 sont mis à disposition. Le jeu de données est disponible sur Moodle (onglet *Données* de la page du cours *Projet Traitement de Données*), le nom du fichier étant `tarifs-tgv-inoui-ouigo.csv`. Vous pouvez télécharger directement le fichier sur le site (onglet *Export*), mais il se peut que les données aient été mises à jour.

Contenu Le site propose ce jeu de données en différents formats (CSV, JSON et Excel), mais nous nous limiterons au format CSV. Chaque ligne correspond aux prix minimum et maximum entre une gare d'origine et une gare de destination pour un transporteur donnée et une classe donnée. Vous pouvez vous inspirez du Listing 1 ou du Listing 2 pour charger ce type de fichier en Python. Les noms des variables sont dans l'ensemble explicites, les variables ne sont donc pas présentées ici. Le seul élément non-trivial à savoir est que le code UIC est un identifiant unique pour chaque gare.

```python
import csv
import os


folder = ''  # dossier où se trouve le fichier
filename = ''  # nom du fichier

with open(os.path.join(folder, filename)) as file:
    data = [row for row in csv.reader(file, delimiter=';')]
```

Listing 1 – Code pour lire un fichier `.csv` du sujet. Chaque ligne de la liste `data` contient une liste des entrées de la ligne correspondante du fichier CSV.

```python
import os
import pandas as pd


folder = ''  # dossier où se trouve le fichier
filename = ''  # nom du fichier

data = pd.read_csv(os.path.join(folder, filename), sep=';')
```

Listing 2 – Code pour lire un fichier `.csv` du sujet. `data` est un objet `pandas.core.frame.DataFrame` dont chaque ligne est un objet `pandas.core.series.Series` contenant les entrées de la ligne correspondante du fichier CSV.

2.2 Tarifs TER

Source Ce jeu de données est disponible sur le site https://ressources.data.sncf.com/explore/dataset/tarifs-ter-par-od/. Les tarifications des TER en vigueur au 1er juin 2022 sont mis à disposition. Le jeu de données est disponible sur Moodle (onglet *Données* de la page du cours *Projet Traitement de Données*), le nom du fichier étant `tarifs-ter-par-od.csv`. Vous pouvez télécharger directement le fichier sur le site (onglet *Export*), mais il se peut que les données aient été mises à jour.

Contenu Le site officiel propose ce jeu de données en différents formats (CSV, JSON et Excel), mais nous nous limiterons au format CSV. Chaque ligne correspond à la tarification pour le trajet entre une gare d'origine et une gare de destination. Vous pouvez vous inspirez du Listing 1 ou du Listing 2 pour charger ce type de fichier en Python. Les noms des variables sont dans l'ensemble explicites, les variables ne sont donc pas présentées ici. Encore une fois, le seul élément non-trivial à savoir est que le code UIC est un identifiant unique pour chaque gare.

2.3 Gare de voyageurs

Source Ce jeu de données est disponible sur le site https://ressources.data.sncf.com/explore/dataset/referentiel-gares-voyageurs/. Des informations, notamment géographiques, sur chaque gare de voyageurs sont mises à disposition. Le jeu de données est disponible sur Moodle (onglet *Données* de la page du cours *Projet Traitement de Données*), le nom du fichier étant `referentiel-gares-voyageurs.csv`. Vous pouvez télécharger directement le fichier sur le site (onglet *Export*), mais il se peut que les données aient été mises à jour.

Contenu Le site officiel propose ce jeu de données en différents formats (CSV, JSON et Excel pour les formats de fichiers plats), mais nous nous limiterons au format CSV. Chaque ligne correspond à une gare de voyageurs et fournit des informations disponibles sur cette gare. Vous pouvez vous inspirez du Listing 1 ou du Listing 2 pour charger ce type de fichier en Python. Les noms des variables sont dans l'ensemble explicites, les variables ne sont donc pas présentées ici. Encore une fois, le seul élément non-trivial à savoir est que le code UIC est un identifiant unique pour chaque gare.

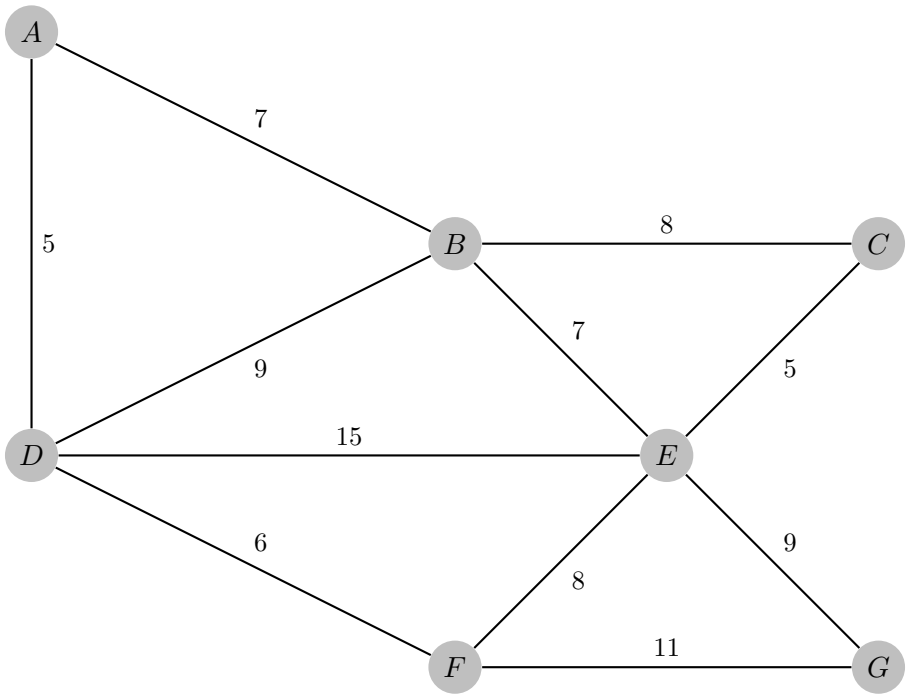

FIGURE 1 – Exemple d'un graphe non orienté avec des poids positifs ou nuls pour chaque arête. Le problème du plus court chemin consiste à trouver la "distance" minimale et un plus court chemin entre deux sommets. Par exemple, pour ce graphe donné, le plus court chemin de A à G est le chemin $A \to D \to F \to G$ et la distance correspondante est égale à 22.

3 Activités sur les données

L'objectif principal du projet est de déterminer le prix minimal pour aller d'une gare TGV d'origine à une autre gare TGV de destination. Vous allez donc travailler sur le problème du plus court chemin appliqué à ces données de la SNCF. Le projet va se diviser en trois grandes parties :

1. implémenter un algorithme permettant de résoudre le problème du plus court chemin,
2. implémenter une chaîne de transformations permettant de passer d'un ensemble de jeux de données brutes à un jeu de données pré-traitées adaptés à l'implémentation de cet algorithme, et
3. utiliser vos implémentations afin d'effectuer des analyses et de répondre à des questions.

3.1 Algorithme de Dijkstra

En théorie des graphes, le problème du plus court chemin consiste à trouver le plus court chemin (ou l'un des plus courts chemins en l'absence d'unicité) pour aller d'un sommet à un autre sommet.

Un graphe est un couple $G = (\mathcal{V}, \mathcal{E})$ où :

— $\mathcal{V}$ est un ensemble de sommets, et

— $\mathcal{E}$ est un ensemble d'arêtes reliant deux sommets entre eux.

Nous considèrerons ici les graphes non orientés, c'est-à-dire que s'il existe une arête allant du sommet A au sommet B, alors cette même arête va également du sommet B au sommet A. De plus, à chaque arête est associé un nombre réel positif ou nul appelé poids, correspondant à la "distance" entre les deux sommet. La distance est considérée symétrique (le poids pour aller du sommet A au sommet B est identique que le poids pour aller du sommet B au sommet A.) La figure 1 donne un exemple du type de graphes considérés.

Afin de résoudre le problème du plus court chemin, vous implémenterez l'**algorithme de Dijkstra**. Vous trouverez les informations nécessaires sur cet algorithme sur internet, par exemple dans la publication originale de l'algorithme, la page Wikipedia en anglais ou la page Wikipedia en français. Vous verrez (notamment sur la page Wikipedia en anglais) qu'il existe deux versions très similaires de l'algorithme dépendant de l'objectif recherché : (1) calculer le plus court chemin entre un *nœud source* et tous les nœuds, ou (2) calculer le plus court chemin entre un *nœud source* et un *nœud destination*. Vous implémenterez ces deux versions. Afin de mettre en avant le côté *programmation orientée objet*, vous implémenterez une seule classe Python pour les deux versions de l'algorithme de Dijkstra.

Vous devrez également choisir comment représenter un graphe (parmi l'ensemble des graphes considérés) en Python, et faire en sorte que votre implémentation fonctionne pour n'importe quel graphe avec la représentation choisie. Cette dernière n'a pas à être optimale en termes de temps d'exécution, mais elle devra être suffisamment pratique afin de faciliter votre implémentation.

Votre implémentation de l'algorithme de Dijkstra devra être testée (avec des tests écrits avec `pytest`). Vous testerez notamment que :

- le graphe donné en entrée a bien le format attendu par votre implémentation,
- chaque nombre réel associé à une arête du graphe est positif ou nul,
- votre implémentation renvoie bien les résultats attendus pour un graphe simple,
- votre implémentation gère le cas où il n'existe aucun chemin reliant deux sommets donnés du graphe,
- le coût minimal pour aller d'un sommet A à un sommet B est le même que le coût minimal pour aller du sommet B au sommet A,
- vous obtenez bien les mêmes résultats si (1) vous calculez le coût minimal et un plus court chemin entre un sommet d'origine et un sommet de destination et si (2) vous calculez le coût minimal entre le sommet d'origine et tous les sommets, puis un plus court chemin entre le sommet de destination.

3.2 Traitement des données

Une part importante du projet est d'implémenter un ensemble d'opérations sur les données de la SNCF précédemment introduites afin d'en effectuer l'analyse. Chaque opération n'a pas à être implémentée sous la forme d'une classe Python, mais la chaîne d'opérations devra être implémentée sous la forme d'une classe Python. La chaîne d'opérations doit renvoyer le graphe qui sera ensuite utilisé en entrée de votre implémentation de l'algorithme de Dijkstra. Vous porterez une attention particulière pour rendre votre code flexible et général. Par exemple, si vous avez deux bouts de code pour traiter deux cas différents mais similaires, il vaut mieux avoir un seul bout de code légèrement plus long qui traite les deux cas. De cette manière, votre code sera plus facilement **réutilisable** et **extensible**.

Votre chaîne d'opérations devra être capable de traiter les hypothèses suivantes :

- **Sélectionner un sous-ensemble de trains** (basé sur des critères tels que le transporteur ou la classe).
- **Sélectionner le prix minimum ou le prix maximum**.
- **Rajouter des correspondances entre certaines gares TGV.** En effet, nous nous limitons dans ce projet aux trajets entre des gares TGV, mais il existe des correspondances entre certaines gares. Par exemple, le métro parisien permet de relier les différentes gares TGV situées dans Paris intra-muros. Il existe également des trains express régionaux (TER) reliant des gares de voyageur. Votre implémentation devra donc permettre de rajouter des correspondances entre des gares TGV si elles se trouvent dans la même ville, le même département ou la même région. Pour simplifier, on pourra considérer que le prix d'une correspondance au sein d'une même ville (respectivement département ou région) ne dépend pas de la ville (respectivement

département ou région). Pour une hypothèse un peu plus réaliste, vous pourrez utiliser les prix moyens des TER au sein de chaque département département et au sein de chaque région.

3.3 Analyses

En utilisant vos implémentations pour les deux parties précédentes, vous effectuerez des analyses assez simples. Dans cette partie, votre code devra être court et succinct étant donné que ce sont vos implémentations pour les deux parties précédentes qui effectuent l'immense majorité du travail. En particulier, vos implémentations pour les deux parties précédentes devront être flexibles, dans le sens où si vous changez les hypothèses effectuées, alors vous n'avez qu'à changer les valeurs des paramètres afin d'obtenir le résultat attendu. Si vous êtes obligés d'écrire beaucoup de lignes de code dans cette partie pour changer une hypothèse par exemple, alors vos implémentations précédentes manquent de flexibilité, il est donc nécessaire de les modifier afin de les rendre plus générales.

3.4 Paquets Autorisés

Vous pourrez utiliser les paquets `numpy`, `matplotlib`, `scipy` et `pandas`. En particulier, si vous décidez d'utiliser `pandas`, il sera attendu d'effectuer le plus d'opérations possibles en utilisant les outils déjà mis à disposition dans ce paquet (et donc d'éviter le plus possible les boucles `for`). Pour les autres paquets ne faisant pas partie de Python, demandez confirmation avant de les utiliser.

Il est interdit d'utiliser une implémentation de l'algorithme de Dijkstra mises à disposition dans un paquet Python, il est attendu que vous codiez votre propre implémentation. Cependant, vous êtes autorisés à utiliser une implémentation existante pour vos tests (afin de vérifier que vous obteniez le même résultat), et **uniquement pour vos tests**.

4 Exemples de questions

Pour fixer les idées, voici une courte liste de questions qu'on souhaiterait étudier avec ces données, et aux lesquelles votre programme informatique permettra de répondre :

- Quel est le prix minimal et un plus court chemin pour aller d'une gare TGV donnée à une autre gare TGV donnée, à hypothèses fixes ?
- Comment évolue le prix minimal et un plus court chemin pour aller d'une gare TGV donnée à une autre gare TGV donnée en faisant varier certaines hypothèses ?
- Est-il vérifié empiriquement qu'il est plus rapide de (1) calculer le prix minimal et un plus court chemin entre une gare TGV d'origine donnée et une autre gare donnée, que de (2) calculer le prix minimal entre une gare TGV d'origine donnée et toutes les autres gares, puis de calculer un plus court chemin entre la gare TGV d'origine donnée et l'autre gare donnée ?
- Est-il vérifié empiriquement qu'il est plus rapide de (1) calculer les prix minimaux entre une gare TGV d'origine donnée et toutes les autres gares (en une seule fois), puis de calculer des plus courts chemins entre cette gare TGV d'origine et toutes les autres gares TGV, que de (2) calculer le prix minimal et un plus court chemin entre une gare TGV d'origine donnée et une autre gare, ce processus étant répété pour toutes les autres gares TGV ?

Vous pourrez ajouter d'autres questions à votre rapport, d'autant plus si elles mettent en valeur la conception de votre programme. À noter que l'étude statistique des données d'entrée ou des résultats n'est pas demandée et ne sera pas évaluée, ne perdez donc pas de temps sur ce point. Il est donc fortement conseillé de ne pas en faire et de ne pas mettre dans votre rapport.

5 Sortie

Votre programme devra permettre de sauvegarder les résultats de son exécution. Il devra sauvegarder dans un fichier ou plusieurs fichiers aux formats adaptés (par exemple CSV pour des données tabulaires ou TXT pour du texte) ces résultats. En particulier, le code correspondant à toutes les analyses et tous les résultats présentés dans le rapport devra être disponible, et tous les résultats devront être sauvegardés dans un ou plusieurs fichiers. Une personne extérieure doit être capable de générer à nouveau tous les fichiers comportant vos résultats simplement en exécutant un ou plusieurs fichiers Python (s'il y a plusieurs fichiers, leur nombre doit être raisonnable).

6 Bonus : Ajout des trajets entre TER

Au lieu de considérer uniquement les gares de voyageurs desservies par des TGV et de faire des hypothèses simplificatrices sur les éventuelles correspondances entre ces différentes gares, vous pourrez considérer l'ensemble des gares de voyageurs et utiliser les correspondances TER. Il restera néanmoins nécessaire de faire des hypothèses simplificatrices au niveau local (typiquement au sein d'une même ville, ou en région parisienne) afin de tenir compte d'autres correspondances possibles (bus, métro, etc.) entre certaines gares de voyageurs.

Le graphe sera donc plus grand et les temps d'exécution de vos programmes seront donc probablement plus élevés. Si l'exécution de vos programmes prend déjà beaucoup de temps, il n'est probablement pas pertinent d'essayer de faire ce bonus.

Rappel : Ce bonus est optionnel et vous n'avez pas à le faire pour avoir une excellente note. Focalisez-vous d'abord sur l'objectif principal du projet avant d'éventuellement le faire. Ce bonus ne se substitue pas au projet principal qui se limite aux gares TGV.