ENSAE 1A – Projet de programmation
Optimisation d'un réseau de livraison

Responsable du cours: Patrick Loiseau

patrick.loiseau@inria.fr

Version du 10 février 2023

1 Instructions et recommandations générales

1.1 Objectif

Le projet de programmation a pour objectif global de vous aider à acquérir une expérience de programmation **par la pratique**, via la résolution d'un problème particulier (décrit dans la section suivante). Plus spécifiquement, il est attendu au cours du projet que vous acquériez un certain nombre de compétences (liste non nécessairement exhaustive) :

- Apprendre à structurer du code pour un projet de moyenne envergure
- Apprendre à chercher en ligne les informations, les documentations, et les librairies utiles
- Apprendre à analyser un problème et à le traduire en terme algorithmique
- Apprendre à analyser la complexité d'un algorithme
- Apprendre à manipuler quelques algorithmes et structures de données classiques
- Apprendre à debugger, tester, et optimiser du code
- Se familiariser avec les bonnes pratiques de code et avec un environnement d'édition.

Le projet est fait en Python (obligatoirement). On utilisera la notion de classe, avec une structure minimale imposée (voir plus loin). Pour l'ensemble du projet, on utilisera le service vscode-python de la plateforme SSP Cloud (hébergée par l'Insee) qui offre un environnement Visual Studio code avec la version 3.10 de python. Pour y accéder, allez sur https://datalab.sspcloud.fr/catalog/ide et sélectionnez `Vscode-python`. Il n'est pas autorisé d'utiliser des notebook jupyter, il est obligatoire de faire les TP et le projet dans des fichiers python d'extension `.py`.

Note : Le service sur sspcloud permet d'accéder à un environnement uniforme avec des logiciels à jour de n'importe où, y compris à la maison. Attention toutefois à bien enregistrer votre travail car un service inutilisé peut être "tué" à tout moment.

1.2 Évaluation et rendus

Le projet doit être fait en binôme ; les deux membres d'un même binôme doivent appartenir au même groupe de TP. Vous devez vous inscrire avec votre binôme, **avant le début de la séance 2**, sur ce formulaire :

https://forms.office.com/e/uRELb0abdQ.

Le projet contient 6 séances de TP. Les trois premières séances de TP seront relativement guidées, les deux suivantes plus ouvertes, la dernière servira à l'évaluation finale. On demande 2 rendus :

Un rendu intermédiaire, dû le vendredi 10/03 (23h59) : Après la 3ème séance de TP, on demande le rendu de l'ensemble du code écrit jusque-là, testé et documenté. Le code doit être déposé dans un dépôt de code github, seul le lien doit être envoyé au chargé de TP (voir section 1.4).

Un rendu final, dû le jeudi 06/04 (23h59) : Une semaine avant la dernière séance de TP, on demande le rendu final de l'ensemble du code du projet, documenté, accompagné d'un rapport d'environ 4-6 pages au format PDF. Le rapport doit décrire succinctement les choix algorithmiques et les résultats et justifier la complexité des algorithmes utilisés. Comme pour le rendu intermédiaire, le code lui-même doit être déposé dans un dépôt github dont le lien doit être donné dans le rapport PDF.

Lors de la dernière séance de TP, le 13/04, chaque chargé de TP passera voir tous les binômes de son groupe pour une séance de questions destinée à évaluer la compréhension des deux membres du binôme. Il ne sera pas demandé de faire une présentation du projet (pas de slide), mais il faudra pouvoir faire une démo si demandé.

La présence aux TPs est obligatoire et sera vérifiée au début de chaque séance. La notation inclura 5 points (sur 20) pour la présence et le rendu intermédiaire. Nous serons particulièrement attentifs au fait que le code soit testé et bien documenté. Le reste (15 points) sera attribué sur la base du rendu final et de la séance de questions lors du dernier TP. Nous tiendrons compte d'un certain nombre de critères : qualité du code, clarté de la documentation, pertinence des choix algorithmiques, analyse des performances algorithmiques, respect des bonnes pratiques, etc.

Note générale : Nous donnons une base de code (classe Graph, etc.) et un certain nombre d'instructions sur la structure du code et les spécifications. Il est crucial de bien respecter ces instructions et de ne pas modifier les noms des fonctions données en les implémentant.

1.3 Ressources utiles

Le projet suppose une connaissance du language Python acquise au premier semestre. Toutefois, en cas de besoin, il existe de nombreux cours et livres sur la programmation en Python, tels que *Introduction to programming in Python* de R. Sedgewick, K Wayne, et R. Dondero. Pour ce qui est des bonnes pratiques, il existe de nombreuses ressources. Citons en particulier :

— Les documents de la communauté Python, e.g., le PEP8 (qui contient une section sur le nommage) et le PEP257 (sur l'écriture de documentation)
— Le Hitchhiker's Guide to Python
— Les supports de Lino Galiana (enseignant du cours Python pour la data science de deuxième année) sur les bonnes pratiques et la qualité du code
— Les pages de Xavier Dupré (ancien responsable du projet et enseignant du cours du premier semestre) donnant des conseils divers

Il pourra être utile de s'y référer au long du projet. Enfin, les élèves pourront être amenés à vouloir consulter des livres d'algorithmique ; il en existe là aussi de nombreux (e.g., *Introduction to Algorithms* de Cormen et al. ou (plus court !) *Algorithms* de Dasgupta, Papadimitriou, et Vazirani).

1.4 Contact des chargés de TP

— Groupe 1 : Lucas Baudin, `lucas.baudin@ensae.fr`
— Groupe 2 : Patrick Loiseau, `patrick.loiseau@inria.fr`
— Groupe 3 : Ziyad Benomar, `ziyad.benomar@ensae.fr`
— Groupe 4 : Quentin Jacquet, `quentin.jacquet@edf.fr`
— Groupe 5 : Salim Chouaki, `salim.chouaki@inria.fr`

2 Description du problème : optimisation d'un réseau de livraison

On considère un réseau routier constitué de villes et de routes entre les villes. L'objectif du projet va être de construire un réseau de livraison pouvant couvrir un ensemble de trajets entre deux villes avec des camions. La difficulté est que chaque route a une limite inférieure de puissance, c'est-à-dire à dire qu'un camion ne peut emprunter cette route que si sa puissance est supérieure ou égale à la limite inférieure de puissance de la route. Il faudra donc déterminer pour chaque couple de villes si un camion d'une puissance donnée peut trouver un chemin possible entre ces deux villes ; puis chercher à optimiser la flotte de camions qu'on achète en fonction des trajets à couvrir.

On représente le réseau routier par un graphe non orienté $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. L'ensemble des sommets $\mathcal{V}$ correspond à l'ensemble de villes. L'ensemble des arêtes $\mathcal{E}$ correspond à l'ensemble des routes existantes entre deux villes. Chaque arête $e \in \mathcal{E}$ est associée à une valeur (ou poids) $\bar{p}_e \geq 0$ qu'on appelle puissance minimale de l'arête e, qui correspond à la puissance minimale d'un camion pour qu'il puisse passer sur la route e. Chaque arête $e \in \mathcal{E}$ est également associée à une deuxième valeur $d_e > 0$ correspondant à la distance entre les deux villes sommets sur l'arête e.

On considère un ensemble $\mathcal{T}$ de trajets, où chaque trajet t est un couple de deux villes distinctes, i.e., $t = (v, v')$ où $v \in \mathcal{V}, v' \in \mathcal{V}, v \neq v'$. L'ensemble $\mathcal{T}$ représente l'ensemble des couples de villes (v, v') pour lesquelles on souhaite avoir un camion qui puisse faire le transport entre v et v'. Notez que le graphe n'est pas orienté et on ne distingue pas la direction du trajet. À chaque trajet t est également associé un profit (ou utilité) $u_t \geq 0$, qui sera acquis par la compagnie de livraison si le trajet t est couvert.

Enfin, le transport est fait par des camions. Chaque camion (noté K) a une puissance p et coûte un prix c. Le transport sur un trajet $t = (v, v') \in \mathcal{T}$ sera possible si et seulement si on peut trouver dans le graphe $\mathcal{G}$ un chemin allant de v à v' sur lequel la puissance minimale de chaque arête est inférieure où égale à p (i.e., le camion a une puissance suffisante pour passer partout sur le chemin). On dit alors que le trajet t peut être couvert par le camion K considéré.

Le projet contient deux parties :

Partie 1 : calcul de la puissance minimale pour un trajet. Dans cette partie, qui couvre environ les trois premières séances, on s'intéresse au problème de calculer la puissance minimale nécessaire d'un camion pour un trajet $t \in \mathcal{T}$ (ainsi que le chemin associé).

Partie 2 : optimisation de l'acquisition de camions. Dans cette deuxième partie, on cherche à calculer quels camions acheter pour optimiser le profit des trajets couverts.

Notation On notera $V = |\mathcal{V}|$ le nombre de sommets du graphe $\mathcal{G}$ (aussi souvent appelé n) ; $E = |\mathcal{E}|$ le nombre d'arêtes du graphe $\mathcal{G}$ (aussi souvent appelé m) ; et $T = |\mathcal{T}|$ le nombre de trajets que l'entreprise souhaite couvrir.

Toutes les ressources associées au projet pour les étudiants (base de code, fichiers d'input, tests préliminaires, etc.) se trouve sur le dépôt github (publique) suivant :

https://github.com/patrickloiseau/ensae-prog23

qui contient également une explication des fichiers fournis.

Séance 1 : 10/2

Dans cette première séance, on va prendre en main la structure générale du code et implémenter quelques premiers algorithmes pour calculer si un camion peut couvrir un trajet (et la puissance minimale d'un camion pour un trajet donné).

Vous devez programmer vous-mêmes les algorithmes de graphe dans la classe `Graph` fournie dans le dépôt git du projet. En particulier, il n'est pas autorisé d'utiliser des paquets comme `networkx` pour la partie algorithmique. Vous pouvez l'utiliser, exclusivement pour la visualisation.

Une base de code, des fichiers de tests pour une partie des questions et des fichiers de données sont disponibles. Pour exécuter les tests, vous pouvez ou bien le faire dans l'onglet « Test » de VS Code, ou bien les exécuter en ligne de commande, par exemple pour la première question :

```
python tests/test_s1q1_graph_loading.py
```

Note : Il est utile de regarder les tests en codant, pour être sûr qu'une fonction retourne le résultat au format attendu par le test. Nous ne fournissons que des tests très minimalistes dont le but est de vous aiguiller vers une bonne structure de tests pour démarrer. Il est essentiel que vous implémentiez d'autres tests par vous-même pour compléter.

Dans cette première séance, on s'intéressera uniquement aux fichiers d'entrés `network.x.in` où `x` est dans $\{00, 01, 02, 03, 04\}$, qui sont des petits fichiers destinés à pouvoir tester que les fonctions implémentés retournent les résultats corrects.

Les fichiers python à compléter dans un premier temps sont `main.py` et `graph.py`. Vous pouvez tester une partie de vos fonctions en appelant les fonctions dans `main.py`.

Question 1. Compléter la définition de la classe Graph en implémentant en particulier la méthode `add_edge` et écrivez une fonction `graph_from_file` qui permet de charger un fichier. Pour commencer, on s'intéresse seulement aux fichiers `network.x.in` où `x` est dans $\{00, 01, 02, 03\}$. Le fichier a le format suivant (comme dans le dossier `inputs`) :

- la première ligne est composée de deux entiers séparés par un espace : le nombre de sommets et le nombre d'arêtes ;
- les lignes suivantes représentent chacune une arête et sont composées de 3 entiers : les 2 numéros des sommets (à partir de 1) suivi de la puissance de l'arête.

Par exemple, le graphe :

$$1 — 2 — 3$$

sera représenté par le fichier (avec des puissances de 10 pour chaque arête) :

```
3 2
1 2 10
2 3 10
```

Question 2. Écrire une méthode qui trouve les composantes connectées du graphe $\mathcal{G}$ et analyser la complexité de l'algorithme en fonction de V et E.

Indice : On pourra implémenter un parcours en profondeur du graphe en utilisant une fonction récursive et on pourra se limiter ici à une méthode qui fonctionne pour les petits graphes.

Question 3. Écrire une fonction qui prend en entrée une puissance de camion p et un trajet t et qui décide si un camion de puissance p peut couvrir le trajet t (et retourne, si c'est possible, un chemin admissible pour le trajet t). En pratique, on retournera le chemin si c'est possible et `None` sinon.

Analyser la complexité de l'algorithme.

Question 4. Modifier la fonction de lecture des fichiers pour lire la distance d'une arête qui est optionnelle. Ainsi la ligne `1 2 10 5` représente une arête qui va de 1 à 2, avec puissance minimale 10 et distance 5. La ligne `1 2 10` représente toujours une arête qui va de 1 à 2, avec puissance minimale 10 et distance 1 (la valeur par défaut). Tester sur le fichier `network.04.in`.

Question 5 (bonus)**.** Modifier la fonction de la question 3 pour qu'elle retourne, lorsque le trajet peut être couvert par le camion, le plus court (au sens de la distance d) chemin admissible pour le trajet t.

Analyser la complexité de l'algorithme ci-dessus.

Question 6. Écrire une fonction qui calcule, pour un trajet t donné, la puissance minimale d'un camion pouvant couvrir ce trajet.

Analyser la complexité de l'algorithme ci-dessus.

Indice : On pourra penser à la recherche binaire.

Question 7. Implémenter une représentation graphique du graphe, du trajet, et du chemin trouvé. On pourra utiliser une bibliothèque pour cela, par exemple `graphviz`, vous pouvez l'installer sur le SSP Cloud avec le script `install_graphviz.sh`, à exécuter dans un terminal.

Question 8. Tester les algorithmes sur les graphes fournis. Implémenter de nouveaux tests des fonctions développées.

Question 9 (bonus, difficile)**.** Télécharger une carte de France. La convertir en un graphe avec des villes et des routes où les puissances minimales sont $\bar{p}_1$ pour les routes départementales et inférieures, $\bar{p}_2 > \bar{p}_1$ pour les routes nationales, et $\bar{p}_3 > \bar{p}_2$ pour les autoroutes. Pour la distance, on pourra prendre soit la distance en km soit le temps de trajet. Pour l'ensemble des trajets $\mathcal{T}$, on prendra l'ensemble des couples de deux villes de plus 100,000 habitants. À chaque trajet, on associera un profit égal au minimum du nombre d'habitants entre les deux villes.

Indice : On pourra utiliser la bibliothèque OSMnx à l'adresse suivante
https://osmnx.readthedocs.io/en/stable/

Si vous êtes arrivé à ce point et que tout fonctionne bien, vous pouvez commencer à tester sur des graphes plus gros (`network.x.in` avec `x` ≥ 1), ainsi qu'en répétant un grand nombre de fois la requête de la question 6. Pour cela, on fournit des ensembles de trajets `network.x.in` avec `x` ≥ 1 associés à chaque graphe correspondant. La structure des fichiers `network.x.in` est la suivante :

— la première ligne contient un entier qui correspond au nombres de trajets dans l'ensemble
— les lignes suivantes contiennent chacune un trajet sous la forme
`ville1 ville2 utilité`