Intelligence artificielle

TP1 : Itineria

Objectifs du TP

Implémenter les méthodes de recherche avec et sans heuristiques vus en cours et en TD au sein d'un système de calcul d'itinéraire.

Comprendre que l'IA s'intègre dans des architectures applicatives, elle n'est pas une fin en soi mais un moyen d'améliorer un traitement.

Coder des fonctionnalités dans un environnement déjà défini en partie

---

Description du TP

Un squelette d'application d'itinéraire vous est fourni sous forme d'un projet eclipse contenu dans le fichier `squelette_itineria.zip` disponible dans Moodle.

L'interface utilisateur se fait pour l'instant par saisie clavier dans une console mais en observant le programme vous verrez que les bases sont prêtes.

Pas besoin d'entrer des données dans l'application, elle contient déjà :

- 95 préfectures de département (et leur position GPS)
- 244 routes permettant de relier chaque préfecture avec les préfectures des départements limitrophes (avec leur longueur en km et leur durée en minutes)

⇒ Entrer à la main (et dans la douleur) par votre enseignant!

Consignes

Étudiez l'architecture du squelette ainsi que les fonctions et classes déjà définis mais également celle qu'il reste à compléter. Normalement, je vous ferai un rapide tour du propriétaire en début de TP.

Quand une fonction existe mais que son corps est vide (ou qu'elle ne contient qu'un `return null` ou un `return 0`) ce n'est pas une erreur, c'est sans doute que vous aurez besoin de remplir cette fonction par la suite, soyez vigilant. Ce sont des aides pour vous.

Lisez bien l'énoncé, il contient beaucoup d'informations et chaque mot est utile!

Aidez-vous de la documentation que je vous indiquerai. Lorsqu'il utilise une nouvelle librairie, un développeur passe beaucoup de temps dans la documentation pour comprendre son fonctionnement. RTFM!

Utilisez Python à 100%, Python est fourni avec un grand nombre d'outils qui facilitent votre travail, utilisez -les!

Pensez **objet**, sachez utiliser les avantages de la programmation objet.

Coder proprement avec des noms de variables et de fonctions/méthodes bien choisis. Ajoutez des commentaires utiles à votre code afin de pouvoir encore le comprendre dans quelques jours.

Respecter les coding style. Je vous conseille de suivre les consignes du PEP8 [1] ou le Google Python Styleguide [2].

1. PEP8 : https://www.python.org/dev/peps/pep-0008/
2. Google Python Styleguide : https://google.github.io/styleguide/pyguide.html

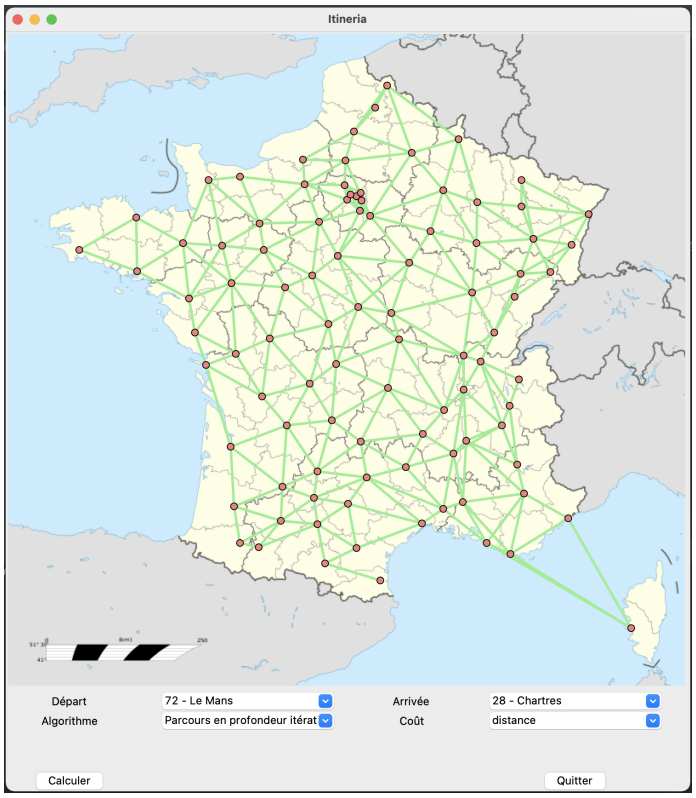
Simple is beautiful. N'essayez pas de coder compliqué, restez simple, cela sera plus efficace et moins sujet aux erreurs.

Réfléchissez avant de coder, prenez un peu de temps pour dessiner/écrire vos idées sur une feuille. Le temps de la réflexion que vous prendrez, vous économisera énormément de temps de debug.

---

1 Squelette et interface graphique

Vous disposez d'un squelette appelé `TP1_Squelette.py` qui charge déjà les données csv (également à télécharger) et qui vous propose une interface graphique entièrement fonctionnelle (cf. capture ci-dessous). La seule chose qu'il vous reste à faire est de coder les fonctions laissées vides.



Les fonds de carte proviennent de Wikipedia et sont disponibles en 3 tailles sur Moodle. Choissisez celle qui convient à la résolution de votre écran (il y a 3 lignes dans le code permettant de choisir la bonne résolution).

Dans un premier temps, familiarisez-vous avec le code existant pour le comprendre et pouvoir insérer efficacement vos algorithmes dedans.

Si vous ne maîtrisez pas trop Python, vous pouvez aller voir ce Google Colab vous présentant les bases de Python.

Pour votre environnement Python, je vous conseille d'installer :
— la distribution Anaconda
— l'IDE PyCharm (Version Pro gratuite avec votre adresse mail UHA)

2 Implémentations des algorithmes

2.1 Parcours en largeur

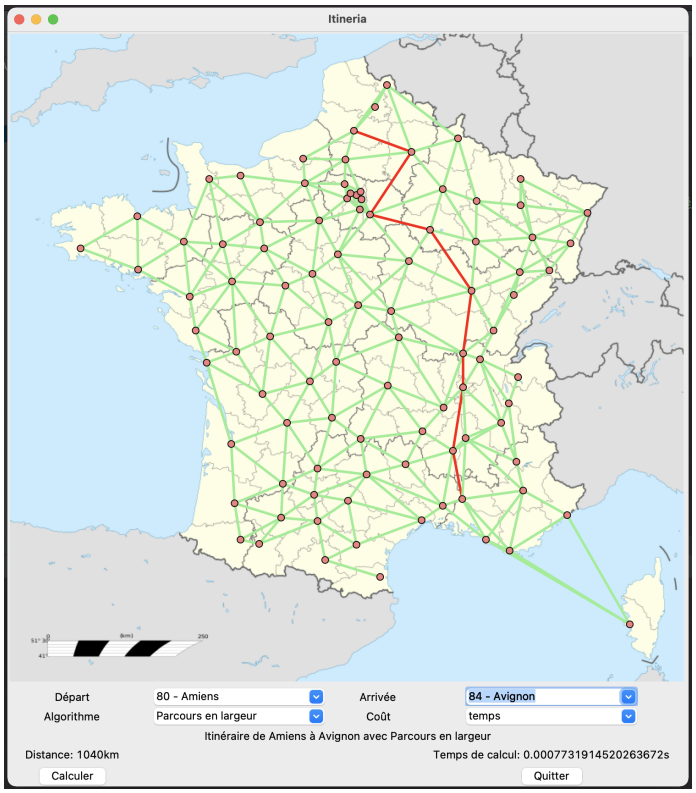
En utilisant l'algorithme du cours, implémentez le parcours en largeur..

⚠Vous aurez besoin du module `Queue` [3]

⚠Vous allez également devoir créer une classe `Node` pour représenter les noeuds.

Si réussi vous devriez obtenir un résultat similaire à la capture ci-dessous.

3. Queue : `https://docs.python.org/3/library/queue.html`



2.2 Parcours en profondeur

Implémentez le parcours en profondeur en version récursive.

⚠Vous aurez besoin du module `LifoQueue` [4] pour stocker les villes parcourues dans le chemin actuel.

2.3 Parcours en profondeur itératif

Implémentez le parcours en profondeur itératif.

⚠Vous aurez besoin du module `LifoQueue` [5]

2.4 Parcours à coût uniforme

Implémentez le parcours à coût uniforme.

⚠Vous aurez besoin du module `PriorityQueue` [6]

⚠Vous devrez modifier votre classe `Node` pour pouvoir comparer les noeuds.

2.5 Recherche gloutonne

Implémentez la recherche gloutonne.

Vous devrez également définir la distance à vol d'oiseau ('As the crow flies').

2.6 A*

Implémentez A*.

⚠Cela ne va pas être trivial de gérer la distance parcourue et l'heuristique!

4. LifoQueue : https://docs.python.org/3/library/queue.html
5. LifoQueue : https://docs.python.org/3/library/queue.html
6. PriorityQueue : https://docs.python.org/3/library/queue.html

3 Bonus

Déjà fini ? Voici quelques améliorations possibles à implémenter.

— Gérez le coût défini en temps au lieu de la distance
— Ajoutez un calcul de coût de la méthode en termes de nœuds développés