SAÉ2.01

Développement orienté objets

BUT Informatique

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL
iut CRÉTEIL-VITRY
DÉPARTEMENT INFORMATIQUE

TABLE DES MATIÈRES

Introduction

Dans cette SAÉ, vous allez concevoir et développer une application orientée objet pour résoudre un problème spécifique en utilisant un langage de programmation orienté objet, tel que Java ou Python.

L'objectif de ce projet est de vous permettre de développer vos compétences en programmation orientée objet, en conception de logiciels et en résolution de problèmes complexes.

Vous devrez concevoir la structure de données et mettre en œuvre les fonctionnalités nécessaires pour résoudre le problème posé, tout en réalisant des tests de validation pour assurer la qualité de votre code. Pour cela, vous pourrez utiliser des outils de développement et de test appropriés.

Le problème spécifique que vous aurez à résoudre implique la recherche du trajet optimal entre deux lieux. Pour résoudre ce problème, vous devrez implémenter des algorithmes tels que l'algorithme de Dijkstra ou l'algorithme de Bellman-Ford pour calculer le plus court chemin entre deux lieux. Vous devrez également concevoir une structure de données pour représenter les différents lieux et leurs relations.

La théorie des graphes pourra être utilisée pour modéliser les relations entre les lieux et pour implémenter les algorithmes de recherche de chemin les plus efficaces. Vous apprendrez à coder ces algorithmes et à les appliquer aux données que vous aurez collectées.

À la fin de ce projet, vous aurez acquis une expérience pratique en conception de logiciels, en programmation orientée objet et en résolution de problèmes complexes, tout en ayant implémenté des algorithmes de graphes pour résoudre un problème concret.

1. Description des sujets

DESCRIPTION du SUJET #1 – Fournir une aide décisionnelle pour visiter des Dispensaires

Dans un pays sans infrastructure routière bien développée, chaque trajet comporte des risques (piste impraticable, pont infranchissable, vols de médicaments sur le chemin, attaques de gangs armés, etc.).

Le responsable des dispensaires de son district doit pourtant acheminer des malades, des médecins/chirurgiens ou des médicaments, d'un hôpital à l'autre de façon régulière, en ambulance. Il aimerait disposer d'un logiciel d'aide à la décision qui lui permette par exemple de connaître les meilleurs chemins à emprunter selon les risques encourus, à partir des probabilités de fiabilité de chaque trajet, qu'il est capable de maintenir régulièrement. Il pourrait aussi vouloir connaître le chemin le plus court entre 2 points, en cas d'urgence vitale nécessitant une opération, par exemple.

On supposera que les trajets se font dans les 2 sens, il peut y avoir plusieurs trajets entre deux dispensaires, et dans cette application, tous sont effectués avec un véhicule (ambulance). En plus de leur fiabilité, les trajets comportent une distance (en km) et une durée moyenne (en minutes).

Chaque dispensaire possède ses propres caractéristiques : certains possèdent un bloc opératoire, d'autre une maternité, d'autre un centre de nutrition. Il faut parfois passer par des sites différents pour un trajet entre deux établissements (ex. distribuer des médicaments à plusieurs centres, passer prendre des soignants pour se rendre à une maternité cible).

La visualisation sur une carte du réseau de dispensaire et du résultat de la recherche pourrait apporter une dimension supplémentaire à l'application, en permettant aux utilisateurs de mieux comprendre la géographie du réseau et de visualiser le trajet optimal proposé par l'algorithme.

Pour ce faire, il est possible d'utiliser une librairie de cartographie comme Leaflet ou Mapbox pour afficher les stations de métro sur une carte interactive, et tracer le trajet optimal entre deux dispensaires en utilisant les coordonnées géographiques de chaque dispensaire.

Cette fonctionnalité est facultative mais recommandée aux étudiants qui maîtrisent déjà la programmation orientée objet et la théorie des graphes, et qui souhaitent explorer un peu plus les technologies de cartographie et de visualisation de données.

DESCRIPTION du SUJET #2 – Recherche de trajets dans le métro parisien

Le métro est l'un des moyens de transport les plus utilisés à Paris, mais il n'est pas toujours facile de trouver le trajet le plus rapide entre deux stations. Dans ce projet, vous allez concevoir et programmer une application permettant de rechercher le trajet optimal entre deux stations de métro en utilisant l'approche objet et la théorie des graphes.

Les données des stations et des relations entre les stations seront fournies dans des fichiers CSV (stations.csv et relations.csv) qui contiennent des informations sur les stations, telles que leur nom et leur ligne, ainsi que les relations entre les stations, telles que le temps de trajet entre deux stations adjacentes.

Vous devrez implémenter une classe pour représenter le graphe du réseau de métro, ainsi que des méthodes pour calculer le plus court chemin entre deux stations à l'aide des algorithmes de Dijkstra ou de Bellman-Ford. Vous pourrez également ajouter des fonctionnalités telles que l'affichage du trajet étape par étape ou la recherche de la station la plus proche d'une station donnée.

Ce projet vous permettra de mettre en pratique vos connaissances en programmation orientée objet, en manipulation de fichiers CSV et en théorie des graphes, tout en vous familiarisant avec le réseau de métro de Paris.

La visualisation sur une carte du réseau de métro et du résultat de la recherche pourrait apporter une dimension supplémentaire à l'application, en permettant aux utilisateurs de mieux comprendre la géographie du réseau et de visualiser le trajet optimal proposé par l'algorithme.

Pour ce faire, il est possible d’utiliser une librairie de cartographie comme Leaflet ou Mapbox pour afficher les stations de métro sur une carte interactive, et tracer le trajet optimal entre deux stations en utilisant les coordonnées géographiques de chaque station.

Cette fonctionnalité est facultative mais recommandée aux étudiants qui maîtrisent déjà la programmation orientée objet et la théorie des graphes, et qui souhaitent explorer un peu plus les technologies de cartographie et de visualisation de données.

2. Fonctionnalités demandées

Certaines des fonctionnalités sont optionnelles. Elles peuvent rapporter un bonus. Vous pouvez également ajouter vos propres fonctionnalités.

Fonctionnalités pour le sujet #1 – Réseau de dispensaires

1. Être capable de lire le réseau des dispensaires et des trajets avec leurs caractéristiques, stockés dans un fichier. Un format est proposé en annexe et un jeu d'essais initial pour les tests est fourni. Chaque équipe doit en plus construire son propre jeu d'essai, qui sera fourni le jour de la démonstration finale. Ce jeu doit comporter au moins 30 nœuds : 3/5 de maternités, 1/5 de centres de santé, 1/5 de blocs opératoires.

   Bonus : affecter des probabilités négatives pour traduire que le trajet comporte un risque létal (attaque d'un gang armé).

2. Afficher les éléments du réseau (graphe) à la demande de l'utilisateur :

   - Les dispensaires d'un type donné (ex. les maternités).
   - Le décompte des nœuds du graphe par type (par exemple : 12 maternités, 6 centres de nutrition, 12 blocs opératoires)
   - Les trajets les plus risqués du graphe (au-delà d'un seuil saisi auprès de l'utilisateur).

3. Donner le chemin le plus fiable entre 2 sites choisis par l'utilisateur.

   Bonus : afficher le graphe et saisir les 2 points sur le graphe ; afficher visuellement le chemin le plus fiable.

4. Donner les chemins les plus courts en kilomètres et en durée, entre 2 sites, sous forme de texte.

   Bonus : afficher le graphe et saisir les 2 points sur le graphe ; afficher visuellement les chemins les plus courts en distance et en durée.

5. Analyse plus poussée du graphe des dispensaires :

   - Les dispensaires proches d'un point donné ou reliés par une arête donnée (analyse 1-distance) ;
   - Dire si deux dispensaires sont reliés à p-distance (la valeur de p étant donnée par l'utilisateur) ;

- Comparer 2 dispensaires A et B : pour chacun, dire lequel est le plus OPERATOIRE que l'autre (plus apte à opérer en cas d'accidents causant un grand nombre de victimes, c'est-à-dire celui qui possède le plus de dispensaires ayant un Bloc opératoire à 2-distance), lequel est le plus NUTRITIONNEL (possède le plus de Centres de Nutrition à 2-distance) et lequel a le plus de MATERNITES (à 2-distance aussi).

6. Bonus : proposer un algorithme pour trouver des cycles hamiltoniens et donner la fiabilité associée à ces cycles. En quoi cette information pourrait être utile pour le responsable des dispensaires ?

Fonctionnalités pour le sujet #2 – Métro parisien

1. Être capable de lire le réseau des lignes de métro avec leurs caractéristiques, stockés dans un fichier. Un format est proposé en annexe et le jeu de données est fourni.

   Bonus : compléter les lignes de métro par les lignes RER en se limitant aux stations intra-muros.

2. Afficher les éléments du réseau métropolitain à la demande de l'utilisateur :

   - La liste des stations d'une ligne donnée.
   - La ou les correspondances possibles entre deux lignes.
   - Les trajets possibles entre deux stations, en indiquant le nombre de stations, le nombre de correspondances, le temps estimé du trajet (on comptera 3 min entre deux stations et 6 min par correspondance).

3. Être capable de fournir, sous forme de texte, le trajet le plus court, ou utilisant le moins de correspondances, entre deux stations.

   Bonus : afficher le graphe et saisir les 2 stations sur le graphe ; afficher visuellement les chemins le plus court en durée ou celui utilisant le moins de correspondances.

4. Être capable de fournir le trajet, le plus court en temps, entre deux stations passant par une troisième station étape.

   Bonus : afficher le graphe et saisir les 2 stations et la station étape sur le graphe ; afficher visuellement le chemin le plus courts en durée ou celui utilisant le moins de correspondances.

5. Analyse plus poussée du graphe des lignes de métro :

- Les stations proches d'une station donnée ou reliées par une arête donnée (analyse 1-distance) ;
- Dire si deux stations sont reliées à p-distance ;
- Comparer 2 stations A et B : pour chacune, dire laquelle est la plus ACCESSIBLE que l'autre (la plus proche d'une correspondance), laquelle est la plus CENTRALE (possède le plus de correspondances à p-distance, la valeur de p étant donnée par l'utilisateur) et laquelle est la plus TERMINALE (plus proche en temps d'un terminus).

6. **Bonus** : proposer un algorithme qui permet de trouver l'arbre couvrant minimum (ACM) reliant toutes les stations. En quoi ce réseau serait-il avantageux ou pas pour la RATP ? et pour les utilisateurs ?

3. Recommandations générales

Quel que soit le sujet 1 ou 2, les données seront représentées sous forme d'un graphe. Graphe chargé en mémoire dans une structure de données dynamique.

Petit rappel sur les objets

La Programmation Orientée Objet (POO) est un outil pour résoudre le problème suivant : "J'ai compris le problème et je sais le résoudre mais comment je le programme ?". En plus de maîtriser un langage de programmation, des structures de données et des algorithmes il faut structurer le code. La POO propose une méthode pour cela. Le but est de poser le problème sous forme d'objets et de leurs interactions. Cela permet d'organiser le code et de découper le problème initial (gros et compliqué) en sous problèmes (plus petits et plus simples). Il permet aussi de réutiliser du code.

La classe Graph en Python

Un graphe est un ensemble de sommets reliés par des arcs. Vous pouvez vous inspirez de l'implémentation de la classe suivante pour représenter les graphes orientés ou non-orientés.

Télécharger le fichier Graph.py mis à disposition sur Eprel. La classe Graph dans ce fichier contient :

- Un dictionnaire _edges représentant un graphe orienté et pondéré. Les clefs de _edges sont les sommets du graphe et pour un sommet n, la valeur _edges[n] est une liste de couples [(n1, w1)..., (nk, wk)] tel que chaque (ni, wi) correspond à un arc de source n, cible ni et poids wi (il s'agit donc plus précisément d'une liste d'adjacence pondérée pour le sommet n).
- Une méthode spéciale __len__() qui retourne le nombre de sommets du graphe. Cette méthode permet d'obtenir la taille (en nombre de sommets) du graphe en utilisant la fonction standard Python len() que vous connaissez déjà.
- Une méthode spéciale __iter__() qui retourne un itérateur sur les sommets du graphe. Cela permet de itérer sue les (étiquettes des) sommets du graphe avec une boucle for standard : for s in graph: ....
- Une méthode spéciale __getitem__() qui permet d'obtenir la liste d'adjacence d'un sommet n avec la syntaxe standard : graph[n].
- Une méthode add_node(u) qui ajoute le sommet au graphe si u n'est pas encore un sommet du graphe.
- Une méthode add_edge(source, target, weight = None) qui prend en entrée deux identifiants de sommets, et un poids (par défaut à None). Si les identifiants n'existent pas, les ajouter avec la méthode add_node(u).
- Une méthode bellman_ford(self, start, destination) qui retourne le plus court chemin entre deux sommets.

Remarque : Le graphe utilisé ici est un graphe non orienté. Dans le cas du sujet #2 concernant les trajets dans le métro, il y a quelques cas de stations qui ne sont desservies que dans un seul sens, sur les lignes 7bis et 10, ces stations ne seront pas prises en compte. La construction d'un arc et celle de son réciproque sont donc dissociées et font l'objet de deux appels à add_edge().

La classe Graph en Java

La classe, et les méthodes associées, sont identiques à celles décrites dans leur version en Python. À savoir :

Classe Graph

Signature de la méthode Python	Signature de la méthode Java
`def __init__(self, directed=False)`	`public Graph(boolean directed)`
`def __len__(self)`	`public int size()`
`def __iter__(self)`	`public Set<String> nodes()`
`def __getitem__(self, node)`	`public List<Pair<String, Integer>> edges(String node)`
`def add_node(self, s)`	`public void addNode(String node)`
`def add_edge(self, source, target, weight=None)`	`public void addEdge(String source, String target, int weight)`
`def __str__(self)`	`public String toString()`
`def bellman_ford(self, start, destination)`	`public Pair<Map<String, Integer>, Map<String, String>> bellmanFord(String start, String destination)`

Remarque : La classe Pair en Java est une classe générique fournie par la bibliothèque standard de Java qui permet de stocker deux valeurs de types différents dans un seul objet. Cette classe est souvent utilisée pour retourner plusieurs valeurs à partir d'une méthode.

4. Livrables et évaluation

Les livrables attendus sont :

1. Le code source de l'application. Une attention particulière sera portée sur la qualité du code (arborescence des dossiers, modularité, commentaires, test unitaires, …) ;
2. Un rapport présentant la problématique, les solutions mises en oeuvre et les résultats obtenus. Une attention particulière sera portée sur la qualité du rapport (rédaction, orthographe, mise en page, illustrations, …) ;
3. Une présentation orale de 15 min s'appuyant sur un visuel Powerpoint et une courte démonstration de l'application. Une attention particulière sera portée sur la qualité de la présentation (synthèse, lisibilité, orthographe, charte graphique, …) ;

Les dates de rendu des livrables seront données par vos enseignants.

L'évaluation finale tiendra compte des parties pratique (50%), écrite (25%) et orale (25%).

Annexes

Disponibilité des données

Sujet #1 – Réseau de dispensaires : Les données du jeu de test sont disponibles sous forme de deux fichiers au format CSV :

1. liste-successeurs.csv : donne la liste des successeurs d'un dispensaire ;
2. liste-adjacence.csv : donne la liste d'adjacence des dispensaires. Pour chaque dispensaire et relation les informations concernant le type du dispensaire, la fiabilité, la distance ainsi que la durée du trajet, sont indiquées.

Sujet #2 – Métro parisien : Les données du jeu de test sont disponibles sous forme de deux fichiers au format CSV :

1. stations.csv : donne la liste des stations de métro. Pour chaque station sont indiqués : l'identifiant numérique de la station, le numéro de la ligne, le fait que la station soit un terminus ou non et le nom de la station.
2. relations.csv : ce fichier contient la liste des relations entre deux stations et le temps de trajet (en secondes) entre ces stations.