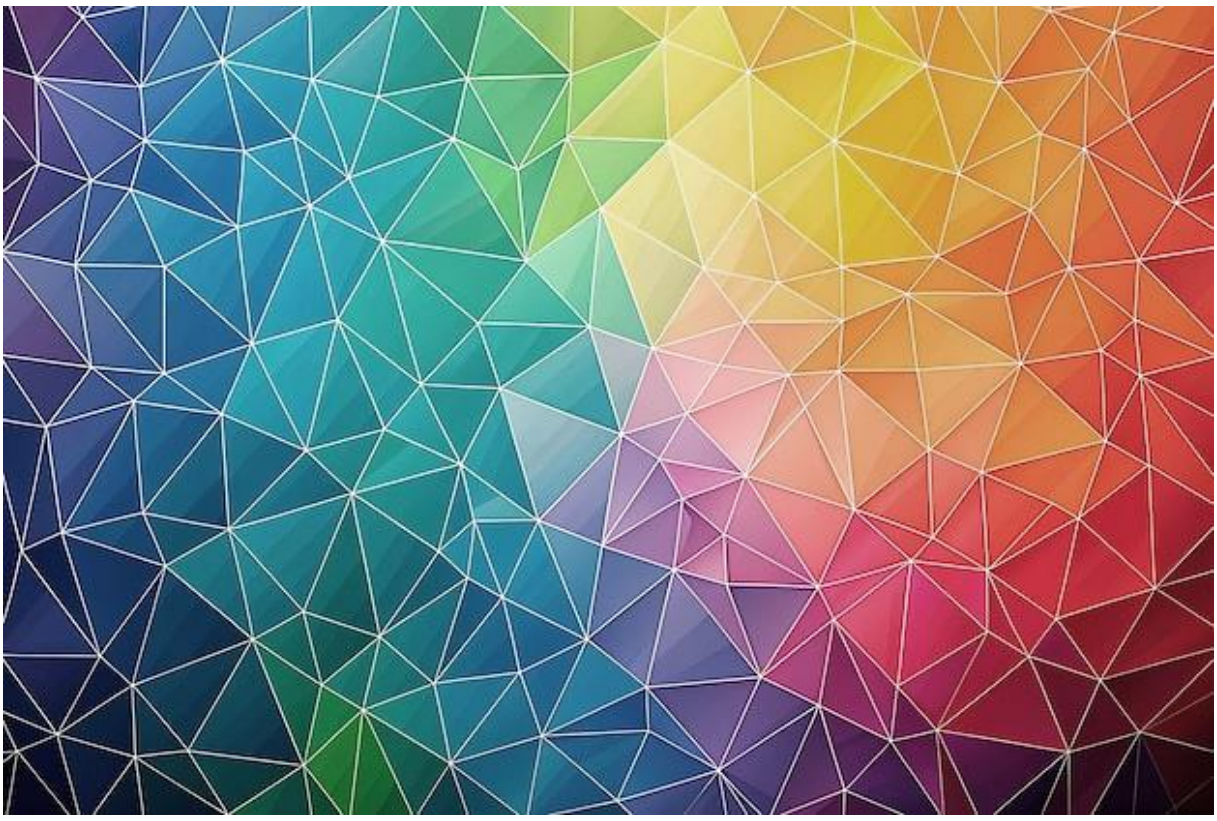


SAÉ 2.02
Exploration algorithmique d'un problème

BUT Informatique

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL
iut
CRÉTEIL-VITRY

Introduction

Le système Vélib' en libre-service est un modèle de mobilité urbaine qui offre aux usagers la possibilité de louer des vélos, y compris des vélos électriques, au sein de la ville de Paris et de ses alentours. Ce réseau de stations, essentiel pour la gestion de la mobilité durable en milieu urbain, repose sur une répartition géographique optimale des stations et sur un approvisionnement énergétique efficace pour ses vélos électriques. Dans un contexte de transition énergétique et d'urbanisation croissante, l'optimisation de ces deux aspects — la répartition des stations et l'alimentation électrique — constitue un enjeu majeur pour garantir à la fois une efficacité logistique et une gestion durable des ressources.

Enjeux de la répartition des stations

L'une des premières problématiques est la **répartition géographique des stations**. Dans une grande métropole comme Paris, il est essentiel de s'assurer que chaque quartier est suffisamment desservi par un nombre adéquat de stations. Une bonne répartition permet de garantir une couverture maximale de la zone sans surcharger certaines zones et sans laisser des zones trop éloignées des stations, ce qui impacte négativement l'accessibilité. Ce problème est en lien direct avec des considérations de **connectivité** et de **densité**, qui peuvent être modélisées mathématiquement par des graphes. En utilisant la **triangulation de Delaunay**, il est possible d'étudier la connectivité du réseau en représentant les stations comme des nœuds d'un graphe et les connexions entre elles comme des arêtes, tout en garantissant une bonne répartition géométrique et en évitant les concentrations excessives ou les zones de desserte insuffisantes. Ce modèle géométrique permettra d'évaluer si le réseau respecte les critères d'une **répartition aléatoire homogène**, comme celle imposée par une **distribution de Poisson**, en vue de garantir une couverture optimale.

Enjeux liés à l'alimentation électrique

Le second enjeu majeur du projet concerne l'**alimentation électrique** des stations Vélib' électriques. Avec l'augmentation de la flotte de vélos électriques, une gestion optimale de l'alimentation devient cruciale pour garantir le bon fonctionnement du système. L'objectif est de minimiser les coûts d'installation et de gestion des infrastructures électriques tout en maintenant une alimentation fiable et performante pour l'ensemble des stations. Le défi ici est de connecter l'ensemble des stations à un réseau électrique centralisé, **en optimisant la longueur des câbles** nécessaires pour relier les stations, tout en maintenant la fiabilité du réseau. Ce problème peut être modélisé par un problème d'**arbre couvrant minimal** (minimum spanning tree, MST), où les stations sont représentées comme des nœuds d'un graphe et les câbles comme des arêtes avec des poids proportionnels à la distance entre les stations. L'objectif est de trouver l'arbre couvrant minimal qui permet de relier toutes les stations, ce qui minimise la longueur totale de câblage, réduisant ainsi les coûts d'infrastructure tout en assurant la continuité du service.

Objectifs du projet

Vous devrez aborder ces deux problématiques complexes sous un angle technique et d'ingénierie en utilisant des outils issus de la théorie des graphes et de la géométrie computationnelle. Plus précisément, le projet sera divisé en deux parties :

1. Analyse de la répartition des stations – La première partie du projet visera à analyser la répartition géographique des stations Vélib' en utilisant la **triangulation de Delaunay** pour évaluer la connectivité du réseau et la qualité de la couverture des zones urbaines. L'objectif est de déterminer si la répartition suit un modèle de **distribution de Poisson** et de proposer des pistes d'amélioration pour optimiser la desserte du territoire.
2. Optimisation de l'alimentation électrique – La seconde partie du projet se concentrera sur l'optimisation du réseau d'alimentation électrique des stations Vélib' électriques. Les étudiants devront modéliser le problème de câblage comme un problème d'**arbre couvrant minimal** (MST) et utiliser des algorithmes comme **Kruskal** ou **Prim** pour minimiser la longueur des câbles nécessaires pour connecter toutes les stations, tout en respectant les contraintes techniques liées à la distribution d'énergie.

Ces deux parties seront abordées dans une démarche d'optimisation combinatoire, où les solutions techniques devront être justifiées par des considérations pratiques (réduction des coûts, couverture optimale, efficacité énergétique) et par une évaluation des impacts sur l'accessibilité et la performance du réseau.

Graphes et algorithmes

Les graphes sont des structures de données puissantes et polyvalentes largement utilisées dans divers domaines. Leur utilisation est omniprésente dans l'informatique, les réseaux, la logistique, la planification et bien d'autres domaines. Un graphe est composé de nœuds (ou sommets) et d'arêtes qui relient ces nœuds. Cette représentation visuelle permet de modéliser des relations complexes entre différentes entités.

Les algorithmes sur les graphes sont essentiels pour résoudre des problèmes tels que la recherche de chemins optimaux, la détection de cycles, la gestion des réseaux sociaux, la planification des itinéraires, et bien plus encore. Grâce à leur flexibilité et à leur capacité à représenter des connexions entre des éléments, les graphes offrent une approche efficace pour résoudre une variété de problèmes, contribuant ainsi à la résolution de nombreux défis du monde réel.

Quelques éléments sur les graphes

Comme dit précédemment, un graphe est une structure de données que l'on peut utiliser pour modéliser la hiérarchie et les relations entre les objets. Il se compose d'un ensemble de **nœuds** (ou sommets) et d'un ensemble d'**arêtes**. Les sommets représentent des objets individuels, tandis que les arêtes illustrent les relations entre ces objets.

Remarque : les termes « nœud » et « sommet » sont souvent utilisés de manière interchangeable. Ici, nous avons choisi d'utiliser le terme « nœud », mais il a la même signification que le terme « sommet ».

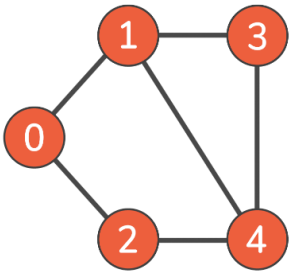
L'information contenue dans un nœud peut être de différente nature en fonction du contexte de l'application. Généralement, les nœuds d'un graphe sont repérés (identifiés) par un numéro. L'**ordre d'un graphe** est égal au nombre de nœuds.

Si chaque arête d'un graphe illustre une connexion bidirectionnelle, nous appelons ce graphe « **graphe non orienté** ». En revanche, si vous pouvez parcourir chaque arête dans une seule direction, le graphe est dit « **orienté** ».

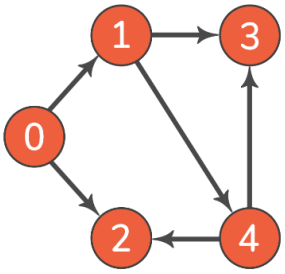
Tous les nœuds d'un graphe n'ont pas besoin d'être connectés aux autres. Si vous pouvez accéder à chaque nœud à partir de n'importe quel autre nœud dans un graphe, nous appelons ce graphe « **graphe connexe** ». Mais parfois, il y a certains nœuds auxquels vous ne pouvez accéder à partir d'aucun autre nœud du graphe, dans ce cas le graphe est dit « **non connexe** ». L'idée fausse, la plus répandue, est que chaque graphe doit être connecté, mais la réalité est que ce n'est pas le cas – en fait, un graphe peut ne contenir aucune arête, juste des nœuds.

Un **poids** peut être affecté à chaque arête d'un graphe. Ce poids représente généralement un coût, celui qu'il faut « dépenser » pour traverser cette arête afin de passer d'un nœud à l'autre. Les graphes avec des poids attribués à leurs arêtes sont appelés **« graphes pondérés »**. Un graphe peut être orienté et pondéré à la fois.

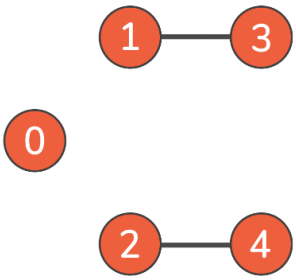
Les représentations visuelles de différents graphes sont données ci-dessous.



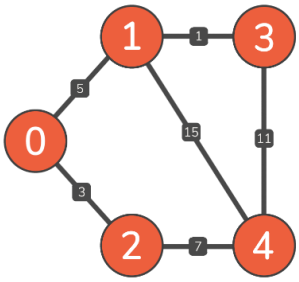
Graphe non orienté à 5 nœuds



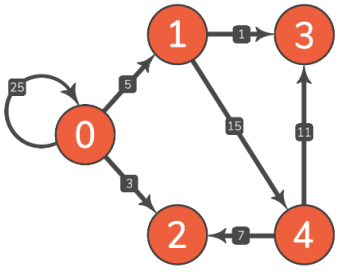
Graphe orienté



Graphe non connexe



Graphe pondéré



Graphe pondéré et orienté

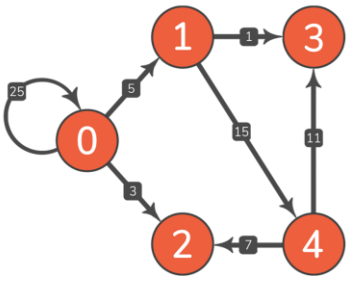
Représentations informatiques d'un graphe

La **représentation informatique d'un graphe** offre une diversité de méthodes, chacune adaptée à des contextes spécifiques selon les besoins et les contraintes du problème à résoudre. L'une des approches les plus courantes est la **matrice d'adjacence**, où une matrice bidimensionnelle est utilisée pour indiquer les relations entre les nœuds du graphe. Cette représentation est particulièrement efficace pour les graphes denses mais peut devenir coûteuse en termes de mémoire pour les graphes plus épars. Une alternative est la **liste d'adjacence**, où chaque nœud est associé à une liste de ses voisins directs. Cette méthode est plus économique pour les graphes moins denses. La représentation la plus simple, mais pas la plus efficace, est la **liste d'arêtes**. Elle est particulièrement efficace pour représenter des graphes creux, où le nombre d'arêtes est significativement inférieur au nombre total d'arêtes possible.

Pratiquement, les structures de données informatiques telles que les **ensembles**, les **dictionnaires**, ou les **tableaux** peuvent être employées pour représenter des graphes, offrant des avantages spécifiques en fonction du contexte algorithmique. Les graphes orientés peuvent être modélisés avec des matrices spécifiques, tandis que les graphes pondérés peuvent nécessiter des attributs supplémentaires pour représenter les poids des arêtes.

La diversité des méthodes de représentation permet aux programmeurs et aux algorithmes de choisir celle qui correspond le mieux aux exigences particulières du problème à traiter. Cette flexibilité est essentielle dans la conception de solutions informatiques efficaces et adaptées à une grande variété de situations.

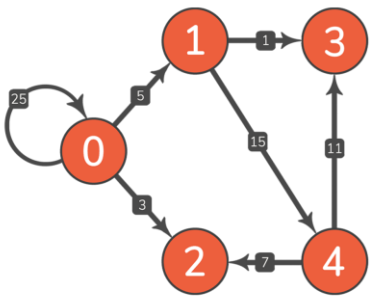
Les représentations possibles d'un graphe (orienté et pondéré) sont illustrées ci-après.



1 ligne = 1 arête

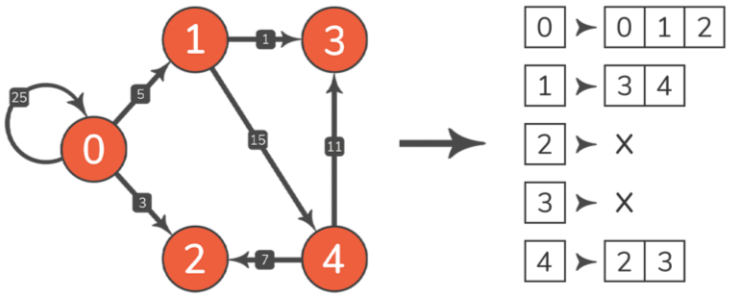
nœud 1	nœud 2	poids
0	0	25
0	1	5
0	2	3
1	3	1
1	4	15
4	2	7
4	3	11

Représentation d'un graphe orienté pondéré par une liste d'arêtes.



	0	1	2	3	4
0	25	5	3	0	0
1	0	0	0	1	15
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	7	11	0

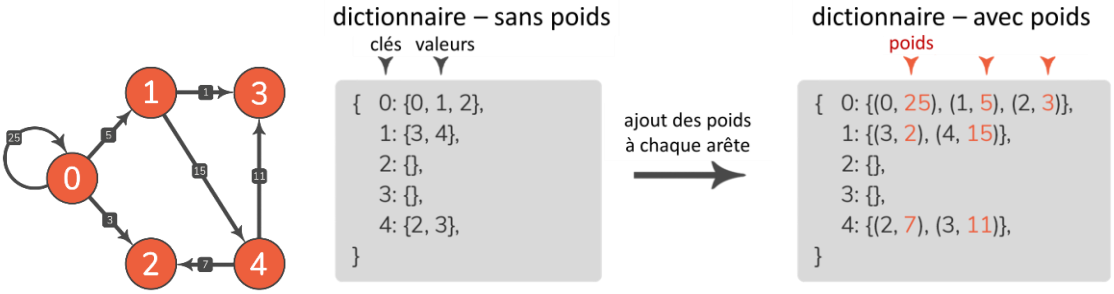
Représentation d'un graphe orienté pondéré par une matrice d'adjacence.



Représentation d'un graphe orienté pondéré par une liste d'adjacence (sans mentionner les poids)

La dernière représentation illustrée, celle faisant appel à des listes d'adjacence, est une représentation de graphe particulièrement facile à implémenter en **Python** en raison de la flexibilité des structures de données disponibles dans ce langage.

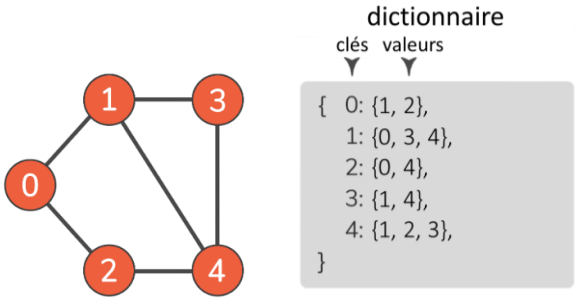
Par exemple, l'utilisation d'un dictionnaire est une approche classique pour implémenter la liste d'adjacence en Python. Dans ce contexte, les clés du dictionnaire représentent les nœuds du graphe, et les valeurs associées sont des listes (ou d'autres structures de données appropriées) contenant les voisins de chaque nœud. Cette approche capitalise sur la capacité des dictionnaires à associer des valeurs à des clés de manière efficace.



Exemple d'utilisation d'un dictionnaire afin d'implémenter une liste d'adjacence d'un graphe orienté non pondéré (à gauche) ou pondéré (à droite).

On peut voir sur la figure ci-dessus, que si le graphe est pondéré, chaque entrée de la liste d'adjacence peut être un tuple (ou une liste) contenant le numéro du nœud voisin et le poids de l'arête.

Une représentation équivalente est possible pour un graphe non orienté et non pondéré.



Exemple d'utilisation d'un dictionnaire afin d'implémenter une liste d'adjacence d'un graphe non orienté et non pondéré.

Mise en œuvre

Pour chacune des problématiques exposées précédemment, il existe un ou plusieurs algorithmes de résolution. Dans cette SAE, vous allez devoir mettre en œuvre ces algorithmes en utilisant le langage Python et les modules spécialisés à cette fin.

Différentes structures de données

Dans le cadre de cette SAE, les données des stations Vélib' sont manipulées à travers plusieurs structures, chacune adaptée aux besoins spécifiques des différentes étapes de l'analyse.

Ces différentes structures de données permettent une manipulation flexible des informations tout au long du processus d'analyse, depuis la collecte initiale jusqu'à l'optimisation algorithmique.

Format JSON Initial

Les données des stations Vélib' sont initialement collectées au format **JSON**. Ce format est choisi pour sa lisibilité et sa facilité de manipulation, permettant de stocker des informations structurées telles que les coordonnées géographiques, la capacité des stations, et d'autres attributs pertinents. Le JSON facilite également l'échange de données entre différents systèmes et outils d'analyse.

Triangulation de Delaunay avec SciPy

Pour modéliser la répartition des stations sous forme de graphe, les coordonnées géographiques sont extraites et passées à la fonction `Delaunay` de la bibliothèque `scipy.spatial`. Cette fonction génère une triangulation de Delaunay, qui est une structure de données sous forme de **simplex**. Chaque simplex représente un triangle formé par trois stations, optimisant ainsi la répartition spatiale des connexions entre les stations. Cette structure est particulièrement adaptée pour des analyses géométriques et topologiques, permettant de visualiser les relations spatiales entre les stations.

Visualisation avec Folium

La triangulation de Delaunay est ensuite utilisée pour créer une visualisation cartographique à l'aide de la bibliothèque `folium`. Les simplex sont représentés sous forme de polygones sur la carte, illustrant les connexions entre les stations. Cette visualisation permet d'identifier rapidement les zones de forte densité de stations et d'analyser la répartition géographique des connexions.

Liste d'Adjacence

Pour répondre aux besoins algorithmiques, notamment l'analyse de la connectivité et l'optimisation des réseaux, la structure de données est transformée en une **liste d'adjacence**. Cette structure représente le graphe sous forme de listes de voisins pour chaque station, facilitant ainsi les opérations de parcours et de recherche. La liste d'adjacence permet de modéliser efficacement les relations entre

les stations et de mettre en œuvre des algorithmes d'optimisation tels que la recherche de l'arbre couvrant minimum.

Le choix de la structure de données représentant le graphe sera donc celui de la liste d'adjacence implémentée à l'aide d'un dictionnaire. Les graphes pourront être orientés ou non, pondérés ou non. Un nœud est identifié soit par un indice allant de 0 à $n-1$ où n est l'ordre du graphe, soit par un caractère, ou un jeu de caractères alphabétiques "A", "B", ..., "AA", "AB", ...

Remarque : Il est possible que vous ailliez à nommer les nœuds par un identifiant (numérique ou alphanumérique) propre à la station Vélib'.

Voici une rédaction détaillée de la partie concernant la collecte des données au format JSON à partir de l'Open Data de Vélib' Métropole. Cette section explique comment obtenir et préparer les données nécessaires pour l'analyse des stations Vélib'.

Collecte des Données

Objectif : Obtenir les informations statiques sur les stations Vélib' à partir de l'Open Data fourni par Vélib' Métropole. Ces données sont essentielles pour l'analyse de la répartition des stations et l'optimisation du réseau.

Étape 1 : Accès aux données en Open Data

Source de données

Les données sont accessibles via l'URL suivante : Velib Metropole Open Data. Cette URL fournit des informations statiques sur les stations Vélib' au format JSON.

Format des données

Les données sont structurées sous la forme d'un objet JSON contenant une liste de stations. Chaque station est décrite par les attributs suivants :

- `capacity` : Nombre de bornettes disponibles dans la station.
- `lat` : Latitude de la station au format WGS84.
- `lon` : Longitude de la station au format WGS84.
- `name` : Nom de la station.
- `station_id` : Identifiant unique de la station au sein du service Vélib' Métropole.

Étape 2 : Collecte des données

Téléchargement des données

Utilisez un script Python pour télécharger les données JSON depuis l'URL fournie. Assurez-vous d'avoir une connexion Internet active pour accéder à l'URL.

Vérification de l'intégrité des données

Après le téléchargement, vérifiez que le fichier JSON contient bien les informations attendues. Ouvrez le fichier et assurez-vous que les champs `capacity`, `lat`, `lon`, `name`, et `station_id` sont présents pour chaque station.

Cette collecte de données constitue la base de l'analyse ultérieure, permettant de modéliser la répartition des stations et d'optimiser le réseau Vélib'.

Génération du graphe des stations

Objectif : Modéliser la répartition des stations Vélib' sous forme de graphe en utilisant la triangulation de Delaunay. Cette méthode permet d'optimiser les connexions entre les stations et de visualiser leur répartition géographique.

Étape 1 : Extraction des coordonnées

Extrayez les coordonnées géographiques (latitude et longitude) des stations à partir des données JSON préalablement collectées. Ces coordonnées sont essentielles pour effectuer la triangulation.

Étape 2 : Triangulation de Delaunay avec SciPy

Génération de la triangulation

Utilisez la fonction `Delaunay` de la bibliothèque `scipy.spatial` pour générer la triangulation à partir des coordonnées des stations. La triangulation de Delaunay maximise le plus petit angle des triangles formés, optimisant ainsi la répartition spatiale des connexions.

Analyse des simplex

La triangulation résultante est composée de simplex, qui sont des ensembles de points formant des triangles. Chaque simplex représente une connexion optimale entre trois stations.

Étape 3 : Visualisation sur une carte

Création de la carte

Utilisez la bibliothèque `folium` pour créer une carte centrée sur Paris. Cette carte servira de support pour visualiser la triangulation de Delaunay.

Ajout des stations et des triangles

Ajoutez des marqueurs pour chaque station et dessinez les triangles formés par la triangulation de Delaunay sous forme de polygones sur la carte.

Ajout d'une légende

Ajoutez une légende à la carte pour expliquer les couleurs et les formes utilisées, facilitant ainsi la compréhension de la visualisation.

Étape 4 : Transcription en liste d'adjacence

Initialisation de la liste d'adjacence

Créez une liste d'adjacence pour représenter les connexions entre les stations sous forme de graphe. Chaque station est un nœud, et les connexions sont représentées par des arêtes.

Construction de la liste d'adjacence

Parcourez les simplex de la triangulation de Delaunay pour ajouter les connexions entre les stations dans la liste d'adjacence.

Affichage de la liste d'adjacence

Affichez la liste d'adjacence pour vérifier les connexions entre les stations. Cette structure est essentielle pour les analyses algorithmiques ultérieures.

Ces étapes permettent de modéliser efficacement la répartition des stations Vélib' et de préparer les données pour des analyses plus approfondies, telles que l'optimisation du réseau.

Analyse du réseau de stations Vélib'

Objectif : Étudier l'optimalité de la distribution des stations Vélib' en mesurant un indice de répartition par station, puis optimiser l'interconnexion électrique entre les stations.

Étape 1 : Mesure de l'indice de répartition

Définition de l'indice de répartition

L'indice de répartition est une mesure qui évalue l'efficacité de la distribution des stations en fonction de leur connectivité et de leur capacité. Un indice proche de 0 indique une bonne répartition, tandis qu'un indice négatif suggère une sous-distribution et un indice positif une sur-distribution.

Calcul de l'indice de répartition

Définissez une fonction pour calculer l'indice de répartition pour chaque station. Cette fonction prend en compte le nombre de voisins (connectivité) et la capacité de la station.

Visualisation des indices de répartition

Utilisez un dégradé de couleurs pour représenter les indices de répartition sur la carte. Les stations avec un indice élevé seront colorées en vert, tandis que celles avec un indice faible seront en rouge.

Étape 2 : Optimisation de l'interconnexion électrique

Algorithme de l'Arbre Couvrant Minimum

Utilisez l'algorithme de Kruskal pour trouver l'arbre couvrant minimum (ACM), qui minimise les coûts de raccordement électrique entre les stations.

Visualisation de l'Arbre Couvrant Minimum

Ajoutez les arêtes de l'arbre couvrant minimum sur la carte pour visualiser l'interconnexion optimale entre les stations.

Ces étapes permettent d'analyser l'efficacité de la distribution des stations Vélib' et d'optimiser leur interconnexion électrique, tout en fournissant des visualisations claires et informatives.

Étude complémentaire

Amélioration de la représentation de l'Indice de répartition

Objectif : Améliorer la visualisation de l'indice de répartition des stations Vélib' en colorant les régions définies par les **cellules de Voronoï** associées à chaque station. Cette approche permet de mieux identifier les zones nécessitant des optimisations.

Relation entre Delaunay et Voronoï

Les triangulations de Delaunay et les diagrammes de Voronoï sont des concepts complémentaires en géométrie computationnelle :

- **Triangulation de Delaunay** : Elle divise un ensemble de points en triangles de manière à maximiser le plus petit angle, optimisant ainsi la répartition spatiale des connexions entre les points.
- **Diagramme de Voronoï** : Il partitionne un plan en régions convexes autour de chaque point, de sorte que chaque région contienne tous les points du plan plus proches de ce point que de tout autre. Les cellules de Voronoï sont duales des triangles de Delaunay, ce qui signifie que chaque arête d'un triangle de Delaunay est perpendiculaire à une arête d'une cellule de Voronoï.

En utilisant ces deux structures ensemble, nous pouvons obtenir une représentation visuelle riche qui montre à la fois les connexions optimales (Delaunay) et les zones d'influence (Voronoï) des stations Vélib'.

Étape 1 : Calcul du diagramme de Voronoï

Utilisez la fonction `Voronoi` de la bibliothèque `scipy.spatial` pour calculer le diagramme de Voronoï à partir des coordonnées des stations.

Étape 2 : Visualisation des cellules de Voronoï sur la carte

Création de la Carte

Utilisez `folium` pour créer une carte centrée sur Paris, qui servira de support pour visualiser les cellules de Voronoï.

Ajout des Cellules de Voronoï

Ajoutez les cellules de Voronoï à la carte en les colorant en fonction de l'indice de répartition calculé pour chaque station. Utilisez un dégradé de couleurs pour représenter les indices, par exemple :

- Vert pour un indice proche de zéro (répartition optimale).
- Rouge pour un indice négatif (sous-représentation des stations).
- Bleu pour un indice positif (sur-représentation des stations).

Cette approche enrichit la compréhension de la répartition des stations Vélib' et permet d'identifier visuellement les zones nécessitant des améliorations pour optimiser le réseau.

Annexes

Livrables

Les livrables sont les suivants :

1. Code source de la partie « Mise en œuvre ». Rendre le dossier archivé du code et des données utiles à l'algorithme.
2. Code source de la partie « Étude complémentaire ».
3. Rapport résumant les différentes études menées sur les graphes dans cette SAE. Ce rapport devra présenter et expliquer chaque algorithme mis en œuvre.

Une séance sera consacrée à une présentation de votre part devant vos enseignants. La durée et les modalités de la soutenance orale seront précisés par vos enseignants.

Les dates de rendus des livrables seront également précisées par vos enseignants.