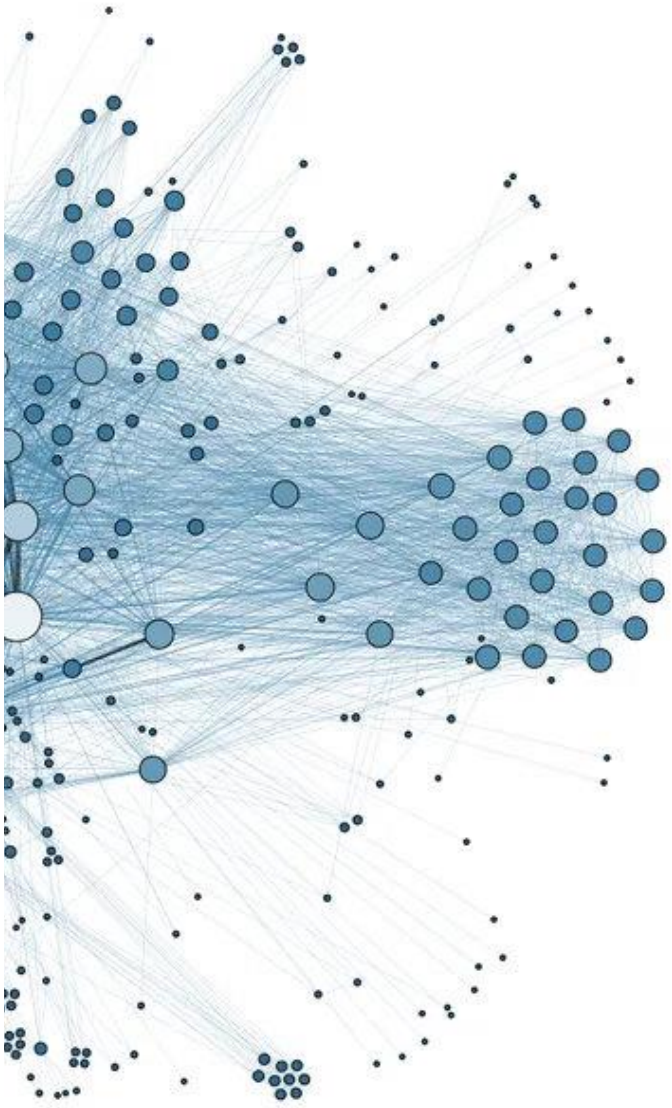
# Rapport : Tables de routage



29 AVRIL **2024**

**Groupe 3 14h45 TD1**
**VALVERDE Victorien**
**YAPO Ebeguy**
**ASSIS Hugo**

# Préambule

## Termes Techniques

Il convient d'expliquer les termes techniques de ce projet avant d'en expliquer son code et son contenu :

1. Graphe
    a. Il s'agit d'un réseau où des entités, comme des nœuds, communiquent au travers de liens. Ces liens représentent par où l'information peut circuler et avec quel coût. Le coût peut être en termes de temps, de distance, de monnaie etc.
2. Matrice d'adjacence
    a. Pour représenter un graphe il existe de nombreuses manières, mais la plus utile dans notre contexte est la matrice d'adjacence. Il s'agit d'une matrice, donc d'un tableau en deux dimensions de même taille, qui contient les informations nécessaires pour définir un graphe. Pour lire une matrice d'adjacence, on se réfère à la ligne et à la colonne.
    b. Il existe des types de matrice d'adjacence, telle qu'une matrice supérieure ou inférieure. On peut faire le choix d'exclure les diagonales.
3. Nœud
    a. Il s'agit d'un composant fondamental d'un graphe. En effet, c'est ce qui permet de représenter des entités qui interagissent entre elles, comme des ordinateurs, des commutateurs, des stations de métro etc.
    b. Dans notre contexte, les nœuds possèdent un nom, un tier, une liste de voisins et une table de routage vers d'autres nœuds.
4. Backbone
    a. Dans ce contexte, le graphe suit une structure particulière. En effet chaque nœud possède un tier, ou un niveau, qui le distingue par les coûts de ses liens. Plus le tier est proche de 0, plus le nœud est rapide.
    b. Le Backbone, 10 nœuds, représentent le squelette du réseau. Ce sont eux qui font transiter l'information le plus rapidement de tous les nœuds et qui relient les différentes parties du réseau.
5. Transit
    a. Les nœuds de transit sont moins rapides que les nœuds du backbone mais ils permettent de rendre le réseau plus connecté et de créer des chemins parfois plus courts.

6. Nœud Régulier
    a. Ces nœuds sont les plus nombreux. Ils ne servent pas d'intermédiaire contrairement aux nœuds de tier I et de tier II, ce qui explique pourquoi leurs liens sont les plus coûteux. C'est eux qu'il faut choisir quand on veut obtenir des routes pertinentes quand on teste les fonctions de plus court chemin.
7. Plus court chemin
    a. Problème classique de la théorie des graphes. Il s'agit de trouver le chemin le plus court reliant deux nœuds. Dans la vie courante, l'équivalent serait de trouver le chemin allant de notre position actuelle à une destination en prenant en compte des paramètres telle que le temps, le trafic etc.

# Décisions

## Structures de données

Nous avons choisi de représenter le graphe et les nœuds à l'aide de classes, ils ont ainsi des attributs et surtout des méthodes qui leurs sont propres. Cependant, il ne suffit pas d'une classe pour représenter un graphe composé d'autant de nœuds, c'est pourquoi nous le représentons à l'aide d'une matrice d'adjacence supérieure, qui est en fait un array Numpy en deux dimensions. Puisque le graphe est non-orienté, stocker les informations dans la partie inférieure serait redondant et complexe à gérer pour la génération des nœuds. Chaque cellule représente une connexion : Si un lien existe, la cellule contiendra sa valeur, sinon elle contiendra 0 si le lien n'existe pas, et l'infini si le lien est interdit (la diagonale).

Pour l'affichage de ce graphe, nous avons été confrontés à un problème : Il y'a beaucoup trop de nœuds pour que le terminal puisse l'afficher. Il existe deux solutions.
La première consiste à exporter dans le dossier ./spreadsheets le graphe dans un tableur. Cela à ses avantages comme le fait de pouvoir tout visualiser et d'appliquer des couleurs. Nous appliquons les couleurs aux lignes et aux colonnes qui correspondent à la fin d'une plage de nœuds, basé sur leur niveau. Il s'agit de la méthode préférée.
La seconde consiste à utiliser le module Panda et afficher une partie de la matrice. Il est déconseillé d'utiliser cette méthode car elle dépend de la police du terminal et elle n'est pas configurée dans le code pour être utilisée clairement. Néanmoins elle existe.

Pour les tables de routages, nous utilisons l'algorithme de Dijkstra pour trouver le plus court chemin. En effet, nous ne traitons pas de liens négatifs entre les nœuds et le graphe est non-orienté, sinon nous aurions recours à d'autres algorithmes, plus complexes et adaptés.
Une fois les tables de routage calculées, nous reconstituons le chemin à l'aide d'une fonction récursive qui s'arrêtent une fois la destination atteinte (Ou si l'utilisateur à rentrer un nœud qui n'existe pas).