TP d'informatique n°15
Comment aller au don du sang ? (ou ailleurs)

PCSI 2023 – 2024

I Objectifs :

- utiliser l'algorithme de Dijkstra pour trouver la sortie/le chemin le plus court jusqu'à un sommet ;
- améliorer le temps de recherche grâce à l'utilisation d'une heuristique : algorithme A^*.

Décommenter rapidement plusieurs lignes code : À plusieurs endroits, il sera demandé de dé-commenter du code fourni pour le compléter. Avec un bon éditeur, il suffit de sélectionner les lignes, de faire un clic droit et de faire "commenter/décommenter" ou équivalent.

II Introduction

Dans ce TP, nous allons travailler sur la carte routière de la ville de Nancy pour essayer de trouver le chemin le plus court depuis le lycée Henri Poincaré jusqu'à l'établissement français du sang de Nancy [1]. Vous pouvez voir le plan et les points concernés sur la figure 1.

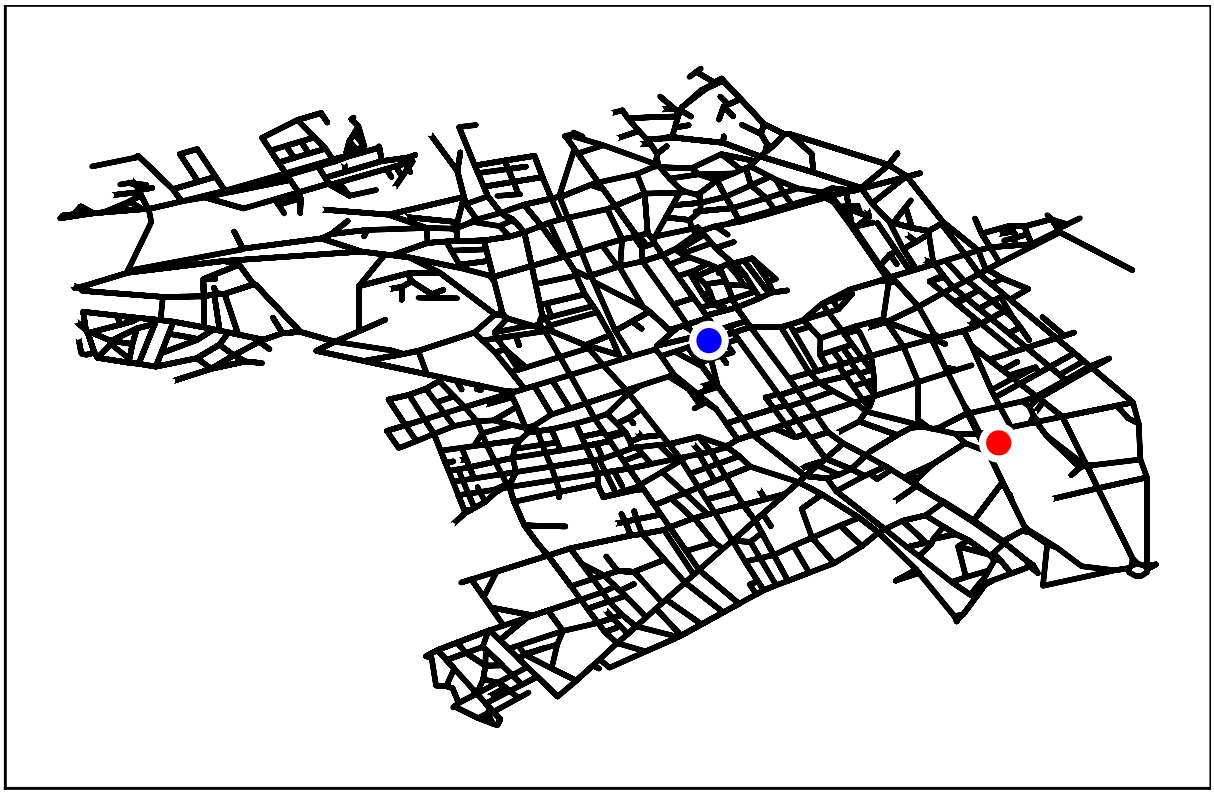

FIGURE 1 – Plan routier de la ville de Nancy ; point de départ et d'arrivée.

1. cette destination est un choix arbitraire et si vous préférez une autre destination, vous n'aurez pas grand chose à changer. Si vous le souhaitez, j'ai aussi les données de la ville de Metz, de Rabat ou de Marrakech. Par contre, les exemples et les coordonnées du point de départ et d'arrivée seront à adapter.

III Mise en œuvre

Dans un premier temps, il vous faut récupérer les données sur internet. Trois fichiers sont à votre disposition :

- Nancy_data_node qui contient les information sur les sommets sous la forme de trois colonnes numéro du sommet abscisse du sommet ordonnée [2]
- Nancy_data_edges qui contient les informations sur les arêtes sous la forme de 5 colonnes :
  1. indice du sommet de départ (noté u) ;
  2. indice du sommet d'arrivée (noté v) ;
  3. longueur de l'arête (en m) ;
  4. vitesse moyenne sur l'arête (en km/h) ;
  5. temps de trajet moyen sur l'arête (en s).
- Un fichier fichier_etudiant.py qui contient un squelette de code, certaines parties étant commentées.

Exercice n°1 Ouvrir les fichiers Nancy_data_node et Nancy_data_edges pour observer la structure des données. Dans le fichier .py, au niveau de la partie «Q1», compléter l'utilisation de la fonction loadtxt pour charger les données sur les nœuds et les arêtes. On pourra utiliser l'argument optionnel skiprows pour ignorer la ou les premières lignes d'en-tête (regarder l'aide de la fonction au besoin). Expliquer ensuite le code de la fonction `affiche_ville_lent` (sauf la ligne avec plt.axis("equal") qui rend simplement le repère orthonormé).

Exercice n°2 Rappeler ce qu'est une liste d'adjacence. On dispose grâce à la question 1 de la liste des arêtes, mais ce n'est pas très pratique pour l'implémentation des algorithmes. On rappelle que le graphe est ici orienté (à cause des sens uniques, fréquents à Nancy). Dé-commentez le code dans la partie Q2 et complétez le pour créer une liste d'adjacence, ainsi que des «dictionnaires» qui, à chaque couple de sommet (u,v) associe les informations de longueur, temps, vitesse moyenne.

Ne vous préoccupez pas du code `DFS_cree_liste_x_et_y`. La fonction `affiche_ville` permet ensuite un affichage plus rapide que celle du début.

Exercice n°3 Pour aller de Poinca jusqu'au don du sang, encore faut-il trouver les sommets correspondant dans nos listes/matrices. Décommentez et complétez le code de la partie Q3 pour déterminer les sommets les plus proches de notre point de départ et de notre destination. Les coordonnées déjà rentrées pour le lycée et la destination ont été trouvée sur internet (coordonnées GPS).

Exercice n°4 Nous allons ensuite vouloir tracer des routes pour pouvoir afficher les chemins que nous allons trouver. Décommentez le code de la partie 4 et complétez le.

Exercice n°5 Le code de l'algorithme de Dijkstra est déjà écrit ... en partie ! Décommentez la partie Q5 et complétez le code pour avoir un algorithme qui fonctionne pour renvoyer la liste des prédécesseurs, puis qui trace la route la moins longue (en terme de distance à parcourir). Vous pouvez comparer votre réponse avec la figure 2 page 3.

2. En fait, il s'agit non pas réellement de abscisse et ordonnée mais de longitude et latitude en degré. Dans votre cours de physique, vous avez manipulé les coordonnées sphériques où θ était la colatitude ($\pi/2$ moins la latitude) et φ était la longitude. Ainsi, pour avoir des abscisses et des ordonnées en mètre, il faudrait faire à chaque fois une manipulation du genre $x' = R_T \cos(y) \times x$ et $y' = R_T \times y$ en convertissant d'abord x et y en radian.

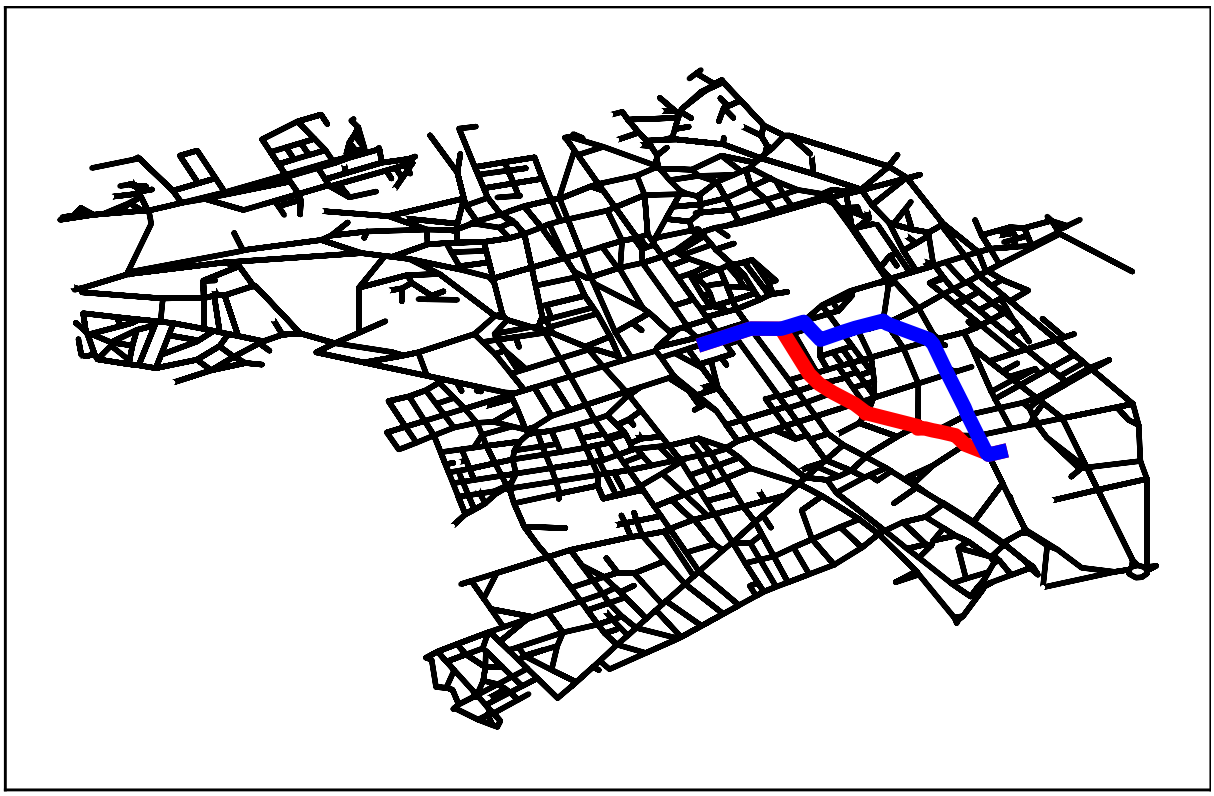

FIGURE 2 – Chemin les plus court en distance (rouge, en «bas») et en temps (bleu en «haut»).

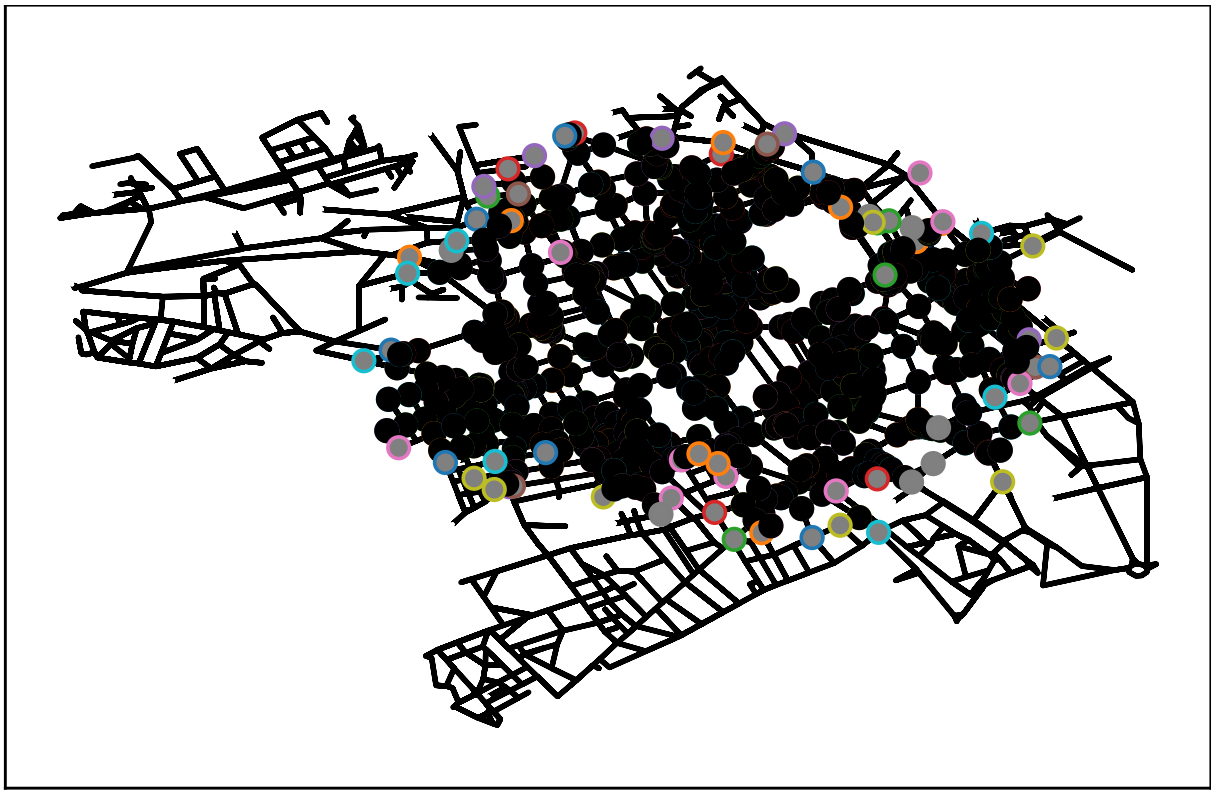

FIGURE 3 – Nœuds explorées par Dijkstra

Exercice n°6 Décommentez la partie Q6 et trouver le chemin le plus court en terme de temps. Vous pouvez comparer votre réponse avec la figure 2 page 3.

Exercice n°7 Décommentez la partie Q7 pour afficher les nœuds visités par l'algorithme de Dijkstra. Peut-être y-a-t'il quelque chose à compléter/corriger. Vous pouvez comparer votre réponse avec la figure 2 page 3.

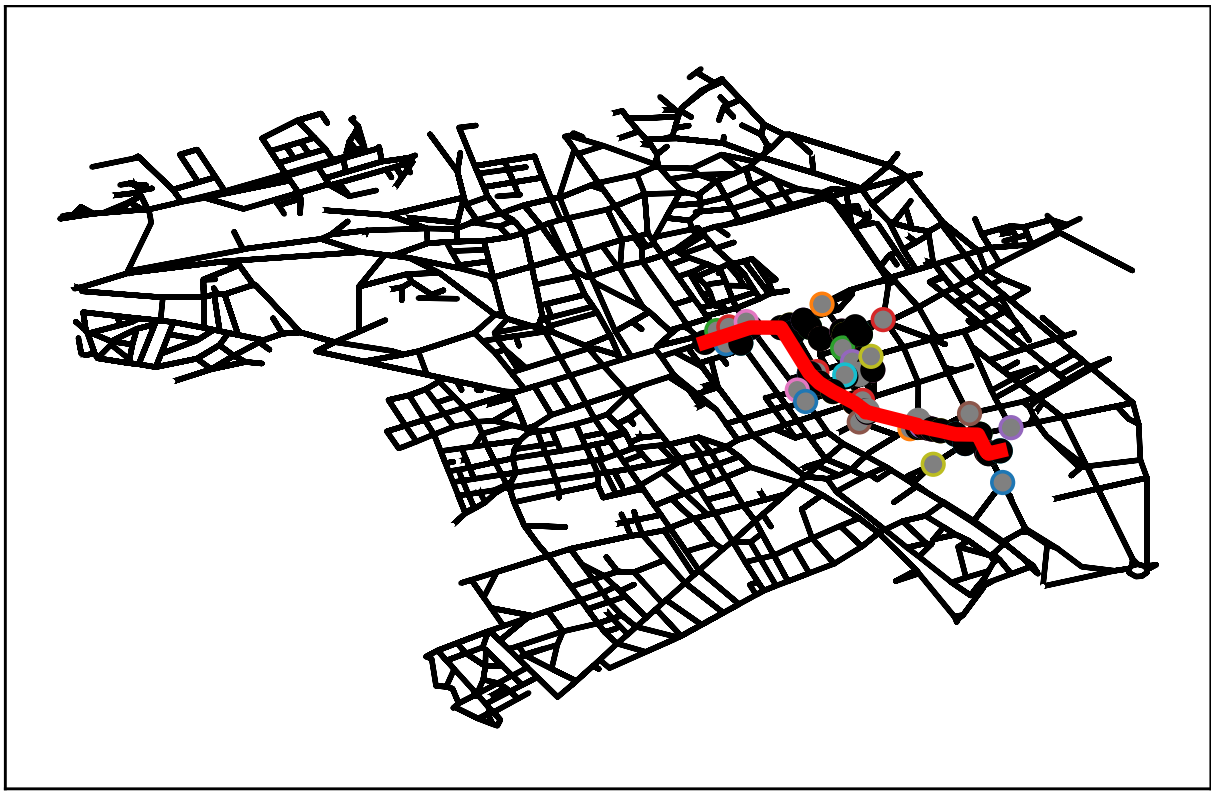

FIGURE 4 – Nœuds explorées par A^*

Exercice n°8 Décommentez la partie Q8 et complétez le code pour implémenter l'algorithme A^* en utilisant comme heuristique pour le chemin restant la distance à vol d'oiseau (norme usuelle de $\mathbb{R}^2$).

Exercice n°9 (Nécessite un peu d'autonomie) Pour réaliser un affichage rapide de la ville, il est préférable de ne pas afficher les arêtes une à une comme cela a été fait avec la fonction `affiche_ville_lent` , mais d'avoir simplement une liste de coordonnées x et une selon y. Pour réaliser cela, on suggère un parcours en profondeur dit «physique», c'est-à-dire ou l'on ajoute les sommets lorsque l'on avance, mais aussi lorsque l'on recule. Ainsi, chaque sommet sera ajoutée plusieurs fois, mais on n'aura bien que des arêtes possibles. Il faut faire attention à ajouter des sommets voisins dans la liste même s'ils sont déjà marqués comme visité pour tracer les routes, mais il ne faut alors pas les ajouter à la liste/pile/file de sommets à traiter.

IV Pour la prochaine fois.

Hélas, le prochain TP sera … l'an prochain ! Le travail pour la prochaine fois est donc «simplement» de tout réviser pour faire bonne impression à votre futur enseignant d'informatique et lui montrer que vous avez été bien formé en première année !