Ophélie Valin
Clémence Bocquet
BUT 1 informatique – Orléans
Groupe 4A
2023-2024

SAé 2.02 Exploitation algorithmique d’un problème

Table of Contents

Lien GitHub

https://github.com/clemencebc1/SAE_2.02_graphes

Réalisation

L’ensemble du travail demandé a été réalisé sur la période de 8 semaines consacrée à la réalisation de cette SAE 2.02 Exploration Algorithmique d’un problème. Chacune des fonctions proposées a pu être testée, à la fois dans son fonctionnement et son efficacité. La docstring est fournie et certains commentaires sont ajoutés lorsque cela est nécessaire, afin de rendre le code plus lisible et compréhensible. Afin de simplifier le temps d’exécution, nous avons utilisé, tout le long de cette SAé, le fichier ‘data_100.txt’ et ‘data_1000.txt’. Les tests se réaliseront avec le fichier contenant 100 lignes.

Toutefois, au cours de la production, des difficultés ont pu apparaître. En effet, en réalisant certaines fonctions, nous avons remarqué que le temps d’exécution était bien trop long. Les fonctions ‘éloignement_max’ et ‘centre_hollywood’ appellaient la fonction ‘centralité’ appellant elle-même la fonction ‘distance’. De ce fait, nous avons mis en place différentes solutions. En premier temps, nous avons tenté de changer de structures de données. Cependant, le changement n’a pas permis de gagner suffisamment de temps d’exécution. De cette façon, nous avons fait en sorte d’alléger la compléxité de ‘distance’. Pareillement, une fonction ‘dicoAtteint’, non demandée dans le sujet, a été ajouté. Celle-ci permet de renvoyer un dictionnaire de distance avec pour clé les acteurs et pour valeurs la distance à partir d’un acteur u passé en paramètre. Ainsi, nous avons résolu le problème du temps d’exécution.

De plus, nous avons commencé par réaliser l’échauffement. Néanmoins, celui a été chronophage et des problèmes d’organisation ont pu se faire voir. Nous avons fait en sorte de produire de quoi fonctionner sans se concentrer, en premier temps, sur la compléxité de celle-ci. De la sorte, nous avons pu réaliser l’ensemble du travail.

Organisation

Lors de cette Saé, nous avons réparti les tâches afin de s'organiser et de produire le travail demandé de manière efficace. Néanmois, certaines d'entre elles ont été faites en binôme. Par exemple, l'échauffement, le rapport ou encore l'étude de l'efficacité ont été étudié de manière commune. Ainsi, ci-dessous se trouve un tableau décrivant l'organisation au sein de notre binôme.

Clémence	Ophélie
Echauffement requetes.py	
Efficacité	
Interraction application sur terminal oracle.py	
Rapport	
Préparation soutenance	
Collaborateurs en communs requetes.py	Collaborateurs proches requetes.py
Qui est au centre d'Hollywood requetes.py	
Une petite famille requetes.py	
Bonus requetes.py	

Détails des questions

6.2 Collaborateurs communs

L'ensemble des collaborateurs communs peut se décrire de cette façon : pour deux nœuds différents u et v, on veut l'ensemble des acteurs qui sont à la fois dans le voisinage de u et dans le voisinage de v.

Formellement, nous pouvons l'écrire de cette manière : pour un graphe G=(V,E) où V est l'ensemble des sommets et E l'ensemble des arêtes de ce graphe, on veut $\{k \mid (u,v,k) \in V^3 \wedge u \neq v \neq k \wedge k \in N(u) \wedge k \in N(v)\}$. En théorie des ensembles, nous pouvons utiliser l'intersection des voisins de A avec les voisins de B, c'est-à-dire tous les voisins à la fois dans A et dans B.

La complexité de cette fonction est : O(1). Nous faisons l'intersection des nœuds dans u et dans v.

6.3 Collaborateurs proches

Le code fournit possède une complexité $O(n^3)$ dans le pire des cas. Nous commençons d'abord par vérifier que l'acteur u est présent en tant que sommet dans le graphe. Ainsi, pour une distance k et pour chacun des collaborateurs d'un acteur à cette distance, on regarde si collaborateur voisin est dans la liste des collaborateurs proches. L'algorithme utilisé est le parcours en largeur Breadth-First Search. À partir d'un sommet donné, on parcourt ses succeceurs puis les succeceurs non explorés de ceux-ci etc.. Pour déterminer la distance k avec un autre acteur, nous pouvons utiliser cet algorithme tant que l'acteur n'est pas exploré, on ajoute 1 à la distance et regarde les succeceurs des succeceurs.

La complexité d'une fonction en ré-utilisant la fonction 'est_proche' est $O(n^5)$. Cette approche semble donc très peu intéressante. En modifiant la fonction fournie, nous pouvons obtenir une complexité de $O(n^2)$.

6.4 Qui est le centre d'Hollywood

Plus la distance est élevée, plus la centralité est faible. Par conséquent, nous cherchons donc pour un acteur u, sa plus faible centralité avec un autre. Pour cela, il est possible d'utiliser un algorithme BFS afin de trouver la plus grande distance possible. Pareillement, nous pouvons utiliser une fonction Networkx 'closeness_centrality(G)' prenant un graphe en paramètre et renvoyant un dictionniaire de la centralité des acteurs. Néanmoins, celle-ci n'a pas été utilisé.

6.5 Une petite famille

Pour déterminer la plus grande distance entre deux acteurs du graphe, nous testons sur un ensemble de couple d'acteurs leur distance. Si elle est supérieure à la plus grande distance déjà enregistrée, alors on stocke la nouvelle valeur. Ainsi, pour le jeu de données 'data_100.txt', la plus grande distance retenue entre deux acteurs est 9. Un acteur u et un acteur v, 9 acteurs les séparent.

Évaluation expérimentale

Différentes implémentations :

Il est possible de calculer la distance entre 2 sommets une unique fois et de l'enregistrer afin de pouvoir réutiliser les données ou de calculer la distance dont on a besoin seulement lorsqu'elle est nécessaire. Dans ce cas, il existe plusieurs méthodes qui pourraient implémenter ces structures de calcul de distance.

Méthode fictive : Enregistrement des parcours déjà effectués pour limiter répétition.

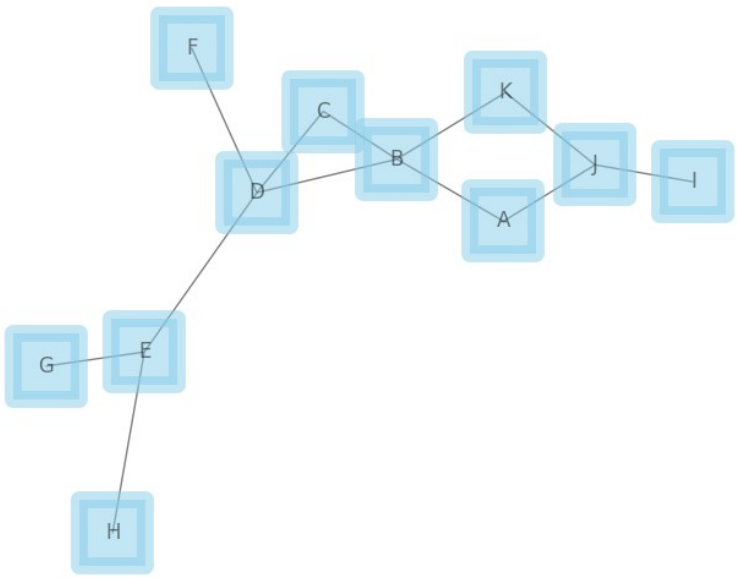
Mémoriser les distances une bonne fois pour toute dans un dictionnaire avec pour clé un couple d'acteurs(trices) et la distance qui les sépare. Ce dictionnaire pourrait même en accompagner un deuxième qui se focaliserait sur le couple d'acteurs(trices) en clé et la liste des acteurs les rejoignant selon le plus court chemin.

Cette méthode nécessiterait donc l'implémentation de potentiellement 2 autres fonctions qui seraient centrées sur l'ajout de données dans leur propre dictionnaire et appelés lorsqu'un parcours a été réalisé. Bien entendu, ces fonctions devront être en capacité de déterminer si les informations doivent être ajoutées ou ignorées selon le contenu déjà présent dans le dictionnaire.

Cela permettrait de demander les données enregistrées au début de chaque fonction de calcul de distances, afin de limiter les parcours inutiles et ne pas réitérer des calculs déjà effectués précédemment. Ainsi, si la distance entre 2 nœuds a déjà été enregistrée dans le dictionnaire des distances, il sera possible de se servir de cette donnée de départ pour rechercher la distance avec un autre acteur.

Exemple : Imaginons le graphe suivant…

Figure 1



Si l’on connaît la distance entre le couple (A,E) = 3, alors on peut réutiliser cette distance en partant de E pour calculer la distance entre le couple (A,H) sans avoir à parcourir de nouveau tout le graphe pour calculer une distance qui à déjà été partiellement trouvée. Ainsi on obtiendrait : distance (A,E) + distance (E,H). Seule la distance (E,H) aurait besoin d’être calculé.

Méthode réalisée :

En premier temps, nous avions réalisé une méthode se basant sur un ensemble, à chaque tour de la boucle nous regardions si l’acteur v a été ajouté. Si oui, nous retournions donc le rang i représentant la distance. Toutefois, la boucle permettant de conserver l’indice peut être évitée. En effet, il est possible, avec un dictionnaire, de garder en mémoire la distance avec un autre nœud. La clé serait donc l’acteur et la valeur sa distance avec l’acteur u. Nous avons donc supprimé celle-ci. Ainsi la complexité est passée de $O(n^3)$ à $O(n^2)$.

Plus loin :

Pour produire réaliser les programmes attendues, nous pouvons utiliser des fonctions Networkx afin de gagner en temps d’exécution et, potentiellement, en complexité.