Rapport : SAE 2.02 Exploration algorithmique d'un problème

LEPAGE Nicolas
CARRAT Killian

Présentation :

Cette SAÉ a été réalisée pour apprendre à optimiser notre code. On a pu voir que la complexité de nos algorithmes doit être la plus petite possible pour qu'on puisse les réutiliser. Pour cela, nous sommes passés par la création d'un graphe à l'aide de fichiers de données en les transformant.

Réalisation :

L'importation de données est le point le plus important. Pour cela, on a créé une fonction qui importe un fichier JSON dans un graphe pour pouvoir ensuite l'utiliser pour les autres fonctions. Pour continuer, on a créé une fonction qui nous permet de voir les acteurs voisins en commun à deux acteurs. On a ensuite implémenté des fonctions qui ont pour but de regarder la distance entre des acteurs. Pour chercher les acteurs étant les plus proches de l'ensemble des autres acteurs, on a créé les fonctions qui donnent la distance maximale qu'un acteur a avec celui qui est le plus loin de lui, pour ensuite voir quel acteur est le plus au centre. Et pour finir, on a implémenté une fonction qui donne la plus grande distance entre 2 acteurs. Celle-ci prend un acteur au hasard dans le graphe, regarde sa centralité et renvoie un acteur qui est à un des bords du graphe, donc on effectue de nouveau la centralité pour trouver sa distance avec un autre bord du graphe, et donc la plus grande distance entre deux acteurs du graphe.

Approfondissement :

La notion liée aux collaborateurs communs est l'intersection de voisins dans un graphe. Il peut être difficile de chercher les collaborateurs si les acteurs ont beaucoup de voisins. L'algorithme caché derrière "collaborateurs_proches" est un algorithme de parcours en largeur. On commence par initialiser la fonction au sommet 'u', puis on fait le BFS (parcours en largeur), puis à chaque nouveau voisin, on appelle la fonction "collaborateurs_communs" pour regarder si les acteurs s'en rapprochent, et on itère sur chaque nouveau voisin jusqu'à trouver le sommet voulu.Nous avons commencer avec la fonction distance_naive qui est une fonction distance avec une très mauvaise complexité $O(n^4)$ nous l'avons créé en utilisant collaborateurs_proches dans une boucle.Puis nous avons cherché à l'améliorer pour obtenir distance en supprimant cette appelle de fonction, qui nous semblait inutil.Nous l'avons remplacer par une partie du code de collaborateurs_proches qui

va chercher les voisins de chaque node parcourut. En faisant ceci nous avons réussi à obtenir une complexité en $O(n^3)$.

Pour la centralité d'un acteur, on cherche le diamètre du graphe, c'est-à-dire la plus grande distance que l'acteur a avec un autre acteur.
Dans notre cas, pour “eloignement_max”, nous ne trouvons pas une valeur inférieure à 6. Ceci peut être dû au jeux de données dont le nombre de personne n’est pas assez conséquent ou bien que nos programmes ne font pas le chemin le plus direct. Pour cette fonction, nous avons fait plusieurs fois le programme pour éviter les erreurs avec le graphe qui n’est pas entièrement connexe et donc permet de minimiser la chance de ne pas fonctionner.

Difficultés :

Pour cette SAÉ, nous avons au maximum essayé de trouver des solutions pour rendre notre code efficace. Cependant, nous n'avons pas pu réduire le temps d'exécution de nos algorithmes pour que la fonction "centre_hollywood" prenne un temps décent pour son exécution avec un jeu de données élevé. On pourrait les améliorer en gardant en mémoire les distances qui les séparent d'autres acteurs pour ne pas refaire plusieurs fois le même chemin. Nous avons aussi remarqué que le graphe avait quelques imperfections, comme le fait que des petits graphes apparaissent, ce qui rend le graphe principal non connexe. Cela peut donc compliquer la recherche des informations dans le graphe.

Travail d'équipe :

Nous nous sommes partagé le travail. Killian a réalisé les fonctions "json_vers_nx" et "eloignement_max", et Nicolas a implémenté les fonctions "collaborateurs_communs", "est_proche", "distance", "centralite" et "centre_hollywood". Killian a créé l'application "Oracle.py". Pour ce qui est de l'optimisation des algorithmes, nous nous sommes tous les deux mis dessus pour essayer de rendre plus efficace l'ensemble de nos programmes. Par rapport au premier programme, les fonctions "json_vers_nx", "centralite" et "eloignement_max" ont pu être optimisées. Nous avons pu réaliser des optimisations du programme "centralite" en utilisant le principe d'inondation, en regardant l'ensemble des voisins des voisins, pour enfin trouver le dernier sommet et donner la distance à laquelle il est le plus éloigné. Pour continuer, on a optimisé la création du graphe en utilisant des ensembles pour créer les nœuds. Et pour finir, "eloignement_max" a finalement utilisé "centralite" pour rendre son programme plus efficace, à la place de "distance".

Nous avons effectué des tests sur nos fonctions en utilisant les fonctions de NetworkX pour vérifier nos résultats, sauf sur "centre_hollywood", qui prend trop de temps et a été testé seulement sur le jeu de données réduit (data_100.txt). Nous avons remarqué la différence d'efficacité des fonctions NetworkX.