Mouad
Zouadi
14B

Rapport SAE 2.02 : Exploration Algorithmique d'un Problème

Pour cette SAE 2.02, on nous demande d'implémenter différentes requêtes, de les tester, et de créer une application autour de ces requêtes via un jeu de données représentant les collaborations entre acteurs et actrices d'Hollywood.

J'ai réalisé ce travail individuellement et réussi à implémenter toutes les requêtes, y compris celles en bonus. J'ai également effectué les tests demandés et créé l'application. En plus de cela, j'ai ajouté quelques fonctionnalités supplémentaires comme la possibilité de choisir son jeu de données ou d'afficher le graphe des collaborations d'un acteur. J'ai aussi créé une interface graphique pour l'application en utilisant l'outil Tkinter. Les éventuelles améliorations possibles seraient sûrement au niveau de l'application : la rendre plus esthétique, ajouter des informations, et améliorer le temps d'exécution des requêtes. Bien que j'aie essayé de rendre les fonctions les plus optimales possibles, il reste une marge d'amélioration. J'ai beaucoup aimé ajouter des fonctionnalités, une interface graphique et améliorer cette application, et j'aurais pu continuer à le faire si j'avais eu plus de temps à disposition.

Partie 6 :

6.1 Pour convertir le jeu de données JSON en un graphe Networkx, j'ai d'abord écrit une fonction qui transforme les données de txt à JSON, puis une fonction qui parcourt les données pour créer des arêtes entre les acteurs ayant collaboré dans un film.

6.2 Cette notion peut être exprimée en termes de théorie des graphes comme l'intersection des ensembles des voisins des deux acteurs dans le graphe. Une borne inférieure sur le temps nécessaire à l'exécution de cette fonction est la complexité O(n), où n est le nombre total de films. Le temps d'exécution moyen est de :

6.3 L'algorithme utilisé est une variante de la recherche en largeur (BFS). Pour déterminer si un acteur se trouve à distance k d'un autre

acteur, on peut utiliser cette fonction pour trouver les acteurs à distance k et vérifier si l'acteur cible est parmi eux, ce qui se fait dans la fonction est_proche. La complexité de cet algorithme est O(N + E), où N est le nombre de sommets dans le graphe et E est le nombre d'arêtes.

6.4 La notion de théorie des graphes utilisée ici est celle de centralité. Pour réaliser cette fonction, j'ai utilisé la fonction de centralité, avec une complexité de O(N+E). Pour le centre d'Hollywood, j'ai créé un dictionnaire avec la centralité par acteur et utilisé le minimum de ce dictionnaire, avec une complexité de $O(N^2+NE)$.

6.5 Pour réaliser cette fonction, j'ai parcouru chaque acteur, calculé sa centralité, puis cherché la centralité maximale.Étant donnée que ma fonction a une complexité de $O(N^2+NE)$ que ma machine prend 0.00003s à faire la centralité d'un acteur et qu'il il y a 610626 acteurs dans le jeu de données cela me prendrait approximativement $0.00003*610626^2 = 11185923.3563$ secondes soit plus de 4 mois pour exécuter la fonction sur "data.txt" .

6.6 Pour réaliser cette fonction, j'ai suivi la même approche que pour la fonction centre_hollywood, mais cette fois, j'ai parcouru seulement les acteurs de s. Pour renvoyer un sous-graphe à partir d'un ensemble, j'ai utilisé la fonction networkX subgraph.

Partie 7 :

En ce qui concerne l'efficacité, j'ai essayé d'obtenir la complexité minimale pour chaque fonction réalisée. Vu la quantité de données à analyser, la complexité est absolument primordiale. J'avais un gros problème de complexité avec la fonction centre d'Hollywood, qui utilise la centralité. J'avais une double boucle avec un appel de la fonction distance à chaque itération, ce qui faisait exploser la complexité. J'ai donc réfléchi et trouvé deux solutions potentielles : ne pas utiliser la fonction distance dans la fonction centralité et plutôt parcourir tous les voisins de l'acteur, puis les voisins de ses voisins, etc. Cette solution a été très efficace et m'a permis de passer d'une complexité de $O(n^2+e)$ à $O(n+e)$ et de plus de 45 secondes de temps d'exécution moyen à 4

secondes. L'autre moyen suggéré dans le sujet est de modifier la fonction distance en utilisant un algorithme de pré-calcul. J'ai donc trois fonctions de calcul de distance que nous allons comparer afin de savoir laquelle est la meilleure dans chaque situation. Nous avons donc distance_naive, qui est un parcours en largeur (BFS) avec une complexité de O(n+e), ensuite shortest_path_length, qui est une fonction de NetworkX aussi en O(n+e) mais plus optimisée que distance_naive, et enfin le pré-calcul, qui est très coûteux en $O(n^3)$, intéressant pour calculer seulement une distance entre deux acteurs mais potentiellement intéressant dans un contexte très particulier. Nous allons donc comparer leur complexité et leur temps d'exécution moyen sur 1000 tentatives sur le fichier "data_2.txt" pour le pré-calcul (car absolument intestable sur les autres) et sur "data_1000.txt" sur les autres pour une distance entre deux acteurs et pour la centralité d'un acteur. Pour la distance entre deux acteurs, voici les résultats :

```
Temps d'execution moyen de distance: 0.000000056 s
Temps d'execution moyen de pre-calcul: 0.000692771 s
```

Pour la centralité d'un acteur

```
Temps d'execution moyen de centralite_dis: 0.0033629900000000023 s
Temps d'execution moyen de centralite_pre_calcul: 0.07565131000000007 s
```

Donc l'hypothèse est bien confirmée : le pré-calcul pour un simple calcul de distance est inefficace, mais pour la centralité où il est utilisé un grand nombre de fois il est plus intéressant. Maintenant, comparons ma fonction de centralité qui visite les voisins des voisins de l'acteur avec celle utilisant le pré-calcul(temps calculé par acteur pour pouvoir tester les 2 sur différents jeux de données).

```
Temps d'execution moyen de centralite_pre_calcul: 0.0007154766355140197 s
Temps d'execution moyen de centralite: 0.00003 s
```

La fonction qui visite les voisins de l'acteur reste la plus efficace et surtout pré calcul est inutilisable dans ce contexte car beaucoup trop lent même si on ne lance qu'une seule il serait meilleur dans un autre contexte ou on a u nombre très élevé de requête par exemple et qu'on

ne le lance qu’une seule fois ce qui n’est pas le cas ici pour une application.

Conclusion et Réflexion Personnelle

J'ai beaucoup apprécié réaliser cette SAE, malgré que je l’aie faite seul. Ce que j’ai préféré, c’était chercher la meilleure complexité pour chaque algorithme et ajouter toujours plus de fonctionnalités à l'application. Cette expérience m'a permis de mobiliser et de renforcer mes connaissances en algorithmique, en théorie des graphes et en développement d'applications Python. J'ai appris à analyser un problème complexe, à comparer et à implémenter différents algorithmes et à optimiser leur performance. Travailler sur ce projet a également renforcé ma capacité à travailler de manière autonome et à gérer efficacement mon temps.

Cette expérience m'a fait apprécier davantage le domaine de l'algorithmique et de la théorie des graphes. J'ai particulièrement aimé chercher la meilleure complexité pour chaque algorithme et ajouter toujours plus de fonctionnalités à l'application. Cela m'a incité à explorer plus profondément ces sujets fascinants et à envisager des applications pratiques pour ces connaissances dans le futur.