Rapport

Projet SAE – À la conquête d'Hollywood
BUT Informatique – 2024-2025

KELES Okan 14A
PAWLOWICZ Valentin 14A

Objectif général

Ce fichier Java contient les fonctions principales permettant de modéliser et analyser un réseau de collaborations entre acteurs et actrices à Hollywood sous la forme d'un graphe. Chaque acteur est représenté comme un sommet, et une arête est ajoutée entre deux acteurs s'ils ont joué dans un même film.

Analyse détaillée des fonctions par question

Question 3.1 :

les méthodes :

- getActorCollaborationMap()
- lireFilms()
- faireMap()
- trouvecollab()
- getGraph()

Le but :

Transformer les données d'un fichier en JSON en graph

Le fonctionnement :

Le main lance la méthode getActorCollaborationMap() qui va appeler la méthode lireFilm() qui est un filereader de fichier GSON puis qui va ensuite appeler la methode faireMap() qui va faire un dictionnaire avec le nom des acteur comme clé et un ensemble (pour éviter les doublons) qui contient les acteur avec qui il a travaillé comme valeur pour ce faire faireMap() va appeler trouvecollab()qui pour un acteur donné va trouver tout les acteur avec qui il a travaillé . Enfin après avoir reçu le dictionnaire le main appel getGraph() pour transformer le dictionnaire en graph

la complexité :

$O(n \times k^2)$, avec n = nombre de films, k = nombre moyen d'acteurs par film.

Question 3.2 :

Les méthodes :

- getCommonCollaborators(String actor1, String actor2)

Le but :

Trouver les collaborateurs communs entre deux acteurs.

Le fonctionnement :

- Récupère la liste des voisins de actor1 et actor2 dans le graphe (i.e., leurs collaborateurs directs).
- Calcule l'intersection des deux ensembles pour obtenir les collaborateurs en commun.

La complexité :

O(d1 + d2), où d1 et d2 sont les degrés (le nombre de voisins) des deux acteurs.

Question 3.3 :

Les méthodes :

- getActorsWithinDistance(String actor, int k)
- getDistance(String actor1, String actor2)

Le but :

- Trouver tous les acteurs à une distance inférieure ou égale à k d'un acteur donné.
- Calculer la distance minimale entre deux acteurs.

Le fonctionnement :

- Les deux fonctions utilisent l'algorithme BFS (parcours en largeur).
- getActorsWithinDistance explore tous les acteurs jusqu'à une profondeur k.
- getDistance explore les voisins de plus en plus loin jusqu'à atteindre l'acteur cible, puis retourne la distance correspondante.

La complexité :

O(V + E) dans le pire cas, mais en pratique limitée par la distance k ou par l'arrêt anticipé (si l'acteur cible est trouvé rapidement).

Question 3.4 :

Les méthodes :

- getActorEccentricity(String actor)
- getGraphCenter()

Le but :

- Calculer la centralité (excentricité) d'un acteur : distance maximale entre lui et un autre acteur.
- Trouver l'acteur le plus central (ayant l'excentricité minimale).

Le fonctionnement :

- getActorEccentricity effectue un BFS depuis l'acteur donné pour enregistrer la plus grande distance rencontrée.
- getGraphCenter applique cette fonction à tous les acteurs et garde celui avec la valeur minimale.

La complexité :

- getActorEccentricity : O(V + E)
- getGraphCenter : O(V × (V + E)), car on répète un BFS pour chaque acteur.

Question 3.5 :

Les méthodes :

- getGraphDiameter()

Le but :

Trouver la distance maximale entre deux acteurs quelconques du graphe, c'est-à-dire le diamètre du graphe.

Le fonctionnement :

- Pour chaque acteur, calcule son excentricité via getActorEccentricity.
- Le diamètre est la valeur maximale obtenue.

La complexité :

O(V × (V + E)), comme pour le centre, car on effectue un BFS depuis chaque sommet.