Bastien Monet
Alexandre Laurin

SAE Algo - A la conquête d'Hollywood

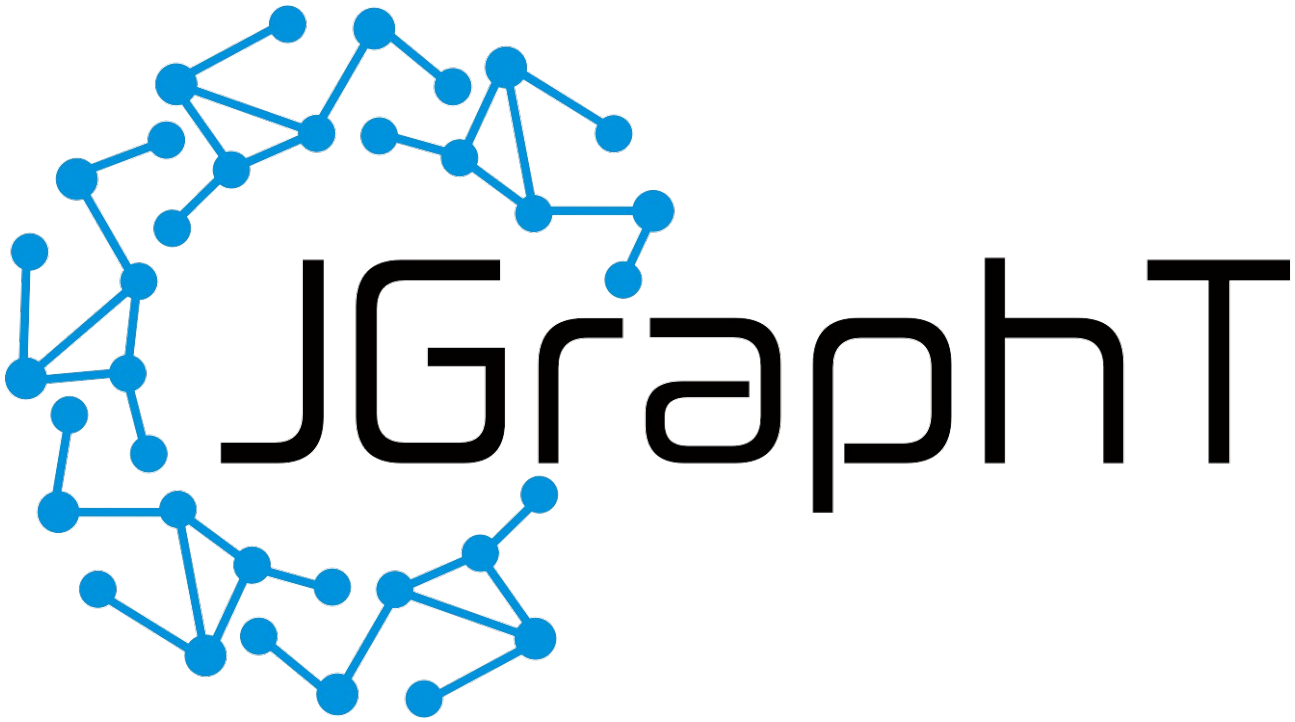


Table des matières

Introduction

Durant cette SAE il nous était demander d'extraire les donnée d'un JSON en java pour les convertir en un graphe dessinable grâce a la fonction donné DOTExporter. Puis réaliser les fonction demandées à l'aide de la librairie jGrapht

Gestion du travail

Toute la SAE a été réaliser sur un dépot git avec l'utilisation de Maven pour résoudre les dépendances manquante en ajoutant des tests à chaque fonctions comme appris en qualité de développement

Extraction des fichiers en un graphe

Pour l'extraction des donnée nous avons choisi la solution Gson de java nous permettant de transformer ligne par ligne le JSON en objet java utilisable. Pour éviter les caractères inutile, une méthode nettoyer existe. Enfin ajouterAuGraphe transforme les objet java en graphe

Fonctions réalisées

nous avons réaliser toute les fonctions, elles se situent dans la bibliothèque « Fonction ». La fonction « collaborateur commun », évaluant les acteurs commun de deux acteurs, « collaborateur proche » permettant de connaître les voisins d'un acteur à une distance donnée.

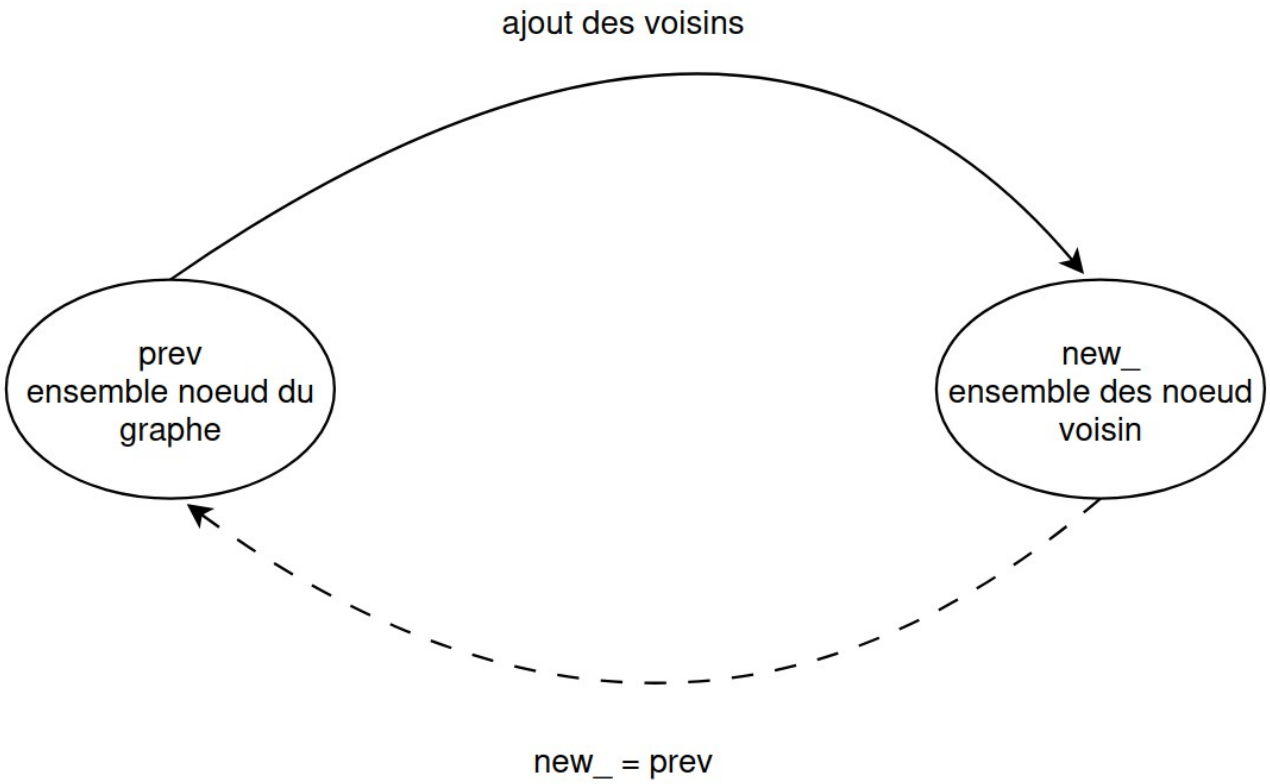
« Centralité » permettant de déterminer l'acteur le plus au centre du graphe. Toute les fonctions passe leur tests respectif

La majorité des fonctions ont été résolu à l'aide d'un BFS, parcours en largeur

Pour finir nous avons réaliser une petite IHM permettant de voir les acteurs disponibles dans le JSON et d'obtenir les résultats de chaque fonctions

Solutions adaptés

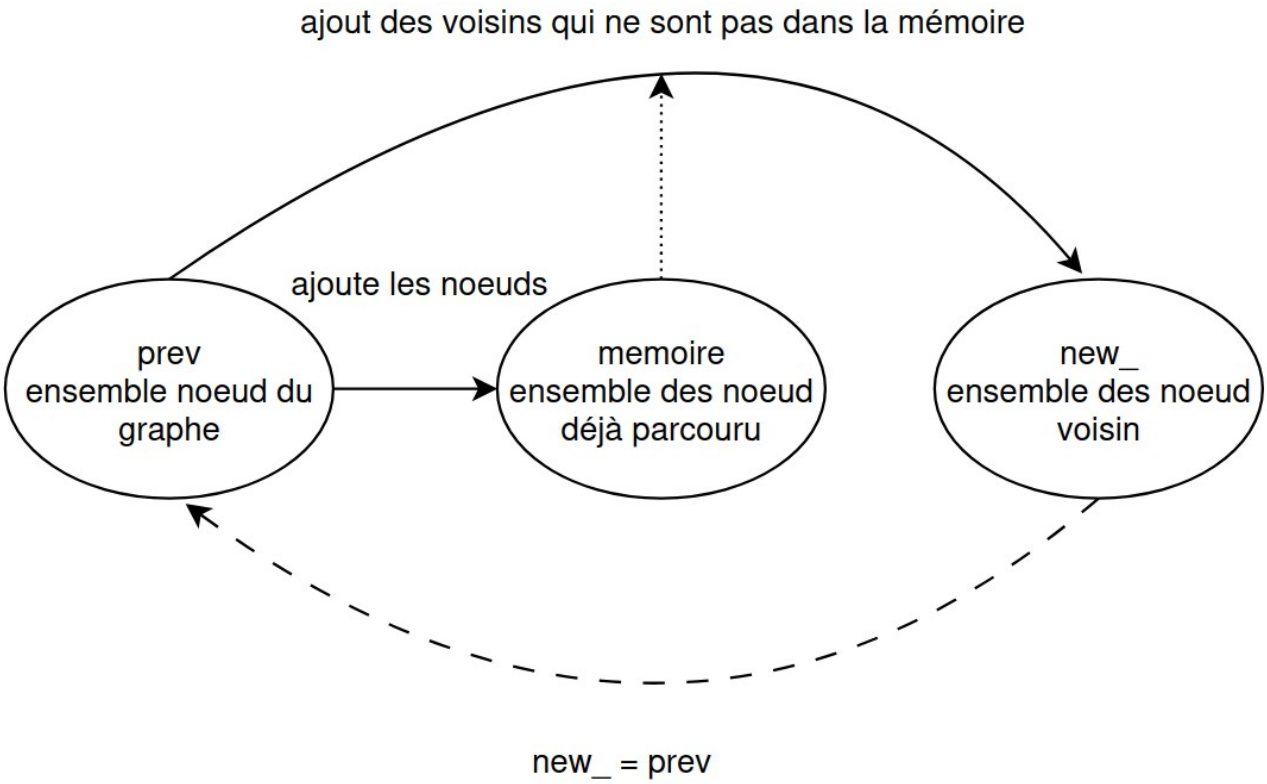
La quasi totalité des fonctions suivent le même schéma d'implémentation ressemblant à celui d'un BFS



les donnée vont être établi dans 2 ensembles prev et new_. Prev va contenir au départ un seul nœud, la fonction va alors ajouté dans new_ ses voisins puis prev va faire une copie de new_ et on répète l'operation.

Grace à cette algorithmes on peut connaître les voisins a distance n iteration

De plus une optimisation à été réaliser sur certaines fonctions.



On gère maintenant un 3ème ensemble nommé mémoire qui va contenir tout les nœud de present dans prev à chaque iteration et lors de l'ajout de prev à new_ on omettra les voisin dans mémoire

Cela nous permet de multiplier par 3 la rapidité du programme

pour plus d'information les complexité sont données dans le code en documentation