

Description des Modules – Projet C/C++

*Projet C/C++*

Sujet : « Réseaux sociaux : Recherche de composantes fortement connexes »

DREYER Quentin

JAMBET Pierre

NGUYEN Michael

Table des matières

[I. Introduction 3](#_Toc262398304)

[II. Schéma général 4](#_Toc262398305)

[Figure 1 : Schéma d’utilisation général 4](file:///C:\Users\Kyrine\Desktop\Projet%20C%20Facebook\doc\description.docx#_Toc262398306)

[III. Schéma détaillé 5](#_Toc262398307)

[1. L’analyseur de graphe, Graph 5](#_Toc262398308)

[Figure 2 : Cheminement des données 6](file:///C:\Users\Kyrine\Desktop\Projet%20C%20Facebook\doc\description.docx#_Toc262398309)

[2. Les structures de graphe, AdjList et AdjMat 7](#_Toc262398310)

[Figure 3 : Traitement par la structure 8](file:///C:\Users\Kyrine\Desktop\Projet%20C%20Facebook\doc\description.docx#_Toc262398311)

[3. Le générateur de fichier d’entrée, Generator 9](#_Toc262398312)

[Figure 4 : Création des données 10](file:///C:\Users\Kyrine\Desktop\Projet%20C%20Facebook\doc\description.docx#_Toc262398313)

# Introduction

Alors que le travail demandé était simplement de calculer les composantes fortement connexes d’un graphe, ainsi que des distances entre des points, nous avons étendu notre programme afin d’ajouter des fonctionnalités pouvant permettre à l’utilisateur un contrôle sur tout ce qui était en lien avec le sujet.

En effet, en plus de faire ce qui a été demandé, nous nous sommes penchés sur les différentes représentations possibles en machine des graphes, c'est-à-dire les matrices d’adjacences et les listes d’adjacences. Chacune de ces représentations possède ses propres avantages et inconvénients selon la composition du graphe, c’est pourquoi nous avons implémenté les deux cas et laissé libre choix à l’utilisateur pour choisir la représentation qu’il désire.

Cependant, dans la conception d’un tel programme, les tests ont une importance capitale. Nous avons donc aussi pris l’initiative d’y ajouter un outil de génération aléatoire de graphe. L’utilisateur pourra donc, en plus de récupérer des données sur des personnes réelles, générer, avec ces informations, des graphes aléatoirement avec un nombre de personnes, de relations, et de distances désiré.

# Schéma général

### Figure 1 : Schéma d’utilisation général

Résultats

Crée un fichier de données standard

Récupère les donnees Facebook

Compte Facebook

OU

AdjMat :

Graphe avec structure matrice d’adjacence

AdjList :

Graphe avec structure liste de voisins

Graph :

Lit et analyse le fichier d’entrée au format standard

Generator :

Fichier d’entrée au format standard

Fichier de noms avec leur id

# Schéma détaillé

## L’analyseur de graphe, Graph

La classe Graph est le cœur du programme. C’est elle qui s’occupe de :

* Recueillir les données brutes (fonction initGraph)

Cette structure n’a besoin que de recevoir le chemin du fichier pour l’ouvrir. Une fois cela effectué, il extrait les données importantes pour l’analyse, c’est-à-dire :

* + - Les informations sur les personnes (nom, id, et fréquence) qui sont stockés dans un tableau.
    - Les relations entre ces personnes, qui selon la structure choisie par l’utilisateur, sera soit dans une matrice de booléen, soit un tableau où chaque case est le tableau des amis.
    - Les questions sur le temps entre deux personnes où l’on stockera pour chaque point de départ distinct, un tableau de point à atteindre.

Améliorations possibles : Vérifier que le fichier donné par l’utilisateur soit au bon format

* Rendre ces données exploitables (fonctions searchSCC et searchDistances)

Selon l’action que l’on veut effectuer (déterminer les composantes fortement connexes ou répondre aux questions), on a besoin tout d’abord de faire modifier les données brutes afin de les rendre exploitables.

Le Graph va donc avant chaque analyse, envoyer les informations à la structure adéquate qui s’occupera de retourner des données dont on pourra tirer plus simplement les informations.

Améliorations possibles : -

* Extraire les résultats (fonctions searchSCC et searchDistances)

Une fois les données rendues exploitables, le Graph va les analyser afin d’extraire les résultats demandés. Ces résultats sont stockés dans des structures afin de les retrouver plus rapidement.

Améliorations possibles : -

* Sauvegarder les résultats (fonctions saveGraph)

Une fois tous les résultats obtenus, le Graph s’occupe de les enregistrer dans un fichier indiqué par l’utilisateur. La sauvegarde se fait avec le format suivant :

*1 ligne nc (le nombre de composantes fortement connexes)*

*nc lignes id1, id2, …*

*q lignes (réponses aux questions)*

*t\_min : id1, id2, … , idk (s’il existe un chemin)*

*Pas de chemin entre id1 et id2 (s’il n’existe pas de chemin)*

Les points importants et points d’articulation sont repérés par « \*id ».

Améliorations possibles : Différencier les points d’articulations des points importants

### Figure 2 : Cheminement des données

Graph

Envoi de données brutes

Ouverture Fichier d’entrée

Zone de stockage

Sauvegarde Fichier de sortie

Réception des données traitées

## Les structures de graphe, AdjList et AdjMat

Les classes AdjList et AdjMat représentent chacune une structure de graphe. Les algorithmes utilisés dans les deux classes sont cependant identiques. La seule chose qui diffère entre les deux classes est la façon dont est repérée une relation entre deux personnes.

Pour AdjList, une personne aura un tableau contenant toutes les personnes à qui il fait confiance, et un autre tableau contenant les personnes qui lui font confiance. L’avantage est que pour des réseaux avec beaucoup de monde mais où les gens se font peu confiance entre eux, il y aura beaucoup moins de données inutiles stockées.

Pour AdjMat, une matrice indiquera ses relations. Pour une personne donnée, pour avoir accès à la liste des personnes à qui il fait confiance, il faudra regarder toute la ligne, un 1 représentera une relation de confiance. Dans le même principe, la liste des personnes qui lui font confiance est accédée en regardant dans la colonne. L’avantage est un temps d’accès rapide, surtout pour des réseaux ou tout le monde se fait confiance.

Ces deux classes n’ont qu’un seul but identique : Traiter les informations pour les rendre exploitables. En effet, leur travail est de modifier le tableau de personne avec des informations utiles afin que la classe Graph puisse les analyser.

* Calcul des composantes fortement connexes (fonction initSCC)

Le calcul de ces composantes ne se fait pas vraiment, grâce à une technique basée sur des parcours en profondeur, on pose des marqueurs de temps sur chaque personne. Ces marqueurs permettront à la classe Graph d’exploiter les composantes fortement connexes.

Durant le calcul de ces composantes, la structure indique aussi si un point peut être considéré comme important ou pas.

Améliorations possibles : -

* Calcul des distances entre les points : (fonction initDist)

De la même manière que pour les composantes fortement connexes, on place des marqueurs de temps sur chaque personne en utilisant l’algorithme de Dijkstra. Ces marqueurs permettront de retrouver le chemin le plus rapide d’un point vers le point de départ de l’algorithme, ainsi que la distance.

Améliorations possibles : Arrêter le parcours dès que les points d’arrivée qui nous intéressent sont explorés.

AdjMat ou AdjList

Envoi du tableau de personne traité

Réception du tableau de personne brut

### Figure 3 : Traitement par la structure

## Le générateur de fichier d’entrée, Generator

La classe Generator contient plusieurs fonctionnalités qui font le lien entre le monde extérieur et le programme. C’est lui qui s’occupe de :

* Récolter les données réelles (fonction generateDatabase)

Cette fonctionnalité utilise le module de récupération d’amis écrit par Mr Dinu. Il permet de récolter, à partir d’un ID Facebook quelconque, la liste de tous ses amis dans la limite d’affichage d’une page.

Il les convertit ensuite et les ajoute dans un fichier « base de données » recensant toutes les personnes recueillies. L’utilisateur peut choisir aussi de créer une nouvelle base de données

Le format pour chaque ligne est :

*Nom et prénom, ID*

Améliorations possibles : Pour chaque ID, recueillir plusieurs niveaux d’amis.

* Créer des fichiers à exploiter par Graph (fonction generateFile)

A partir d’un fichier créé par generateDatabase, on peut utiliser cette fonctionnalité pour créer un fichier qui sera analysé par Graph. L’utilisateur entre un nombre de personnes, un nombre de relations et un nombre de questions et le fichier sera généré avec le format adéquat (voir *II.1*)

Améliorations possibles : Nombre de relations et de questions générés plus précis. Moins d’aléatoire dans les relations.

### Figure 4 : Création des données

Generator

Enregistrement dans la base de données

ID facebook

Base de données

Nombre de personnes, de relations et de questions

Création d’un fichier d’entrée