

Rapport de Synthèse – Projet C/C++

*Projet C/C++*

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

DREYER Quentin

JAMBET Pierre

NGUYEN Michael

Table des matières

[**Introduction** 3](#_Toc262504665)

[**1.** **Sujet** 3](#_Toc262504666)

[**2.** **Organisation du travail** 4](#_Toc262504667)

[**Travail réalisé** 5](#_Toc262504668)

[**1.** **Outils utilisés** 5](#_Toc262504669)

[**2.** **Représentation** 6](#_Toc262504670)

[Figure 1 : Graphe exemple 6](#_Toc262504671)

[Figure 3 : Liste d'adjacence 6](#_Toc262504672)

[Figure 2 : Matrice d'adjacence 6](file:///E:\Mes%20documents\Bureau\ig3projetc\trunk\doc\synthèse.docx#_Toc262504673)

[**3.** **Les classes** 7](#_Toc262504674)

[1) Graph 7](#_Toc262504675)

[2) AdjMat 7](#_Toc262504676)

[3) AdjList 8](#_Toc262504677)

[4) Generator 8](#_Toc262504678)

[5) Summit 9](#_Toc262504679)

[**4.** **Les tests** 10](#_Toc262504680)

[**Conclusion** 11](#_Toc262504681)

[**Annexes** 12](#_Toc262504682)

[**1.** **Remerciements** 12](#_Toc262504683)

[**2.** **Bibliographie** 12](#_Toc262504684)

[**3.** **Webographie** 12](#_Toc262504685)

# **Introduction**

## **Sujet**

#### Travail demandé

Le sujet du projet portait sur l’analyse des réseaux sociaux comme Facebook par exemple. Dans ce cadre il nous était demandé d’analyser des données extraites de Facebook avec des outils issus de la théorie des graphes.

L’objectif de ce projet était de créer un programme en C ou C++ dont la principale fonctionnalité est de parcourir un graphe afin d’effectuer sur ce dernier plusieurs calculs. Le premier calcul étant la décomposition des composantes fortement connexes, CFC. Il est ensuite demander de chercher des « plus courts chemins » à l’intérieur de ce graphe entre deux sommets donnés, chaque sommet étant pondéré, ici chaque personne est pondérée par la fréquence d’actualisation de son compte Facebook.

Le thème principal du projet étant l’exploration de graphes nous avons choisi de nommer notre programme LAGER.

\_

| | \_\_ \_ \_\_ \_ \_\_\_ \_ \_\_

| | / \_` |/ \_` |/ \_ \ '\_\_|

| |\_\_| (\_| | (\_| | \_\_/ |

|\_\_\_\_\_\\_\_,\_|\\_\_, |\\_\_\_|\_|

|\_\_\_/

Lager Ain't a Graph Explorer for Rookies !

#### Fichier d’entrée

Le fichier d’entrée que notre programme doit lire doit être de la forme suivante :

**1 ligne** n *(le nombre de personnes dans le réseau social)*

**n lignes** nom personne, id, fréquence

**1 ligne** m *(le nombre de relations entre les personnes)*

**m lignes** id1, id2 *(personne id1 a confiance en id2)*

**1 ligne** q *(le nombre de questions que l’on se pose)*

**q lignes** id1 -> id2

4

Pierre Bollampally, 12331, 24

Gouman Rin, 564, 2

Jnkeea van der Zwaan, 76549, 8

Popescu George, 99, 1

6

12331, 564

564, 76549

76549, 12331

564, 12331

99, 564

99, 76549

3

99 -> 564

99 -> 76549

99 -> 12331

Une partie du travail a donc consisté à générer des fichiers d’entrée respectant cette nomenclature afin qu’ils puissent être lus par le programme.

## **Organisation du travail**

Le choix du langage fut la première question auquel nous avons du faire face, en effet le projet pouvait être réalisé en C ou C++, ayant tous trois des connaissances dans ces deux langages nous avons choisi le C++. Ce choix est justifié par coté « Orienté Objet » du C++.

Ce choix peut paraître surprenant car nous n’avons utilisé pour ce projet qu’une infime partie des possibilités du C++, nous n’avons pas utilisé d’héritages ni de polymorphismes ou autres techniques avancées. La « simple » utilisation des classes ainsi que de la Bibliothèque standard STL (Standard Template Library), nous ont été très utiles. De plus, nous avons pu aussi utiliser des fonctions du langage C très utiles : atoi ou fscanf par exemple.

# **Travail réalisé**

## **Outils utilisés**

#### Développement

Le projet a été développé conjointement sur Windows ainsi que sur Gnu/Linux, pour plusieurs raisons. Nous avions tous des machines fonctionnant sous Windows chez nous, cependant, certains outils n’étant disponibles que sur Gnu/Linux, comme Valgrind par exemple et comme les ordinateurs de Polytech’ fonctionnent en grande partie sur Debian, nous avons utilisé les deux systèmes d’exploitation. Ceci nous a permis d’obtenir un code portable.

#### Contrôle de versions

Afin de gérer au mieux le travail de groupe, et pour optimiser le travail collaboratif nous avons choisi d’utiliser un gestionnaire de version. Notre choix s’est porté sur SubVersion(SVN), car l’outil TortoiseSVN pour Windows est extrêmement pratique. Nous avons donc hébergé notre code sur un serveur fourni gratuitement par Google (URL : <http://code.google.com/>).

#### Editeurs

Les outils que nous avons utilisés pour le développement étaient soit un simple éditeur de texte, NotePad++ sur Windows ou Gedit sur Ubuntu, ainsi qu’un IDE tel que CodeBlocks que nous avons retenu pour sa portabilité ainsi que sa gratuité.

#### Tests unitaires

Afin de créer un code robuste et facile à modifier, nous avons choisi d’utiliser un framework permettant de mettre en place des tests unitaires. Nous avons choisi le framework Google Test, pour sa simplicité d’utilisation. Ainsi a chaque modification du code, nous pouvions utiliser les tests pour vérifier que chaque fonction se comportait correctement, indépendamment du reste du programme.

Un test s’écrit simplement de la manière suivante :

*Sur cet exemple, on teste premièrement que la condition cond est vraie et on teste ensuite l’égalité des valeurs de val1 et val2.*

TEST(nom\_categorie\_test, nom\_test) {  
 ASSERT\_TRUE(cond);  
 EXPECT\_EQ(val1, val2);  
}

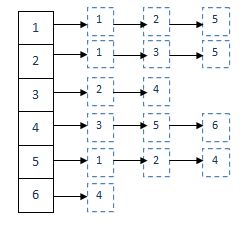
On a donc pu écrire une succession d’assertions vérifiant les propriétés de nos fonctions.

## **Représentation**

Nous avons choisi de représenter les graphes de deux manières différentes, qui sont les deux façons les plus classiques, la matrice d’adjacence, et la liste d’adjacence. Voici un exemple illustrant ces deux représentations. On notera l’intérêt de la liste en termes de stockage mémoire car on ne stocke pas les sommets « non voisins » d’un sommet donné.

## E:\Mes documents\Bureau\ig3projetc\trunk\doc\Fichier6n-graph2.png

### Figure 1 : Graphe exemple

\begin{pmatrix}
1 & 1 & 0 & 0 & 1 & 0\\
1 & 0 & 1 & 0 & 1 & 0\\
0 & 1 & 0 & 1 & 0 & 0\\
0 & 0 & 1 & 0 & 1 & 1\\
1 & 1 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & 0\\
\end{pmatrix}.

### Figure 2 : Matrice d'adjacence

### Figure 3 : Liste d'adjacence

Nous avons donc pu effectuer les calculs avec les deux représentations et comparer les temps d’exécution et occupation de mémoire en fonction de cette représentation.

## **Les classes**

Les classes et la façon dont elles interagissent entre elles sont détaillés à l’aide de schéma dans le rapport intitulé « description ». Nous nous contenterons ici d’un très bref résumé du fonctionnement de chacune.

Il est tout de même important de noter que dans tout le code, l’information essentielle est le tableau de sommet. En effet, pour toutes les données sur ces personnes, on ne les désignera pas par leur id, unique mais difficile à retrouver, mais par leur emplacement dans ce tableau.

Ainsi, pour la représentation matrice, on pourra faire plus simplement le lien entre la ligne/colonne x, qui nous enverra sur les liens de la x-ième personne du tableau. De même, si la représentation est en liste, la x-ième liste (qui peut être vide) indiquera les voisins de la x-ième personne du tableau. Cette liste contiendra de la même manière les numéros des personnes, et non des personnes ou les id.

Le seul endroit où l’on va repasser de ces nombres aux personnes réelles sera lors de la sauvegarde dans le fichier de sortie.

### Graph

Cette classe est la classe principale du projet. Elle contient la plus grande partie des fonctions que l’on appellera de l’extérieur. Elle fait office d’interface entre les deux classes qui gèrent la représentation des graphes (AdjMat et AdjList) et l’utilisateur.

/\* ===========================================================================

PROJET : projet C ig3 - FACEBOOK

NOM DU FICHIER : Graph.h

OBJET : Structure de graphe

-------------------------------------------------------------------------------

DATE DE CREATION : 30/04/2010

AUTEUR : Quentin DREYER / Pierre JAMBET / Michael NGUYEN

-------------------------------------------------------------------------------

DETAILS : SCC (Strongly Connected Component) = CFC (Composante Fortement Connexe)

============================================================================ \*/

Son importance vient aussi du fait qu’elle va contenir toutes les informations données et résultats :

* Les personnes : Nous avons choisi de les représenter sous forme d’un vecteur de Summit (voir 3.5). En effet, grâce à l’efficacité de cette structure, on peut ajouter rapidement les personnes dans le tableau en avançant au fur et à mesure dans le fichier d’entrée.
* Les relations :
  + Structure matrice : Grâce au lien entre la place d’une personne, et une ligne/colonne dans une matrice, nous n’avons pas eu besoin d’une structure plus complexe qu’un simple tableau
  + Structure liste : pour les listes, nous avons exploité le caractère « tableau dynamique » du vecteur afin d’ajouter les relations au fur et à mesure. Cependant, un tableau dynamique de vecteur nous posait quelques soucis lors de la destruction de l’objet, c’est pourquoi nous avons opté pour un vecteur de vecteur. Afin d’optimiser la vitesse, au dépend de l’espace mémoire, nous avons décidé de créer à la fois les listes « à qui je fais confiance » et « qui me fait confiance ».
* Les composantes fortement connexes : Cette fois ci, nous avons à nouveau exploité le caractère dynamique du vecteur. En effet, on ne peut pas prédire à l’avance le nombre de CFC, ainsi que le nombre de personne dans chacune. C’est pourquoi, nous avons opté pour un vecteur de vecteur : chaque vecteur différent représente une CFC, et chaque CFC est composée de personnes (ou plutôt, de numéro renvoyant vers les personnes)
* Les questions : Afin d’éviter plusieurs calculs de distances inutiles, nous avons jugé intéressant de regrouper les questions ayant le même point de départ. Le container map était ici approprié. En effet, pour une clé unique (le point de départ), nous avons associé un vecteur de point, les points d’arrivée.
* Les distances : De la même manière, comme on peut retrouver les distances à partir des valeurs π et d, nous avons pour chaque point de départ (identique que pour les questions), associé ces tableaux (que l’on peut récupérer avec la structure Sumit).
* Les chemins : Un nouveau problème s’est posé. Comment repérer deux chemins différents à partir d’un même point ? Nous avons donc exploité la structure de multimap permettant à une clé (point de départ) d’avoir plusieurs valeurs (chemins vers des points d’arrivée différents).

Avec cette structure en tête, il fut alors simple de répartir les tâches des fonctions :

* Initiation :
  + Remplir le tableau de questions
  + Remplir la structure de graphe selon le choix de l’utilisateur (par défaut la structure est matrice).
  + Remplir le map correspondant aux questions
* Recherche des CFC :
  + Initialiser et stocker le tableau de Summit
  + Parcourir le tableau et remplir au fur et à mesure les CFC
* Recherche des distances :
  + Pour chaque point de départ différent, stocker le tableau de Summit
  + Pour chaque question, parcourir le tableau correspondant et stocker le chemin
* Sauvegarde :
  + Lire dans les données dans les structures définies.

### AdjMat

La classe AdjMat contient les fonctions permettant d’effectuer les calculs sur un graphe sous forme de Matrice. Les fonctions qu’elle définie sont les mêmes que celles de AdjList, elles ne différent qu’au niveau de la représentation des relations entre les sommets du graphe.

/\* ===========================================================================

PROJET : projet C ig3 - FACEBOOK

NOM DU FICHIER : AdjMat.h

OBJET : matrice d'adjacence

-------------------------------------------------------------------------------

DATE DE CREATION : 30/04/2010

AUTEUR : Quentin DREYER / Pierre JAMBET / Michael NGUYEN

-------------------------------------------------------------------------------

DETAILS : SCC (Strongly Connected Component) = CFC (Composante Fortement Connexe)

: DFS (Depth First Search) = PPG(Parcours en Profondeur du Graphe)

============================================================================ \*/

Avec cette structure, afin de déterminer toutes les personnes à qui la personne x fait confiance, il est nécessaire de parcourir tout la ligne. Cependant, on peut déduire de la même matrice toutes les personnes qui font confiance à x en regardant tout simplement la colonne.

En suivant donc les algorithmes proposés dans le sujet et adapté le parcours de façon à être utilisable par cette structure. Nous avons pu implémenter simplement le parcours en profondeur. Cependant, dans un souci d’optimisation, nous avons préféré utiliser des algorithmes de tri contenus dans la librairie standard. Le seul inconvénient était de devoir sauvegarder les id des personnes afin de pouvoir les retrouver plus tard.

Le calcul des points importants se fait naturellement pendant le calcul des CFC. Après avoir réfléchi à des critères pouvant permettre à un sommet d’obtenir le statut d’important, nous en avons déduis qu’un point était important si :

* Il n’était ni source, ni puits
* Il n’était pas relié par le reste du graphe par le même sommet
* Il possédait un fils (resp. père) dans la même CFC qui ne possède pas d’autre père (resp. fils) de la même CFC

Pour le calcul des distances, nous avons adapté l’algorithme de Dijkstra afin d’obtenir les meilleures performances. En effet, nous avons ajouté une condition permettant à la fonction de s’arrêter dès qu’il n’y a plus de points atteignables, même si tous les points n’ont pas été réellement atteints.

### AdjList

Comme il l’est précisé ci dessus, cette classe réalise l’ensemble des calculs qui permettent d’obtenir les composantes connexes d’un graphe représenté par une des listes d’adjacence.

/\* =============================================================================

PROJET : projet C ig3 - FACEBOOK

NOM DU FICHIER : AdjList.h

OBJET : representation de graphes sous forme de liste de voisins

--------------------------------------------------------------------------------

DATE DE CREATION : 30/04/2010

AUTEUR : Quentin DREYER / Pierre JAMBET / Michael NGUYEN

--------------------------------------------------------------------------------

DETAILS : SCC (Strongly Connected Component) = CFC (Composante Fortement Connexe)

: DFS (Depth First Search) = PPG(Parcours en Profondeur du Graphe)

============================================================================= \*/

Les algorithmes utilisés sont identiques à ceux pour AdjMat. La seule différence est que nous avons un accès direct à tous les voisins d’une personne, et donc, peut faciliter certains parcours. Le problème est que pour les graphes comportant beaucoup de relations, il peut arriver que l’on stocke plus d’information que pour une matrice.

### Generator

Afin de générer des fichiers dits « d’entrée » qui respectent la forme imposée par le sujet nous avons créée une classe dédiée à la génération qui peut venir de plusieurs sources.

/\* =============================================================================

PROJET : projet C ig3 - FACEBOOK

NOM DU FICHIER : Generator.h

OBJET : Genere des fichiers de donnee

--------------------------------------------------------------------------------

DATE DE CREATION : 19/04/2010

AUTEUR : Quentin DREYER / Pierre JAMBET / Michael NGUYEN

--------------------------------------------------------------------------------

============================================================================= \*/

Le générateur est composé de deux fonctions principales :

* La génération d’un fichier d’entrée : A partir d’une base de données fournie, ou créée par l’utilisateur, un nombre de personnes dans la limite des stocks disponibles, un nombre de relations et de questions, cette fonction va piocher dans la base de données le nombre de personnes désiré. A partir de là, nous avons créé une matrice que l’on remplira. Par exemple, du nombre de relation, on peut déduire qu’il faut que la matrice soit remplie avec un certain pourcentage, et donc que chaque case de la matrice possède ce même pourcentage d’être à 1. Il nous suffit alors de parcourir la matrice afin de déterminer les relations. On fait de même pour les questions. Le nombre de relations et de questions obtenu est égal au nombre demandé à 5% près, du à l’aléatoire.
* La génération d’une base de données : Pour cette fonction, nous avons utilisé le module de Mr Dinu. Nous avons cependant remarqué qu’il n’était nécessaire de se connecter à Facebook qu’une seule fois, juste pour récupérer les cookies. Après, nous pouvions récupérer les données de n’importe qui à volonté. Afin de générer une base de données, il faut tout d’abord savoir si l’utilisateur désire compléter une base déjà existante ou bien en créer une nouvelle. S’il s’agit du premier choix, il faut au préalable stocker les éléments déjà dans la base (que nous faisons à l’aide d’un map). Ensuite, à chaque nouvelle personne que l’on veut insérer, on peut facilement vérifier si elle est déjà présente. On évite ainsi de se retrouver avec plusieurs personnes ayant le même ID.

### Summit

Ce fichier n’est en réalité pas une classe. Nous avons fait ce choix car nous trouvions superflu d’utiliser le mécanisme complexe d’une classe alors que nous ne voulions seulement définir une structure qui nous serait utile pour stocker des informations concernant les sommets. Il était aussi nécessaire de définir quelques fonctions associées à cette structure,

/\* =============================================================================

PROJET : projet C ig3 - FACEBOOK

NOM DU FICHIER : Summit.h

OBJET : Structure de sommet

--------------------------------------------------------------------------------

DATE DE CREATION : 30/04/2010

AUTEUR : Quentin DREYER / Pierre JAMBET / Michael NGUYEN

--------------------------------------------------------------------------------

DETAILS : Ce fichier définit la structure "struct\_summit" qui permet de faire toutes les operations sur les   
graphes : parcours, calcul de CFC ...============================================================================= \*/

Les informations stockées sont simple :

* Le nom en chaine de caractère
* L’ID en chaine de caractère aussi étant donné qu’une ID Facebook peut contenir jusqu’à 20 caractères, trop long pour un int normal
* La fréquence

En plus de cela, nous avons d’autres attributs qui vont nous servir dans le calcul des CFC et des distances :

* Beg : Pour le calcul des CFC, il nous indique le temps de début. Pour le calcul des distances, on peut y lire directement la distance de ce sommet au point de départ.
* End : Pour le calcul des CFC, il indique le temps de fin d’exploration. Pour le calcul des distances, il indique le père le plus proche, -1 s’il s’agit du point de départ.
* Num : Il nous indique l’emplacement par rapport au tableau de sommet initial, en cas de réorganisation.

## **Les tests**

Les détails des tests sont fournis dans le document intitulé « test » nous nous contenterons ici de lister les différents types de tests ainsi que d’expliquer leur utilité au sein du projet.

Nous pouvons séparer nos tests en deux catégories **fonctionnels** et **non-fonctionnels**

* **Les tests fonctionnels**

Ces tests ont pour but de vérifier le bon fonctionnement du programme ou plus en détail de chaque fonction. Avec de tels tests nous nous sommes assuré que le programme répondait à nos attentes.

Ces tests ont été effectués avec des « tests unitaires » que nous avons réalisés avec le FrameWork Google Test. Nous avons ainsi testé chaque portion important de notre code. De cette manière nous pouvions savoir à tout moment de l’évolution du projet si les résultats programme étaient « corrects » ou non.

* **Les tests non-fonctionnels**

Ces tests nous ont permis de contrôler l’exécution de notre programme. Nous avons réalisé pour ceci deux tests. Premièrement nous avons écris un script shell qui exécutait un très grand nombre de fois le programme avec des paramètres différents a chaque fois. Nous avons ainsi obtenu des statistiques sur le temps de calcul des composantes fortement connexes et pour les calculs des chemins. Ces statistiques, sous formes de graphiques, sont disponibles en annexe, dans le document intitulé « test ».

Nous avons aussi utilisé l’outil Valgrind, principalement son module « memcheck » que nous avons utilisé afin de traquer les fuites mémoire de notre programme.

L’utilisation des ces tests non-fonctionnels nous a permis de travailler sur l’efficacité de notre programme. Nous sommes ainsi arrivés à diminuer le temps d’exécution tout en évitant les problèmes liés a la mémoire.

# **Conclusion**

# **Annexes**

## **Remerciements**

Nous tenons à remercier Polytech‘ Montpellier, plus particulièrement l’ensemble de nos professeurs, dont les enseignements nous ont permis de mener à bien ce projet. Cependant nous pensons qu’il est important de remercier aussi les auteurs des outils qui nous ont été indispensables pendant le déroulement du projet.

Merci à :

* Google pour : Google Code, Google Test et Google
* Richard Stallman pour GCC
* Razvan Dinu grace a qui nous avons pu tester notre programme sur des données « réelles ».
* Mathieu Triay pour ses conseils avisés

## **Bibliographie**

Support de cours

The C Programming Language (K & R)

## **Webographie**

* <http://www.siteduzero.com/>
* <http://www.cplusplus.com/>
* <http://www.developpez.com/>
* et en particulier :
* <http://gl.developpez.com/tutoriel/outil/makefile/>
* <http://fr.wikipedia.org/>
* <http://code.google.com/p/googletest/>