Projet IN608

Manipulation de Graphes orientés
Compte Rendu

AIRIAU Vincent, DAFRANE Salsabil, DE FREITAS Alexandre, GODINEAU Camille,
MALO Amandine, NARDIN Théo, OILLO Sébastien, PEUGNET Guillaume, TEDESCHI Hugo

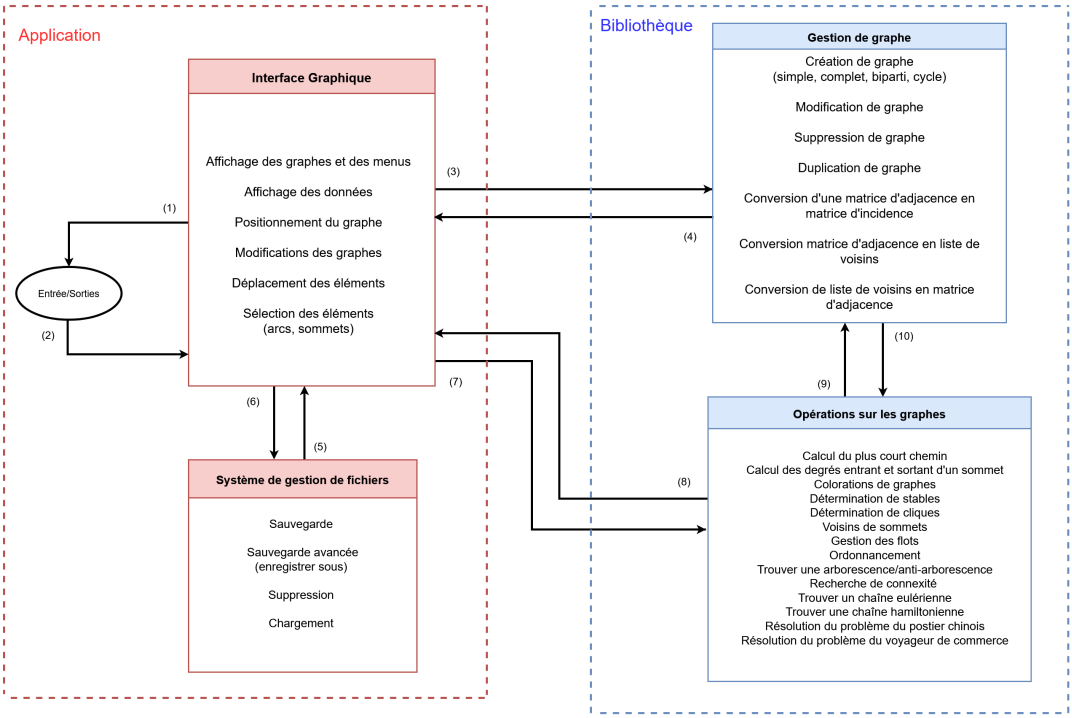
2019 - 2020

Sommaire

1 Introduction

Nous avons choisis le sujet des graphes orientés car c'est un sujet pluridisciplinaire et dont les applications sont larges et multiples.
Au cours de notre cursus à l'université de Versailles, la théorie des graphes fut une des connaissances incontournables de notre formation. C'est pourquoi nous avons choisis de traiter ce sujet au travers d'une application permettant la modélisation de graphes orientés et d'une bibliothèque permettant de les manipuler fut une manière de traiter tous ces acquis sous un aspect concret. Selon nous, avoir à développer une application mettant en pratique la théorie des graphes est aussi une manière de consolider nos acquis, mais aussi d'en apprendre plus.
A la suite de ce compte rendu relatif à l'implémentation, nous aborderons le fonctionnement de l'architecture de cette application et de la bibliothèque. Au cours d'explications techniques, nous présenterons les points épineux du projet ayant été mis en évidence pendant la phase de développement. Enfin, nous proposerons un bilan rétrospectif sur notre travail ainsi que sur l'organisation au sein de l'équipe de conception

2 Architecture



1 :	Clics et touches du clavier	6 :	Nom du fichier, message de validation (suppression, sauvegarde) ou contenu du fichier
2 :	Affichage	7 :	Graphe, sommet du graphe, message de demande d'actions
3 :	Sommets sélectionnés, arcs sélectionnés, graphe, choix des interactions, taille du graphe	8 :	Liste des sommets, sous-graphe, graphe, message de validation de l'opération
4 :	Graphe	9 :	Graphe modifié
5 :	Nom du fichier, graphe	10 :	Graphe

L'organigramme ci-dessus présente les quatre modules essentiels au développement de notre application et de notre bibliothèque. L'architecture de ces derniers est composé :
- d'une gestion de graphe, permettant de créer, supprimer et manipuler des graphes orientés.
- d'opération sur les graphes, dans lequel des algorithmes complexes sont implémentés pour être appliqués aux graphes.
- d'une interface graphique, le noyau de notre projet. Il communique avec tous les autres modules en interaction avec l'utilisateur.
- d'un système de gestion de fichiers, servant à stocker ou récupérer des données sauvegardées par l'utilisateur via la bibliothèque RapidJson.

3 Description du fonctionnement

Nous allons dans cette partie étudier le fonctionnement interne de notre programme.

3.1 Prérequis

On suppose qu'un utilisateur doit avoir des connaissances en C++, s'il souhaite utiliser notre bibliothèque afin d'effectuer son propre projet. En revanche il ne lui est pas nécessaire d'avoir des connaissances en programmation s'il souhaite seulement utiliser notre application graphique.

Dans le cas où l'utilisateur souhaite utiliser notre bibliothèque, deux fichiers à inclure sont conçus à cet effet:
- Le fichier *liste.hh*, qui lui permettra d'avoir accès aux algorithmes de graphes orientées que l'on a implémenté.
- Le fichier *classes.hh*, qui donne accès aux classes et structures essentielles à la représentation de graphes.

3.2 Fonctionnement des modules

3.2.1 Gestion de graphes

Le module de gestion de graphes (*classes.hh*) est composé de différentes classes et structures primordiales à la composition d'un graphe orienté. Il va permettre, grâce à la combinaison de ces différentes classes, la création de graphes orientés manipulables et représentables.
Nous avons d'abord la classe Sommet (*Sommet.hh*), qui représente les sommets d'un graphe. On y retrouve toutes les caractéristiques d'un sommet (ID, étiquette, coordonnées, charge utile) mais aussi la liste des arcs sortants de ce sommet, nécessaire pour l'optimisation du code. De la même manière, la classe Arc (Arc.hh) représente les arcs du graphe grâce à son ID, l'étiquette, la charge utile et les ID des sommets départ et arrivée qui a nouveau font le lien entre ces deux classes. Ces dernières sont les éléments-clés pour construire un graphe orienté. On peut remarquer que dans la classe graphe (*graphe.hh*), ces classes expriment la majeure partie de la représentation par *liste_Arcs* et *liste_Sommets* qui symbolisent l'ensemble des arcs et des sommets présents dans le graphe. Le graphe possède aussi une étiquette ainsi qu'un chemin qui sera utilisé lors de la sauvegarde.
Il s'agit ici du noyau du projet car il est important d'avoir des objets biens définis et facilement manipulables pour une implémentation et une exécution des algorithmes cohérentes. C'est pourquoi une classe Matrice (*Matrice.hh*) est également présente pour récupérer et stocker les informations essentielles, ce qui facilite l'accès aux objets et par conséquent permet une meilleure optimisation.
Dans notre programme, nous définissons également des tests associés à ce module. Ils vérifient une création correcte et une bonne manipulation des différents objets évoqués précédemment. Cette vérification se fait sur les constructeurs de classe, les surcharges d'opérateur, les méthodes ainsi que sur les getters et les setters.

3.2.2 Opération sur les graphes

Le module d'opérations sur les graphes (*liste.hh*) est composé de l'ensemble des algorithmes qui peuvent être appliqués sur un graphe orienté. L'appel de ces algorithmes se fait soit via l'interface graphique, si l'utilisateur utilise l'application, soit via un appel de fonction classique, dans un code, pour un utilisateur utilisant la bibliothèque dans son propre programme.
Chacune des fonctionnalités implémentée dans ce module va permettre de manipuler un graphe et va renvoyer le résultat sous la forme de variables (vecteur, Graphe, entier, ...) ou modifier directement les attributs du graphe courant. Dans le cas d'un appel par l'interface graphique, les valeurs de retour seront affichées dans la console de l'application ou le graphe sera redessiné, en affichant, par exemple, les sommets de couleurs différentes si la coloration est appelée.

3.2.3 Gestion de fichiers

Pour le module de gestions de fichiers (*gestion_fichier.hh*), l'utilisateur a la possibilité d'enregistrer son graphe dans un document de type ".json". Ce document est créé à partir du graphe engendré par l'utilisateur et de ses composants tels que l'emplacement de son fichier, son nom, la liste de ses sommets ainsi que la liste de ses arcs. Le document généré

possède le même format que celui du schéma que nous avons défini (cf graphschema.json). Le fichier étant facilement lisible par un humain, l'utilisateur a la possibilité de le modifier, directement, à tout moment.
Le chargement d'un fichier se fait à l'aide du chemin vers ce dernier. Dans un premier temps le programme vérifie que le fichier sélectionné est bien un fichier Json, puis que son contenu est conforme au schéma de graphe que nous utilisons. Le programme crée alors un objet graphe à partir des informations stockées dans le fichier, dans lequel chacun des objets est initialisé avec les valeurs du fichier.

3.2.4 Interface Graphique

Le module interface graphique permet d'afficher le graphe, ses caractéristiques ainsi que le résultat des algorithmes. Cette interface permet aussi à l'utilisateur de créer un graphe et de le modifier, en le dessinant à la souris, mais aussi de faire appel aux différentes fonctionnalités.
Dans le cas de l'ouverture et de l'enregistrement d'un fichier, cette interface permet à l'utilisateur de naviguer dans le système de gestion de fichiers de son ordinateur. Enfin, ce module possède une fonction de positionnement qui permet, dans le cas où les sommets ne possèdent pas de coordonnées ou des coordonnées ne permettant pas une bonne visualisation, de les placer de manière à ce qu'ils ne se superposent pas. Chacun des graphes sera dessiné dans un onglet qui lui sera propre, sur la partie droite, l'utilisateur pourra sélectionner l'outil qu'il souhaite utiliser (soit l'ajout, la suppression ou la sélection pour les sommets et les arcs).
Lorsque l'utilisateur sélectionne un sommet ou un arc, les caractéristiques de ce dernier sont affichées sur la partie droite de l'écran. L'utilisateur a aussi à sa disposition, sur le haut de l'écran des menus déroulants desquels il pourra gérer les fichiers (ouverture, sauvegarde), dupliquer, supprimer, fermer un graphe, extraire un sous graphe et arranger les sommets grâce à la fonction de positionnement.
Il y a par ailleurs un menu regroupant l'ensemble des algorithmes disponibles et enfin un menu proposant l'accès à la documentation et au dépôt GitHub du projet. Le résultat des algorithmes est affiché sur la partie droite de l'écran, si il s'agit de valeurs, et dans le cas d'un nouveau graphe, un nouvel onglet est ouvert. Dans le cas de PERT, avant l'exécution de l'algorithme, une nouvelle fenêtre est ouverte dans laquelle l'utilisateur pourra rentrer l'ensemble des tâches sur lesquelles il veut travailler ainsi que leurs contraintes d'antériorité. De plus l'interface appellera dans un premier temps *calcul_posterite* qui va permettre de vérifier que les contraintes soient respectées et de calculer les postériorités.

4 Points délicats de la programmation

4.1 Bibliothèques

Le premier objectif que nous avions été de nous familiariser avec les bibliothèques choisies, car elle nous furent totalement inédites voire très peu expérimentée.
La première qu'on a pu exploiter est la bibliothèque de test Catch Catch2, qui se vend comme une bibliothèque moderne supportant toutes les versions de C++. Elle nous a été très utile pour vérifier, au préalable, le comportement de nos méthodes et de nos constructeurs via des tests que l'on a codé à cet effet.
La seconde à laquelle on a eu recoure est la bibliothèque RapidJSON, qui est consacré à l'analyse et la génération de fichier Json. Nous l'avons appliquer, comme expliqué plus haut, dans notre module gestion de fichier. En effet il permet de pouvoir sauvegarder, modifier, ou charger des données dans un document type.
La troisième, qui est essentiel à notre application est Qt. C'est une bibliothèque multiplateforme sur laquelle repose un environnement graphique important. C'est également bien plus que ça, c'est aussi un ensemble de bibliothèque, appelé aussi framework. Elle nous permet de pouvoir afficher nos graphes orientés à l'écran et de pouvoir donner la possibilité à l'utilisateur de manier chacune des fonctionnalités, par modules, visuellement.

4.2 Classes Graphe et Matrice

Une autre des difficultés rencontré a été relaté lors de l'implémention des classes Graphe et Matrice. En effet, dans la classe Matrice la méthode *Graphe conversionGraphe ()* renvoie un Graphe et nécessite pour cela son constructeur. De même que dans la classe Graphe, les constructeurs de Matrice sont nécessaires dans les méthodes *Matrice conversion_vers_Matrice_adj()* et *Matrice conversion_vers_Matrice_inc()*.
Pour remédier à cela, nous avons pris la décision de regrouper les classes Matrice et Graphe dans un seul et même fichier nommé *GrapheMatrice*.

4.3 Algorithmes

Durant l'élaboration de notre module *Opérations sur les graphes*, certains algorithmes étaient plus délicats à implémenter que d'autres. Parmis ces algorithmes nous pouvons citer PERT, l'algorithme d'Edmond Karp soit la gestion de flots d'un

Graphe, le problème du postier chinois et l'algorithme de Little. Ces fonctions ont demandées de nombreuses heures de travail. PLus précisement dans le cas de l'algorithme de Little, la première difficulté éprouvée fut aussi de passer du pseudo-code à l'implémentation, en prenant en compte nos objets. En outre, il a été difficile de par sa compréhension et des itérations à réaliser.
L'écriture des algorithmes de PERT et d'Edmond Karp en pseudo-code avait été réalisée pour le rendu du Cahier des Charges. Ceux-ci restaientt cependant difficile à retranscrire en code étant donné la complexité de ces derniers. L'autre difficulté de l'algorithme d'ordonnancement était l'accès difficile aux objets depuis la fonction à cause de nombreuses encapsulations.

4.4 Interface graphique

L'algorithme de placement inspiré de ForceAtlas2 à été difficile à implémenté de par sa complexité et aussi du au fait que le code de base n'était pas fait pour coder cet algorithme. C'est pour ça que nous n'avons pas implémenté ForceAtlas2 mais un algorithme qui s'en rapproche prenant en compte les spécificités de notre implémentation.

Un autre point fut particulièrement délicat a implémenter : la modélisation de graphe. En effet les objets fournit par Qt bien que très pratique sont parfois complexe. Les éléments qui posèrent le plus de problème furent la QGraphicsScene et la QGraphicsView à cause des positions différentes entre chacune compliquées à appréhender dans un premier temps. De la même façon le boudingRect des arcs principalement fut compliqué à mettre en place car il était compliqué de se représenter graphiquement à l'avance ce que produirait les fonctions. De nombreux problèmes ont été soulevés lors de la compilation des tests avec Qt, de plus leur écriture particulièrement chronophage et pas forcément représentative nous ont mené à leur préférer les tests graphiques de visu principalement pour MainWindow, MODialog ou OCDialog, l'utilisation de QDebug a aussi permis de gagner du temps.

5 Changements mineurs sur les spécifications

Durant la phase de programmation, nous avons décidé d'ajouter ou de changer les signatures de certaines fonctions, pour des raisons d'optimisation ou de non répétition de code. Par ailleurs, certains cas de tests ont été modifiés ou rajoutés pour couvrir un plus grand nombre de cas et anticiper le plus de bugs possibles. Enfin certaines fonctions du module Interface graphique ont été déplacées d'un fichier vers un autre ou dans les *public slots* afin que les données du graphe puissent être récupérées pour être traitées.

5.1 Changements de signature

- Nous avons tout d'abord modifié les surcharges d'opérateurs avec le retrait d'un des deux paramètres dans l'ensemble de nos classes.

  bool operator==(Sommet const S1)const
  bool operator==(Arc constA) const
  bool operator==(Matrice const M1)const
  bool operator==(Graphe const G1) const

  bool operator!=(Arc constA)
  bool operator!=(Sommet const S1)
  bool operator!=(Matrice M1)
  bool operator!=(Graphe const G1)

  Sommet operator=(Sommet const S1)
  Arc operator=(Arc constA1)
  Matrice operator=(Matrice const M1)
  Graphe operator=(Graphe const G1)

- Dans la fonction de gestion de fichiers *verif_file*, *path* est le chemin du fichier à vérifier. Nous avons réalisé ce changement car une variable de type Document ne peux être passé en argument. Pour pallier à cela, nous avons remplacé dans les paramètres le document par son chemin ce qui nous permet de récupérer le document à l'intérieur de la fonction et de travailler dessus. La fonction *verif_file* devient:

bool verif_file(string path)

- Nous avons aussi modifié la fonction *couleur_adjacente* pour qu'elle renvoie en plus du nombre de voisins colorés la liste des couleurs présente dans ces sommets. Nous avons procédé à ce changement pour des raisons d'optimisation. En effet, pour la résolution du problème de coloration de graphe, nous devions dans un premier temps calculer le nombre de voisins colorés et puis, un peu plus loin dans la fonction, calculer les couleurs des voisins pour choisir la couleur du sommet étudié. Nous avons donc décidé de regrouper ces deux parties pour éviter une répétition et optimiser le code puisque pour chaque sommet à traiter, la matrice du graphe sera parcourue une fois au lieu de deux. Nous avons donc:

  pair<int, vector<int>> couleur_adjacente(int id, vector<int> v, Matrice M)

  La fonction prend en entrée l'ID du sommet sur lequel on veut travailler, un vecteur v dans lequel est renseigné l'ensemble des couleurs des sommets du graphe (mis à 0 si le sommet n'a pas encore été coloré, un entier compris entre 1 et n, avec n le nombre de couleurs déjà utilisées, qui représente le numéro de la couleur) et enfin une matrice d'adjacence représentant le graphe.

  En sortie de cette fonction, nous récupérons une paire contenant un entier et et un vecteur d'entier. L'entier représente le nombre de voisins colorés et le vecteur contient l'ensemble des couleurs des voisins triées par ordre croissant.

- Dans la partie graphique, nous avons changé les paramètres des fonctions *QZoneDeDessin::Afficher_Sommet(Sommet s)* et *QZoneDeDessin::Afficher_Arc(Arc a)*. Nous avons eu besoin de l'entièreté de l'objet et non seulement de son ID pour pouvoir afficher correctement l'objet avec les bonnes coordonnées ainsi que le reste de ces attributs.

5.2 Ajout de fonctions

- Tout d'abord, nous avons ajouté l'opérateur d'égalité aux deux structures que nous avions crées (VectVal et pert_row). Nous avions besoin de pouvoir comparer facilement ces structures entre elles, d'une part pour faciliter la comparaison entre deux sommets ou deux arcs et par extension deux graphes et d'autre part pour faciliter les tests. Nous avons aussi ajouté l'opérateur d'affectation pour la structure *VectVal* car sans elle nous ne pouvions affecter directement un **VectVal** et nous devions passer par chacun des paramètres. L'ajout de cet opérateur nous a donc permis une optimisation du code.

  VectVal operator=(VectVal v2)

  inline bool operator== (VectVal v1, VectVal v2)

  exinline bool operator== (pert$_r$*owv*1, *pert$_r$owv*2)

- Nous avions oublié d'indiquer la fonction *QArc::paint* dans les spécifications. Il s'agit d'une fonction primordiale qui permet l'affichage des Arcs sur la zone de dessin. Nous l'avions cependant prévue puisque le test de cette fonction était déjà défini.

  *void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)*

  La fonction prend en paramètres *painter*, un pointeur sur un QPainter qui permet le dessin de l'arc, *option* un pointeur constant sur un QStyleOptionGraphicsItem et enfin un pointeur sur un QWidget représentant la fenêtre de dessin.

- Nous avons ajouté des constructeurs de copie dans les classes QSommet et QArc. Nous avons eu la nécessité d'ajouter ces deux fonctions car certains des constructeurs des bibliothèques QT en avaient besoin. Nous avons donc:

  QArc (QArc const A)

  Ce constructeur prend en paramètre le QArc à copier et renvoie une copie de ce dernier.

  QSommet(QSommet const S)

  Ce constructeur prend en paramètre le QSommet à copier et envoie une copie de ce dernier.

- La fonction *void mouseDoubleClickEvent(QGraphicsSceneMouseEvent *event)* a aussi été ajoutée dans la classe QSommet afin de permettre le déplacement des sommets dans la zone de dessin.

- Enfin, la fonction *QZoneDeDessin::distanceForce(QSommet a, QSommet b)* a été complété la liste de nos fonctions. Elle est appelée au cours de la fonction QZoneDeDessin::placementSommet() et permet à deux sommets de ne pas être supperposés.

6 Comparaison entre l'estimation et la réalité

Gestion de graphe		
Suppression de graphes	estimation: 50 lignes implémentation: 4 lignes	estimation: 1h implémentation: 5 min
Création de graphes	estimation: 150 lignes implémentation: 140 lignes	estimation: 3h implémentation: 15h
Modification de graphes	estimation: 300 lignes implémentation: 213 lignes	estimation: 3h implémentation: 10h
Duplication de graphes	estimation: 100 lignes implémentation: 28 lignes	estimation: 1h implémentation: 30 min
Conversion de la liste de voisins en graphe	estimation: 50 lignes implémentation: 14 lignes	estimation: 1h implémentation: 30 min
Conversion de graphe en liste de voisins	estimation: 50 lignes implémentation: 7 lignes	estimation: 5h implémentation: 1h
Conversion de la matrice d'incidence en graphe	estimation: 30 lignes implémentation: 32 lignes	estimation: 1h implémentation: 1h
TOTAL	estimation: 730 lignes implémentation: 438 lignes	estimation: 11h implémentation: 28h05min
Système de gestion de fichiers		
Suppression de fichiers	estimation: 100 lignes implémentation: 3 lignes	estimation: 1h implémentation: 10m
Chargement de fichiers	estimation: 200 lignes implémentation: 145 lignes	estimation: 2h implémentation: 10h
Sauvegarde de fichiers	estimation: 200 lignes implémentation: 163 lignes	estimation: 2h implémentation: 3h
Sauvegarde avancée de fichiers	estimation: 100 lignes implémentation: 155 lignes	estimation: 1h implémentation: 15m
TOTAL	estimation: 600 lignes implémentation: 466 lignes	estimation: 6h implémentation: 13h25m

Opérations sur les graphes		
Coloration de graphe	estimation: 150 lignes implémentation: 159 lignes	estimation: 3h implémentation: 20h
Détection de stables	estimation: 20 lignes implémentation: 17 lignes	estimation: 1h implémentation: 30 min
Détection de cliques	estimation: 20 lignes implémentation: 14 lignes	estimation: 1h implémentation: 15 min
Ordonnancement	estimation: 250 lignes implémentation: 326 lignes	estimation: 4h implémentation: 35h
Calcul des plus courts chemins	estimation: 200 lignes implémentation: 168 lignes	estimation: 4h implémentation: 6h
Arborescence et Anti-Arborescence	estimation: 100 lignes implémentation: 128 lignes	estimation: 3h implémentation: 3h
Chaînes euleriennes	estimation: 100 lignes implémentation: 87 lignes	estimation: 3h implémentation: 5h
Chaînes hamiltoniennes	estimation: 100 lignes implémentation: 58 lignes	estimation: 3h implémentation: 30min
Calcul des degrés d'un sommet	estimation: 20 lignes implémentation: 20 lignes	estimation: 1h implémentation: 30min
Calcul des voisins d'un sommet	estimation: 20 lignes implémentation: 13 lignes	estimation: 3h implémentation: 30min
Résolution du problème du postier chinois	estimation: 150 lignes implémentation: 144 lignes	estimation: 4h implémentation: 10h
Résolution du problème de Little	estimation: 200 lignes implémentation: 477 lignes	estimation: 4h implémentation: 60h
Recherche de la connexité	estimation: 100 lignes implémentation: 69 lignes	estimation: 2h implémentation: 1h30
Gestion des flots	estimation: 200 lignes implémentation: 125 lignes	estimation: 3h implémentation: 8h
TOTAL	estimation: 1630 lignes implémentation:	estimation: 39h implémentation: 150h 45 min
Interface graphique		
Affichage des graphes et des menus	estimation: 1800 lignes implémentation:2000 lignes	estimation: 20h implémentation:26h
Positionnement du graphe	estimation: 600 lignes implémentation: 116 lignes	estimation: 6h implémentation: 35h
Affichage des données	estimation: 200 lignes implémentation:20 lignes	estimation: 4h implémentation:1h
Modification des graphes	estimation: 100 lignes implémentation:150 lignes	estimation: 4h implémentation:5h
Déplacement des éléments	estimation: 500 lignes implémentation:15 lignes	estimation: 5h implémentation: 20 minutes
Sélection des éléments	estimation: 100 lignes implémentation:60	estimation: 3h implémentation: 2h
TOTAL	estimation: 3300 lignes implémentation: 2361 lignes	estimation: 42h implémentation: 69h 20 minutes

7 Conclusion technique

Le produit final de notre projet réponds bien aux objectifs attendus : la manipulation de graphes orientés et l'application d'algorithmes sur ceux-ci. Pour cela notre projet est divisé en deux parties, l'interface graphique et la bibliothèque. Nous avions choisis comme langage le C++, choix qui s'est avéré satisfaisant puisque nous n'avons eu aucun problème de ce côté là lors de l'implémentation.

Notre application qui est donc fonctionnelle n'est cependant pas sans défauts. Certaines améliorations pourraient êtres réalisées, telles que la gestion d'erreurs qui se fait actuellement individuellement dans chaque fonction, mais pourrais être regroupé dans un même fichier.

Pour améliorer notre application, l'ajout de nouveaux algorithmes applicables sur les graphes orientés est une possibilité.

On peut aussi envisager d'étendre la méthode PERT en ajoutant des contraintes de ressources. La représentation graphique du déroulement des algorithmes, étapes par étapes, peut aussi être réalisé, ce qui permettrait par exemple a des étudiants d'avoir une première approche de la théorie des graphes.

Il subsiste très certainement dans notre application des bugs que nous n'avons pas encore rencontrés. Cependant nous avons pu en consigner quelques-uns qui font partie de ce qu'il faut corriger pour améliorer notre projet. Il y a notamment:

- La fonction de placement de sommets qui fonctionne bien pour les sommets n'étant pas reliés par des arcs mais dysfonctionne lorsque les sommets sont reliés et qu'ils sont relativement éloignés l'un de l'autre car la force d'attraction devient trop forte.
- La fonction de création de graphe PERT ne produit le résultat attendu graphiquement parlant.
- Parfois (sans qu'un cas type puisse être isolé) il est impossible de créer des arcs entre deux sommets
- L'utilisation de l'algorithme de Bellman avec aucune séléction de sommet fait crasher l'application.
- L'algorithme de Bellman n'affiche pas les bons résultats malgré qu'il passe son test.
- L'algorithme de résolution du postier chinois produit un segmentation fault quand on l'utilise graphiquement malgré qu'il passe son test haut la main.
- La recherche de chaine hamiltonienne de fonctionne pas graphiquement malgré la validation de son test unitaire.
- L'implémentation de l'algorithme de Little, pour la résolution du voyageur de commerce comporte des problèmes de résolution. Malgré la bonne réalisation du pseudo-code, il y a encore des problèmes d'implémentation liés à l'analyse du résultat obtenu. Il pourrait aussi être optimisé pour gagner de la légèreté en terme de lignes de code.

8 Conclusion sur l'organisation interne du projet

Tout au long de ce projet de dernière année de licence informatique, nous avons travaillé en groupe de 9, ce qui est une première pour la majorité des membres du groupe. Cette expérience nous a permis de nous regrouper plusieurs fois par semaine, que ce soit en présentiel, ou à distance si la situation ne le permettait pas, afin de mettre nos idées en commun et débattre autour de notre sujet. Pendant la période de confinement, nous avons réussi à garder des points quasiment quotidiens ce qui a permis d'apporter de l'aide le plus tôt possible à ceux qui en avaient besoin et ainsi d'avancer rapidement.

Lors des premières réunions, nous avons pris la décision de ne pas avoir de chef de projet afin que tous les membres aient autant de responsabilités et que chacun puisse donner son avis. Cette décision s'est avérée efficace, puisque la réalisation du projet dans l'ensemble s'est bien déroulé. En effet, chacun des membres du groupe a pris ses responsabilités, permettant ainsi au projet d'avancer dans la bonne direction tout en gardant une entente amicale.