Projet IN608

Manipulation de graphes orientés

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

2019 - 2020

Table des matières

1 Fondement du projet

1.1 Historique

1.1.1 Personnages importants

— **Euler (1707-1783)** : Père de la théorie des graphes.
— **Al-Adli (IXe siècle)** : Théoricien arabe, créateur du problème du cavalier.
— **Arthur Cayley (1821 - 1895)** : Mathématicien britannique de la fin du XIXème siècle. Il travaillera sur le sujet des arbres en théorie des graphes et sur le problème de coloration des cartes.

1.1.2 Rappels théoriques

Un graphe est un ensemble d'objets appelés sommets qui sont mis en relation. Il existe plusieurs types de graphes, les graphes étudiés au cours de ce projet seront les *graphes orientés*.

Dans un graphe orienté, les sommets sont reliés entre eux par des arcs de manière asymétrique. Cela signifie que ces arcs ne sont empruntables que dans un unique sens. Un graphe orienté est représenté sous forme de diagramme où des ensembles de points représentant les sommets sont joints entre eux par des flèches représentant les arcs.

Dans un graphe, il existe le concept de sous-graphe partiel qui est un sous-ensemble de sommets d'un graphe nommé G. C'est-à-dire un nouveau graphe formé de certains sommets sélectionnés, des arcs reliant ces sommets, ainsi que d'arcs qui sont également choisis.

1.2 But du projet

Notre objectif est de créer une bibliothèque logicielle permettant à un développeur tiers de manipuler les graphes orientés sans avoir à les ré-implémenter. Cette bibliothèque permettra donc une manipulation aisée des graphes orientés dans d'autres programmes, dit consommateurs, grâce à une interface logicielle. Pour accompagner la bibliothèque nous allons aussi créer une application graphique de création de graphes permettant de modéliser toutes sortes de graphes orientés et d'y appliquer des algorithmes. Cette application permettra à un utilisateur lambda de créer des graphes orientés et de les manipuler.

1.2.1 Périmètre du projet

Notre projet s'articule à travers deux périmètres, celui de l'application et celui de la bibliothèque.
La bibliothèque apportera les fonctionnalités suivantes :
— Créer, supprimer et modifier des graphes
— Les algorithmes qui pourront être appliqués sur les graphes.

L'application doit permettre de :
— Dessiner graphiquement les graphes
— Sélectionner graphiquement les sommets d'un graphe pour en faire un sous-graphe partiel
— Choisir quel algorithme appliquer au graphe
— Lancer la sauvegarde
— Sauvegarder les graphes et de les charger

1.3 Utilisateurs du produit

Les utilisateurs finaux du produit vont différer selon l'usage. En effet, pour la bibliothèque le client final est un développeur tiers qui utilise notre travail dans le sien. Dans le cas de cette application, l'utilisateur final est une personne lambda , c'est à dire sans compétences informatiques particulières, qui souhaite modéliser un problème de graphe orienté.

2 Contraintes sur le projet

2.1 Contraintes imposées

— La solution doit pouvoir s'appliquer à tout graphe orienté. C'est pourquoi il est nécessaire de pouvoir générer un ensemble de sommets reliés par des arcs.
— Il est nécessaire de pouvoir manipuler un graphe orienté mais aussi ses sous-graphes partiels. Ici, les graphes employés seront visibles via l'application mais seront abstraits. C'est-à-dire qu'ils seront représentés par leur matrice d'adjacence.
— Chaque sommet devra comporter un numéro d'identification et deux coordonnées dans le plan et pourront porter un nombre quelconque de réels et d'entiers.
— Les arcs du graphes pourront porter un nombre quelconque de réels et d'entiers.
— L'affectation entre graphes ou sous-graphes partiels doit être possible

2.2 Glossaire et convention de dénomination

— Sommet : un point du graphe. Dans ce document, soit s un sommet et V l'ensemble de tous les sommets.
— Arc : relation entre deux sommets consécutifs. Il a obligatoirement un sens. C'est-à-dire un point de départ et un point d'arrivée. Soit a un arc et E l'ensemble des arcs.
— Arc entrant : arc entrant vers un sommet.
— Arc sortant : arc sortant d'un sommet.

— Voisins : deux sommets sont voisins s'ils sont reliés par un arc.
— Successeur : t est successeur de s si il existe un arc entre les sommets s et t et que t se trouve à l'extrémité finale de l'arc.
— prédécesseur : s est prédécesseur de t si il existe un arc entre les sommets s et t et que s se trouve à l'origine de l'arc.
— Graphe : ensemble de sommets qui sont mis en relation par des arêtes formant un réseau.
— Graphe orienté : les sommets d'un graphe orienté sont reliés entre eux par des arcs qui ont un sens.
— Sous-graphe : graphe ayant pour sommets un sous-ensemble V des sommets d'un autre graphe G et pour arcs uniquement ceux de ce même graphe G.
— Sous-graphe partiel : graphe ayant pour sommets un sous-ensemble V des sommets d'un graphe G et pour arcs un sous-ensemble de ceux de G joignant.
— matrice : tableau de nombres permettant d'interpréter en termes calculatoires ou opérationnels des résultats.
— matrice d'adjacence : matrice qui représente le graphe en indiquant si il existe un lien entre deux sommets.
— matrice d'incidence : c'est une matrice qui décrit le graphe en indiquant a quels sommets sont liés les arcs.
— un arbre : graphe particulier dont la racine est unique, ses sommets adjacents droits et gauches sont toujours de degrés supérieurs à 1 et ses feuilles sont de degré 1. Ici, dans l'univers des graphes orientés, on parlera d'arborescence ou d'anti-arborescence pour préciser le sens des branches de l'arbre.
— Mandelbrot : Programme permettant de dessiner un ensemble de Mandelbrot. Ce sont des fractales.
— n-body : "Le problème à N corps consiste à résoudre les équations du mouvement de Newton de N corps interagissant gravitationnellement, connaissant leurs masses ainsi que leurs positions et vitesses initiales. Par extension cette appellation a été conservée dans le cas où l'on s'intéresse à un ensemble de particules liées par un potentiel quelconque."[9].

3 Exigences fonctionnelles

3.1 Portée du travail

Cette application doit permettre d'appliquer un ensemble d'algorithmes de théorie des graphes à un des cas d'utilisation abstrait ou concret. Abstrait dans un cas de génération aléatoire d'un graphe, concret dans un cas de génération d'un graphe à partir d'une matrice pré-existante.

3.2 Exigences fonctionnelles et exigences sur les données

— Interface graphique et mise en place d'un drag & drop. Cette interface permet à l'utilisateur de choisir parmi les fonctionnalités de l'application et d'avoir un aperçu des graphes. Sauvegarde de fichiers Une sauvegarde de la matrice d'adjacence dans un fichier externe pourra être effectué si voulu. Chacun des résultats des modules pourra également être enregistré dans un fichier.
— Création de graphes L'utilisateur a la possibilité de créer un graphe en ajoutant directement les sommets et les arcs. Il peut aussi importer une matrice d'adjacence (de taille N x N) ou bien une liste de voisins déjà remplie ou en créant directement une matrice d'adjacence sur l'interface graphique. Création de sous-graphes Ce module permet à l'utilisateur de sélectionner des sommets pour créer un nouveau graphe qui sera le sous-graphe contenant ces sommets et les arcs reliants ces sommets.
— Modification de graphes Il sera possible de modifier les graphes en supprimant ou en ajoutant un sommet. Il sera aussi possible de déplacer, ajouter ou supprimer des arcs. Les noms, valeurs et coordonnées pourront aussi être affichés, effacés et modifiés.
— Graphes aléatoires Des matrices de tailles et valeurs aléatoires pourront être créées et affichées sous forme de graphe. Les coordonnées de chacun des sommets seront calculées à l'aide d'un algorithme.
— Suppression de graphes. Les matrices pourront être supprimées, le graphe disparaîtra de l'interface graphique le cas échéant.

4 Exigences non fonctionnelles

4.1 Interface utilisateur

Au niveau de l'interface utilisateur, une distinction devra être prise en compte entre celle destinée à l'utilisateur et l'usage de la bibliothèque.

En effet, l'interface entre l'application et l'utilisateur sera une interface graphique permettant à une personne sans connaissance en informatique de modéliser un graphe orienté et d'appliquer des algorithmes sur ce graphe.

Grâce à l'interface graphique l'utilisateur pourra naviguer dans son système de fichiers pour sélectionner un fichier sauvegardé.

Cependant, l'interface de programmation de la bibliothèque vise un public de développeurs et offrira la possibilité de l'utiliser pour la production de leur application.

4.2 Performance

L'application devra pouvoir gérer des graphes possédant un grand nombre de sommets (Maximum de *Force Atlas 2 = 1 Million*) ainsi que plusieurs graphes à la fois.

Nous exigerons également un temps de réponse équivalent à la complexité de l'algorithme. Notre application devra être fluide quand nous effectuons des actions au sein de celle-ci. Les fichiers seront composés majoritairement de la matrice ce qui signifie que la taille de nos fichiers sera plus ou moins équivalente à la taille de la matrice contenue dans le fichier, le reste sera consacré à d'autres données.

4.3 Adéquation du produit avec son environnement

La bibliothèque devra nous permettre d'appliquer des algorithmes sur des graphes. La modification et la suppression de graphes seront également possibles. L'utilisateur devra avoir accès à internet afin de pouvoir télécharger la bibliothèque pour l'utiliser, ainsi que de télécharger l'application s'il souhaite avoir accès à l'interface graphique.

4.4 Maintenance, support, portabilité, installation du produit

Ce projet est réalisé dans le cadre universitaire, de ce fait nous allons mener une maintenabilité minime, sauf pour des résolutions de bugs mineurs et l'ajout d'algorithmes. Les différentes améliorations seront apportées par l'utilisateur s'il souhaite réaliser des modifications en fonction de ses besoins.

4.5 Sécurité

Notre application n'aura accès qu'aux fichiers désignés par l'utilisateur et ne procède à aucune récolte de données.

4.6 Mentions légales, politiques et culturelles

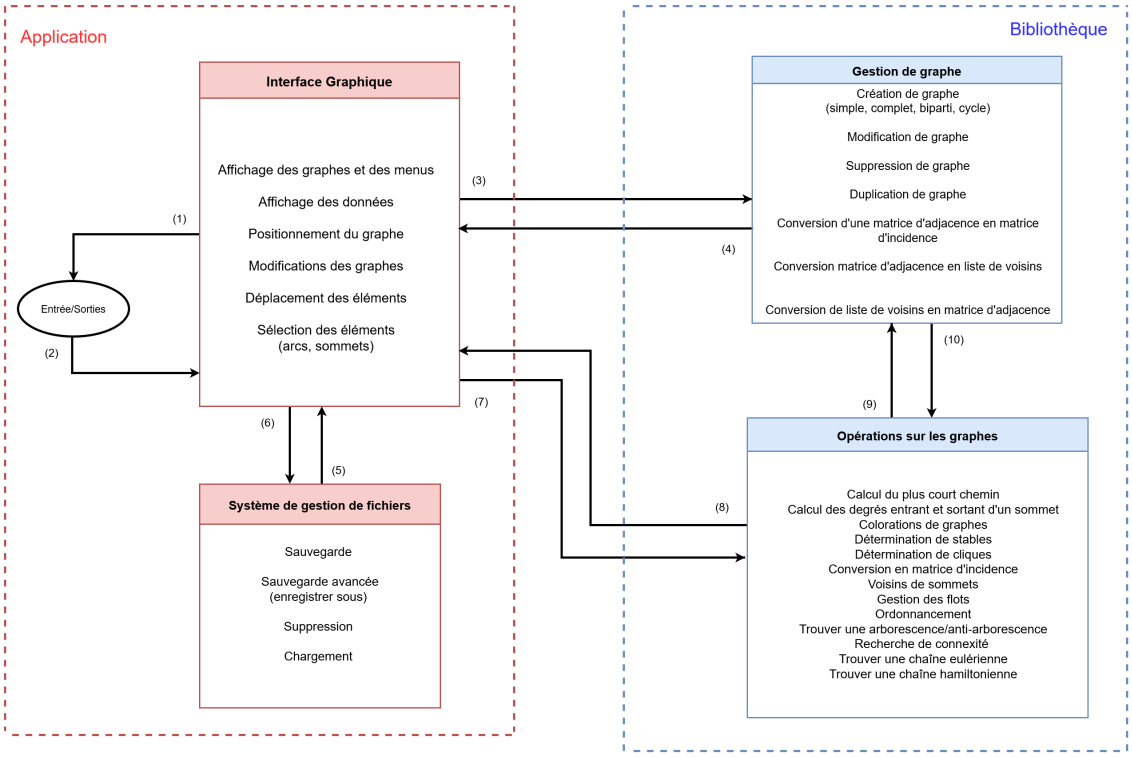
Les fichiers gérés par l'application n'étant pas des données sensibles et comme notre application ne récolte aucune information, aucune loi particulière n'est à appliquer.

La documentation de la bibliothèque et de l'application seront écrites en français car cette application est produite pour des utilisateurs francophones.

Le code sera open source et en licence MIT [15].

5 Modules

5.1 Organigramme



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

5.2 Fonctionnalités et algorithmes associés

5.2.1 Gestion des graphes

- **Création de graphe :** Créer un graphe qui peut être aléatoire, d'un type spécifique (complet, biparti ou cycle) ou à partir d'une liste de voisins fournie en amont.
- **Modification de graphe :** Permet de modifier un graphe, en rajoutant un sommet par exemple.
- **Duplication de graphe :** Copie un graphe.
- **Suppression de graphe :** Supprimer un graphe précédemment créée.
- **Conversion d'une matrice d'adjacence en liste de voisins :** Permet à partir d'une matrice d'adjacence de créer une liste de voisins et inversement.
- **Conversion en matrice d'incidence :** Conversion d'une matrice d'adjacence en matrice d'incidence[2] : Prend en paramètre une matrice d'adjacence M1 et permet de la convertir dans la matrice d'incidence M2 équivalente en $O(E^2)$.

5.2.2 Système de fichiers

- **Sauvegarde :** Permet de sauvegarder dans un fichier l'ensemble des informations propres au graphe courant, son nom, les positions de ses sommets, son nombre de sommets, leur nom, le sens des arcs, leur destination et leur poids.
- **Sauvegarde avancée (enregistrer sous) :** Permet de sauvegarder le graphe courant sous un nouveau nom et à un autre emplacement.
- **Chargement :** Permet de charger un fichier dans lequel est sauvegardé un graphe et de l'afficher.
- **Suppression :** Permet de supprimer un fichier dont le lieu de stockage est placé en paramètre.

5.2.3 Interface graphique

- **Affichage des graphes et des menus :** Affiche à l'écran les différents menus et le dessin en cours.
- **Affichage des données :** Afficher le nom des sommets et le poids des arcs.
- **Positionnement du graphe :** Permet dans le cas où le graphe est aléatoire de placer les sommets de façon à ce qu'ils ne se superposent pas, mais aussi, de réorganiser à la demande de l'utilisateur les positions de ses sommets de façon à maximiser la lisibilité du graphe courant. Un algorithme de force du nom de Force Atlas 2 [10] ayant une complexité de $O(V * log(V))$ sera utilisé.
- **Création et modifications des graphes :** Permet à l'utilisateur de créer un graphe (en ajoutant des sommets ou des arcs), d'affecter le poids des arcs ou de nommer les sommets ou de changer le poids des arcs ou le nom des sommets.
- **Déplacement des éléments :** Permet à l'utilisateur de déplacer à l'aide de son curseur les points et de changer les destinations et le départ des arcs.
- **Sélection des éléments (arcs, sommets) :** Permet de sélectionner un certain nombre de sommets et les arcs qui y sont liés pour pouvoir travailler sur un sous-graphe du graphe courant.

5.2.4 Opérations sur les graphes

Ces algorithmes ont été choisis car ils sont plus optimaux.

- **Calcul du plus court chemin :** Permet d'appliquer l'algorithme de Floyd-Warshall [5], qui permet de trouver l'ensemble des plus courts chemins en $O(V^3)$. On préférera Ford-Bellman[6] pour le plus court chemin entre un sommet et tous les sommets en $O(V^2)$ plutôt que Dijkstra car il permet la présence d'arcs de poids négatifs .
- **Calcul des degrés entrant et sortant d'un sommet** Permet de calculer le degré d'un sommet avec une complexité de $O(V^2)$.
- **Coloration de graphe, détermination de stables et de cliques :** Pour trouver une coloration du graphe mis en entrée, l'algorithme de DSATUR [1] va être ici utilisé. Il est en $O(V^2)$ et donne une coloration moins mauvaise dans le pire des cas que celui de Welsh-Powell. Pour trouver les stables, le même algorithme sera utilisé. Quant aux cliques, la matrice du graphe sera inversée puis l'algorithme de DSATUR sera appliqué. Ce sera donc en $O(V + V^2)$.

Algorithme 1 : Conversion d'une matrice d'adjacence en matrice d'incidence

```
Entrées : Matrice d'adjacence A du graphe G
Sorties : Matrice d'incidence I du graphe G
pour toutes les lignes de A faire
    pour toutes les colonnes de A faire
        si il existe un arc entre le sommet s et un sommet voisin t alors
            Ajouter une colonne dans la matrice I puis mettre 1 pour la ligne de s ainsi que -1 pour la ligne de t
        sinon
            Ajouter 0 dans la matrice I entre s et t et t et s
```

— **Gestion des flots :** Permet grâce à l'algorithme d'Edmond-Karp [7] de résoudre le problème de flot maximal sur le graphe placé en entrée en $O(V * E^2)$. Il a été choisi car c'est une spécialisation d'algorithme de Ford-Fulkerson. Il est plus efficace dans le cas de graphe denses.
— **Ordonnancement :** Permet, à partir d'un formulaire rempli par l'utilisateur, de créer un diagramme de PERT, avec le calcul des tâches et du chemin critique en $O(V^2)$.
— **Trouver l'arborescence/anti-arborescence :** Permet de trouver une arborescence couvrante du graphe G et l'anti-arborescence si on l'applique sur le graphe inverse de G en $O(V * E)$.
— **Recherche de connexité :** Permet de vérifier la connexité du graphe grâce à l'arborescence et à l'anti-arborescence en $O(V * E)$.
— **Trouver une chaîne eulérienne :** Permet de trouver les chaînes eulériennes du graphe placé en entrée en $O(V!)$.
— **Trouver une chaîne hamiltonienne :** Permet de trouver les chaînes hamiltoniennes du graphe placé en entrée en $O(V^2)$.
— **Résoudre le problème du voyageur de commerce :** Permet de résoudre le problème du voyageur de commerce grâce à l'algorithme de Little [8] en $O(V^2)$ sur un graphe placé en entrée.

Algorithme 2 : Calcul d'un flot maximal - algorithme d'Edmond-Karp

Entrées : graphe G, s la source et p le puits
Sorties : Flot maximum F
F est fixé à 0.
tant que *il existe un chemin améliorant* **faire**
  Un parcours en largeur à partir du sommet s
  si *il existe un chemin améliorant entre s et p* **alors**
    - on ajoute la valeur du chemin améliorant au flot maximum
    - on met à jour les valeurs résiduelles de chacun des arcs du chemin améliorant
  sinon
    On renvoie F

Algorithme 3 : Calcul de degrés entrants et sortants d'un sommet

Entrées : Sommet s, Graphe G
Sorties : Degrés entrants, Degrés sortants
Initialisation du successeur et du prédécesseur à 0;
pour *toutes les lignes de la matrice de G* **faire**
  pour *toutes les colonnes de la matrice de G* **faire**
    si *il existe un arc entre s et un autre sommet du graphe G* **alors**
      Augmenter le nombre de successeurs
    si *il existe un arc entre un sommet du graphe G et le sommet s* **alors**
      Augmenter le nombre de prédécesseurs

Algorithme 4 : Ordonnancement (PERT)

Entrées : nom et durée des tâches, contraintes d'antériorité
Sorties : graphe avec date au plus tôt et au plus tard, tâches critiques
pour *toute tâche t* **faire**
  pour *tout antécédent n de t* **faire**
    ajouter t aux successeurs de n.
Créer un sommet "départ" et lui affecter la valeur 0 comme date au plus tôt.
pour *toute tâche t n'ayant aucune contrainte d'antériorité* **faire**
  Créer le sommet t et un arc a allant de départ vers t.
  Affecter à a la durée de t.
  La date au plus tôt de t est égale à la durée de t.
tant que *il existe une tâche non traitée* **faire**
  pour *toute tâche t ayant une unique contrainte d'antériorité sur une tâche x déjà traitée* **faire**
    Créer le sommet t et un arc a allant du sommet x vers s.
    Affecter à a la durée de t.
    La date au plus tôt de t est égale à la date au plus tôt de x plus la durée de t.
  pour *tout tâche t ayant plus d'une contrainte d'antériorité sur des tâches x1, x2, ..., xn toutes déjà traitées* **faire**
    si *la postériorité du xi (avec $1 \leq i \leq n$) ayant la valeur au plus tôt la plus élevée est égal à 1* **alors**
      Créer le sommet t et un arc allant du sommet xi ayant la date au plus tôt la plus élevée au nouveau sommet.
      Créer un arc fictif entre chaque sommet antérieur et le sommet xi.
      Affecter à a la durée de t.
      La date au plus tôt de t est égale à la date au plus tôt de x plus la durée de t.
    sinon
      Créer un nouveau sommet x1 + x2 ... + xn = SS.
      Créer un arc fictif entre chaque sommet antérieur et le sommet SS.
      Affecter au nouveau sommet SS la valeur de la date au plus tôt du sommet xi le plus grand.
      Créer un nouveau sommet t et un arc a allant du sommet SS au sommet t.
      Affecter à a la durée de t.
      La date au plus tôt de t est égale à la date au plus tôt de SS plus la durée de t.

// Suite de l'algorithme d'ordonnancement (PERT) //
Créer un sommet "arrivée", créer un arc fictif entre tous les sommets n'ayant aucun successeur et ce sommet.
Affecter au sommet d'arrivée la valeur de la date au plus tôt la plus élevée de ses prédécesseurs, comme date au plus tôt et au plus tard.
pour *chaque sommet s* **faire**
    la date au plus tard de s est égale à la date au plus tard la moins élevée de ses successeurs moins la valeur de l'arc les rejoignant.
    si *la date au plus tôt est égale à la date au plus tard* **alors**
        s est une tâche critique

Algorithme 5 : Calcul du plus court chemin (Floyd-Warshall)

Entrées : Matrice d'adjacence A, Matrice père P
Sorties : Matrice avec les distances et la matrice P remplie
On recopie A dans A' ; **pour** *tout les sommets K de la matrice A'* **faire**
    pour *toutes les lignes S de la matrice A'* **faire**
        pour *toutes les colonnes T de la matrice A'* **faire**
            On compare les chemins direct entre S et T et les chemins utilisant K comme intermédiaire (S - K et K - T) ; **si** *le chemin intermédiaire est plus court que le chemin direct* **alors**
                Remplacement de ce chemin dans la matrice A' et K devient le sommet père de T dans la matrice P

Algorithme 6 : Trouver une arborescence de poids minimum

Entrées : Graphe G
Sorties : Arborescence de poids minimum
pour *tous les sommets de G* **faire**
    Prendre l'arc entrant de plus petit poids et soustraire son poids à chacun des arcs entrants
Créer un nouveau graphe contenant tous les sommets et les arcs de poids 0

Algorithme 7 : Coloration de graphe (DSATUR)

Entrées : Liste L des sommets du graphe G
Sorties : Liste des sommets et leur couleur associée
Ordonner L par ordre décroissant de degrés ; s = Sommet de degré maximum dans L ; Colorer s avec la couleur 1 **tant que** *tous les sommets de L ne sont pas colorés* **faire**
    s_0 = Sommet avec le DSAT le plus important dans L (Si égalité prendre le sommet de degrés maximum)
    Colorer s_0 avec la plus petite couleur possible (absent de son voisinage)
DSAT (sommet s) = nombre de couleurs différentes dans les sommets adjacents à s.

5.3 Coûts

Gestion de graphe		
Suppression de graphes	MALO Amandine	50
Création de graphes		150
Modifications de graphes	TEDESCHI Hugo	300
Duplication de graphes	GODINEAU Camille	100
Conversion de la liste de voisins en matrice d'adjacence et inversement	AIRIAU Vincent	100
Conversion matrice d'incidence du graphe	OILLO Sébastien	30
TOTAL		685
Opérations sur les graphes		
Coloration de graphe	GODINEAU Camille	150
Detection de stables		20
Detection de cliques		20
Ordonnancement		250
Calcul des plus courts chemins	MALO Amandine, PEUGNET Guillaume	200
Arborescence et Anti-Arborescence	MALO Amandine	100
Chaines eulerienne		100
Chaines hamiltonienne		100
Calcul des degrés d'un sommet	DAFRANE Salsabil	20
Calcul des voisins d'un sommet		20
Résolution du problème du postier chinois		150
Résolution du problème de Little	AIRIAU Vincent	200
Recherche de la connexité		100
Gestions des flots	TEDESCHI Hugo	200
TOTAL		1630

Gestionnaire de fichiers		
Suppression de fichiers	OILLO Sébastien	100
Chargment de fichiers		200
Sauvegarde de fichiers		200
Sauvegarde avancé de fichiers		100
TOTAL		600
Interface graphique		
Affichage des graphes et des menus	DE FREITAS Alexandre, NARDIN Théo, PEUGNET Guillaume	1800
Positionnement du graphe	DE FREITAS Alexandre, NARDIN Théo, GODINEAU Camille	600
Affichage des données	TEDESCHI Hugo	200
Modification des graphes	DAFRANE Salsabil	100
Déplacement des éléments	DAFRANE Salsabil, AIRIAU Vincent	500
Séléction des éléments	OILLO Sébastien	100
TOTAL		3300
NOMBRE TOTAL DE LIGNES DE CODE		6215

6 Autres aspects du projet

6.1 Salle d'attente : idées pour les futures versions

— Rajouter la possibilité à l'utilisateur de créer des algorithmes à appliquer sur ses graphes grâce à un système de "boites" à la façon de Scratch.
— Permettre l'ajout de graphes non orientés.

6.2 Choix du langage

Pour notre projet nous allons avoir besoin de gérer des arcs, des graphes et des sommets. Chacun de ces éléments devra pouvoir contenir un grand nombre de données différentes.
Le graphe par exemple devra contenir un ensemble d'arcs et de sommets, en plus d'autres de ses caractéristiques. Chacun de ces objets possédera aussi des fonctionnalités qui leur seront propres, comme la création, la suppression, le déplacement ou la modification de ses caractéristiques. Pour cela, l'usage d'un langage orienté objet est plus adapté. En effet, les structures de données présentes dans les langages procéduraux comme le C sont trop simples et ne nous donneraient pas la même aisance pour modifier en temps réel les graphes. Nous avons besoin d'objets initialisés avec des paramètres différents suivant l'utilisation demandée. Cela nécessite le polymorphisme et donc un langage orienté objet. De plus ces objets pourront être manipulés par les algorithmes qui nécessitent un paradigme procédural impératif. En résumé nous avons besoin d'un langage multi-paradigme, qui permet d'utiliser autant des notions procédurales et des notions d'objets telles que les classes.

L'autre chose importante dont a besoin notre application est la performance. En effet notre application doit permettre à l'utilisateur de créer des graphes composés de nombreux sommets et arcs sans que cela n'affecte de manière significative les performances de l'application. Ce qui nous amène à comparer les performances des langages. Le C++ est plus indiqué que le python (qui permet lui aussi le multi-paradigmes) pour des raisons de performance. D'après le projet "The Computer Language Benchmarks Game" le C++ met 1.51 secondes à exécuter le programme mandelbrot contre 259.50 secondes pour le python[11]. De la même façon, le programme n-body met 7.70 secondes à être exécuté en C++ contre 865.18 secondes en python[11]. De manière générale le C++ est plus rapide que les autres langages. En effet, si on consulte les autres benchmark[11][12][13][14] présents sur le site nous pouvons observer que le C++ est parfois plus rapide que le C ou que le Rust et est nettement plus rapide que le java et le Python.

Cela nous amène donc à choisir le C++ comme langage pour développer notre application et la bibliothèque.

7 Conclusion

Notre projet vise à la création d'une bibliothèque et d'une application permettant la manipulation de graphes orientés et de sous-graphes partiels.
Il était demandé de permettre la création d'un nombre arbitraire de graphes, de les manipuler et créer des sous-graphes partiel. Pour répondre à ces attentes, nous allons réaliser une bibliothèque regroupant les méthodes permettant les manipulations basiques sur les graphes et différents algorithmes à appliquer sur ces derniers. Une application permettra, par le biais d'une interface graphique, à l'utilisateur de dessiner des graphes ou de sélectionner des sous-graphes et d'y appliquer les algorithmes présents dans la bibliothèque.

8 Références

8.1 Algorithmes

1 Daniel Brélaz, New methods to color the vertices of a graph, Communication of ACM, Avril 1979
2 Phillipe Lacomme, Christian Prins, Marc Sevaux ; algorithmes de graphes ; Eyrolles ; 2èm édition 2003
3 Cormen, Leiserson, Rivest, Stein ; Algorithmique ; Dunod ; 3èm édition 2010
4 (Algorithme de conversion de la matrice d'adjacence vers matrice d'incidence) Mathilde Hurand, Algorithmique — M1 TD 1 : Graphes et représentations, http ://www.lix.polytechnique.fr/ hurand/Cours/Master1Algo/TD1.pdf, 2007
5 (Algorithme de Floyd-Warshall) Bernard Roy, « Transitivité et connexité. », C. R. Acad. Sci. Paris, vol. 249, 1959, p. 216–218
6 (Algorithme de Ford-Bellman) Richard Bellman, On a routing problem, Quarterly of Applied Mathematics, vol. 16, 1958, p. 87–90
7 (Algorithme d'Edmond Karp) Jack Edmonds et Richard M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, Journal of the ACM, Association for Computing Machinery (ACM), vol. 19, no 2, 1972, p. 248–264
8 (Algorithme de little) John D. C Little, An algorithm for the traveling salesman problem, Cambridge, M.I.T., 1963

8.2 Définitions

9 Problème à N corps, https ://fr.wikipedia.org/wiki/Probl%C3%A8me_%C3%A0_N_corps , 2004

8.2.1 Algorithme de dessin de graphe

10 Rémi Damlencour, Algorithmes de modélisation de graphes,
http ://www-igm.univ-mlv.fr/ dr/XPOSE2012/visualisation_de_graphes/algorithmes.html, 2013

8.3 Choix du langage

11 The Computer Language Benchmarks Game, C++ g++ versus C gcc fastest programs, https ://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/cpp.html
12 The Computer Language Benchmarks Game, C++ g++ versus Java fastest programs, https ://benchmarksgame-team.pages.debi
java.html
13 The Computer Language Benchmarks Game, C++ g++ versus Rust fastest programs, https ://benchmarksgame-team.pages.debi
rust.html
14 The Computer Language Benchmarks Game, C++ g++ versus Python 3 fastest programs, https ://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/gpp-python3.html

8.4 Licence

15 MIT licence, https ://mit-license.org/