Julien ARNOLD – Adrien COUCHOT – Jean-Philippe DUBUC – Romain TEULÉ

UniversitÉ de haute alsace  Année 2017

Projet Graphe Algo

Sommaire

[Introduction 2](#_Toc480646452)

[Description 3](#_Toc480646453)

[Description des classes 3](#_Toc480646454)

[Description des fonctions 3](#_Toc480646455)

[Problèmes rencontrés 4](#_Toc480646456)

[Démonstration 5](#_Toc480646457)

[Conclusion 6](#_Toc480646458)

# Introduction

La notion de graphes apparait dans de nombreux domaines comme les réseaux de communication, relation binaire, règles de jeu ou encore problèmes d’évaluation. Les algorithmes appliqués à des graphes permettent de résoudre des problèmes dans tous ces domaines.

# Organisation

Afin de nous organiser, nous avons utilisé plusieurs outils :

* Pour la communication, un groupe Facebook Messenger.
* GitHub, pour gérer nos versions et le développement du projet.
* DevCpp ainsi que VisualStudio, des environnements de développement pour coder notre projet.

Pour ce qui est de la répartition des tâches, elle s’est faite au fur et à mesure de l’avancement du projet, et en fonction des disponibilités de chacun. Dans le code du projet sont passés en commentaire les personnes qui ont contribué à chaque partie.

# Description

## Description des classes

#### OBJET - Adrien

Cette classe gère le graph sous toutes les formes sous lesquels il peut être.

L'information sur sa forme est conservée dans l'entier "type" :

-1 correspond à la matrice d'adjacence

-2 file successeur

-et 3 à la représentation par liste chaînée.

Le type est relié à un pointeur, un seul peut être utilisé à la fois car l'objet peut être d'un seul type à la fois : "t1", "t2" ou "t3".

L'objet gère la conversion d'un type à un autre grâce aux fonctions "convertToType1()","convertToType2()" et "convertToType3()".

On peut afficher un graph grâce à la fonction "afficher()" qui est supportée par toutes les formes de l'objet.

Pour la modification du graph, l'objet est convertie en type 1 et est géré par la fonction "modifierTableau()"

#### MATRICE D'ADJACENCE - Adrien

La matrice d'adjacence est la forme 1 de l'objet, le constructeur "adjacence(int taille)" initialise une matrice carré de taille "taille" remplie de 0.

La méthode "afficher()" est utilisée par la classe objet afin d'afficher la matrice.

La fonction "dansTableau(int i, int j)" renvoit vrai si la valeur (i,j) ne dépasse pas le tableau, elle est utilisée par "get/setValeurLien(int i, int j)" qui permet de retourner/modifier la valeur en (i,j)

#### FILE DE SUCCESSEURS - Adrien

La forme file de successeurs est la 2 de l'objet, "fileSuccesseurs(int nbSommets, int nbArcs)" permet d'initialiser fs et aps.

"mettreAJourAps()" permet de réactualiser aps en fonction de fs.

La méthode "afficher()" est utilisée par la classe objet afin d'afficher la file de successeurs.

#### Représentation sous formes de listes chainées - Romain

Les graphes peuvent être représentées sous la forme de listes chainées, plus précisément avec une liste principale contenant les différents sommets du graphe et plusieurs listes secondaires représentant les successeurs de chaque sommet.

Pour cette représentation, 3 classes sont utilisées :

- la classe chainon, représentant un sommet dans la liste principale. Chaque chainon contient une valeur, un pointeur sur le chainon suivant, et un pointeur sur son premier successeur.

- la classe chainonSuccesseur, qui sert dans les listes secondaires, représente un arc. Elle contient un pointeur sur un chainon de la liste principale, indiquant ainsi que ce sommet est le successeur, et un pointeur sur le successeur suivant

- la classe pointeurs, qui est la classe principale. Elle contient seulement un pointeur sur la tête de liste, pointeur qui lui permet d'accéder à l'intégralité de la liste. Elle possède également les fonctions permettant de gérer la liste, d'obtenir des informations sur les sommets et sur leurs successeurs.

Cette représentation présente deux intérêts :

- elle permet facilement de rajouter des informations que ce soit sur un sommet ou sur un arc, en modifiant les classes chainon et chainonSuccesseur en leur rajoutant des informations (couts, détails supplémentaires, etc.)

- les listes chainées sont très efficace pour les opérations d'insertion et de suppression. Ainsi, pour des algorithmes nécessitant beaucoup de ces opérations, cette représentation peut se montrer plus intéressantes qu'une sous forme de matrice ou de fs/aps.

#### FICHIER - Adrien

Cette classe permet de sauvegarder un graph dans un fichier avec le nom définit dans le constructeur de l'objet "fichier" > "fichier(string nomFichier)". On peut changer/connaitre le nom du fichier grâce à "setNomFichier(string nomFichier)" et "getNomFichier()".

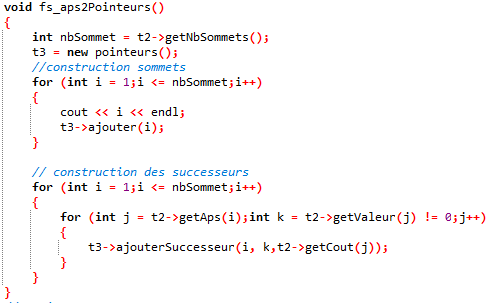
Elle fonctionne avec l'objet sous la forme type 1, la matrice d'adjacence.

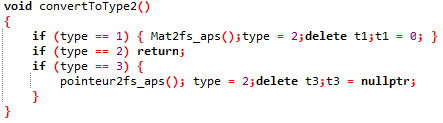
On peut sauvergarder ou charger le graph à partir un fichier texte "sauvegarder(objet & o)" / "charger(objet & o)".

## Description des fonctions implémentées :

#### Fonctions de conversion

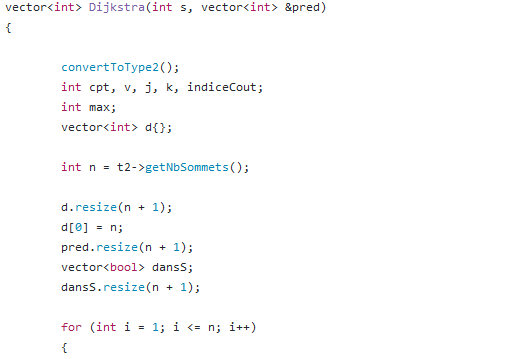
Afin de passer d’un type à l’autre : matrice d’adjacence <-> file de successeurs <-> pointeurs, on utilise différentes fonctions de la classe objet.h.

Ces 4 fonctions Mat2fs\_aps - fs\_aps2Mat - fs\_aps2Pointeurs() - pointeur2fs\_aps() permettent de changer la façon dont les données sont représentées :

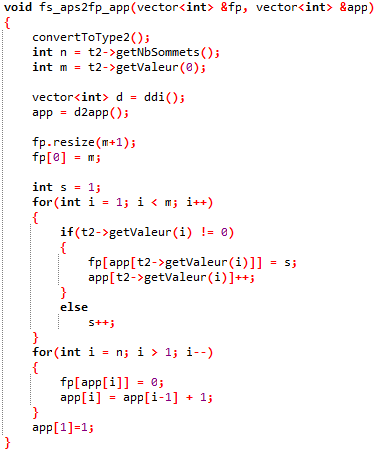
Pour changer de type, on appelle les fonctions ConvertToType 1 2 ou 3 qui choisissent en fonction du type actuel la bonne procédure à appeler.

#### Dijkstra

L’algorithme de Dijkstra permet de trouver le chemin le plus court entre un sommet passé en paramètre et tous les autres. La fonction prend en paramètre le sommet s (entier) et un vecteur pour récupérer la file des prédécesseurs.



La fonction utilise la représentation en file de successeurs et appelle donc le convertisseur au début.



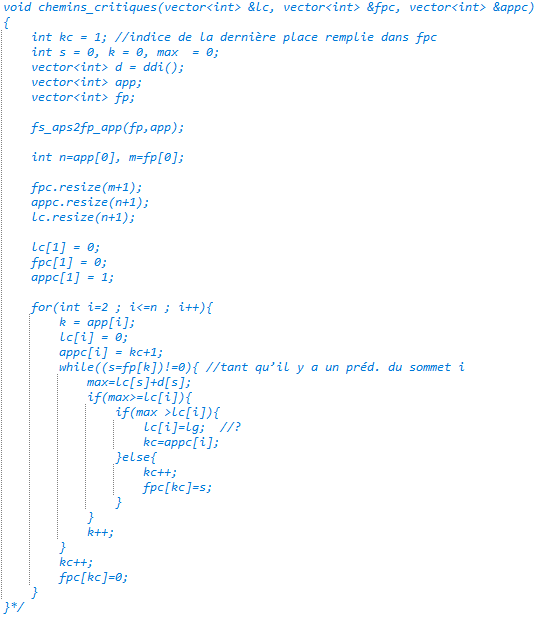
#### File des prédécesseurs

Cette fonction permet de récupérer deux vecteurs passés en paramètre : la file des prédécesseurs et leur adresse, en fonction de la représentation par fs/aps.

Elle utilise aussi d’autres fonctions comme ddi() qui permet de calculer de demi degré intérieur et app() pour calculer l’adresse des premiers prédécesseurs en fonction du ddi.

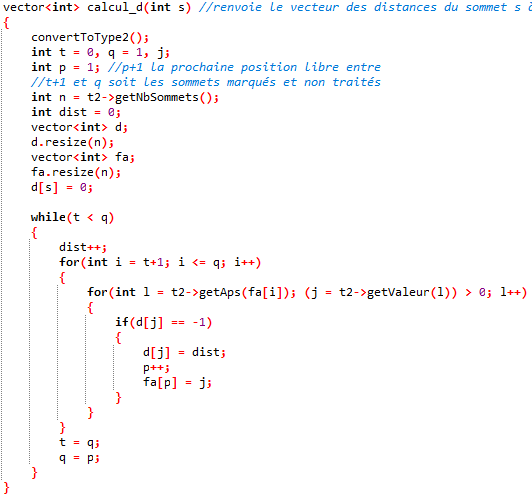
Une fonctionne pour trouver la file des prédécesseurs à partir de la représentation fs – app est aussi implémentée, ainsi qu’une fonction pour renvoyer le nombre de prédécesseurs.

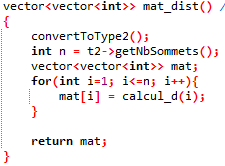
#### Chemins critiques

La fonction permettant de trouver les chemins critiques d’un graphe a été commencée mais pas terminée.

#### Calcul des distances

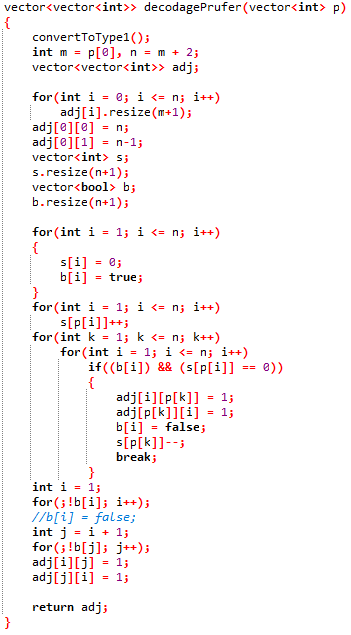
Cette fonction permet de calculer la distance d’un sommet s passé en paramètre aux autres.





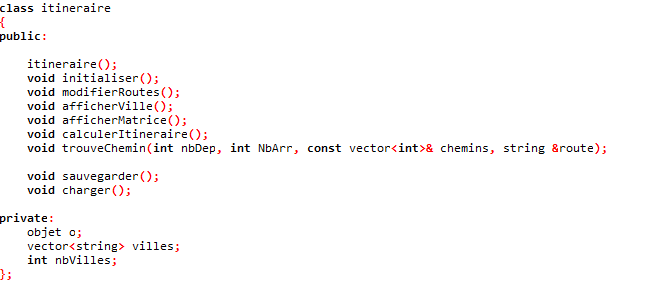
Elle est appelée par une autre fonction mat\_dist() qui crée la matrice des distances du graphe :

#### Codage de Prüfer

Cette fonction renvoie un vecteur d’entiers contenant le codage de prüfer du graphe en utilisant la matrice d’adjacence. Une autre fonction permet de décoder ce codage en le prenant en paramètre et en renvoyant un tableau de vecteur d’entiers :

## Description du cas concret

Afin d’utiliser les graphes dans une application concrète, nous avons créé une classe permettant le calcul d’itinéraire entre plusieurs villes (ou lieux). En effet, la classe itineraire permet la saisie de villes, de routes, et permet ensuite de calculer l’itinéraire le plus rapide entre deux villes, à l’aide de l’algorithme de Djikistra.



Elle contient également des fonctions de sauvegarde et de chargement de fichier, permettant d’enregistrer les configurations de villes crées par l’utilisateur.

# Problèmes rencontrés

Pendant le déroulement de ce projet, nous avons dû faire face à quelques problèmes à surmonter :

* La gestion du temps : des disponibilités différentes ont retardé l’avancement du projet (vacances,…)
* La répartition des tâches a posé quelques problèmes au début, on ne savait pas par où commencer. Une fois le projet commencé l’organisation fut plus simple.
* L’implémentation d’une classe permettant une manipulation des algorithmes et un passage simple entre les différents modes de représentations en mémoire a également source de nombreuses réflexions et difficultés

# Démonstration

# Conclusion

Ce projet nous a permis d’approfondir les notions acquises tout au long de l’année et de développer notre travail en équipe. L’implémentation de fonctions étudiées pendant le cours dans le but de les utiliser dans un cas concret a été une tâche intéressante qui a permis de nous montrer la puissance et l’utilité de ces algorithmes.