Rapport - Projet Individuel n° 20

Test de la connexité d'un graphe

COUCHOUD Thomas COLEAU Victor

27 mai 2017

Table des matières

Chapitre 1

Présentation du sujet

L'objectif de ce projet est de réaliser l'implémentation d'une fonction permettant de déterminer si un graphe est connexe tout en se basant sur la librairie que nous avons développée précédemment, lors du projet n° 2.

Nous nous intéressons plus particulièrement aux graphs non-orientés mais il est à noter que la méthode implémentée peut aussi s'appliquer sur des graphs orientés.

La connexité d'un graphe se détermine selon les chemins réalisables. En effet, si pour toute paire de sommets du graphe il existe un chemin entre eux, alors le graphe est dit connexe. De manière plus visuelle, un graphe connexe est un graphe dont tous les sommets font partie du même "bloc" (il n'y a pas plusieurs regroupements de sommets, mais un unique).

Chapitre 2

Architecture

Afin de répondre à cette question, nous avons décidé de créer une nouvelle classe appelée CGraphToolbox. Celle-ci contient l'ensemble des méthodes permettant des actions avancées sur un objet de la classe CGraph. Grâce à cela, nous pourrons, par la suite, ajouter d'autres méthodes utilisant les graphs sans modifier la classe CGraph elle-même.

Un objet de la classe CGraphToolbox s'instancie par un constructeur prenant en argument un objet CGraph. La ToolBox va alors dupliquer le graph pour ensuite effectuer ses actions sur cette copie. Ce choix sera expliqué par la suite.

La classe CGraphToolbox ne contient pour l'instant que 3 méthodes principales : GRTtransformNonOriented, GRThasPath et GRTisConnex.

La méthode GRTtransformNonOriented transforme le graph actuel de la ToolBox en un graph non-orienté. Cela signifie que pour tous points reliés par un unique arc (dans un seul sens), ils seront dorénavant reliés dans les deux sens. Cette action est irréversible.

La méthode GRThasPath prend en paramètres 2 indices de sommets (le départ et l'arrivée) et renvoie un booléen indiquant l'existence d'un chemin entre ces 2 points. Si l'un des ces 2 sommets n'existe pas dans le graph, une exception sera levée.

Il est à noter que l'exécution de cette méthode se fait par appels récursifs, mais que la première récursion se fait sur une surcharge de cette même méthode prenant en paramètre supplémentaire un tableau contenant les indices des sommets déjà visités par l'algorithme.

Cet algorithme peut se décomposer de la manière suivante :

- Ajout du sommet courant (celui dit de départ) au tableau des indices déjà explorés.
- Pour chaque sommet atteignable depuis le sommet courant on va vérifier/faire :
  - S'il s'agit du sommet d'arrivée, on renvoie vrai. Ceci va fermer toutes les récursions et terminer la méthode.
  - S'il est dans la liste des sommets déjà explorés, on l'ignore et passe au suivant. Cela permet d'éviter les boucles infinies où l'on tournerait entre plusieurs sommets sans jamais voir les autres.
  - Si aucune des conditions précédentes n'est remplie, on lance une récursion avec pour nouveau sommet de départ le sommet atteignable.
- Si aucune des récursions précédentes n'a renvoyé vrai, on renvoie alors faux. Cela termine la méthode.

Cet algorithme est assez efficace pour deux raisons : premièrement, il s'arrête dès qu'il trouve un chemin, et deuxièmement, s'il venait à devoir parcourir presque tout le graph, la liste des sommets déjà explorés permet de réduire considérablement les possibilités de chemin à chaque nouvelle itération.

Enfin la méthode GRTisConnex ne fait que peu d'instructions elle-même. Elle va simplement appeler la méthode GRThasPath pour chaque couple de sommets. Il est à noter que les points d'arrivée sont choisis parmi les points qui n'ont pas encore été des sommets de départ. Ceci permet d'éviter des recherches inutiles car si le chemin entre les sommets X et Y existe, inutile de tester celui entre Y et X.

Là encore, l'algorithme s'arrête dès qu'un des chemins n'existe pas car cela signifie que le graph n'est pas connexe.

Afin de simplifier l'exécution des méthodes précédemment présentées, nous avons rajouté deux méthodes dans les classes CGraph et CVertex.

GRAgetRechableIndices renvoie dans un tableau la liste des sommets atteignables à partir d'un indice passé en paramètre. Lève une exception si le sommet n'existe pas dans le graph de la ToolBox.

VERgetRechableIndices renvoie dans un tableau la liste des sommets atteignables à partir du sommet en cours.

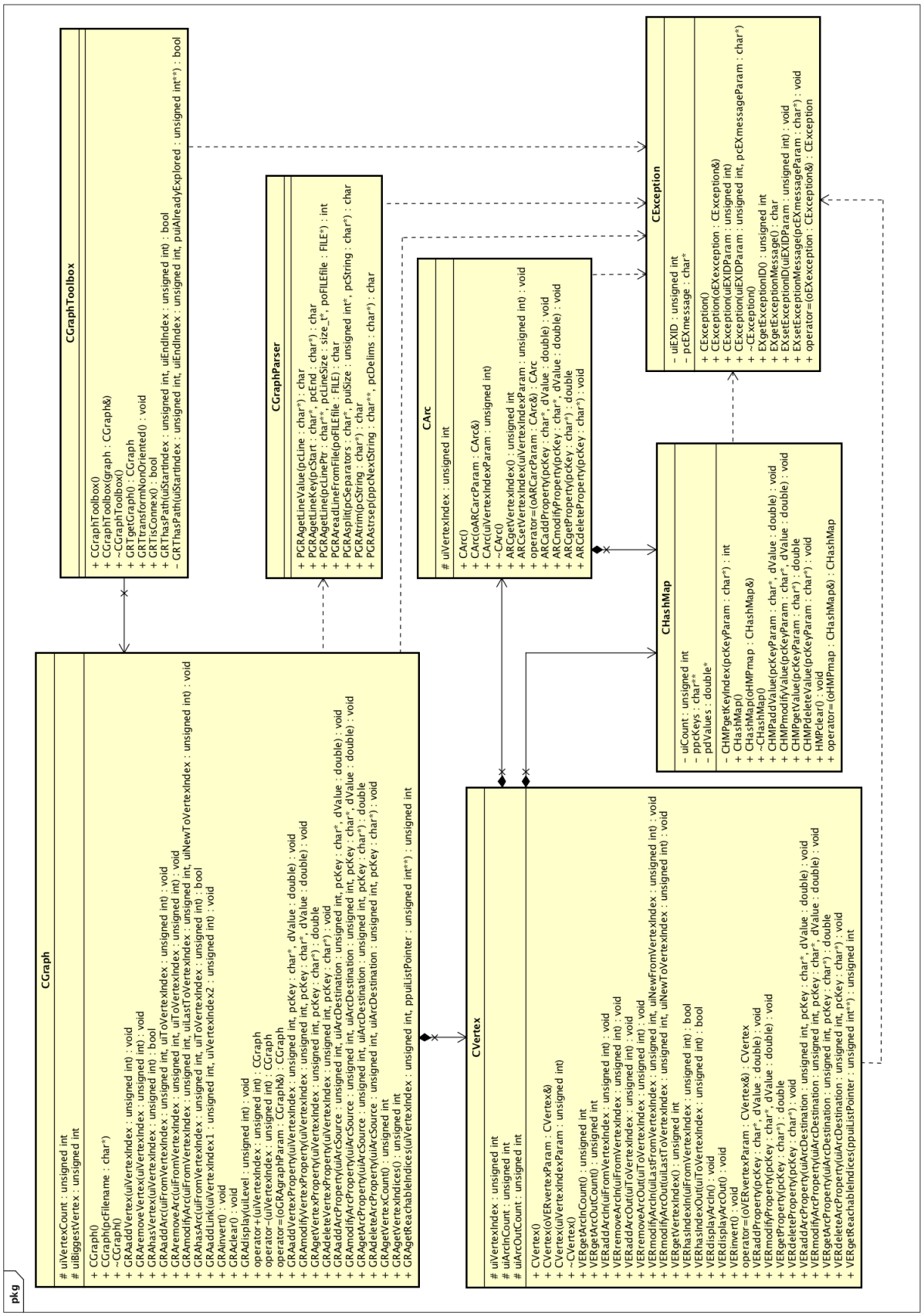

FIGURE 2.1 – Diagramme de classes

Chapitre 3

Choix de codage

Le premier choix que nous avons du faire est celui de la recopie (ou non) du graph lors de la création de la ToolBox. L'autre solution aurait été de ne garder qu'une référence (un pointeur) sur l'objet graph en courant. Cela aurait économisé du temps de recopie et de la mémoire. Cependant, certaines méthodes de la ToolBox modifient de manière irréversible le graph. C'est à cause de cela que nous avons prit la décision de dupliquer le graph afin que l'original ne soit jamais modifié malgré les actions de la ToolBox.

Un deuxième choix à expliquer est celui de la forme du tableau des indices déjà explorés. En effet, on remarque qu'aucun paramètre représentant la taille de ce tableau n'est passé à chaque appel de récursivité. Ceci s'explique par le fait que la taille est contenue dans la première case de ce tableau de 'unsigned int'.

Cette méthode à deux avantages :

- On économise le passage d'un argument à chaque récursion, économisant de la mémoire. (avantage assez limité)
- Mais surtout, le tableau contient ainsi toujours au moins une case, celle de la taille. En effet, même si aucune réelle valeur n'est contenue, la première case aura la valeur 0. Ceci permet de facilité l'utilisation des Malloc, Réalloc et Free car il est dorénavant utile de vérifier la taille à chaque modification du tableau (Realloc s'il contient déjà des valeurs, Malloc si c'est un nouveau tableau, Free s'il est vidé, etc..). On utilise un Malloc la toute première fois puis uniquement des Realloc car la taille ne deviendra jamais nulle.

Enfin, malgré les consignes du sujet, nous avons remarqué que notre méthode permettent de connaitre la connexité d'un graph ne demandait pas forcement un graph non-orienté et fonctionnait aussi sur un orienté.

Nous avons donc choisi de séparer la méthode de connexité de celle transformant la graph en non-orienté. Cela augmente ainsi les possibilités de la librairie en distinguant bien deux actions différentes mais toujours compatibles.

Chapitre 4

Tests effectués

Afin de pouvoir valider le fonctionnement de notre code, nous avons réalisé quelques tests. Certains ont été réalisés manuellement mais un bon nombre d'entre eux ont été écrits sous forme de "test unitaires". Ceux-ci sont présents dans les fichiers CXXXTest.cpp qui vont respectivement tester leur classe XXX. Une grosse partie de ceux-ci proviennent des tests effectués lors de la partie 2 du projet. Seule la classe CGraphToolboxUnit a été ajoutée.

Celle-ci contient les tests suivants

- Test de la transformation d'un graph orienté en un graph non-orienté.
- Test de la fonction indiquant si un chemin existe entre deux sommets.
- Test de la fonction indiquant si un chemin existe entre deux sommets avec pour sommet de départ un sommet n'existant pas.
- Test de la fonction indiquant si un chemin existe entre deux sommets avec pour sommet d'arrivée un sommet n'existant pas.
- Test de la fonction indiquant la connexité d'un graph.

De plus, un test valgrind est réalisé sur la phase d'execution des tests.

Chapitre 5

Conseils d'utilisation

Le programme a été conçu et compilé de sorte que l'exécutable puisse prendre en arguments un fichier source de graph formaté comme défini dans le sujet.

Suite à cela, il va exécuter les instructions suivantes :

- Affichage de la connexité du graph.
- Affichage de la connexité du graph après transformation en un graph non orienté.

Pour une utilisation en tant que librairie, tout se fait à partir d'un objet CGraphToolbox. Celle-ci dupliquera le graph passé en paramètre lors de sa création et offrira différentes fonctionnalités sur ce dernier. Cela comprend :

- Transformation du graph en un graph non-orienté.
- Savoir si un chemin existe entre deux sommets.
- Savoir si le graph est connexe.

Pour toute information supplémentaire sur les méthodes, se référer aux cartouches d'entête présents dans les fichiers .h.