Laboratoire : Algorithmique de graphes

1 Objectifs

Le but de ce laboratoire est d'implémenter quelques algorithmes de base associés aux graphes.

2 Travail à faire

Vous devez réutiliser l'implémentation du laboratoire sur les graphes, et ajouter les 4 algorithmes suivants :

1. Parcours en profondeur
2. Parcours en largeur
3. Vérification de connexité faible
4. Tri topologique

Vous devez utiliser l'interface suivante :

```
std::vector<unsigned int> parcoursProfondeur(unsigned int) const;
std::vector<unsigned int> parcoursLargeur(unsigned int) const;
bool estConnexe() const;
std::vector<unsigned int> triTopologique() const;
```

Vous devez implémenter ces algorithmes en vous assurant que ceux-ci sont fonctionnels peu importe le choix d'implémentation du graphe : par des listes d'adjacence, ou une matrice d'adjacence. Vous devez donc implémenter ces méthodes en n'utilisant que les méthodes publiques préalablement définies dans le laboratoire sur l'implémentation des graphes.

3 Astuces

3.1 Parcours en largeur et en profondeur

Le code pour les parcours en profondeur et en largeur se ressemble beaucoup. La différence est que vous aurez certainement à utiliser une pile pour le parcours en profondeur et une file pour le parcours en largeur.

3.2 Tri topologique

Au début de certains livres, les auteurs indiquent les dépendances chronologiques entre les chapitres en les représentant par un diagramme. Par exemple, on voit sur ce diagramme que pour lire le chapitre 16, il faut avoir lu les chapitres 4, 8 et 15. Un lecteur courageux veut lire le strict minimum pour appréhender le chapitre 21. Il faut donc qu'il transforme l'ordre partiel indiqué par les dépendances du diagramme en un ordre total déterminant la liste des chapitres nécessaires au chapitre 21. Bien sûr, **ceci n'est pas possible si le graphe de dépendance contient un cycle**.

L'opération qui consiste à mettre ainsi en ordre les nœuds d'un graphe dirigé sans circuit (souvent appelés sous leur dénomination anglaise *DAGs* pour *directed acyclic graphs*) est appelée le *tri topologique*. Le tri topologique consiste donc à ordonner les sommets d'un DAG en une suite dans laquelle l'origine de chaque arc apparaît avant son extrémité. Nous vous proposons ci-bas un « pseudo-code » sur lequel vous pouvez vous baser pour implémenter le tri topologique.

Soit g un graphe et Q une file vide et V un vecteur de solution vide.

1. Pour chaque sommet s de g, trouver son ordre d'entrée $e(s)$
2. Enfiler dans Q tous les sommets s tels que $e(s) = 0$
3. Tant que Q n'est pas vide :
   (a) Défiler un sommet n de Q
   (b) Ajouter n à la fin de V
   (c) Pour tout voisin m de n:
      i. Mettre à jour $e(m) = e(m) - 1$
      ii. Si $e(m) = 0$, enfiler m dans Q
4. V contient la solution du tri topologique. Si V ne contient pas tous les sommets, c'est qu'il y avait un cycle dans le graphe.

4 Documentation

Voir la section Documentation/Normes sur le site Web du cours. Vous y trouverez la description des commentaires attendus dans un programme ainsi que des normes de programmation en vigueur dans notre cours.

5 Important

1. Nous vous fournissons des tests unitaires *Google Test* que vous pouvez utiliser pour vérifier votre implémentation.
2. Vous êtes tenu de faire la gestion des exceptions dans les méthodes que vous avez à implémenter. Référez-vous à la documentation *Doxygen* fournie avec l'énoncé pour connaître les types d'exception que vous avez à gérer pour chacune d'elles. Vous devez utiliser le cadre de la théorie du contrat que nous avons préparé dans les fichiers `ContratException.cpp` et `ContratException.h` disponibles sur le site du cours.
3. Vous devez documenter chaque méthode, que vous avez à implémenter, avec les commentaires de *Doxygen*. Référez-vous à section de Doxygen sur le site Web du cours ainsi qu'aux exemples de cette semaine pour découvrir les commentaires que vous avez à écrire. Assurez-vous de respecter les normes de programmation en vigueur disponibles également sur le site du cours.
4. Vous devez générer la documentation en format HTML. À cette fin, nous vous fournissons un fichier de configuration `sdd.doxyfile` que vous pouvez utiliser tel quel.
5. Nous vous encourageons à utiliser au maximum la partie privée de la classe afin d'y ajouter des fonctions utilitaires privées. Elles permettent d'augmenter la lisibilité du code et de réduire la duplication de code identique.

Bon travail !