R2.07 Graphes

TP1 – Introduction à JGrapht

2024-2025

Prérequis

Avant de commencer, on va s'assurer que vous avez bien tout ce qu'il faut pour faire fonctionner le code du TP.

Afin de faciliter le processus de compilation et la gestion des dépendances, nous utiliserons l'outils maven. Cet outil est déjà installé sur les machines du département et peut être utilisé avec la commande `mvn <action>`.

Dans notre cas, nous utiliserons les deux commandes suivantes :

```
mvn compile
mvn exec:java -Dexec.mainClass="MonExec"
```

Allez sur Célène et récupérez l'archive `tp1.zip`. Décompressez l'archive puis lancer la compilation. La première fois, cette étape peut être longue car `maven` se connecte à un registre distant afin de récupérer les bibliothèques indiquées dans le fichier `pom.xml` à la racine du projet.

Ensuite, exécutez (via vscode) la classe `Executable`. Cela doit se passer "sans encombre", et vous devez obtenir un nouveau fichier, nommé `graph.dot`.

Transformez ce fichier en un .pdf en exécutant : `dot -T pdf graph.dot -o graph.pdf`

Si vous en êtes ici avec un fichier graph.pdf c'est que tout est ok pour démarrer la suite!

Introduction

Au cours de ce TP on utilisera la bibliothèque Java JGgraphT. Cette bibliothèque permet de représenter toute sorte de graphes (orienté/ non orienté / simple / multigraphe / etc.) et dans ce TP on se contentera de graphes simples non orientés pour lesquels les sommets sont des chaînes de caractères.

Pour déclarer une variable de type “graphe simple non orienté pour laquelle les sommet sont des chaines de caractères”, on fera :

```
Graph<String, DefaultEdge> graph;
```

Et pour l'instancier :

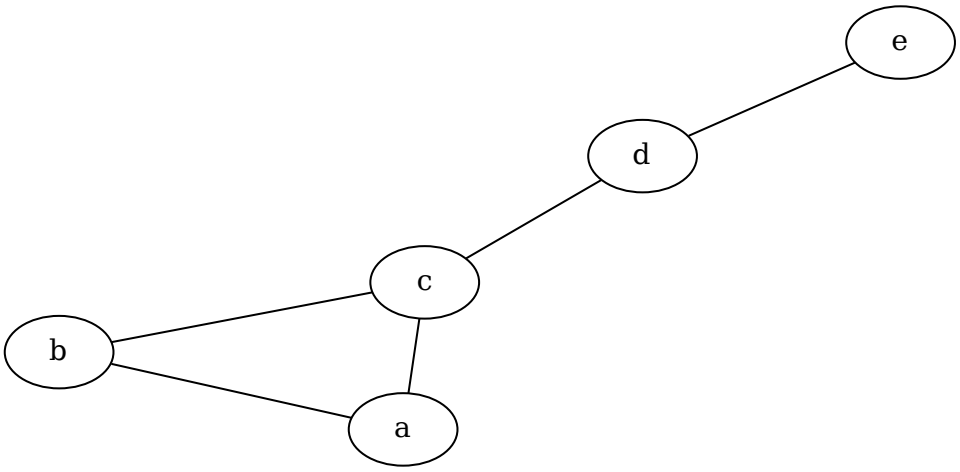
```
graph = new SimpleGraph<>(DefaultEdge.class);
```

Les opérations possibles sur un graphe sont documentées ici : https://jgrapht.org/javadoc/org.jgrapht.core/org/jgrapht/Graph.html

En particulier, vous aurez besoin des méthodes :

```
addVertex(V v)  // pour ajouter un noeud
addEdge(V source, V destination) // pour ajouter une arête
```

1. Créez le graphe suivant :



2. En suivant la même procédure qu'en début de TP, générez un fichier “grapheExo1.pdf” : utilisez la méthode export pour générer un fichier “grapheExo1.dot” puis utilisez la commande dot pour générer un fichier pdf : `dot -T pdf grapheExo1.dot -o grapheExo1.pdf`
3. Essayez la méthode `getEdge` de votre graphe. Cela renvoie un objet de type `DefaultEdge`, dont la documentation est disponible ici :https://jgrapht.org/javadoc/org.jgrapht.core/org/jgrapht/graph/DefaultEdge.html

Manipulations

Pour les questions suivantes, vous appliquerez la méthode TDD pour votre développement.

Pour commencer, créez un dépôt git dans le répertoire courant avec `git init`. Ensuite, créez deux fichiers `ProprietesGraphe.java` et `ProprieteGrapheTest.java`.

Pour rappel, la démarche TDD pour le développement d'une fonctionnalité consiste à :

— Ecrire un test qui ne passe pas
— Ecrire le minimum de code permettant de faire passer le test
— Lorsque le test passe, faire un commit et passer à la fonctionnalité suivante.

Faites une branche par fonctionnalité puis fusionnez-la à la branche `main` lorsque son développement est terminé.

Représentation textuelle

On souhaite écrire une méthode qui renvoie une représentation textuelle d'un graphe. Par exemple, pour le graphe précédent, on souhaiterait avoir la chaine :

```
{a, b, c, d, e, f}, {a--b, a--c, b--c, c--d, d--e}
```

Écrivez une méthode

```
public String getString(Graph<String, DefaultEdge> graph)
```

qui réalise cela.

Vous aurez besoin pour cela des méthodes :

```
vertexSet()
edgeSet()
getEdgeTarget(E e)
getEdgeSource(E e)
```

Degré maximal

Écrivez une méthode

```
public int
getDegreMax(Graph<String, DefaultEdge> graph)
```

qui renvoie le degré maximal du graphe.

Nombre de triangles

Écrivez une méthode

```
public int
getNbTriangles(Graph<String, DefaultEdge> graph)
```

qui renvoie le nombre de triangles du graphe, c'est-à-dire le nombre de sous-ensembles $\{a, b, c\} \subseteq V$ tels que :

— $\{a, b\} \in E, \{b, c\} \in E$ et $\{a, c\} \in E$
— $a \neq b \wedge a \neq c \wedge b \neq c$

Chaines

Écrivez une méthode

```
public boolean
estChaine(Graph<String, DefaultEdge> graph, List<String> chaine)
```

renvoyant vrai si `chaine` est une chaîne du graphe et faux sinon. On rappelle qu'une chaîne est une suite de sommets $u_1, u_2, \ldots, u_k$ tel que pour tout $i < k$, alors l'arête $\{u_i, u_{i+1}\}$ est une arête du graphe.

Métro de Vienne

1. Le fichier vienna_subway.csv contient le graphe des relations entre stations de métro de Vienne. Exportez ce graphe au format dot puis générez une visualisation de ce graphe en pdf. Pour rappel, voici la ligne de commande :

```
dot -T pdf vienna_subway.dot -o vienna_subway.pdf
```

2. Testez les méthodes précédentes sur ce graphe.