TD/TP 1 - Représentation d'un graphe

Gilles Simonin

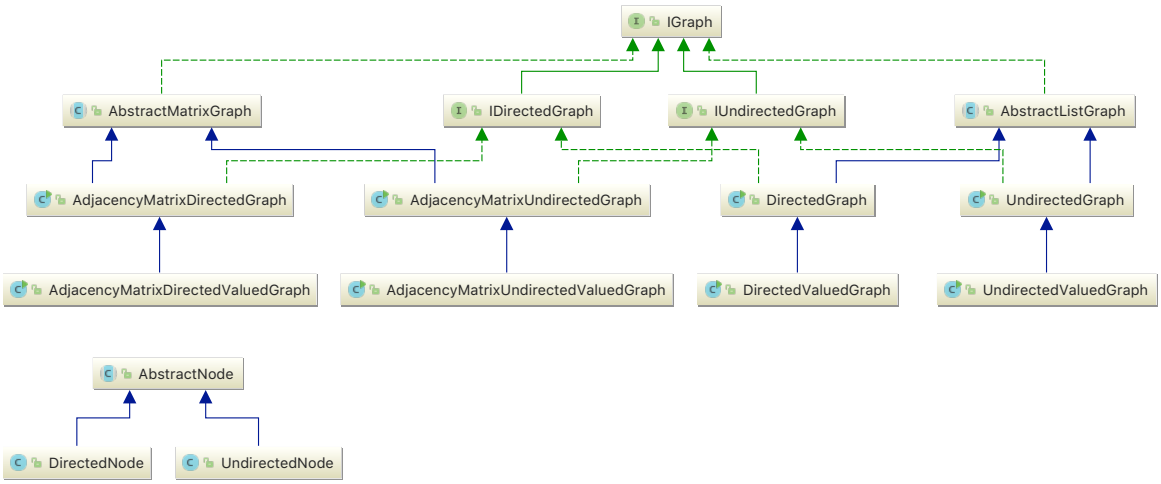
28 septembre 2021

Résumé

Votre travail lors de cette séance de 1h15 est de tester votre capacité à mettre en oeuvre différentes représentations de graphe vues en cours. Ce travail devra être effectué au sein d'un projet Eclipse que vous utiliserez tout au long des séances de TD/TP, et potentiellement pour vos évaluations.

1 Un framework générique

Nous allons tout d'abord poser les bases d'un framework pouvant utiliser différentes représentations de graphes. Afin de pouvoir manipuler des graphes non-orientés et orientés, nous vous proposons une architecture basée sur l'UML suivant. Récupérez le projet puis importez-le dans Eclipse, ce projet comprend plusieurs packages et certaines classes déjà implémentées. C'est le cas pour les interfaces qui vous sont présentées `IGraph`, `IUndirectedGraph` et `IDirectedGraph`.



Dans le package *GraphAlgorithms* et la classe `GraphTools`, vous trouverez une méthode statique `generateGraphData(int n, int m, boolean multi, boolean s)` qui génère un graphe aléatoire simple si `multi` est faux et symètrique si `s` est vrai, en prenant en paramètre l'ordre `n` de ce graphe et son nombre d'arêtes/arcs `m`. Le graphe retourné est sous la forme d'une matrice d'entiers 0/1 (sa matrice d'adjacente) pour les graphes simples, et une matrice d'entiers $0/\{1, 2, 3\}$ pour les multi-graphes.

Nous verrons plus tard qu'il existe également une méthode qui génère, en fonction d'un graphe, une matrice de coût pour chaque arête/arc existant.

1.1 Implémentation sous forme de matrice d'adjacence

Dans le package *AdjacencyMatrix*, nous vous proposons un implémentation quasi-complète de la structure de graphe en utilisant la représentation sous forme de matrice d'adjacence :

— `AdjacencyMatrixUndirectedGraph` implémente l'interface `IUndirectedGraph`,
— `AdjacencyMatrixDirectedGraph` implémente l'interface `IDirectedGraph`.

Question 1 *Nous vous proposons dans un premier temps de vous familiariser avec l'implémentation tournée objets qui a été choisie. Afin de mieux gérer les informations sur les sommets et les arêtes/arcs, une classe abstraite* `AbstractNode` *est définie. Une fois cette compréhension faite, il vous est demandé de compléter des méthodes de bases pour les graphes non orientés :* `isEdge`(), `removeEdge`(), `addEdge`() *; puis les méthodes de bases pour les graphes orientés :* `isArc`(), `removeArc`(), `addArc`() *et* `computeInverse`(). *Enfin vous testerez vos méthodes en complétant le* `main`().

1.2 Implémentation sous forme de liste d'adjacence

Nous proposons ici de mettre en oeuvre une implémentation de la structure de graphe en utilisant la représentation sous forme de liste d'adjacence.

Question 2 *Complétez la classe concrète* `UndirectedGraph` *implémentant l'interface* `IUndirectedGraph` *et héritant de la classe abstraite* `AbstractListGraph` *qui représente un graphe non orienté par sa liste de voisins. Il vous faudra compléter :*

— *Le constructeur de cette classe prenant en paramètre la matrice d'entiers donnée.*
— *Les méthodes suivantes :*
  — `isEdge(A x, A y)`,
  — `removeEdge(A x, A y)`
  — `addEdge(A x, A y)`
  — `toAdjacencyMatrix()`

Enfin vous testerez votre ces méthodes en complétant le `main`().

Question 3 *De la même manière, complétez la classe concrète* `DirectedGraph` *implémentant l'interface* `IDirectedGraph` *et héritant de la classe abstraite* `AbstractListGraph` *pour représenter un graphe orienté par ses listes de successeurs et prédécesseurs. Complétez les méthodes équivalentes, ainsi que* `computeInverse`().

2 Vers d'autres représentations

Cette partie est à faire chez soi, nous proposons ici de mettre en oeuvre différentes implémentations de la structure de graphe. La première porte sur la gestion des poids d'arêtes/arcs dans les graphes, la suite sur une structuration par matrice d'incidence.

Question 4 *Complétez les classes concrètes* `AdjacencyMatrixDirectedValuedGraph` *et* `AdjacencyMatrixUndirectedValuedGraph`, *qui permettent de manipuler des graphes avec des poids sur les arêtes/arcs dans une représentation par matrice d'adjacence. Faites de même pour les classes* `DirectedValuedGraph` *et* `UndirectedValuedGraph` *dans une représentation par listes d'adjacence.*

Question 5 (Pour aller plus loin) *Proposez une classe concrète* `IncidentMatrixUndirectedGraph`, *implémentant l'interface* `IUndirectedGraph` *et héritant d'une nouvelle classe abstraite* `AbstractIncidentGraph`, *qui représente un graphe par sa matrice d'incidence. Développez deux constructeurs, l'un basé sur la méthode* `generateGraphData`, *l'autre basé sur une instance de* `IUndirectedGraph`. *Puis développez les accesseurs et méthodes habituels.*

Reprenez cette dernière question du TP dans le cadre des graphes orientés. Vous vous baserez cette fois-ci sur l'interface `IDirectedGraph`.