SAE 2.02 Exploration Algorithmique d'un problème Parcours de graphes

Contact : Mikal.Ziane@u-paris.fr

Introduction

Dans le cadre de la SAE 2.02 vous devez réaliser une application en Java par équipes de 3. La note de ce travail ne sera pas individualisée, si l'équipe fonctionne sans problème majeur, mais lors des DST une série de questions porteront sur ce travail et la note (individuelle) aura le même coefficient.

Ce travail est composé d'une série d'exercices de difficulté croissante qui portent sur la notion de sous-typage et de polymorphisme qui sont vus dans l'enseignement IQ02Y040 Qualité de développement. Tout le développement sera fait en Java et devra pouvoir tourner sur les machines de l'IUT.

Le travail à faire vous est donné au fur et à mesure.

Partie I Représentation des graphes

Quand ces notions auront été vues en cours, la notion de sous-typage permettra de masquer, à un programmeur peu expert en la matière, les différentes façons d'implémenter un graphe : matrice d'adjacence, liste d'adjacence etc., tout en lui permettant de manipuler des graphes malgré tout.

D'ici là, les différentes façons d'implémenter un graphe seront simplement codées chacune dans une classe dédiée **sans héritage ni sous-typage pour le moment**. Une série de tests unitaires vous est donnée pour contribuer à spécifier ce que doit faire chaque classe.

Exercice 1 Matrice d'adjacence

Ecrivez une classe GrapheMA qui représente un graphe à l'aide d'une matrice d'adjacence, par exemple via un tableau de booléens à 2 dimensions. La classe devra permettre de valider les tests unitaires page suivante.

Exercice 2 Liste d'adjacence

Ecrivez une classe GrapheLA qui stocke un graphe sous forme de liste d'adjacence, par exemple une liste de liste d'entiers.
Les tests sont similaires sauf pour ce qui concerne toString.

```
assertTrue(g.toString().contentEquals(
                    "1 -> 2 3 4 5 \n"
                  + "2 -> 5 \n"
                  + "3 -> \n"
                  + "4 -> 4 \n"
                  + "5 -> 1 \n"
                  + "6 -> \n"
                  ));
```

Test spécifique à GrapheLA

```java
class GrapheMATest {
	private final static int NB_NOEUDS = 6;
	@Test
	void test() {
		GrapheMA g = new GrapheMA(NB_NOEUDS);
		assertEquals(NB_NOEUDS, g.getNbNoeuds());
		g.ajouterArc(1,2);
		g.ajouterArc(1,3);
		g.ajouterArc(1,4);
		g.ajouterArc(1,5);
		g.ajouterArc(2,5);
		g.ajouterArc(4,4);
		g.ajouterArc(5,1);

		assertTrue(g.aArc(1,5));
		assertTrue(g.aArc(4,4));
		assertTrue(g.aArc(5,1));

		assertFalse(g.aArc(4,1));
		assertFalse(g.aArc(6,6));

		assertEquals(4,g.dOut(1)); // degré sortant
		assertEquals(1,g.dOut(2));
		assertEquals(0,g.dOut(3));
		assertEquals(1,g.dOut(5));
		assertEquals(0,g.dOut(6));

		assertEquals(1, g.dIn(1)); // degré entrant
		assertEquals(2, g.dIn(4));
		assertEquals(2, g.dIn(5));
		assertEquals(0, g.dIn(6));

		assertTrue(g.toString().contentEquals(
				"0 1 1 1 1 0 \n"+
				"0 0 0 0 1 0 \n"+
				"0 0 0 0 0 0 \n"+
				"0 0 0 1 0 0 \n"+
				"1 0 0 0 0 0 \n"+
				"0 0 0 0 0 0 \n"));
	}
}
```

Test unitaires pour GrapheMA