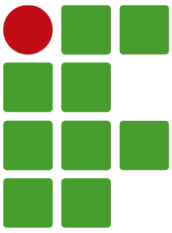


INSTITUTO FEDERAL
SUDESTE DE MINAS GERAIS | Campus Juiz de Fora

TESTES DE SOFTWARE – SISTEMA GRAFÃO

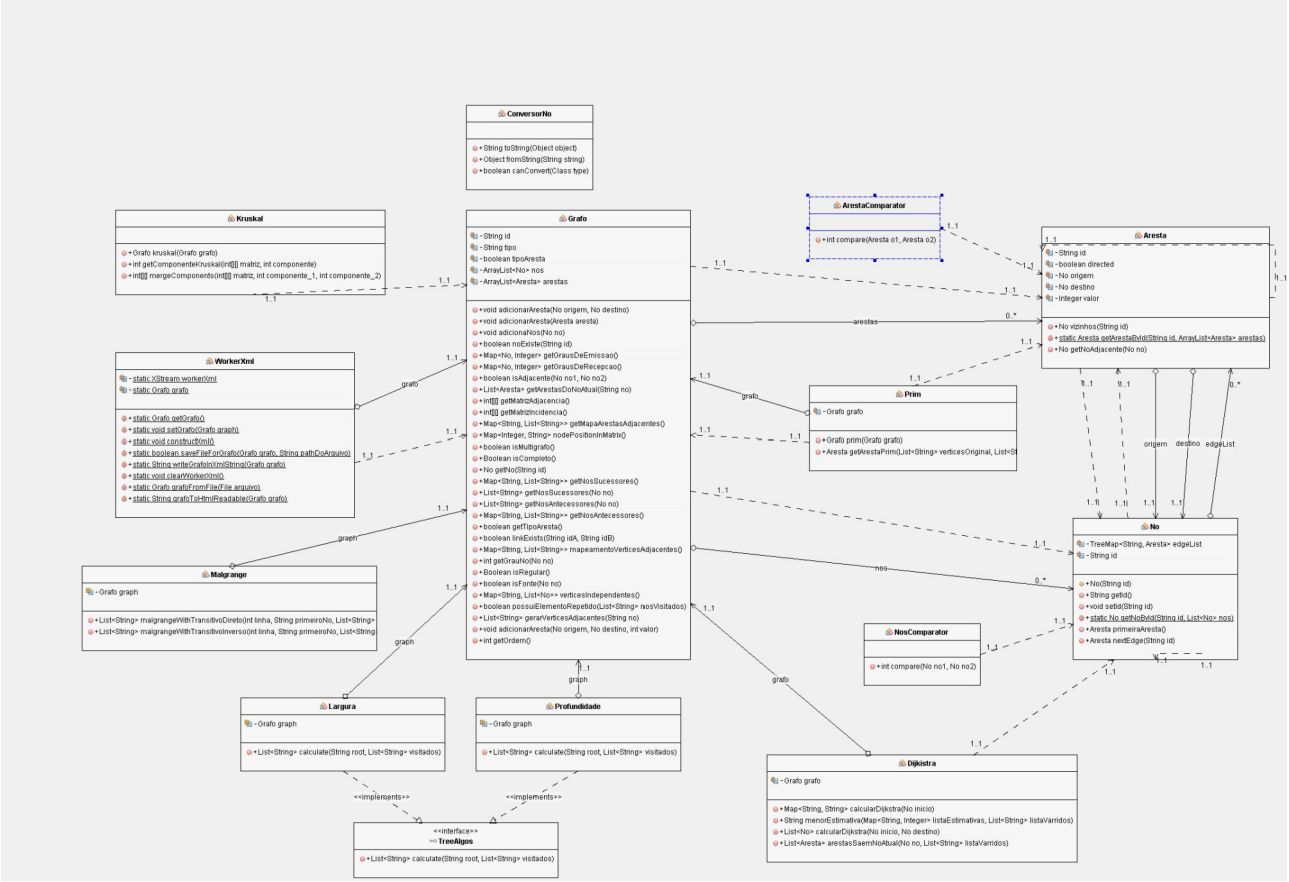
Gabriel Beto Rocha
Matheus Oliveira Rodrigues

Juiz de Fora
2018

1 - Resumo do sistema

O sistema utilizado foi desenvolvido com fins acadêmicos, para a matéria Teoria dos Grafos, onde fora implementado técnicas e algoritmos para a construção de um grafo e para se obter informações através dele e seus atributos, como distâncias, menores distâncias, ordem, grau de seus vértices, entre outras informações. O sistema roda na plataforma Web, com um servidor Tomcat e foi desenvolvido em Java. A forma de utilizar é bem simples por possuir uma interface bem intuitiva, onde basta preencher os dados do grafo e ele te retorna todas as suas informações diretamente, também é possível exportar esse grafo criado no sistema para um arquivo XML e salva-lo em seu computador, facilitando a reutilização do mesmo posteriormente no sistema.

Diagrama de classes do sistema:



2 - Métricas de Chidamber & Kemerer e Lorenz & Kidd :

	WMC	DIT	NOC	CBO	CS	NOO	NOA
Aresta	3	0	0	4	8	1	0
ArestaComparator	1	0	0	1	1	0	0
ConversorNo	3	0	0	0	3	0	0
Dijkstra	4	0	0	2	5	0	0
Grafo	30	0	0	13	35	0	0
Kruskal	3	0	0	1	3	0	0
Largura	1	0	0	2	2	0	0
Malgrange	2	0	0	1	3	0	0
No	3	0	0	9	5	0	0
NosComparator	1	0	0	1	1	0	0
Prim	2	0	0	3	3	0	0
Profundidade	1	0	0	2	2	1	1
TreeAlgos	1	0	0	0	1	0	0
WorkerXml	6	0	0	2	8	0	0

Calculando-se 20% das classes para serem testadas, aproximadamente devem ser testadas 3 classes. As classes escolhidas foram Aresta, Grafo e No, por possuírem o maior WMC, CBO e CS, destacando-se das demais.

3 - Métrica de Complexidade Ciclomática de McCabe:

Classe::Metodo	Complexidade Ciclomática
Aresta::vizinhos	5
Aresta::getArestaById	2
Aresta::getNoAdjacente	3
ArestaComparator::compare	3
ConversorNo::toString:	1
ConversorNo::fromString:	1
ConversorNo::canConvert:	1
Dijkistra::calcularDijkstra:	10
Dijkistra::menorEstimativa:	2
Dijkistra::calcularDijkstra:	2
Dijkistra::arestasSaemNoAtual:	1
Grafo::isMultigrafo:	9
Grafo::getMapaArestasAdjacentes:	6
Grafo::isCompleto:	6
Grafo::isAdjacente:	5
Grafo::getGrauNo:	5
Grafo::mapeamentoVerticesAdjacentes:	5
Grafo::linkExists:	4
Grafo::isRegular:	3
Grafo::getArestasDoNoAtual:	2
Grafo::getGrausDeRecepcao:	2
Grafo::gerarVerticesAdjacentes:	2
Grafo::getGraus:	2
Grafo::verticesIndependentes	2
Grafo::getGrausDeEmissao	2
Grafo::adicionarAresta:	1
Grafo::getNosSucessores:	1
Grafo::getNosAntecessores:	1
Grafo::getMatrizIncidencia:	1
Grafo::getMatrizAdjacencia:	1
Grafo::getNosSucessores:	1
Grafo::possuiElementoRepetido:	1
Grafo::nodePositionInMatrix:	1
Grafo::isFonte:	1
Grafo::getTipoAresta:	1
Grafo::noExiste:	1
Grafo::adicionaNos:	1
Grafo::getNo:	1
Grafo::getNosAntecessores:	1
Grafo::getOrdem:	1

Kruskal::kruskal:	3
Kruskal::mergeComponents:	3
Kruskal::getComponenteKruskal:	1
Largura::calculate:	2
Malgrange::malgrangeWithTransitivoInverso:	6
Malgrange::malgrangeWithTransitivoDireto:	6
No::nextEdge:	3
No::getNoById:	2
No::primeiraAresta:	2
NosComparator::compare:	1
Prim::getArestaPrim:	3
Prim::prim:	2
Profundidade::calculate:	1
WorkerXml::grafoFromFile:	2
WorkerXml::saveFileForGrafo:	2
WorkerXml::setGrafo:	1
WorkerXml::writeGrafoInXmlString:	1
WorkerXml::grafoToHtmlReadable:	1
WorkerXml::clearWorkerXml:	1

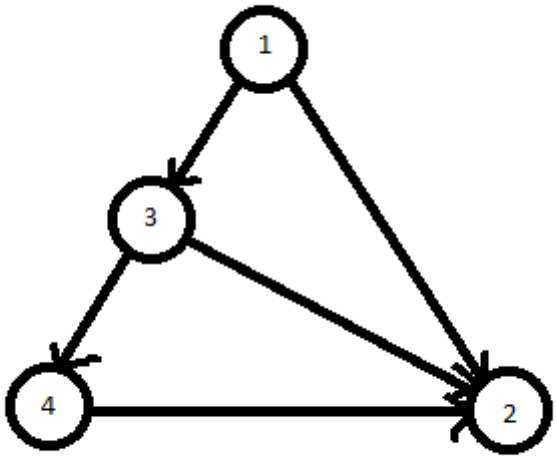
4 - Baseado nessas métricas identifique classes e métodos que devem ser mais testados:

Analizando a tabela acima, é possível constatar que as classes a serem mais testadas são: Aresta, Grafo, No e Djikistra, e fazendo uma análise de seus métodos, 60 ao todo, calculando que 20% deles devem ser mais testados, obtivemos os seguintes resultados:

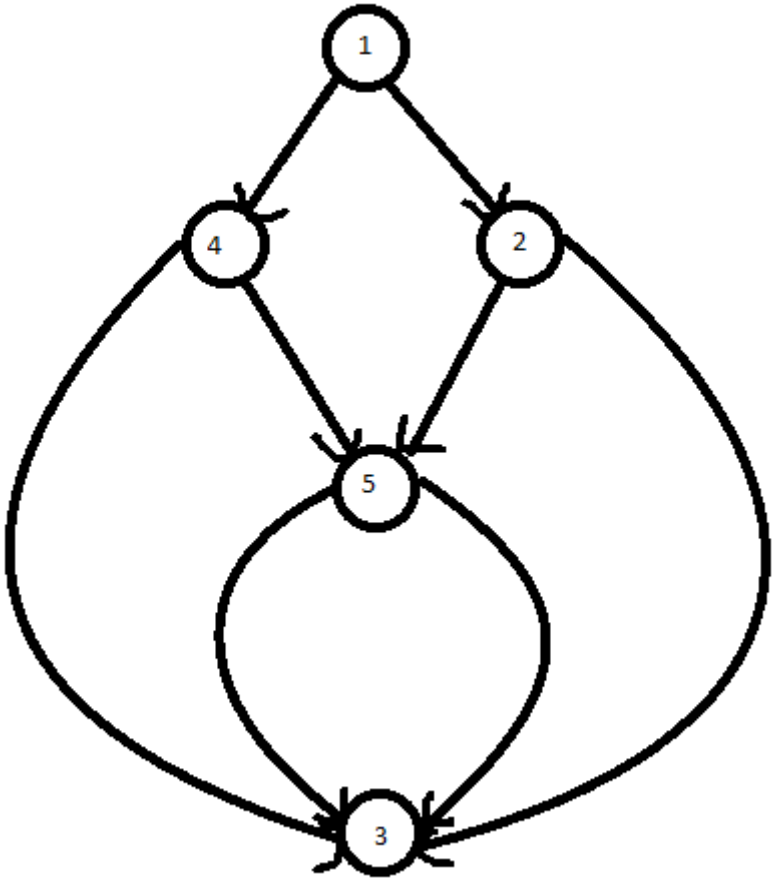
Classe::Metodo	Complexidade Ciclomática
Dijkistra::calcularDijkstra:	10
Grafo::isMultigrafo:	9
Grafo::getMapaArestasAdjacentes:	6
Grafo::isCompleto:	6
Grafo::isAdjacente:	5
Grafo::getGrauNo:	5
Grafo::mapeamentoVerticesAdjacentes:	5
Grafo::linkExists:	4
No::nextEdge:	3
No::primeiraAresta:	2
Aresta::vizinhos	5
Aresta::getNoAdjacente	3

5 - Comparação dos gráficos de complexidade ciclomática dos métodos acima, comparando os gráficos feitos manualmente com os gráficos obtidos por ferramentas.

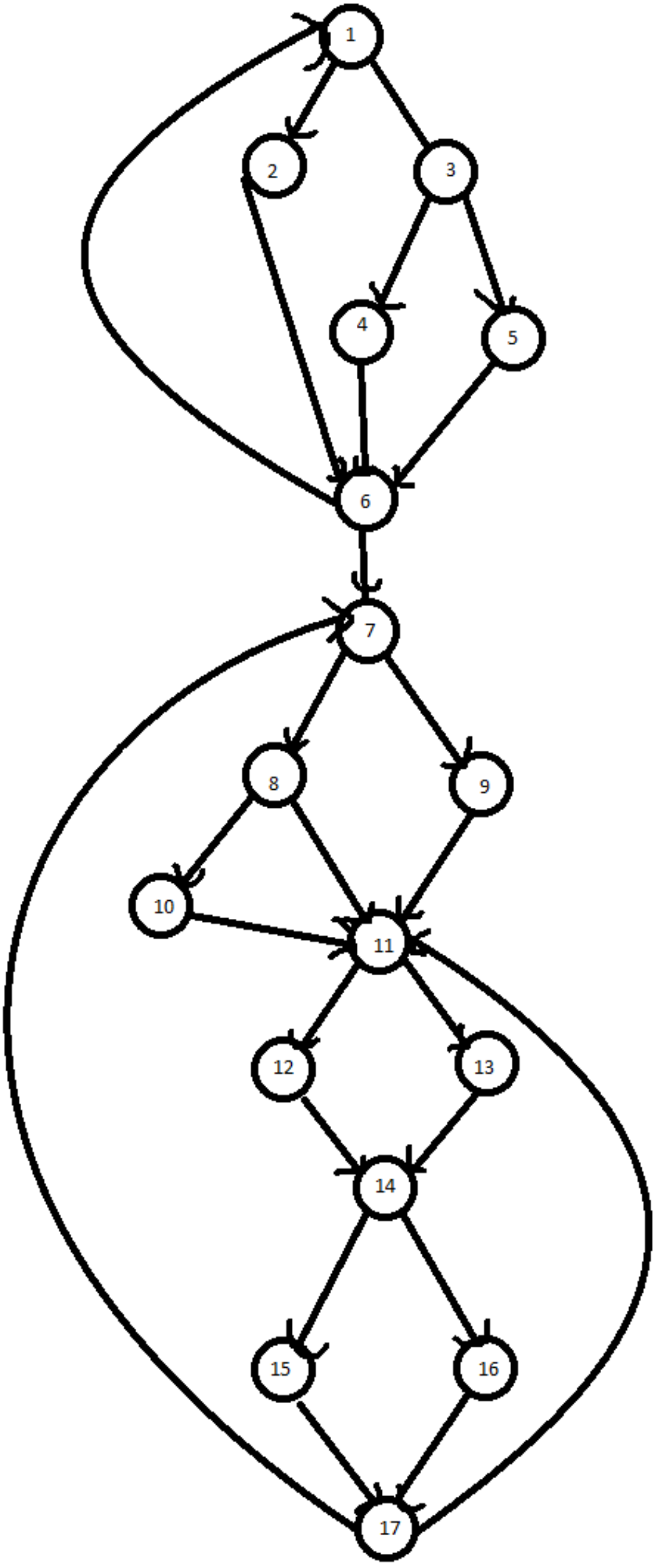
Aresta :: getNoAdjacente



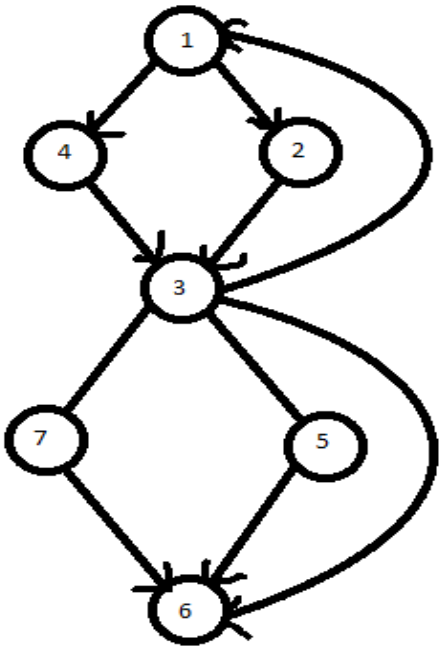
Aresta :: vizinhos



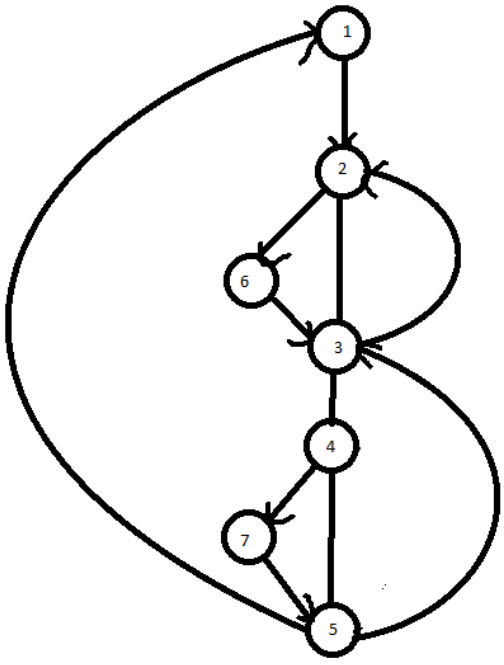
Dijkstra :: calcularDijkstra



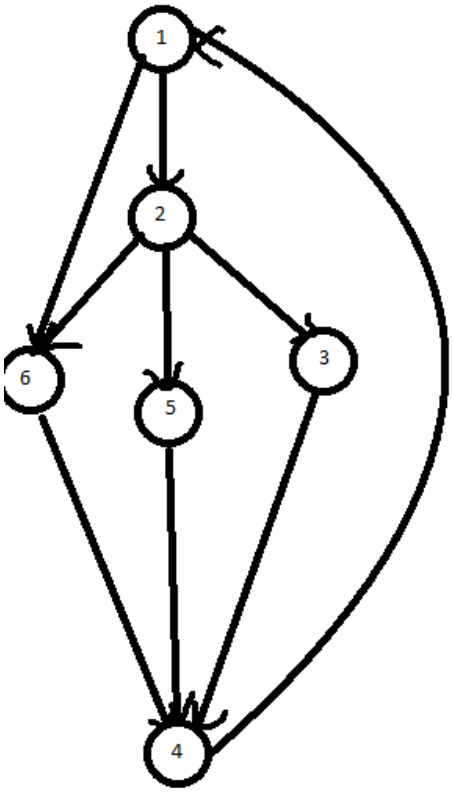
Grafo :: getGrauNo



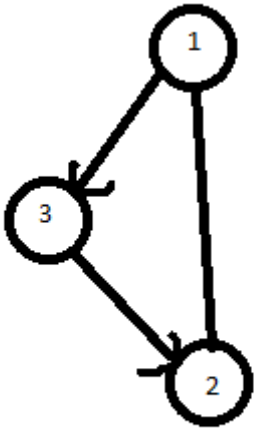
Grafo :: getMapaArestaADjacentes



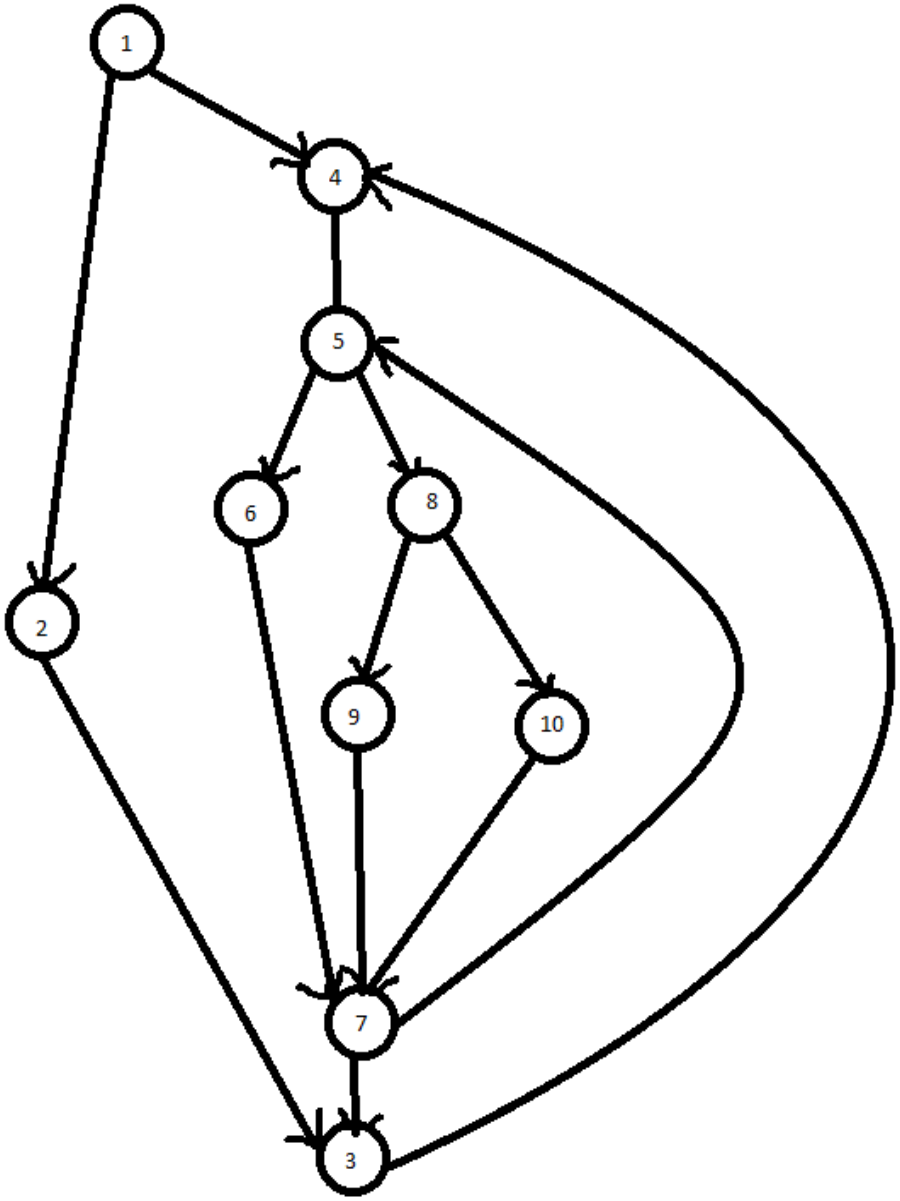
Grafo :: is Adjacente



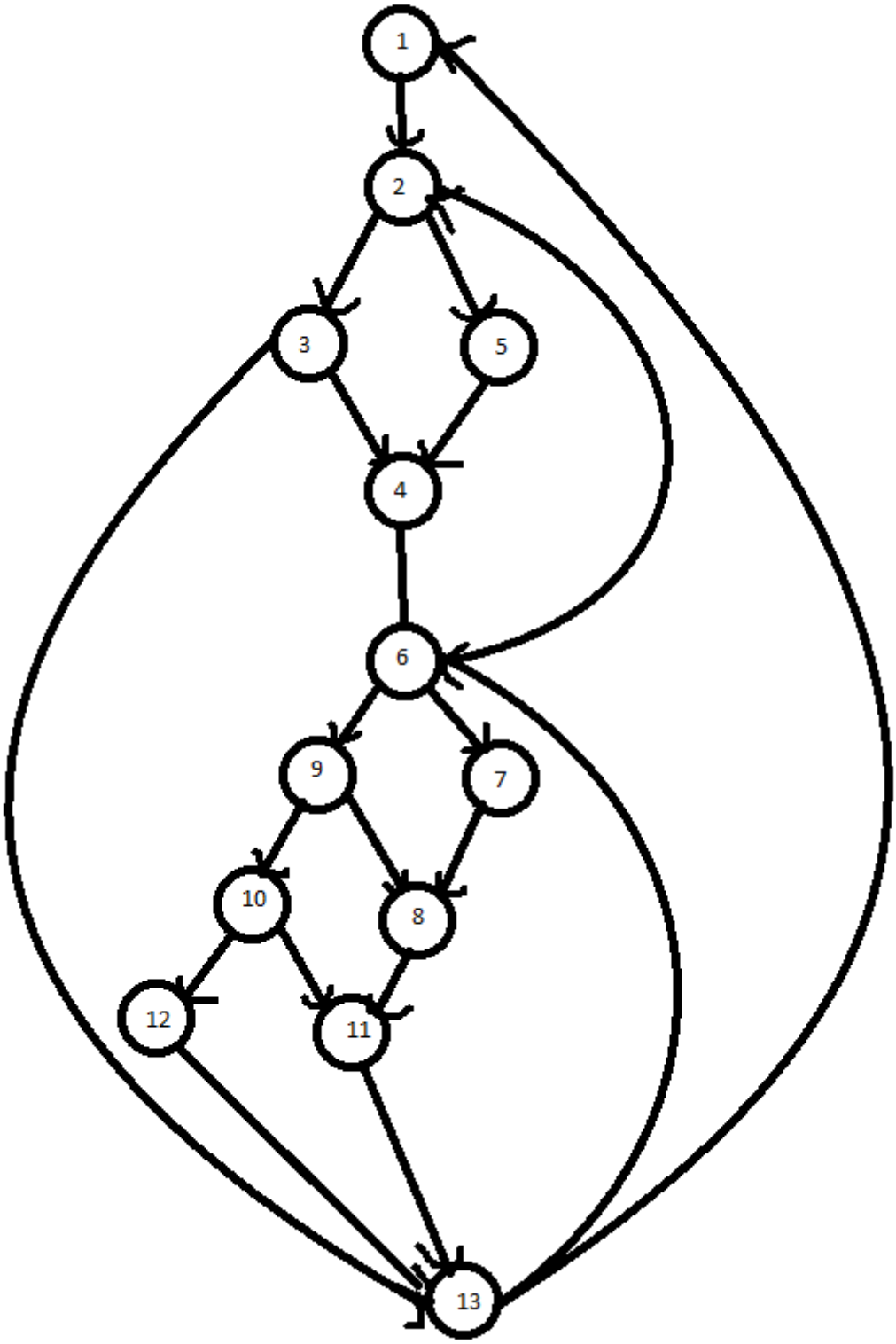
No :: primeiraAresta



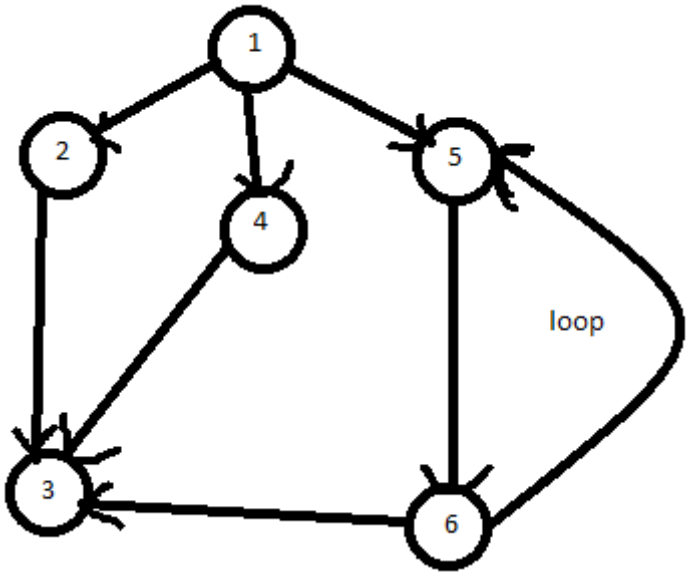
Grafo :: isCompleto



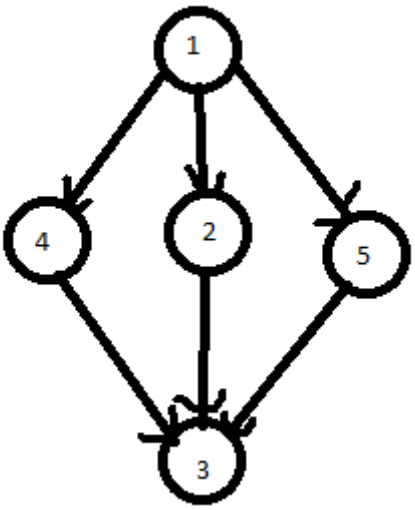
Grafo :: isMultigrafo



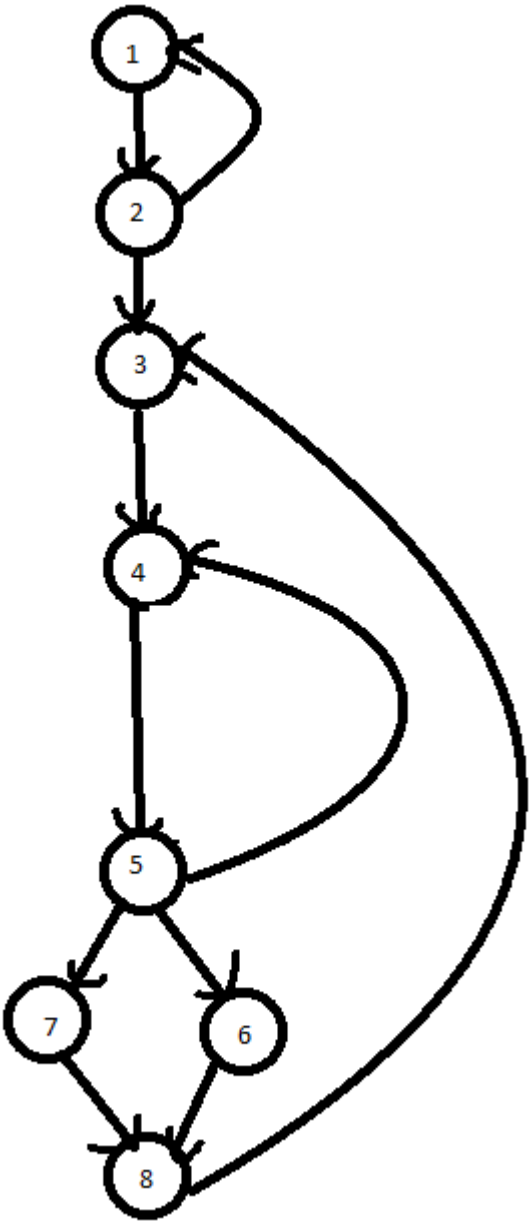
Grafo :: linkExiste



No :: nextEdge



Grafo :: mapeamentoVerticesAdjacentes



6 - Casos de teste, utilizando o Junit:

Classe Aresta:

vizinhos()

```java
@Test
public void vizinhosTest() {
    No no1 = new No("1");
    No no2 = new No("2");
    No no3 = new No("3");

    Aresta aresta1 = new Aresta("a1", no1, no2, 50, true);
    Aresta aresta2 = new Aresta("a2", no2, no3, 10, true);
    Aresta aresta3 = new Aresta("a3", no3, no1, 30, true);

    TreeMap<String, Aresta> mapa1 = new TreeMap();
    mapa1.put("1", aresta1);
    mapa1.put("2", aresta3);

    TreeMap<String, Aresta> mapa2 = new TreeMap();
    mapa2.put("1", aresta1);
    mapa2.put("2", aresta2);

    TreeMap<String, Aresta> mapa3 = new TreeMap();
    mapa3.put("1", aresta2);
    mapa3.put("2", aresta3);

    ArrayList<Aresta> arestas = new ArrayList();
    arestas.add(aresta1);
    arestas.add(aresta2);
    arestas.add(aresta3);

    ArrayList<No> nos = new ArrayList();
    nos.add(no1);
    nos.add(no2);
    nos.add(no3);

    no1.setEdgeList(mapa1);
    no2.setEdgeList(mapa2);
    no3.setEdgeList(mapa3);

    Assert.assertEquals(no2, aresta1.vizinhos("1"));
}
```

noAdjacente()

```java
@Test
public void noAdjacente(){
    No no1 = new No("1");
    No no2 = new No("2");
    No no3 = new No("3");

    Aresta aresta1 = new Aresta("a1", no1, no2, 50, true);
    Aresta aresta2 = new Aresta("a2", no2, no3, 10, true);
    Aresta aresta3 = new Aresta("a3", no3, no1, 30, true);

    TreeMap<String, Aresta> mapa1 = new TreeMap();
    mapa1.put("1", aresta1);
    mapa1.put("2", aresta3);

    TreeMap<String, Aresta> mapa2 = new TreeMap();
    mapa2.put("1", aresta1);
    mapa2.put("2", aresta2);

    TreeMap<String, Aresta> mapa3 = new TreeMap();
    mapa3.put("1", aresta2);
    mapa3.put("2", aresta3);

    ArrayList<Aresta> arestas = new ArrayList();
    arestas.add(aresta1);
    arestas.add(aresta2);
    arestas.add(aresta3);

    ArrayList<No> nos = new ArrayList();
    nos.add(no1);
    nos.add(no2);
    nos.add(no3);

    no1.setEdgeList(mapa1);
    no2.setEdgeList(mapa2);
    no3.setEdgeList(mapa3);

    Assert.assertEquals(no2, aresta1.getNoAdjacente(no1));
}
```

Classe Grafo:

multiGrafo()

```java
@Test
public void MultiGrafoTest() {
    INo no1 = new No("1");
    INo no2 = new No("2");
    INo no3 = new No("3");
    INo no4 = new No("4");
    Aresta aresta1 = new Aresta("a1", no1, no2, 50, true);
    Aresta aresta2 = new Aresta("a2", no2, no3, 10, true);
    Aresta aresta3 = new Aresta("a3", no3, no4, 30, true);
    Aresta aresta4 = new Aresta("a4", no4, no1, 100, true);
    Aresta aresta5 = new Aresta("a5", no1, no3, 10, true);
    Aresta aresta6 = new Aresta("a6", no3, no1, 50, true);
    Aresta aresta7 = new Aresta("a7", no1, no3, 5, true);
    Aresta aresta8 = new Aresta("a8", no4, no1, 20, true);

    ArrayList<Aresta> arestas = new ArrayList();
    arestas.add(aresta1);
    arestas.add(aresta2);
    arestas.add(aresta3);
    arestas.add(aresta4);
    arestas.add(aresta5);
    arestas.add(aresta6);
    arestas.add(aresta7);
    arestas.add(aresta8);

    ArrayList<INo> nos = new ArrayList();
    nos.add(no1);
    nos.add(no2);
    nos.add(no3);
    nos.add(no4);

    Grafo grafo = new Grafo("grafo", "teste", true, nos, arestas);
    Assert.assertTrue(grafo.isMultigrafo());

}
```

getMapaArestasAdjacentes()

```java
@Test
public void getMapaArestasAdjacentesTest() {
    INo no1 = new No("1");
    INo no2 = new No("2");
    INo no3 = new No("3");
    INo no4 = new No("4");
    Aresta aresta1 = new Aresta("a1", no1, no2, 50, true);
    Aresta aresta2 = new Aresta("a2", no2, no3, 10, true);
    Aresta aresta3 = new Aresta("a3", no3, no4, 30, true);
    Aresta aresta4 = new Aresta("a4", no4, no1, 100, true);
    Aresta aresta5 = new Aresta("a5", no1, no3, 10, true);
    Aresta aresta6 = new Aresta("a6", no3, no1, 50, true);
    Aresta aresta7 = new Aresta("a7", no1, no3, 5, true);
    Aresta aresta8 = new Aresta("a8", no4, no1, 20, true);

    ArrayList<Aresta> arestas = new ArrayList();
    arestas.add(aresta1);
    arestas.add(aresta2);
    arestas.add(aresta3);
    arestas.add(aresta4);
    arestas.add(aresta5);
    arestas.add(aresta6);
    arestas.add(aresta7);
    arestas.add(aresta8);

    ArrayList<INo> nos = new ArrayList();
    nos.add(no1);
    nos.add(no2);
    nos.add(no3);
    nos.add(no4);

    Grafo grafo = new Grafo("grafo", "teste", true, nos, arestas);

    /*
    a1 : [a2, a4, a5, a6, a7, a8]
    a2 : [a1, a3, a5, a6, a7]
    a3 : [a2, a4, a5, a6, a7, a8]
    a4 : [a1, a3, a5, a6, a7, a8]
    a5 : [a1, a2, a3, a4, a6, a7, a8]
    a6 : [a1, a2, a3, a4, a5, a7, a8]
    a7 : [a1, a2, a3, a4, a5, a6, a8]
     Map<String, List<String>> mapa = new HashMap();
     List<String> listaAuxiliar1 = new ArrayList();
     listaAuxiliar1.add("a2");
     listaAuxiliar1.add("a4");
     listaAuxiliar1.add("a5");
     listaAuxiliar1.add("a6");
     listaAuxiliar1.add("a7");
     listaAuxiliar1.add("a8");
     mapa.put("a1", listaAuxiliar1);
```

Continuação na próxima página…

```
    List<String> listaAuxiliar2 = new ArrayList();
    listaAuxiliar2.add("a1");
    listaAuxiliar2.add("a3");
    listaAuxiliar2.add("a5");
    listaAuxiliar2.add("a6");
    listaAuxiliar2.add("a7");
    mapa.put("a2", listaAuxiliar2);

    List<String> listaAuxiliar3 = new ArrayList();
    listaAuxiliar3.add("a2");
    listaAuxiliar3.add("a4");
    listaAuxiliar3.add("a5");
    listaAuxiliar3.add("a6");
    listaAuxiliar3.add("a7");
    listaAuxiliar3.add("a8");
    mapa.put("a3", listaAuxiliar3);

    List<String> listaAuxiliar4 = new ArrayList();
    listaAuxiliar4.add("a1");
    listaAuxiliar4.add("a3");
    listaAuxiliar4.add("a5");
    listaAuxiliar4.add("a6");
    listaAuxiliar4.add("a7");
    listaAuxiliar4.add("a8");
    mapa.put("a4", listaAuxiliar4);

    List<String> listaAuxiliar5 = new ArrayList();
    listaAuxiliar5.add("a1");
    listaAuxiliar5.add("a2");
    listaAuxiliar5.add("a3");
    listaAuxiliar5.add("a4");
    listaAuxiliar5.add("a6");
    listaAuxiliar5.add("a7");
    listaAuxiliar5.add("a8");
    List<String> listaAuxiliar6 = new ArrayList();
    listaAuxiliar6.add("a1");
    listaAuxiliar6.add("a2");
    listaAuxiliar6.add("a3");
    listaAuxiliar6.add("a4");
    listaAuxiliar6.add("a5");
    listaAuxiliar6.add("a7");
    listaAuxiliar6.add("a8");
    mapa.put("a6", listaAuxiliar6);

    List<String> listaAuxiliar7 = new ArrayList();
    listaAuxiliar7.add("a1");
    listaAuxiliar7.add("a2");
    listaAuxiliar7.add("a3");
    listaAuxiliar7.add("a4");
    listaAuxiliar7.add("a5");
    listaAuxiliar7.add("a6");
    listaAuxiliar7.add("a8");
    mapa.put("a7", listaAuxiliar7);

    List<String> listaAuxiliar8 = new ArrayList();
    listaAuxiliar8.add("a1");
    listaAuxiliar8.add("a3");
    listaAuxiliar8.add("a4");
    listaAuxiliar8.add("a5");
    listaAuxiliar8.add("a6");
    listaAuxiliar8.add("a7");
    mapa.put("a8", listaAuxiliar8);

    Assert.assertEquals(mapa, grafo.getMapaArestasAdjacentes());

}
```

isCompleto()

```java
@Test
public void isCompletoTest() {
    INo no1 = new No("1");
    INo no2 = new No("2");
    INo no3 = new No("3");

    Aresta aresta1 = new Aresta("a1", no1, no2, 50, false);
    Aresta aresta2 = new Aresta("a2", no2, no3, 10, false);
    Aresta aresta3 = new Aresta("a3", no3, no1, 30, false);

    ArrayList<Aresta> arestas = new ArrayList();
    arestas.add(aresta1);
    arestas.add(aresta2);
    arestas.add(aresta3);

    ArrayList<INo> nos = new ArrayList();
    nos.add(no1);
    nos.add(no2);
    nos.add(no3);

    Grafo grafo = new Grafo("grafo", "teste", false, nos, arestas);

    Assert.assertTrue(grafo.isCompleto());
}
```

isAdjacente()

```java
@Test
public void isAdjacenteTeste() {
    INo no1 = new No("1");
    INo no2 = new No("2");
    INo no3 = new No("3");

    Aresta aresta1 = new Aresta("a1", no1, no2, 50, false);
    Aresta aresta2 = new Aresta("a2", no2, no3, 10, false);
    Aresta aresta3 = new Aresta("a3", no3, no1, 30, false);

    ArrayList<Aresta> arestas = new ArrayList();
    arestas.add(aresta1);
    arestas.add(aresta2);
    arestas.add(aresta3);

    ArrayList<INo> nos = new ArrayList();
    nos.add(no1);
    nos.add(no2);
    nos.add(no3);

    Grafo grafo = new Grafo("grafo", "teste", false, nos, arestas);

    Assert.assertTrue(grafo.isAdjacente(no1, no2));
}
```

getGrauNo()

```java
@Test
public void getGrauNo() {
    INo no1 = new No("1");
    INo no2 = new No("2");
    INo no3 = new No("3");

    Aresta aresta1 = new Aresta("a1", no1, no2, 50, false);
    Aresta aresta2 = new Aresta("a2", no2, no3, 10, false);
    Aresta aresta3 = new Aresta("a3", no3, no1, 30, false);

    ArrayList<Aresta> arestas = new ArrayList();
    arestas.add(aresta1);
    arestas.add(aresta2);
    arestas.add(aresta3);

    ArrayList<INo> nos = new ArrayList();
    nos.add(no1);
    nos.add(no2);
    nos.add(no3);

    Grafo grafo = new Grafo("grafo", "teste", false, nos, arestas);
    Assert.assertEquals(2, grafo.getGrauNo(no3));
}
```

mapeamentoVerticesAdjacentes()

```java
@Test
public void mapeamentoVerticesADjacentes() {
    INo no1 = new No("1");
    INo no2 = new No("2");
    INo no3 = new No("3");
    INo no4 = new No("4");
    Aresta aresta1 = new Aresta("a1", no1, no2, 50, true);
    Aresta aresta2 = new Aresta("a2", no2, no3, 10, true);
    Aresta aresta3 = new Aresta("a3", no3, no4, 30, true);
    Aresta aresta4 = new Aresta("a4", no4, no1, 100, true);
    Aresta aresta5 = new Aresta("a5", no1, no3, 10, true);
    Aresta aresta6 = new Aresta("a6", no3, no1, 50, true);
    Aresta aresta7 = new Aresta("a7", no1, no3, 5, true);
    Aresta aresta8 = new Aresta("a8", no4, no1, 20, true);

    ArrayList<Aresta> arestas = new ArrayList();
    arestas.add(aresta1);
    arestas.add(aresta2);
    arestas.add(aresta3);
    arestas.add(aresta4);
    arestas.add(aresta5);
    arestas.add(aresta6);
    arestas.add(aresta7);
    arestas.add(aresta8);

    ArrayList<INo> nos = new ArrayList();
    nos.add(no1);
    nos.add(no2);
    nos.add(no3);
    nos.add(no4);

    Grafo grafo = new Grafo("grafo", "teste", true, nos, arestas);
    */
    Map<String, List<String>> mapa = new HashMap();

    List<String> listaAuxiliar1 = new ArrayList();
    listaAuxiliar1.add("2");
    listaAuxiliar1.add("3");
    listaAuxiliar1.add("4");
    mapa.put("1", listaAuxiliar1);

    List<String> listaAuxiliar2 = new ArrayList();
    listaAuxiliar2.add("1");
    listaAuxiliar2.add("3");
    mapa.put("2", listaAuxiliar2);

    List<String> listaAuxiliar3 = new ArrayList();
    listaAuxiliar3.add("1");
    listaAuxiliar3.add("2");
    listaAuxiliar3.add("4");
    mapa.put("3", listaAuxiliar3);

    List<String> listaAuxiliar4 = new ArrayList();
    listaAuxiliar4.add("1");
    listaAuxiliar4.add("3");
    mapa.put("4", listaAuxiliar4);

    Assert.assertEquals(mapa, grafo.mapeamentoVerticesAdjacentes());
}
```

linkExiste()

```java
@Test
public void linkExistTest() {
    INo no1 = new No("1");
    INo no2 = new No("2");
    INo no3 = new No("3");

    Aresta aresta1 = new Aresta("a1", no1, no2, 50, true);
    Aresta aresta2 = new Aresta("a2", no2, no3, 10, true);
    Aresta aresta3 = new Aresta("a3", no3, no1, 30, true);

    TreeMap<String, Aresta> mapa1 = new TreeMap();
    mapa1.put("1", aresta1);
    mapa1.put("2", aresta3);

    TreeMap<String, Aresta> mapa2 = new TreeMap();
    mapa2.put("1", aresta1);
    mapa2.put("2", aresta2);

    TreeMap<String, Aresta> mapa3 = new TreeMap();
    mapa3.put("1", aresta2);
    mapa3.put("2", aresta3);

    ArrayList<Aresta> arestas = new ArrayList();
    arestas.add(aresta1);
    arestas.add(aresta2);
    arestas.add(aresta3);

    ArrayList<INo> nos = new ArrayList();
    nos.add(no1);
    nos.add(no2);
    nos.add(no3);

    no1.setEdgeList(mapa1);
    no2.setEdgeList(mapa2);
    no3.setEdgeList(mapa3);

    Grafo grafo = new Grafo("grafo", "teste", true, nos, arestas);

    Assert.assertTrue(grafo.linkExists("1", "2"));

}
```

Classe No:

nextEdge()

```java
@Test
public void nextEdgeTest() {
    No no1 = new No("1");
    No no2 = new No("2");
    No no3 = new No("3");

    Aresta aresta1 = new Aresta("a1", no1, no2, 50, true);
    Aresta aresta2 = new Aresta("a2", no2, no3, 10, true);
    Aresta aresta3 = new Aresta("a3", no3, no1, 30, true);

    TreeMap<String, Aresta> mapa1 = new TreeMap();
    mapa1.put("1", aresta1);
    mapa1.put("2", aresta3);

    TreeMap<String, Aresta> mapa2 = new TreeMap();
    mapa2.put("1", aresta1);
    mapa2.put("2", aresta2);

    TreeMap<String, Aresta> mapa3 = new TreeMap();
    mapa3.put("1", aresta2);
    mapa3.put("2", aresta3);

    ArrayList<Aresta> arestas = new ArrayList();
    arestas.add(aresta1);
    arestas.add(aresta2);
    arestas.add(aresta3);

    ArrayList<No> nos = new ArrayList();
    nos.add(no1);
    nos.add(no2);
    nos.add(no3);

    no1.setEdgeList(mapa1);
    no2.setEdgeList(mapa2);
    no3.setEdgeList(mapa3);

    Assert.assertEquals(aresta1, no1.nextEdge("1"));
}
```

nextPrimeiraAresta()

```java
@Test
public void nextPrimeiraArestaTest() {
    No no1 = new No("1");
    No no2 = new No("2");
    No no3 = new No("3");

    Aresta aresta1 = new Aresta("a1", no1, no2, 50, true);
    Aresta aresta2 = new Aresta("a2", no2, no3, 10, true);
    Aresta aresta3 = new Aresta("a3", no3, no1, 30, true);

    TreeMap<String, Aresta> mapa1 = new TreeMap();
    mapa1.put("1", aresta1);
    mapa1.put("2", aresta3);

    TreeMap<String, Aresta> mapa2 = new TreeMap();
    mapa2.put("1", aresta1);
    mapa2.put("2", aresta2);

    TreeMap<String, Aresta> mapa3 = new TreeMap();
    mapa3.put("1", aresta2);
    mapa3.put("2", aresta3);

    ArrayList<Aresta> arestas = new ArrayList();
    arestas.add(aresta1);
    arestas.add(aresta2);
    arestas.add(aresta3);

    ArrayList<No> nos = new ArrayList();
    nos.add(no1);
    nos.add(no2);
    nos.add(no3);

    no1.setEdgeList(mapa1);
    no2.setEdgeList(mapa2);
    no3.setEdgeList(mapa3);

    Assert.assertEquals(aresta1, no1.primeiraAresta());

}
```

Classe Djikistra:

```java
@Test
public void testeCalcularDijkistra() {
    No no1 = new No("1");
    No no2 = new No("2");
    No no3 = new No("3");
    No no4 = new No("4");
    No no5 = new No("5");
    Aresta aresta1 = new Aresta("a1", no1, no2, 50, true);
    Aresta aresta2 = new Aresta("a2", no1, no5, 10, true);
    Aresta aresta3 = new Aresta("a3", no1, no3, 30, true);
    Aresta aresta4 = new Aresta("a4", no1, no4, 100, true);
    Aresta aresta5 = new Aresta("a5", no5, no4, 10, true);
    Aresta aresta6 = new Aresta("a6", no3, no4, 50, true);
    Aresta aresta7 = new Aresta("a7", no3, no2, 5, true);
    Aresta aresta8 = new Aresta("a8", no4, no2, 20, true);

    ArrayList<Aresta> arestas = new ArrayList();
    arestas.add(aresta1);
    arestas.add(aresta2);
    arestas.add(aresta3);
    arestas.add(aresta4);
    arestas.add(aresta5);
    arestas.add(aresta6);
    arestas.add(aresta7);
    arestas.add(aresta8);

    ArrayList<INo> nos = new ArrayList();
    nos.add(no1);
    nos.add(no2);
    nos.add(no3);
    nos.add(no4);
    nos.add(no5);

    Grafo grafo = new Grafo("grafo", "teste", true, nos, arestas);

    Dijkistra dijkistra = new Dijkistra(grafo);


    Map<String, String> respostaEsperada = new HashMap<String, String>();
    respostaEsperada.put("1", "1");
    respostaEsperada.put("2", "3");
    respostaEsperada.put("3", "1");
    respostaEsperada.put("4", "5");
    Assert.assertEquals(respostaEsperada, dijkistra.calcularDijkstra(no1));

}
```

7 - Relatório da ferramenta de Teste de cobertura:

JaCoCoverage analysis of project "SourceCode" (powered by JaCoCo from EclEmma)

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
controller		0%		0%	89	89	340	340	74	74	14	14
model		43%		39%	141	210	258	448	67	109	6	9
planaridade		0%		0%	95	95	147	147	19	19	4	4
model.algoritmos		0%		0%	72	72	200	200	30	30	6	6
utils		0%		0%	26	26	76	76	21	21	2	2
Total	4.973 of 5.905	16%	399 of 478	17%	423	492	1.021	1.211	211	253	32	35

Grafo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
verticesIndependentes()		0%		0%	5	5	22	22	1	1
isMultigrafo()		43%		25%	8	9	6	11	0	1
gerarVerticesAdjacentes(String)		0%		0%	2	2	13	13	1	1
lambda$getGraus$0(Map, INo)		0%		0%	4	4	7	7	1	1
adicionarAresta(No, No, int)		0%		n/a	1	1	6	6	1	1
getArestasDoNoAtual(String)		0%		0%	2	2	5	5	1	1
adicionarAresta(No, No)		0%		n/a	1	1	4	4	1	1
lambda$getGrausDeRecepcao$2(Map, INo)		0%		0%	3	3	7	7	1	1
lambda$getGrausDeEmissao$1(Map, INo)		0%		0%	3	3	7	7	1	1
nodePositionInMatrix()		0%		0%	2	2	7	7	1	1
isAdjacente(INo, INo)		70%		62%	4	9	3	8	0	1
possuiElementoRepetido(List)		0%		0%	2	2	2	2	1	1
lambda$gerarVerticesAdjacentes$18(String, Aresta)		0%		0%	3	3	1	1	1	1
lambda$getArestasDoNoAtual$3(String, Aresta)		0%		0%	3	3	1	1	1	1
getNosSucessores(INo)		0%		n/a	1	1	3	3	1	1
getNosAntecessores(INo)		0%		n/a	1	1	3	3	1	1
adicionarAresta(Aresta)		0%		n/a	1	1	4	4	1	1
getGraus()		0%		n/a	1	1	3	3	1	1
getGrausDeEmissao()		0%		n/a	1	1	3	3	1	1
getGrausDeRecepcao()		0%		n/a	1	1	3	3	1	1
getNosSucessores()		0%		n/a	1	1	3	3	1	1
getNosAntecessores()		0%		n/a	1	1	3	3	1	1
noExiste(String)		68%		33%	3	4	3	6	0	1
lambda$gerarVerticesAdjacentes$20(String, INo)		0%		0%	2	2	1	1	1	1
lambda$getNosAntecessores$16(Map, INo)		0%		n/a	1	1	2	2	1	1
lambda$getNosSucessores$11(Map, INo)		0%		n/a	1	1	2	2	1	1
linkExists(String, String)		77%		40%	5	6	1	5	0	1
getOrdem()		0%		n/a	1	1	1	1	1	1
isRegular()		91%		75%	1	3	1	5	0	1
getNo(String)		90%		67%	2	4	2	7	0	1
getId()		0%		n/a	1	1	1	1	1	1
getTipo()		0%		n/a	1	1	1	1	1	1
isTipoAresta()		0%		n/a	1	1	1	1	1	1
getTipoAresta()		0%		n/a	1	1	1	1	1	1
getMapaArestasAdjacentes()		100%		100%	0	8	0	32	0	1
mapeamentoVerticesAdjacentes()		100%		90%	1	6	0	21	0	1
getMatrizIncidencia()		100%		100%	0	3	0	16	0	1
getMatrizAdjacencia()		100%		100%	0	2	0	10	0	1
getGrauNo(INo)		100%		100%	0	5	0	9	0	1
lambda$getMatrizAdjacencia$7(int[][], Map, Aresta)		100%		n/a	0	1	0	2	0	1
lambda$getMatrizIncidencia$9(int[][], Map, Map, Aresta)		100%		n/a	0	1	0	2	0	1
lambda$getMatrizAdjacencia$8(int[][], Map, Aresta)		100%		n/a	0	1	0	2	0	1
Grafo(String, String, boolean, ArrayList, ArrayList)		100%		n/a	0	1	0	7	0	1
lambda$getMatrizIncidencia$10(int[][], Map, Map, Aresta)		100%		n/a	0	1	0	2	0	1
setId(String)		100%		n/a	0	1	0	2	0	1
setTipo(String)		100%		n/a	0	1	0	2	0	1
setTipoAresta(boolean)		100%		n/a	0	1	0	2	0	1
setNos(ArrayList)		100%		n/a	0	1	0	2	0	1
setArestas(ArrayList)		100%		n/a	0	1	0	2	0	1
getNos()		100%		n/a	0	1	0	1	0	1
getArestas()		100%		n/a	0	1	0	1	0	1
Total	764 of 1.561	51%	76 of 148	49%	88	139	151	298	41	65

Aresta

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
vizinhos(String)		32%		17%	6	7	3	5	0	1
getArestaById(String, ArrayList)		0%		0%	3	3	5	5	1	1
Aresta(String, INo, INo)		0%		n/a	1	1	5	5	1	1
getNoAdjacente(INo)		77%		50%	2	3	2	5	0	1
getValor()		0%		n/a	1	1	1	1	1	1
Aresta(String, INo, INo, Integer, boolean)		100%		n/a	0	1	0	7	0	1
Aresta(String, INo, INo, Integer)		100%		n/a	0	1	0	6	0	1
setDirected(boolean)		100%		n/a	0	1	0	2	0	1
setId(String)		100%		n/a	0	1	0	2	0	1
setOrigem(INo)		100%		n/a	0	1	0	2	0	1
setDestino(INo)		100%		n/a	0	1	0	2	0	1
setValor(Integer)		100%		n/a	0	1	0	2	0	1
isDirected()		100%		n/a	0	1	0	1	0	1
getId()		100%		n/a	0	1	0	1	0	1
getOrigem()		100%		n/a	0	1	0	1	0	1
getDestino()		100%		n/a	0	1	0	1	0	1
Total	63 of 156	60%	16 of 20	20%	13	26	16	48	3	16

No

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
nextEdge(String)		0%		0%	5	5	4	4	1	1
setId(String)		0%		n/a	1	1	2	2	1	1
getNoById(String, List)		85%		50%	2	3	2	5	0	1
primeiraAresta()		86%		50%	1	2	1	3	0	1
No(String)		100%		n/a	0	1	0	3	0	1
setEdgeList(TreeMap)		100%		n/a	0	1	0	2	0	1
getId()		100%		n/a	0	1	0	1	0	1
Total	42 of 84	50%	11 of 14	21%	9	14	9	20	2	7

Dijkistra

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
calcularDijkstra(INo)		0%		0%	11	11	39	39	1	1
menorEstimativa(Map, List)		0%		0%	4	4	10	10	1	1
calcularDijkstra(INo, INo)		0%		0%	2	2	9	9	1	1
arestasSaemNoAtual(INo, List)		0%		n/a	1	1	3	3	1	1
lambda$arestasSaemNoAtual$0(INo, List, Aresta)		0%		0%	3	3	1	1	1	1
Dijkistra(Grafo)		0%		n/a	1	1	4	4	1	1
lambda$arestasSaemNoAtual$1(List, Aresta)		0%		n/a	1	1	2	2	1	1
Total	336 of 336	0%	32 of 32	0%	23	23	67	67	7	7

8 - Testes funcionais:

Teste Grafo completo direcionado:

1	*open*	http://localhost:8080/SourceCode/	
2	*click*	id=nomeGrafo	
3	*type*	id=nomeGrafo	Orientado direcionado
4	*click*	id=gDirecionado	
5	*click*	id=no	
6	*type*	id=no	a
7	*click*	id=adiconaCampo	
8	*mouse over*	id=adiconaCampo	
9	*mouse out*	id=adiconaCampo	
10	*type*	id=no1	b
11	*click*	id=adiconaCampo	
12	*type*	id=no2	c
13	*click*	id=aresta	
14	*type*	id=aresta	a,b
15	*click*	id=adiconaAresta	
16	*type*	id=aresta1	a,c
17	*click*	id=adiconaAresta	
18	*mouse over*	id=adiconaAresta	
19	*mouse out*	id=adiconaAresta	
20	*type*	id=aresta2	b,a
21	*click*	id=adiconaAresta	
22	*mouse over*	id=adiconaAresta	
23	*mouse out*	id=adiconaAresta	
24	*type*	id=aresta3	b,c
25	*click*	id=adiconaAresta	
26	*type*	id=aresta4	c,a
27	*click*	id=adiconaAresta	
28	*type*	id=aresta5	c,b
29	*click*	name=bntIncluir	
30	*click*	css=input.button.button-cyan	
31	*assert text*	css=.col-md-4:nth-child(7) li:nth-child(2) li:nth-child(1)	Vértice Adjacente: A

Teste Grafo completo não direcionado

1	*open*	http://localhost:8080/SourceCode/	
2	*set window size*	836x963	
3	*click*	id=nomeGrafo	
4	*type*	id=nomeGrafo	grafo completo nao direcionado
5	*click*	id=no	
6	*type*	id=no	a
7	*click*	id=adiconaCampo	
8	*mouse over*	id=adiconaCampo	
9	*mouse out*	id=adiconaCampo	
10	*type*	id=no1	b
11	*click*	id=adiconaCampo	
12	*type*	id=no2	c
13	*click*	id=aresta	
14	*type*	id=aresta	a,b
15	*click*	id=adiconaAresta	
16	*mouse over*	id=adiconaAresta	
17	*mouse out*	id=adiconaAresta	
18	*type*	id=aresta1	a,c
19	*click*	id=adiconaAresta	
20	*type*	id=aresta2	b,c
21	*click*	id=adiconaAresta	
22	*click*	css=p:nth-child(5) > #removerArest a	
23	*click*	name=bntIncluir	
24	*click*	css=input.button.button-cyan	
25	*assert text*	css=.col-md-4:nth-child(7) li:nth-chil	Vértice Adjacente: A

Teste Grafo conexo direcionado:

1	*open*	http://localhost:8080/SourceCode/	
2	*set window size*	836x963	
3	*click*	id=nomeGrafo	
4	*type*	id=nomeGrafo	Grafo conexo direcionado
5	*click*	id=gDirecionado	
6	*click*	id=no	
7	*type*	id=no	a
8	*click*	id=adiconaCampo	
9	*mouse over*	id=adiconaCampo	
10	*mouse out*	id=adiconaCampo	
11	*type*	id=no1	b
12	*click*	id=adiconaCampo	
13	*type*	id=no2	c
14	*click*	id=aresta	
15	*type*	id=aresta	a,b
16	*click*	id=adiconaAresta	
17	*mouse over*	id=adiconaAresta	
18	*mouse out*	id=adiconaAresta	
19	*type*	id=aresta1	b,a
20	*click*	id=adiconaAresta	
21	*type*	id=aresta2	b,c
22	*click*	id=adiconaAresta	
23	*type*	id=aresta3	c,b
24	*click*	name=bntIncluir	
25	*click*	css=input.button.button-cyan	
26	*click*	css=small	
27	*assert text*	css=small	- Grafo direcionado
28	*assert text*	css=.col-md-4:nth-child(2) li:nth-child(3)	A3: B - C.

Teste Grafo conexo não direcionado:

1	*open*	http://localhost:8080/SourceCode/	
2	*set window size*	836x963	
3	*click*	id=nomeGrafo	
4	*type*	id=nomeGrafo	Grafo conexo
5	*type*	id=no	a
6	*click*	id=adiconaCampo	
7	*type*	id=no1	b
8	*click*	id=adiconaCampo	
9	*type*	id=no2	c
10	*click*	id=aresta	
11	*type*	id=aresta	a,b
12	*click*	id=adiconaAresta	
13	*type*	id=aresta1	b,c
14	*click*	name=bntIncluir	
15	*click*	css=input.button.button-cyan	
16	*assert text*	css=.col-md-4:nth-child(2) li:nth-child(1)	A1: A - B.

Teste Nó com grau positivo:

1	*open*	http://localhost:8080/SourceCode/	
2	*set window size*	836x963	
3	*click*	id=nomeGrafo	
4	*type*	id=nomeGrafo	Grafo
5	*click*	id=no	
6	*type*	id=no	a
7	*click*	id=adiconaCampo	
8	*type*	id=no1	b
9	*click*	id=adiconaCampo	
10	*type*	id=no2	c
11	*click*	id=aresta	
12	*type*	id=aresta	b,c
13	*click*	name=bntIncluir	
14	*click*	css=input.button.button-cyan	
15	*assert text*	css=.col-md-4:nth-child(3) li:nth-child(2)	B: 1

Teste Grafo simples:

3	*click*	id=nomeGrafo	
4	*type*	id=nomeGrafo	Grafo simples
5	*click*	id=no	
6	*type*	id=no	a
7	*click*	id=adiconaCampo	
8	*mouse over*	id=adiconaCampo	
9	*mouse out*	id=adiconaCampo	
10	*type*	id=no1	b
11	*click*	id=adiconaCampo	
12	*mouse over*	id=adiconaCampo	
13	*mouse out*	id=adiconaCampo	
14	*type*	id=no2	c
15	*click*	id=aresta	
16	*type*	id=aresta	a,b
17	*click*	id=adiconaAresta	
18	*mouse over*	id=adiconaAresta	
19	*mouse out*	id=adiconaAresta	
20	*type*	id=aresta1	b,c
21	*click*	id=adiconaAresta	
22	*mouse over*	id=adiconaAresta	
23	*mouse out*	id=adiconaAresta	
24	*type*	id=aresta2	c,a
25	*click*	name=bntIncluir	
26	*click*	css=input.button.button-cyan	
27	*assert text*	css=.col-md-4:nth-child(5) li:nth-child(2) > strong:nth-child(1)	B

Teste No com grau positivo e com laço:

1	*open*	http://localhost:8080/SourceCode/	
2	*set window size*	836x963	
3	*click*	id=nomeGrafo	
4	*type*	id=nomeGrafo	Grafo
5	*click*	id=no	
6	*type*	id=no	a
7	*click*	id=adiconaCampo	
8	*mouse over*	id=adiconaCampo	
9	*mouse out*	id=adiconaCampo	
10	*type*	id=no1	b
11	*click*	id=adiconaCampo	
12	*type*	id=no2	c
13	*click*	id=aresta	
14	*type*	id=aresta	a,a
15	*click*	id=adiconaAresta	
16	*type*	id=aresta1	a,b
17	*click*	id=adiconaAresta	
18	*mouse over*	id=adiconaAresta	
19	*mouse out*	id=adiconaAresta	
20	*type*	id=aresta2	b,c
21	*click*	name=bntIncluir	
22	*click*	css=input.button.button-cyan	
23	*assert text*	css=.col-md-4:nth-child(3) li:nth-child(1)	A: 2

Teste Nó com grau zero:

1	*open*	http://localhost:8080/SourceCode/	
2	*set window size*	836x963	
3	*click*	id=nomeGrafo	
4	*type*	id=nomeGrafo	Grafo
5	*click*	id=no	
6	*type*	id=no	a
7	*click*	id=adiconaCampo	
8	*mouse over*	id=adiconaCampo	
9	*mouse out*	id=adiconaCampo	
10	*type*	id=no1	b
11	*click*	id=adiconaCampo	
12	*mouse over*	id=adiconaCampo	
13	*mouse out*	id=adiconaCampo	
14	*type*	id=no2	c
15	*click*	id=aresta	
16	*type*	id=aresta	c,b
17	*click*	name=bntIncluir	
18	*click*	css=input.button.button-cyan	
19	*mouse down at*	css=.col-md-4:nth-child(3) li:nth-child(1)	27.5,8.600006103515625
20	*mouse move at*	css=.col-md-4:nth-child(3) li:nth-child(1)	27.5,8.600006103515625
21	*mouse up at*	css=.col-md-4:nth-child(3) li:nth-child(1)	27.5,8.600006103515625
22	*assert text*	css=.col-md-4:nth-child(3) li:nth-child(1)	A: 0

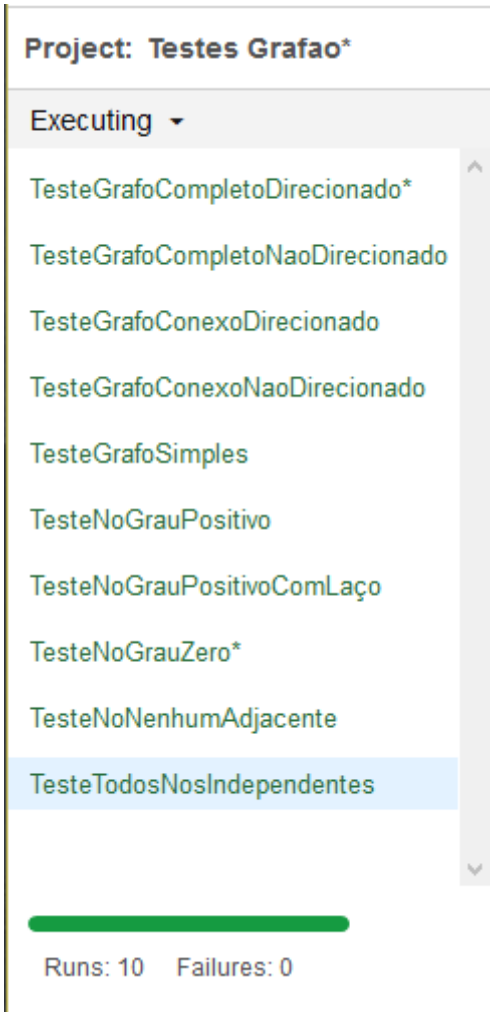
Teste Nó com nenhum outro nó adjacente:

1	*open*	http://localhost:8080/SourceCode/	
2	*click*	id=nomeGrafo	
3	*type*	id=nomeGrafo	grafo
4	*type*	id=no	a
5	*click*	id=adiconaCampo	
6	*type*	id=no1	b
7	*click*	id=adiconaCampo	
8	*type*	id=no2	c
9	*click*	id=aresta	
10	*type*	id=aresta	b,c
11	*click*	id=adiconaAresta	
12	*type*	id=aresta1	c,b
13	*click*	name=bntIncluir	
14	*click*	css=input.button.button-cyan	
15	*click*	css=.col-md-4:nth-child(6) li	
16	*assert text*	css=.col-md-4:nth-child(6) li	A: 0
17	*set window size*	836x768	

Teste todos os nós independentes:

1	*open*	http://localhost:8080/SourceCode/	
2	*click*	id=nomeGrafo	
3	*type*	id=nomeGrafo	grafao
4	*type*	id=no	a
5	*click*	id=adiconaCampo	
6	*type*	id=no1	b
7	*click*	id=adiconaCampo	
8	*type*	id=no2	c
9	*click*	id=aresta	
10	*type*	id=aresta	a,a
11	*click*	id=adiconaAresta	
12	*type*	id=aresta1	b,b
13	*click*	id=adiconaAresta	
14	*type*	id=aresta2	c,c
15	*click*	name=bntIncluir	
16	*click*	css=input.button.button-cyan	
17	*click*	css=.col-md-4:nth-child(9) > ul > li:nth-child(1)	
18	*click*	css=.col-md-4:nth-child(9) > ul > li:nth-child(1)	
19	*double click*	css=.col-md-4:nth-child(9) > ul > li:nth-child(1)	
20	*assert text*	css=.col-md-4:nth-child(9) > ul > li:nth-child(1)	A\nB\nC
21	*click*	css=.col-md-4:nth-child(9) > ul > li:nth-child(2)	
22	*click*	css=.col-md-4:nth-child(9) > ul > li:nth-child(2)	
23	*double click*	css=.col-md-4:nth-child(9) > ul > li:nth-child(2)	
24	*assert text*	css=.col-md-4:nth-child(9) > ul > li:nth-child(2)	B\nA\nC
25	*click*	css=.col-md-4:nth-child(9) li:nth-child(3)	
26	*click*	css=.col-md-4:nth-child(9) li:nth-child(3)	
27	*double click*	css=.col-md-4:nth-child(9) li:nth-child(3)	
28	*assert text*	css=.col-md-4:nth-child(9) li:nth-child(3)	C\nA\nB
29	*set window size*	652x768	

Todos os testes aprovados:



9 - Testes com objetos Mock:

A classe escolhida para realizar os testes Mock foi a classe No, para isso foi criada a interface INo.

Interface INo:

```
public interface INo {

    public void setEdgeList(TreeMap<String, Aresta> edgeList);

    public String getId();

    public void setId(String id);

    public static No getNoById(String id, List<No> nos) {
        for (No no : nos) {
            if (no.getId().equals(id)) {
                return no;
            }
        }
        return null;

    }

    public Aresta primeiraAresta();

    public Aresta nextEdge(String id);
}
```

Classe No:

```java
public class No implements INo {

    private TreeMap<String, Aresta> edgeList;
    private String id;

    public No(String id) {
        this.id = id;
    }

    public void setEdgeList(TreeMap<String, Aresta> edgeList) {
        this.edgeList = edgeList;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public static INo getNoById(String id, List<INo> nos) {
        for (INo no : nos) {
            if (no.getId().equals(id)) {
                return no;
            }
        }
        return null;
    }

    public Aresta primeiraAresta() {
        if (edgeList.isEmpty()) {
            return null;
        }
        return edgeList.get(edgeList.firstKey());
    }

    public Aresta nextEdge(String id) {

        if (!edgeList.containsKey(id) || (edgeList.lastKey() == null ? id == null : edgeList.lastKey()
        .equals(id))) {
            return null;
        }
        SortedMap<String, Aresta> map = edgeList.tailMap(id);
        return edgeList.get(map.firstKey());
    }
}
```

Classe de teste Mock:

```java
public class MockTest {

    @Test
    public void noExisteTeste() {
        INo no = EasyMock.createMock(INo.class);
        EasyMock.expect(no.getId()).andReturn("n1");

        EasyMock.replay(no);

        Aresta aresta = new Aresta("a1", no, null, 50);
        ArrayList<Aresta> arestas = new ArrayList();
        ArrayList<INo> nos = new ArrayList();
        arestas.add(aresta);
        nos.add(no);
        Grafo grafo = new Grafo("1", "teste", true, nos, arestas);

        Assert.assertTrue(grafo.noExiste("n1"));
        EasyMock.verify(no);

    }


    @Test
    public void getNoByIdTest(){
        INo no = EasyMock.createMock(INo.class);
        EasyMock.expect(no.getId()).andReturn("n1");

        EasyMock.replay(no);

        List<INo> nos = new ArrayList();
        nos.add(no);

        Assert.assertEquals(no, No.getNoById("n1", nos) );
        EasyMock.verify(no);

    }
}
```