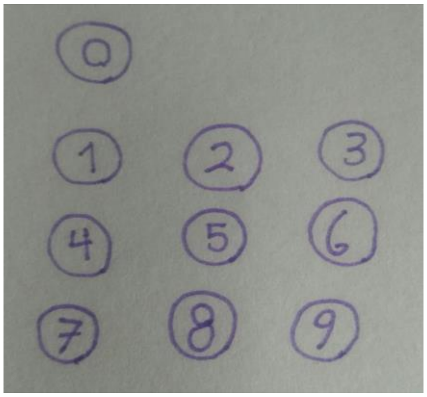
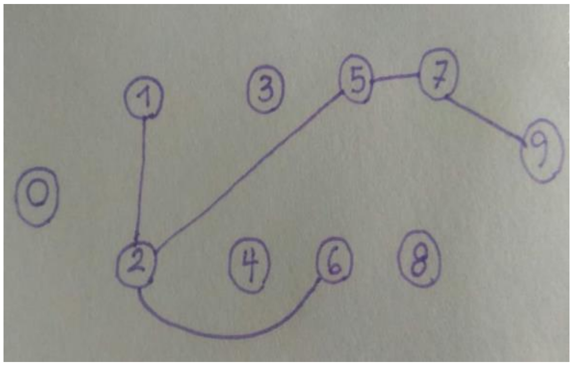
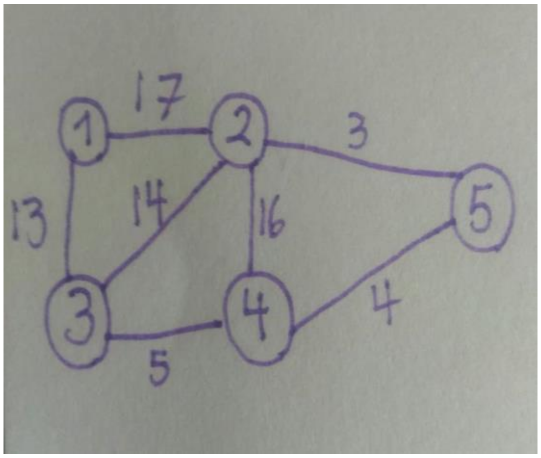
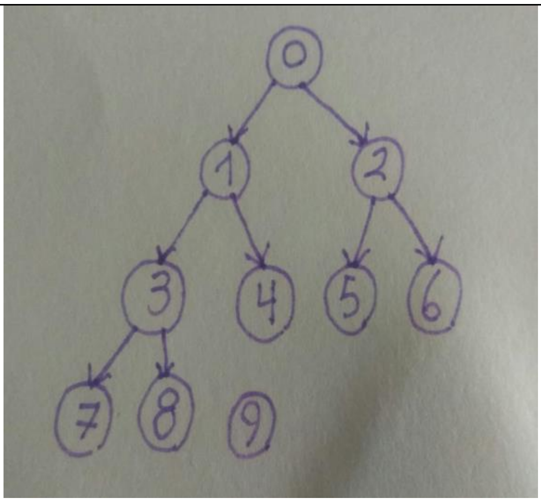
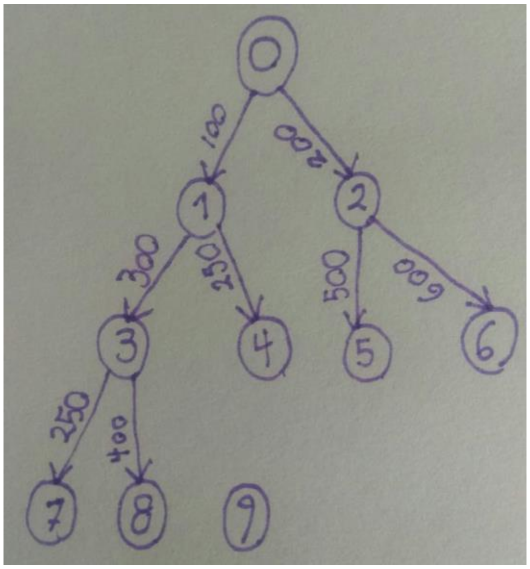
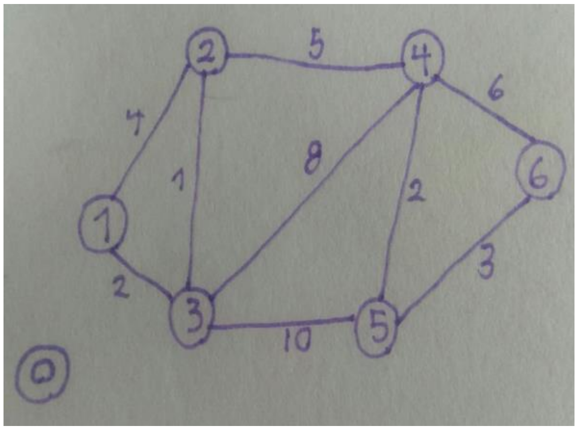
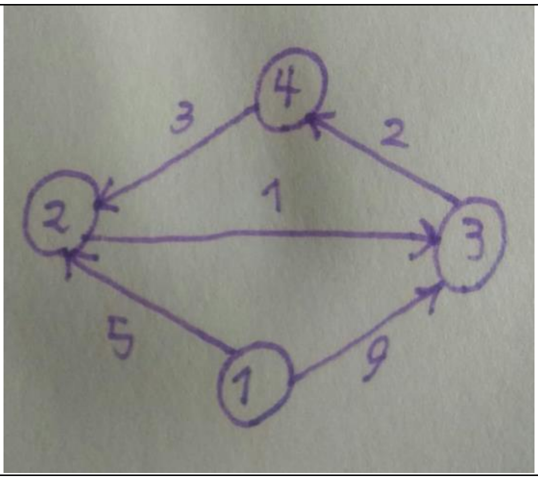
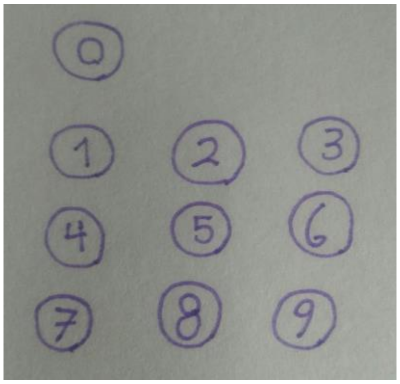
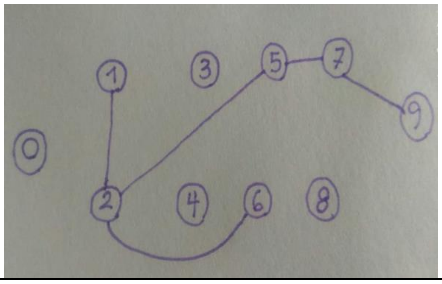
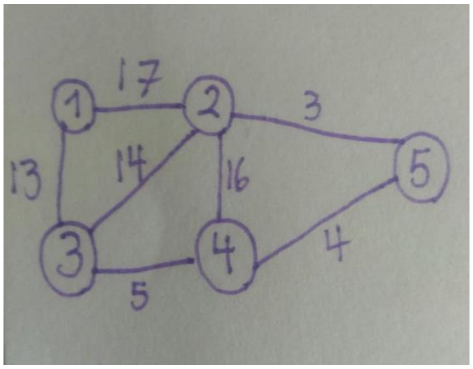
Scenario configuration

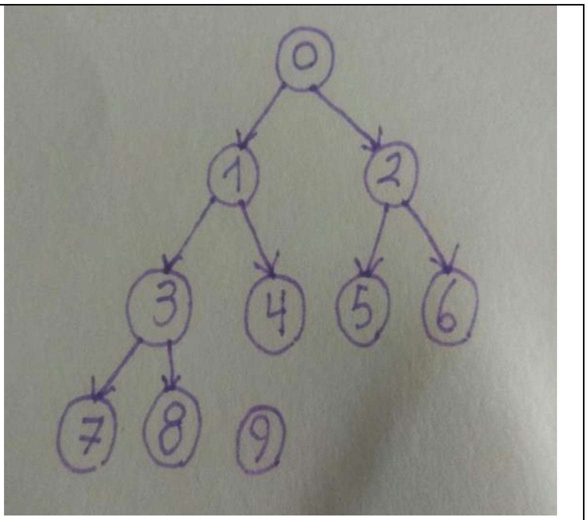
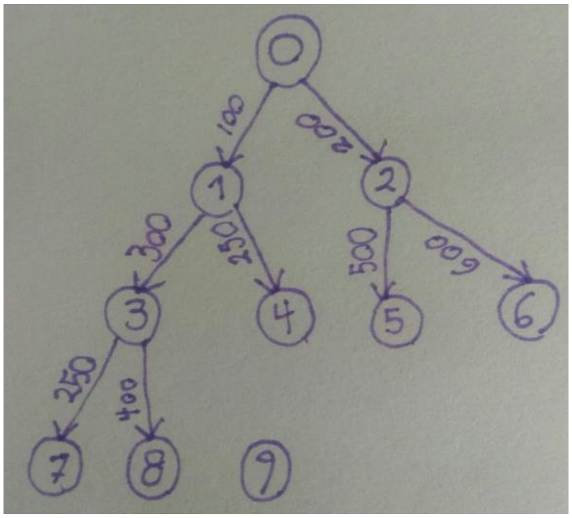
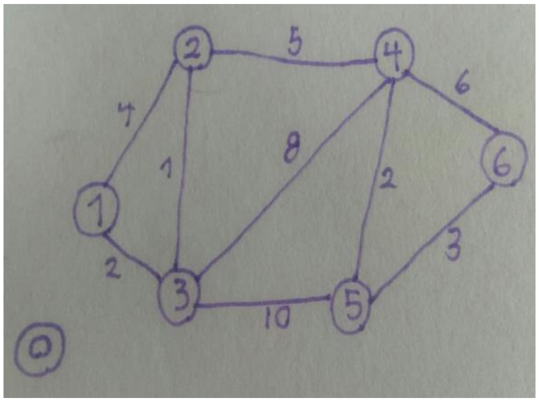
Class	Class under test	Scenario´s name	Scenario
TestAdjacencyMatrix	AdjacencyMatrix	setupScenario1()	
TestAdjacencyMatrix	AdjacencyMatrix	setupScenario2()	
TestAdjacencyMatrix	AdjacencyMatrix	setupScenario3()	
TestAdjacencyMatrix	AdjacencyMatrix	setupScenario4()	 17 1 2 3 13 14 16 5 3 4 4 5

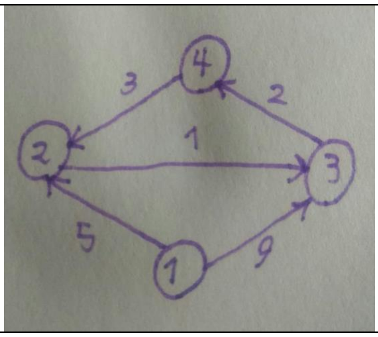
xTestAdjacencyMatrix	**AdjacencyMatrix**	**setupScenario5()**	
TestAdjacencyMatrix	**AdjacencyMatrix**	**setupScenario6()**	 0 100 200 1 2 300 250 500 600 3 4 5 6 250 400 7 8 9
TestAdjacencyMatrix	**AdjacencyMatrix**	**setupScenario7()**	 2 5 4 4 6 1 8 2 6 1 2 3 3 10 5 0

TestAdjacencyMatrix	**AdjacencyMatrix**	**setupScenario8()**	

Scenario configuration

Class	Class under test	Scenario´s name	Scenario
TestAdjacencyList	AdjacencyList	setupScenario1()	
TestAdjacencyList	AdjacencyList	setupScenario2()	
TestAdjacencyList	AdjacencyList	setupScenario3()	
TestAdjacencyList	AdjacencyList	setupScenario4()	 17 1 2 3 13 14 16 5 3 4 4 5

TestAdjacencyList	**AdjacencyList**	**setupScenario5()**	
TestAdjacencyList	**AdjacencyList**	**setupScenario6()**	 0 100 200 1 2 300 250 500 600 3 4 5 6 250 400 7 8 9
TestAdjacencyList	**AdjacencyList**	**setupScenario7()**	 2 5 4 6 4 1 8 6 2 1 3 2 3 10 5 0

TestAdjacencyList	**AdjacencyList**		

AdjacencyMatrix Test cases

Test Objective: Verifying the correct creation of an undirected graph.				
adjacencyMatrixTest1()				
Class under test	**Method**	**Scenario**	**Input**	**Outcome**
AdjacencyMatrix	**adjacencyMatrix()**	**Scenario1**		**The adjacencyMatrix must be different than null.**

Test Objective: Verifying the correct creation of a directed graph.				
adjacencyMatrixTest2()				
Class under test	**Method**	**Scenario**	**Input**	**Outcome**
AdjacencyMatrix	**adjacencyMatrix()**	**Scenario2**		**The adjacencyMatrix must be different than null.**

Test Objective: Verifying the correct adding of a Vertex to the graph when it is empty (using Adjacency Matrix to represent).				
addVertexTest1()				
Class under test	**Method**	**Scenario**	**Input**	**Outcome**
AdjacencyMatrix	**addVertex()**	**Scenario1**	**Vertex<v> v**	**The size of the graph is one.**

Test Objective: Verifying the correct adding of a Vertex to the graph when it has Vertexes already (using Adjacency Matrix to represent).

addVertexTest2()

Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	**addVertex()**	**Scenario2**	**Vertex<v> v**	**The size of the graph is the amount off vertexes added in the creation of the scenario plus one.**

Test Objective: Verifying the correct adding of a Vertex to the graph when its matrix has already reached its limit (it`s full) (using Adjacency Matrix to represent).

addVertexTest3()

Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	**addVertex()**	**Scenario3**	**Vertex<v> v**	**The size of the graph is the amount off vertexes added in the creation of the scenario plus one.** **Besides the Matrix of adjacency must increase its original size in order to fit more nodes when needed.**

Test Objective: Verifying the correct removing of a Vertex from the graph when the graph is empty (using matrix of adjacency to represent).

removeVertexTest1()

Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	**removeVertex()**	**Scenario1**	**Vertex<v> v //the object we want to remove**	**The size of the graph is the amount off vertexes added in the creation of the scenario minus one. Since the graph is empty in this scenario, nothing cant be deleted.**

Test Objective: Verifying the correct removing of a Vertex from the graph when the graph is empty (using matrix of adjacency to represent).				
removeVertexTest2()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	removeVertex()	Scenario2	Vertex<v> v //the object we want to remove	The size of the graph is the amount off vertexes added in the creation of the scenario minus one.

Test Objective: Verifying the correct removing of a Vertex from the graph when the graph is not empty, and its matrix is full (using matrix of adjacency to represent).				
removeVertexTest3()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	removeVertex()	Scenario7	Vertex<v> v // the vertex that we want to remove.	The size of the graph is the amount off vertexes added in the creation of the scenario minus one.

Test Objective: Verify the correct adding of an edge to an undirected graph (the graph is empty).				
addEdgeTest1()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	addEdge()	Scenario1	Vertex<v> v1 && Vertex<v> v2	The addEdge() method must return true. Since the graph is empty and one can´t add an edge when there is no Vertexes this test will return false.

Test Objective: Verify the correct adding of an edge to a directed graph.				
addEdgeTest2()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	addEdge()	Scenario2	Vertex<v> v1 && Vertex<v> v2	The addEdge() method must return true. If the edge was properly added, the adjacency matrix in the indexes (v1, v2) must be 1, otherwise the edge was not added. If the graph is directed the adjacency matrix in the indexes (v1, v2) must be 1& (v2,v1) must be 0.

Test Objective: Verify the incorrect adding of an edge to the graph. An edge will be added between an existing and an inexistent Vertex.				
addEdgeTest3()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	addEdge()	Scenario3	Vertex<v> v1 && Vertex<v> v2	The addEdge() method must return false. In this test two vertex will be passed but one of them will not exist in the graph, therefore the edge cannot be added.

Test Objective: Verify the correct adding of a weighted edge to the graph (the graph is empty).				
addWeightedEdgeTest1()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	addEdge()	Scenario1	Vertex<v> v1 && Vertex<v> v2	The addEdge() method must return true. Since the graph is empty and one can´t add an edge when there is no Vertexes this test will return false.

Test Objective: Verify the correct adding of a weighted edge to the graph.

addWeightedEdgeTest2()

Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	addEdge()	Scenario2	-Vertex<v> v1 && Vertex<v> v2 -double weight	The addEdge() method must return true. The index of the weights matrix in the position [v1][v2] must be equal to the weight of the edge the adjacency matrix in the indexes [v1][v2] && [v2][v1] must be 1

Test Objective: Verify the correct adding of an edge to the graph.

addWeightedEdgeTest3()

Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	addEdge()	Scenario3	-Vertex<v> v1 && Vertex<v> v2	The addEdge() method must return true. The index of the weights matrix in the position [v1][v2] must be equal to the weight of the edge

Test Objective: Verify the correct functioning of the bfs algorithm.

bfsTest1()

Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	bfs1()	Scenario4	Vertex<v> origin	A list of previous indexes (the predecessor of a node in a certain index) is output. Since we built the scenario we can assert the expected list. If the output list matches the expected list the assertion must be true.

Test Objective: Verify the correct functioning of the bfs algorithm.

bfsTest2()

Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	bfs()	Scenario3	Vertex<v> origin	A list of previous indexes (the predecessor of a node in a certain index) is output. Since we built the scenario we can assert the expected list. If the output list matches the expected list the assertion must be true.

Test Objective: Verify the correct functioning of the bfs algorithm.				
bfsTest3()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	bfs()	Scenario7	Vertex<v> origin	A list of previous indexes (the predecessor of a node in a certain index) is output. Since we built the scenario we can assert the expected list. If the output list matches the expected list the assertion must be true.

Test Objective: Verify the correct functioning of the bfs algorithm.				
dfsTest1()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	dfs()	Scenario4	Vertex<v> origin	A list of previous indexes (the predecessor of a node in a certain index) is output. Since we built the scenario, we can assert the expected list. If the output list matches the expected list the assertion must be true.

Test Objective: Verify the correct functioning of the dfs algorithm.				
dfsTest2()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	dfs()	Scenario3	Vertex<v> origin	A list of previous indexes (the predecessor of a node in a certain index) is output. Since we built the scenario, we can assert the expected list. If the output list matches the expected list the assertion must be true.

Test Objective: Verify the correct functioning of the dfs algorithm.				
dfsTest3()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	dfs()	Scenario7	Vertex<v> origin	A list of previous indexes (the predecessor of a node in a certain index) is output. Since we built the scenario, we can assert the expected list. If the output list matches the expected list the assertion must be true.

<table>
<tr><td colspan="5"><b>Test Objective: Verify the correct functioning of the Prim´s algorithm.</b></td></tr>
<tr><td colspan="5"><b>primsAlgorithmTest1()</b></td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td><b>AdjacencyMatrix</b></td><td><b>buildMSTPrim()</b></td><td><b>Scenario3</b></td><td><b>Vertex<v> origin</b></td><td><b>A list of previous Nodes (the predecessor of a certain node) is output.
Since we built the scenario, we can assert the expected list.
If the output list matches the expected list the assertion must be true.</b></td></tr>
</table>

<table>
<tr><td colspan="5"><b>Test Objective: Verify the correct functioning of the Prim´s algorithm.</b></td></tr>
<tr><td colspan="5"><b>PrimsAlgorithmTest2()</b></td></tr>
<tr><th>primsAlgorithmTest1()</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td><b>AdjacencyMatrix</b></td><td><b>buildMSTPrim ()</b></td><td><b>Scenario7</b></td><td><b>Vertex<v> origin</b></td><td><b>A list of previous indexes (the predecessor of a node in a certain index) is output.
Since we built the scenario, we can assert the expected list.
If the output list matches the expected list the assertion must be true.</b></td></tr>
</table>

<table>
<tr><td colspan="5"><b>Test Objective: Verify the correct functioning of the Kruskal´s algorithm.</b></td></tr>
<tr><td colspan="5"><b>kruskalTest1()</b></td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td><b>AdjacencyMatrix</b></td><td><b>kruskal()</b></td><td><b>Scenario1</b></td><td><b>Vertex<v> origin</b></td><td><b>A list of previous Nodes (the predecessor of a certain node) is output.
Since we built the scenario, we can assert the expected list.
If the output list matches the expected list the assertion must be true.</b></td></tr>
</table>

<table>
<tr><td colspan="5"><b>Test Objective: Verify the correct functioning of the Kruskal´s algorithm.</b></td></tr>
<tr><td colspan="5"><b>kruskalTest2()</b></td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyMatrix</td><td>kruskal()</td><td>Scenario1</td><td>Vertex<v> origin</td><td>A list of previous Nodes (the predecessor of a certain node) is output.
Since we built the scenario, we can assert the expected list.
If the output list matches the expected list the assertion must be true.</td></tr>
</table>

<table>
<tr><td colspan="5"><b>Test Objective: Verify the correct functioning of the Dijkstra´s algorithm.</b></td></tr>
<tr><td colspan="5"><b>dijkstraTest1()</b></td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyMatrix</td><td>dijkstra()</td><td>Scenario7</td><td>Vertex<v> origin</td><td>A list of previous Nodes indexes (the predecessor of a certain node) is output.
Since we built the scenario, we can assert the expected list.
If the output list matches the expected list the assertion must be true.</td></tr>
</table>

<table>
<tr><td colspan="5"><b>Test Objective: Verify the correct functioning of the Dijkstra´s algorithm.</b></td></tr>
<tr><td colspan="5"><b>dijkstraTest2()</b></td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyMatrix</td><td>dijkstra()</td><td>Scenario1</td><td>Vertex<v> origin</td><td>A list of previous Nodes indexes (the predecessor of a certain node) is output.
Since we built the scenario, we can assert the expected list.
If the output list matches the expected list the assertion must be true.</td></tr>
</table>

Test Objective: Verify the correct functioning of the Floyd Warshall´s algorithm.				
fwTest1()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	floydWarshall()	Scenario4	<none>	A bidimensional matrix with the cheapest path between every pair of nodes is output. Since we built the scenario, we can assert the expected matrix. If the output matrix matches the expected one the assertion must be true.

Test Objective: Verify the correct functioning of the Floyd Warshall´s algorithm.				
fwTest2()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	floydWarshall()	Scenario8	<none>	A bidimensional matrix with the cheapest path between every pair of nodes is output. Since we built the scenario, we can assert the expected matrix. If the output matrix matches the expected one the assertion must be true.

Test Objective: Verify the method used for returning the indexes of the adjacent vertexes from an specific vertex.				
adjacentsTest1()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	adjacents()	Scenario6	Int index	A list with the indexes of adjacent nodes. Since we know the scenario, we can assert the expected indexes from the output list. The output list must match the expected list.

Test Objective: Verify the method used for returning the indexes of the adjacent vertexes from an specific node.				
adjacentsTest2()				
Class under test	**Method**	**Scenario**	**Input**	**Outcome**
AdjacencyMatrix	**adjacents()**	**Scenario2**	**Int index**	**A list with the indexes of adjacent nodes. Since we know the scenario, we can assert the expected indexes from the output list. The output list must match the expected list.**

Test Objective: Verify the correct functioning of the search index method that returns the index of an object. (the graph is empty)				
searchIndexTest1()				
Class under test	**Method**	**Scenario**	**Input**	**Outcome**
AdjacencyMatrix	**searchIndex ()**	**Scenario1**	**Vertex<v> v**	**The index of the first occurrence of the object we give as a parameter should be the output of the method. Since we created the scenarios, we know what the index of the parameter is. Since the graph is empty, it´ll return -1, which is the default result for when the object one give as a parameter is not in the graph.**

Test Objective: Verify the correct functioning of the search index method that returns the index of an object.				
searchIndexTest2()				
Class under test	**Method**	**Scenario**	**Input**	**Outcome**
AdjacencyMatrix	**searchIndex ()**	**Scenario2**	**Vertex<v> v**	**The index of the first occurrence of the object we give as a parameter should be the output of the method. Since we created the scenarios, we know what the index of the parameter is. We test that this numbers are equal and the result of the test must be true.**

<table>
<tr><td colspan="5"><b>Test Objective: Verify the correct functioning of the search index method that returns the index of an object.</b></td></tr>
<tr><td colspan="5"><b>searchIndexTest3()</b></td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyMatrix</td><td>searchIndex ()</td><td>Scenario3</td><td>Vertex<v> v</td><td>The index of the first occurrence of the object we give as a parameter should be the output of the method. Since we created the scenarios, we know what the index of the parameter is.
We test that this numbers are equal and the result of the test must be true.</td></tr>
</table>

<table>
<tr><td colspan="5"><b>Test Objective: Verify the cost of going from a certain node to another one in the graph using the Floyd Warshall algorithm.</b></td></tr>
<tr><td colspan="5"><b>pathCostTest()</b></td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyMatrix</td><td>pathCost()</td><td>Scenario4</td><td>City c1, City c2</td><td>The method outputs a Double type value, since we built the scenario we can assert the expected output.
The output of the method should be the same as the expected.</td></tr>
</table>

<table>
<tr><td colspan="5"><b>Test Objective: Verify the cost of going from a certain node to another one in the graph using the Floyd Warshall algorithm.</b></td></tr>
<tr><td colspan="5"><b>pathCostTest1()</b></td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyMatrix</td><td>pathCost()</td><td>Scenario4</td><td>City c1, City c2</td><td>The method outputs a Double type value, since we built the scenario, we can assert the expected output.
The output of the method should be the same as the expected.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the proper functioning of the path for the BFS algorithm .</td></tr>
<tr><td colspan="5">bfsPathTest()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyMatrix</td><td>bfs()</td><td>Scenario3</td><td>City c1, City c2</td><td>The method outputs a list of the nodes in order of the algorithm route.
The result of the execution of the method should be the same as the expected.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the proper functioning of the path for the BFS algorithm .</td></tr>
<tr><td colspan="5">bfsPathTest1()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyMatrix</td><td>bfs()</td><td>Scenario3</td><td>City c1, City c2</td><td>The method outputs a list of the nodes in order of the algorithm route.
The result of the execution of the method should be the same as the expected.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the proper functioning of the path for the BFS algorithm</td></tr>
<tr><td colspan="5">dijkstraPathTest()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyMatrix</td><td>dijkstraPath()</td><td>Scenario7</td><td>City c1, City c2</td><td>The method outputs a list of predecessor nodes denoting the path the algorithm follow in order to go from c1 to c2 .
Since we built the
The output of the method should be the same as the expected.</td></tr>
</table>

Test Objective: Verify the proper functioning of the path for the BFS algorithm				
dijkstraPathTest1()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyMatrix	dijkstraPath()	Scenario4	City c1, City c2	The method outputs a list of predecessor nodes denoting the path the algorithm follow in order to go from c1 to c2 . Since we built the The output of the method should be the same as the expected.

AdjacencyList Test cases

Test Objective: Verifying the correct creation of an a graph using an adjacency list.				
testAdjacencyList()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	AdjacencyList<V>()	Scenario2	Vertex<V> v	We make the assertion that the adjacency list is different than null and this one must be true.

Test Objective: Verifying the correct creation of a directed graph using an adjacency list.				
testAdjacencyList1()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	AdjacencyList<V>()	Scenario2	Vertex<V> v	We make the assertion that the adjacency list is different than null and this one must be true. We also make the assertion that the graph is directed and this assertion must be true.

Test Objective: Verifying the correct adding of a Vertex to the graph when it is empty (using Adjacency List to represent).

addVertexTest1()

Class under test	Method	Scenario	Input	Outcome
AdjacencyList	addVertex()	Scenario1	Vertex<v> v	The size of the graph is one.

Test Objective: Verifying the correct adding of a Vertex to the graph when it has Vertexes already (using Adjacency List to represent).

addVertexTest2()

Class under test	Method	Scenario	Input	Outcome
AdjacencyList	addVertex()	Scenario2	Vertex<v> v	The size of the graph is the amount off vertexes added in the creation of the scenario plus one.

Test Objective: Verifying the correct adding of a Vertex to the graph when its matrix has already reached its limit (it`s full) (using Adjacency Matrix to represent).

addVertexTest3()

Class under test	Method	Scenario	Input	Outcome
AdjacencyList	addVertex()	Scenario3	Vertex<v> v	The size of the graph is the amount off vertexes added in the creation of the scenario plus one. Besides the Matrix of adjacency must increase its original size in order to fit more nodes when needed.

Test Objective: Verifying the correct removing of a Vertex from the graph when the graph is empty (using list of adjacency to represent).

removeVertexTest1()

Class under test	Method	Scenario	Input	Outcome
AdjacencyList	removeVertex()	Scenario1	Vertex<v> v	The size of the graph is the amount off vertexes added in the creation of the scenario minus one. Since the graph is empty in this scenario, nothing cant be deleted. It must return false

Test Objective: Verifying the correct removing of a Vertex from the graph when the graph is not empty (using list of adjacencies to represent).				
removeVertexTest2()				
Class under test	**Method**	**Scenario**	**Input**	**Outcome**
AdjacencyList	**removeVertex()**	**Scenario2**	**Vertex<v> v**	**The size of the graph is the amount off vertexes added in the creation of the scenario minus one. We make the assertion that the graph does not contains the removed node, if the operation was successful the assertion must be true.**

Test Objective: Verifying the correct removing of a Vertex from the graph when the graph is not empty (using list of adjacency to represent).				
removeVertexTest3()				
Class under test	**Method**	**Scenario**	**Input**	**Outcome**
AdjacencyList	**removeVertex()**	**Scenario3**	**Vertex<v> v**	**The size of the graph is the amount off vertexes added in the creation of the scenario minus one. We make the assertion that the graph does not contains the removed node, if the operation was successful the assertion must be true.**

Test Objective: Verify the correct adding of an edge to the graph (the graph is empty).				
addEdgeTest1()				
Class under test	**Method**	**Scenario**	**Input**	**Outcome**
AdjacencyList	**addEdge()**	**Scenario1**	**Vertex<v> v1 && Vertex<v> v2**	**The addEdge() method must return true. Since the graph is empty and one can´t add an edge when there is no Vertexes this test will return false.**

<table>
<tr><td colspan="5">Test Objective: Verify the correct adding of an edge to the graph (the graph is not empty).</td></tr>
<tr><td colspan="5">addEdgeTest2()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>addEdge()</td><td>Scenario2</td><td>Vertex<v> v1 && Vertex<v> v2</td><td>The addEdge() method must return true.
If one gets the adjacent of v1,must return v2 in order for the test to prove the proper functioning of the method.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the correct adding of an edge to a directed graph (the graph is not empty).</td></tr>
<tr><td colspan="5">addEdgeTest3()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>addEdge()</td><td>Scenario2</td><td>Vertex<v> v1 && Vertex<v> v2</td><td>The addEdge() method must return false.
If one gets the adjacent of v1,must return v2 and the adjacent of v2,must return v1 in order for the test to prove the proper functioning of the method.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the correct adding of an edge to an undirected graph (the graph is not empty).</td></tr>
<tr><td colspan="5">addEdgeTest4()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>addEdge()</td><td>Scenario5</td><td>Vertex<v> v1 && Vertex<v> v2</td><td>The addEdge() method must return true.
If one gets the adjacent of v1 must be v2 in order for the test to prove the proper functioning of the method.</td></tr>
</table>

Test Objective: Verify the correct adding of an weighted edge to the graph (the graph is not empty).

addWeightedEdgeTest1()

Class under test	Method	Scenario	Input	Outcome
AdjacencyList	addEdge()	Scenario5	Vertex<v> v1 && Vertex<v> v2	The assertion that the edge was added must be true if the method output true. If one gets the adjacent of v1 must be v2 in order for the test to prove the proper functioning of the method.

Test Objective: Verify the correct adding of and edge to the graph.

addEdgeTest2()

Class under test	Method	Scenario	Input	Outcome
AdjacencyList	addEdge()	Scenario2	Vertex<v> v1 && Vertex<v> v2	The assertion that the edge was added must be true if the method output true must return true.

Test Objective: Verify the correct adding of and edge to the graph.

addEdgeTest3()

Class under test	Method	Scenario	Input	Outcome
AdjacencyList	addEdge()	Scenario3	Vertex<v> v1 && Vertex<v> v2	The assertion that the edge was added must be true if the method output true must return false. In this test two vertex will be passed but one of them will not exist in the graph, therefore the edge cannot be added.

<table>
<tr><td colspan="5"><b>Test Objective: Verify the correct adding of a weighted edge to the graph (the graph is empty).</b></td></tr>
<tr><td colspan="5"><b>addWeightedEdgeTest1()</b></td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>addEdge()</td><td>Scenario1</td><td>Vertex<v> v1 && Vertex<v> v2</td><td>The addEdge() method must output true.
The weight of the added edge must be equal to the expected weight.
Since the graph is empty and one can´t add an edge when there is no Vertexes this test will return false.</td></tr>
</table>

<table>
<tr><td colspan="5"><b>Test Objective: Verify the correct adding of and edge to the graph.</b></td></tr>
<tr><td colspan="5"><b>addWeightedEdgeTest2()</b></td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>addEdge()</td><td>Scenario2</td><td>Vertex<v> v1 && Vertex<v> v2</td><td>The weight of the added edge must be equal to the expected weight.
Since the graph is empty and one can´t add an edge when there is no Vertexes this test will return false.</td></tr>
</table>

<table>
<tr><td colspan="5"><b>Test Objective: Verify the correct adding of and edge to the graph.</b></td></tr>
<tr><td colspan="5"><b>addWeightedEdgeTest3()</b></td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>addEdge()</td><td>Scenario3</td><td>Vertex<v> v1 && Vertex<v> v2</td><td>The addEdge() method must return false.
In this test two vertex will be passed but one of them will not exist in the graph, therefore the edge cannot be added.</td></tr>
</table>

Test Objective: Verify the method used for returning the indexes of the adjacent vertexes from an specific vertex in a weak connected graph .				
adjacentsTest1()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	adjacents()	Scenario6	Int index	A list with the indexes of adjacent nodes. Since we know the scenario, we can assert the expected indexes from the output list. The output list must match the expected list.

Test Objective: Verify the method used for returning the indexes of the adjacent vertexes from an specific vertex in a strongly connected graph.				
adjacentsTest2()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	adjacents()	Scenario6	Int index	A list with the indexes of adjacent nodes. Since we know the scenario, we can assert the expected indexes from the output list. The output list must match the expected list.

Test Objective: Verify the method used for returning the indexes of the adjacent vertexes from an specific vertex in a strongly connected graph.				
adjacentsTest3()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	adjacents()	Scenario6	Int index	A list with the indexes of adjacent nodes. Since we know the scenario, we can assert the expected indexes from the output list. The output list must match the expected list.

Test Objective: Verify the method used for returning the indexes of the adjacent vertexes from an specific node.				
adjacentsTest2()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	adjacents()	Scenario2	Int index	A list with the indexes of adjacent nodes. Since we know the scenario, we can assert the expected indexes from the output list. The output list must match the expected list.

Test Objective: Verify the correct functioning of the search index method that returns the index of an object. (the graph is empty)				
searchIndexTest1()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	searchIndex ()	Scenario1	Vertex<v> v	The index of the first occurrence of the object we give as a parameter should be the output of the method. Since we created the scenarios, we know what the index of the parameter is. Since the graph is empty, it´ll return -1, which is the default result for when the object one give as a parameter is not in the graph.

Test Objective: Verify the correct functioning of the search index method that returns the index of an object.				
searchIndexTest2()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	searchIndex ()	Scenario2	Vertex<v> v	The index of the first occurrence of the object we give as a parameter should be the output of the method. Since we created the scenarios, we know what the index of the parameter is. We test that this numbers are equal and the result of the test must be true.

Test Objective: Verify the correct functioning of the search index method that returns the index of an object.				
searchIndexTest3()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	searchIndex ()	Scenario3	Vertex<v> v	The index of the first occurrence of the object we give as a parameter should be the output of the method. Since we created the scenarios, we know what the index of the parameter is. We test that this numbers are equal and the result of the test must be true.

Test Objective: Verify the correct functioning of the BFS algorithm.				
bfsTest1()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	bfs()	Scenario4	Vertex<v> origin	A list of previous indexes (the predecessor of a node in a certain index) is output. Since we built the scenario, we can assert the expected list. If the output list matches the expected list the assertion must be true.

Test Objective: Verify the correct functioning of the BFS algorithm.				
bfsTest2()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	bfs()	Scenario6	Vertex<v> origin	A list of previous indexes (the predecessor of a node in a certain index) is output. Since we built the scenario, we can assert the expected list. If the output list matches the expected list the assertion must be true.

<table>
<tr><td colspan="5">Test Objective: Verify the correct functioning of the BFS algorithm.</td></tr>
<tr><td colspan="5">bfsTest3()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>bfs()</td><td>Scenario7</td><td>Vertex<v> origin</td><td>A list of previous indexes (the predecessor of a node in a certain index) is output.
Since we built the scenario we can assert the expected list.
If the output list matches the expected list the assertion must be true.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the correct functioning of the DFS algorithm.</td></tr>
<tr><td colspan="5">dfsTest1()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>dfs()</td><td>Scenario4</td><td>int origin</td><td>A list of previous indexes (the predecessor of a node in a certain index) is output.
Since we built the scenario, we can assert the expected list.
If the output list matches the expected list the assertion must be true.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the correct functioning of the DFS algorithm.</td></tr>
<tr><td colspan="5">dfsTest2()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>dfs()</td><td>Scenario3</td><td>int origin</td><td>A list of previous indexes (the predecessor of a node in a certain index) is output.
Since we built the scenario, we can assert the expected list.
If the output list matches the expected list the assertion must be true.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the correct functioning of the dfs algorithm.</td></tr>
<tr><td colspan="5">dfsTest3()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>dfs()</td><td>Scenario3</td><td>int origin</td><td>A list of previous indexes (the predecessor of a node in a certain index) is output.
Since we built the scenario, we can assert the expected list.
If the output list matches the expected list the assertion must be true.</td></tr>
</table>

Test Objective: Verify the correct functioning of the Prim´s algorithm.				
primsAlgorithmTest1()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	buildMSTPrim()	Scenario7	City origin	A list of previous Nodes (the predecessor of a certain node) is output. Since we built the scenario, we can assert the expected list. If the output list matches the expected list the assertion must be true.

Test Objective: Verify the correct functioning of the Prim´s algorithm.				
PrimsAlgorithmTest2()				
primsAlgorithmTest1()	Method	Scenario	Input	Outcome
AdjacencyList	buildMSTPrim ()	Scenario4	Vertex<v> origin	A list of previous indexes (the predecessor of a node in a certain index) is output. Since we built the scenario, we can assert the expected list. If the output list matches the expected list the assertion must be true.

Test Objective: Verify the correct functioning of the Kruskal´s algorithm.				
kruskalTest1()				
Class under test	Method	Scenario	Input	Outcome
AdjacencyList	buildMSTKruskal ()	Scenario7	Vertex<v> origin	A list of previous Nodes (the predecessor of a certain node) is output. Since we built the scenario, we can assert the expected list. If the output list matches the expected list the assertion must be true.

<table>
<tr><td colspan="5">Test Objective: Verify the correct functioning of the Kruskal´s algorithm.</td></tr>
<tr><td colspan="5">kruskalTest2()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>buildMSTKruskal ()</td><td>Scenario1</td><td>Vertex<v> origin</td><td>A list of previous Nodes (the predecessor of a certain node) is output.
Since we built the scenario, we can assert the expected list.
If the output list matches the expected list the assertion must be true.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the correct functioning of the Dijkstra´s algorithm.</td></tr>
<tr><td colspan="5">dijkstraTest1()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>dijkstra()</td><td>Scenario7</td><td>Vertex<v> origin</td><td>A list of previous Nodes indexes (the predecessor of a certain node) is output.
Since we built the scenario, we can assert the expected list.
If the output list matches the expected list the assertion must be true.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the correct functioning of the Dijkstra´s algorithm.</td></tr>
<tr><td colspan="5">dijkstraTest2()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>dijkstra()</td><td>Scenario7</td><td>Vertex<v> origin</td><td>A list of previous Nodes indexes (the predecessor of a certain node) is output.
Since we built the scenario, we can assert the expected list.
If the output list matches the expected list the assertion must be true.</td></tr>
</table>

Test Objective: Verify the correct functioning of the Floyd Warshall´s algorithm.

fwTest1()

Class under test	Method	Scenario	Input	Outcome
AdjacencyList	floydWarshall()	Scenario4	<none>	A bidimensional matrix with the cheapest path between every pair of nodes is output. Since we built the scenario, we can assert the expected matrix. If the output matrix matches the expected one the assertion must be true.

Test Objective: Verify the correct functioning of the Floyd Warshall´s algorithm.

fwTest2()

Class under test	Method	Scenario	Input	Outcome
AdjacencyList	floydWarshall()	Scenario8	<none>	A bidimensional matrix with the cheapest path between every pair of nodes is output. Since we built the scenario, we can assert the expected matrix. If the output matrix matches the expected one the assertion must be true.

Test Objective: Verify the cost of going from a certain node to another one in the graph using the Floyd Warshall algorithm.

pathCostTest()

Class under test	Method	Scenario	Input	Outcome
AdjacencyList	pathCost()	Scenario4	City c1, City c2	The method outputs a Double type value, since we built the scenario we can assert the expected output. The output of the method should be the same as the expected.

<table>
<tr><td colspan="5">Test Objective: Verify the cost of going from a certain node to another one in the graph using the Floyd Warshall algorithm.</td></tr>
<tr><td colspan="5">pathCostTest1()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>pathCost()</td><td>Scenario4</td><td>City c1, City c2</td><td>The method outputs a Double type value, since we built the scenario, we can assert the expected output.
The output of the method should be the same as the expected.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the proper functioning of the path for the BFS algorithm.</td></tr>
<tr><td colspan="5">bfsPathTest()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>bfs()</td><td>Scenario3</td><td>City c1, City c2</td><td>The method outputs a list of the nodes in order of the algorithm route.
The result of the execution of the method should be the same as the expected.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the proper functioning of the path for the BFS algorithm.</td></tr>
<tr><td colspan="5">bfsPathTest1()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>bfs()</td><td>Scenario3</td><td>City c1, City c2</td><td>The method outputs a list of the nodes in order of the algorithm route.
The result of the execution of the method should be the same as the expected.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the proper functioning of the path for the BFS algorithm</td></tr>
<tr><td colspan="5">dijkstraPathTest()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>dijkstraPath()</td><td>Scenario7</td><td>City c1, City c2</td><td>The method outputs a list of predecessor nodes denoting the path the algorithm follow in order to go from c1 to c2 .
Since we built the
The output of the method should be the same as the expected.</td></tr>
</table>

<table>
<tr><td colspan="5">Test Objective: Verify the proper functioning of the path for the BFS algorithm</td></tr>
<tr><td colspan="5">dijkstraPathTest1()</td></tr>
<tr><th>Class under test</th><th>Method</th><th>Scenario</th><th>Input</th><th>Outcome</th></tr>
<tr><td>AdjacencyList</td><td>dijkstraPath()</td><td>Scenario4</td><td>City c1, City c2</td><td>The method outputs a list of predecessor nodes denoting the path the algorithm follow in order to go from c1 to c2 .
Since we built the
The output of the method should be the same as the expected.</td></tr>
</table>