**TEST CASES HASHTABLE**

| **Name** | **Class** | **Scenery** |
| --- | --- | --- |
| setUp0 | TestListGraph | empty |
| setUp1 | TestListGraph | new AdjacencyListGraph g |
| setUp2 | TestListGraph | new AdjacencyListGraph with vertex 1 |
| setUp3 | TestListGraph | new AdjacencyListGraph with vertices 1, 2, 3 |
| setUp1 | TestMatrixGraph | Has a the initialization of the graph |
| setUp2 | TestMatrixGraph | Has the initialization of the matrix graph and also add 5 different vertex |
| setUp3 | TestMatrixGraph | has the initialization of the matrix graph, add 4 new different vertex to the graph |
| setUp4 | TestMatrixGraph | has the initialization of the matrix graph, add 4 new different vertices and finally add 3 edges between the vertices. These edges are directional |

**TEST CASES FOR LIST GRAPH**

| **Objective of the test: Test if an element can be added to the graph** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| ListGraph | add | setUp1 | An integer with value 1 | The element is properly added into the graph |

| **Objective of the test: Test if an element that is null can be added to the graph** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| ListGraph | add | setUp1 | null | The element can not be added to the graph as it is null |

| **Objective of the test: Test if an element can be found in the graph** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| ListGraph | searchVertex | setUp3 | An integer with value 2 | The element is correctly found in the graph |

| **Objective of the test: Test if the value of element can be found in the graph** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| ListGraph | searchValue | setUp3 | An integer with value 1 | The value of the element is correctly found in the graph |

| **Objective of the test: Test if an element can be found in the graph** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| ListGraph | searchVertex | setUp3 | An integer with value 1 | The element can not be found in the graph as it was not added to it |

| **Objective of the test: Test if an edge can be added to the graph** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| ListGraph | addEdge | setUp3 | A graph with three integers 1, 2, 3 | The edges are properly added to the vertices |

| **Objective of the test: Test if a vertex can be deleted from the graph** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| ListGraph | deleteVertex | setUp3 | A graph with three integers 1, 2, 3 | The edges are properly deleted from the vertices |

| **Objective of the test: Test if a null vertex can be deleted from the graph** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| ListGraph | deleteVertex | setUp1 | null | The edges are not deleted as they are not elements in the graph |

| **Objective of the test: Test if a vertex can be deleted from the graph** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| ListGraph | addEdge | setUp1 | null | The edges are not deleted as there aren’t enough elements in the graph to create an edge |

**TEST CASES MATRIX**

| **Objective of the test: Test if the method add, add correctly a vertex or not in different scenery** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MatrixGraph | addVertex | setup1 | The initialization of the graph | The method returns true indicating that the vertex was correctly added |
| MatrixGraph | addVertex | - | Non initialization of the graph | The method should return a null pointer exception indicating that is correct |
| MatrixGRaph |  | setUp1 | initialized graph | Assert false because the value can never be null |

| **Objective of the test: Test if the method searches correctly a vertex or not in different scenery** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MatrixGraph | SearchVertex | setup3 | The initialization of the graph  mGraph.addVertex(1);  mGraph.addVertex(2);  mGraph.addVertex(3); | The method returns the value of the node that we were searching |
| MatrixGraph | SearchVertex | setUp3 | setUp1 and  mGraph.addVertex(1);  mGraph.addVertex(2);  mGraph.addVertex(3); | The method should return null because the vertex does not exists |
| MatrixGRaph | SearchVertex | setUp1 | initialized graph | The method asser null because the graph is empty |

| **Objective of the test: Test if the method ad edges correctly** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MatrixGraph | addEdge | setup3 | The initialization of the graph  mGraph.addVertex(1);  mGraph.addVertex(2);  mGraph.addVertex(3); | The method returns true for each case if the edges are correct |
| MatrixGraph | addEdge | setUp3 | setUp1 | The method should return false because there are non index for the edge |
| MatrixGRaph | addEdge | setUp1 | The initialization of the graph | The method the method should assert false because the graph is empty |

| **Objective of the test: Test if the method deletes correctly** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MatrixGraph | deleteVertex | setup3 | The initialization of the graph  mGraph.addVertex(1);  mGraph.addVertex(2);  mGraph.addVertex(3); | The method returns true for the first assert, and returns null when we search the deleted value again |
| MatrixGraph | deleteVertex | setUp1 | The initialization of the graph | The method returns false because is deleting a vertex that does not exists in the graph |
| MatrixGRaph | deleteVertex | setUp1 | The initialization of the graph | The method returns false because is deleting a vertex that does not exists in the graph,in this case is null |

| **Objective of the test: Test if the method recognize the neighbors of each node** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MatrixGraph | getNeighbors | setup4 | The initialization of the graph  mGraph.addVertex(1);  mGraph.addVertex(2);  mGraph.addVertex(3); | The method returns the exact number of the amount of neighbors that the vertex has for each case. |

| **Objective of the test: Test if the dijkstra shows correctly the shortest way between two nodes** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MatrixGraph | dijkstra | setUp1() | The initialization of the graph | The method add edges between nodes, creates the drain and the source in a given index and assert if the shortest path has as it first element the water source and if tha size of the list is 1 |
| MatrixGraph | dijkstra | setUp1 | The initialization of the graph | The method should assert when the path is not empty given that add the different nodes to the path |
| MatrixGraph | dijkstra | setUp1 | The initialization of the graph | The method should assert when the path size is zero because th drain and the source has the same index |