**Test design**

**Configuration of the Scenarios**

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| setUpStage1 | Graph | A Graph with:  5 vertexes:   1. {value: “Server1”} 2. {value: “Server 2”} 3. {value: “Server3”} 4. {value: “Server4”} 5. {value: “server5”}   5 edges:   1. Vertex1—2—Vertex2 2. Vertex1—4—Vertex3 3. Vertex2—3—Vertex4 4. Vertex2—6—Vertex3 5. Vertex3—2—Vertex5 |
| setUpStage2 | Graph | A Graph with:  5 vertexes:  1. {value: “Server1”}  2. {value: “Server2”}  3. {value: “Server3”,}  4. {value: “Server4”}  5. {value: “Server5”}  3 edges:  1. Vertex1—3—Vertex2  2. Vertex1—7—Vertex3  3. Vertex4—6—Vertex5 |
| setUpStage3 | Graph | A Graph with:  5 vertexes:  1. {value: “Server1”}  2. {value: “Server2”}  3. {value: “Server3”}  4. {value: “Server4”}  5. {value: “Server5”}  10 edges:  1. Vertex1—3—Vertex2  2. Vertex1—7—Vertex3  3. Vertex1—4—Vertex4  4. Vertex1—3—Vertex5  5. Vertex2—9—Vertex3  6. Vertex2—4—Vertex4  7. Vertex2—1—Vertex5  8. Vertex3—5—Vertex4  9. Vertex3—3—Vertex5  10. Vertex4—2—Vertex5 |
| setUpStage4 | Graph | A Graph with:  5 vertexes:  1. {value: “Server1”}  2. {value: “Server2”}  3. {value: “Server3”}  4. {value: “Server4”}  5. {value: “Server5”}  0 edges |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: the purpose of this test is to test everything related to the BFS method.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Graph** | **BFS** | setUpStage4 | all vertexes are kept in white color |
| **Graph** | **BFS** | setUpStage2 | a distance of +1 over the number of Vertex is expected and a partial route where only the first part that is connected is done, which in this case is 1-2 and 1-3. |
| **Graph** | **BFS** | setUpStage1 | normal enumeration is expected |
| **Graph** | **BFS** | setUpStage3 | a distance of +2 over the number of the Vertex is expected. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: the purpose of this test is to test everything related to the DFS method.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Graph** | **DFS** | setUpStage4 | all vertex end distances should be a +2 of each vertex number |
| **Graph** | **DFS** | setUpStage2 | all vertex end distances should be a +2 of each vertex over the distance |
| **Graph** | **DFS** | setUpStage1 | normal enumeration is expected |
| **Graph** | **DFS** | setUpStage3 | normal enumeration is expected |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: the purpose of this test is to test everything related to the Deleteedge method.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Graph** | **Deleteedge** | setUpStage3 | the result expected is true, and the connection deleted. |
| **Graph** | **Deleteedge** | setUpStage4 | the result expected is false because in the 4 graph we don’t have edges. |
| **Graph** | **Deleteedge** | setUpStage4 | the result expected is false because you can’t delete a connection which is null. |
| **Graph** | **Deleteedge** | setUpStage4 | the result expected is false because you can’t delete a connection between vertex null with connection to a vertex null. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: the purpose of this test is to test everything related to the DeleteVertex method.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Graph** | **DeleteVertex** | setUpStage4 | the result expected is false because you can’t delete a vertex null |
| **Graph** | **DeleteVertex** | setUpStage4 | the result expected is false because you can’t delete a vertex what doesn’t exist. |
| **Graph** | **DeleteVertex** | setUpStage2 | the result expected is true and a decrement in the list of vertexes to 4. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: the purpose of this test is to test everything related to the addVertex method.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Graph** | **addVertex** |  | the expected result is that initially the size of the list of the list of Vertexes increases from 0 to 1 and that when adding the same vertex again it remains 1 |
| **Graph** | **addVertex** |  | the expected result is that initially the size of the list of the list of Vertexes increases from 0 to 3 after adding 3 vertexes |
| **Graph** | **addVertex** |  | the expected result is that initially the size of the list of the list of Vertexes increases from 0 to 0 because you can’t add a null vertex |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: the purpose of this test is to test everything related to the addEdge method.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Graph** | **addEdge** |  | it is expected that when adding a null edge to a vertex the adjacency list or its adjacency matrix will not change. |
| **Graph** | **addEdge** |  | it is expected that when adding the same edge to a vertex the adjacency list or its adjacency matrix will not change. |
| **Graph** | **addEdge** |  | it is expected that when adding edge to a vertex the adjacency list or its adjacency matrix will change to adjust to that change. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: the purpose of this test is to test everything related to the dijkstra method.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Graph** | **dijkstra** | setUpStage4 | the method should return either null because it is impossible to reach the target from any Vertex other than itself. |
| **Graph** | **dijkstra** | SetupStage2 | the method should return null because even if a certain part is reached, the Vertex objective cannot be reached. |
| **Graph** | **dijkstra** | SetUpStage3 | The method should return the next list:  {“Server1”,”Server2”,”Server5”} |
| **Graph** | **dijkstra** | SetUpStage3 | The method should return the next list:  {“Server4”,”Server5”,”Server2”} |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: the purpose of this test is to test everything related to the floydWarshall method.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Graph** | **floydWarshall.** | setUpStage4 | The distance between every node is infinity |
| **Graph** | **floydWarshall.** | SetupStage2 | The distance between 3 and 4 is infinity |
| **Graph** | **floydWarshall.** | SetUpStage3 | A matrix with all the respective distance between vertex |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: the purpose of this test is to test everything related to the Prim method.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Graph** | **Prim.** | setUpStage4 | should return a graph with only one vertex |
| **Graph** | **Prim.** | SetupStage2 | should return a graph with 3 Vertex and two edges |
| **Graph** | **Prim.** | SetUpStage3 | Should return a graph with the 5 vertex and 4 edges |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: the purpose of this test is to test everything related to the Kruskal method.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Graph** | **Kruskal.** | setUpStage4 | Should return a graph with 4 vertexes but without edges |
| **Graph** | **Kruskal.** | SetupStage2 | Should return a graph with two parts:   1. 3 Vertex and 2 edges 2. 2 Vertex and 1 edge |
| **Graph** | **Kruskal.** | SetUpStage3 | Should return a graph with the 5 vertex and 4 edges |