**Scenarios**

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| setupStage1 | GraphTest | An empty graph |
| setupStage2 | GraphTest | A graph with the following vertexes:  vertex1("A")  vertex2("B")  vertex4("D")  vertex5("E")  vertex6("F")  vertex7("G")  vertex8("H")  vertexP(“C”)  This vertexP must be a global variable in the test |
| setupStage3 | GraphTest | A graph with the following vertexes:  vertex1("A")  vertex2("B")  vertex4("D")  vertex5("E")  vertex6("F")  vertex7("G")  vertex8("H")  vertexP(“C”)  This vertexP must be a global variable in the test  The vertexes must have the connections:  vertex1, vertex2, weight: 3  vertex1, vertexP, weight: 5  vertex2, vertex4, weight: 1  vertex2, vertex5, weight: 8  vertexP, vertex6, weight: 12  vertexP, vertex7, weight: 5 |
| setupStage4 | GraphTest | A graph with the following vertexes  vertexA(“A”)  this vertexA must be a global variable in the test  vertexB(“B”)  vertexC(“C”)  vertexD(“D”)  vertexZ(“Z”)  The vertexs must have the connections:  vertexA, vertexB, 4  vertexA, vertexC, 2  vertexB, vertexD, 5  vertexB, vertexC, 1  vertexC, vertexD, 8  vertexC, vertexE, 10  vertexD, vertexE, 2  vertexD, vertexZ, 6  vertexE, vertexZ, 3 |

**Tests**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective: Verify that the insert vertex method of the graph works correctly and doesn’t cause any execution errors or exceptions** | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Awaited response** |
| Graph | insertVertex | setupStage1 | new vertex value=”A” | The vertex is added succesfully and the method returns the value of the added vertex. |
| Graph | insertVertex | setupStage1 | vertex value =”A” two times | The vertex is added succesfully the first time but it’s not added the second time, the method should return null. |
| Graph | insertVertex | setupStage2 | new vertex value=”Z” | The vertex is added succesfully and the method returns the value of the added vertex. |
| **Test objective: Verify that the insert edge method of the graph works correctly and doesn’t cause any execution errors or exceptions** | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Awaited response** |
| Graph | insertEdge | setupStage1 | new vertex value=”A”  new vertex value=”B” | The Edge is not added because those vertexes are not in the graph. |
| Graph | insertEdge | setupStage1 | Vertex1 = new vertex(“A”)  Vertex2 = new vertex(“B”)  insertVertex(vertex1)  insertVertex(vertex2)  insertEdge(vertex1, vertex2, 0) | The Edge is added succesfully |
| Graph | insertEdge | setupStage2 | Vertex1 = new vertex(“A”)  Vertex2 = new vertex(“B”)  insertVertex(vertex1)  insertVertex(vertex2)  insertEdge(vertex1, vertex2, 0)  insertEdge(vertex1, vertex2, 0) | The Edge is added succesfully the first time, but it’s not added the second time because it arleady exists. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective: Verify that the delete vertex method of the graph works correctly and doesn’t cause any execution errors or exceptions** | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Awaited response** |
| Graph | deleteVertex | setupStage1 | Vertex1 = Vertex(“A”)  InsertVertex(vertex1)  deleteVertex(vertex1) | The vertex is added and deleted succesfully |
| Graph | deleteVertex | setupStage1 | Vertex1 = Vertex(“A”)  Vertex2 = Vertex(“B”)  InsertVertex(vertex1)  InsertVertex(vertex2)  deleteVertex(vertex1)  deleteVertex(vertex2) | The two vertexes are added and deleted succesfully |
| Graph | deleteVertex | setupStage1 | Vertex1 = Vertex(“A”)  Vertex2 = Vertex(“B”)  InsertVertex(vertex1)  deleteVertex(vertex2) | The vertex is not deleted because it doesn’t exist in the graph |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective: Verify that the delete edge method of the graph works correctly and doesn’t cause any execution errors or exceptions** | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Awaited response** |
| Graph | deleteEdge | setupStage1 | Vertex1 = Vertex(“A”)  Vertex2 = Vertex(“B”)  deleteEdge(vertex1, vertex2) | The Edge is not deleted because it doesn’t exist |
| Graph | deleteEdge | setupStage1 | Vertex1 = Vertex(“A”)  Vertex2 = Vertex(“B”)  InsertVertex(vertex1)  InsertVertex(vertex2)  insertEdge(vertex1, vertex2)  deleteEdge(vertex1, vertex2) | The Edge is deleted. |
| Graph | deleteEdge | setupStage1 | Vertex1 = Vertex(“A”)  Vertex2 = Vertex(“B”)  InsertVertex(vertex1)  deleteEdge(vertex1, vertex2) | The Edge is not deleted because it doesn’t exist. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective: Verify that the BFS method of the graph works correctly, returns a binary Tree and this binary tree’s preOrder method returns the expected String and doesn’t cause any execution errors or exceptions** | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Awaited response** |
| Graph | BFS | setupStage2 | BFS(vertexP) | The BFS method returns the String: “CABDEFG” |
| Graph | BFS | setupStage3 | BFS(vertexP) | The BFS method returns a different String tan: “CABDEFGH” because the H is not connected to vertexP in any way |
| Graph | BFS | setupStage2 | BFS(vertexP) | The BFS methor returns the Sring: “C” because the vertexes are not connected between eachOther |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective: Verify that the DFS method of the graph works correctly, returns a list of binary Trees and each of these binary trees’s preOrder method returns the expected String and doesn’t cause any execution errors or exceptions** | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Awaited response** |
| Graph | DFS | setupStage2 | DFS() | The first tree in the list returns the String: “ABDECFG”  The second tree in the list returns the String: “H” |
| Graph | DFS | setupStage3 | DFS() | The first tree in the list returns a different String than: “ABDECFGH” because H is in a different tree |
| Graph | DFS | setupStage2 | DFS() | The method returns 8 trees, the preorder of these trees should be:  firstTree = “A”  firstTree = “B”  firstTree = “C”  firstTree = “D”  firstTree = “E”  firstTree = “F”  firstTree = “G”  firstTree = “H” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective: Verify that the Dijkstra method of the graph works correctly, returns the list of previous and the list of distances and doesn’t cause any execution errors or exceptions** | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Awaited response** |
| Graph | dijkstra | setupStage3 | dijkstra(vertexP) | The method returns the corresponding list of previous and the corresponding list of distances.  Expected previous list:  "C A nill B B C C nill"  Expected distances list:  "5.0 8.0 0.0 9.0 16.0 12.0 5.0 inf" |
| Graph | dijkstra | setupStage2 | dijkstra(vertexP) | The method returns the corresponding list of previous and the corresponding list of distances.  Expected previous list:  "nill nill nill nill nill nill nill nill"  Expected distances list:  "inf inf 0.0 inf inf inf inf inf" |
| Graph | dijkstra | setupStage4 | dijkstra(vertexA) | The method returns the corresponding list of previous and the corresponding list of distances.  Expected previous list:  "nill C A B D E"  Expected distances list:  "0.0 3.0 2.0 8.0 10.0 13.0" |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective: Verify that the Floyd Warshall method of the graph works correctly, returns the list of distances and doesn’t cause any execution errors or exceptions** | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Awaited response** |
| Graph | floydWarshall | setupStage3 | dijkstra(vertexP) | The method returns the corresponding list of distances.  Expected distances list:  {0, 3, 5, 4, 11, 17, 10, inf},  {3, 0, 8, 1, 8, 20, 13, inf},  {5, 8, 0, 9, 16, 12, 5, inf},  {4, 1, 9, 0, 9, 21, 14, inf},  {11, 8, 16, 9, 0, 28, 21, inf},  {17, 20, 12, 21, 28, 0, 17, inf},  {10, 13, 5, 14, 21, 17, 0, inf},  {inf, inf, inf, inf, inf, inf, inf, 0} |
| Graph | floydWarshall | setupStage2 | dijkstra(vertexP) | The method returns the corresponding list of distances.  Expected distances list:  {0, inf, inf, inf, inf, inf, inf, inf},  {inf, 0, inf, inf, inf, inf, inf, inf},  {inf, inf, 0, inf, inf, inf, inf, inf},  {inf, inf, inf, 0, inf, inf, inf, inf},  {inf, inf, inf, inf, 0, inf, inf, inf}  {inf, inf, inf, inf, inf, 0, inf, inf}  {inf, inf, inf, inf, inf, inf, 0, inf}  {inf, inf, inf, inf, inf, inf, inf, 0} |
| Graph | floydWarshall | setupStage4 | dijkstra(vertexA) | The method returns the corresponding list of distances.  Expected distances list:  {0, 3, 2, 8, 10, 13},  {3, 0, 1, 5, 7, 10},  {2, 1, 0, 6, 8, 11},  {8, 5, 6, 0, 2, 5},  {10, 7, 8, 2, 0, 3},  {13, 10, 11, 5, 3, 0} |