**Format of scenarios and use cases - Integrative Task II**

**Scenario Configuration**

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| testInsertVertex | AdjacentMatrixGraphtest | Verify that vertex insertion in the graph works correctly by adding cities, in this case representing vertices |
| testAddRepeatedVertex | AdjacentMatrixGraphtest | It checks if the graph correctly handles the insertion of repeated vertices, meaning the 4 added cities, and throws the expected exception when trying to add a vertex that already exists in the graph. |
| testAddingMultipleVertex | AdjacentMatrixGraphtest | Check if the graph can correctly handle the addition of multiple vertices, in this case 5 cities and if the size of the graph after adding the vertices is as expected. |
| testDeleteVertexInPseudoGraph | AdjacentMatrixGraphtest | Verifies whether the graph correctly handles vertex removal, including detection of missing vertices and correct updating of the vertex list and adjacency matrix after removal. |
| testDeleteCertexInSimpleGraph | AdjacentMatrixGraphtest | Checks whether the simple graph correctly handles vertex removal, including updating the vertex list and adjacency matrix after deletion. |

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| testAddEdgeInSimpleGraph | AdjacentMatrixGraphtest | Verifies whether the graph correctly handles adding edges and throws expected exceptions in case of duplicate edges, loops, or vertices not found. |
| testAddEdgeInPseudoGraph | AdjacentMatrixGraphtest | Checks if the pseudo graph handles the addition of edges correctly and if the weights of the edges are recorded correctly in the adjacency matrix. |
| testAddEdgeinDirectedGraph | AdjacentMatrixGraphtest | Verifies whether the directed graph correctly handles edge addition and whether edge weights are recorded correctly in the adjacency matrix |
| testDeleteEdgeInSimpleGraph | AdjacentMatrixGraphtest | It checks if the simple graph correctly handles edge removal, and if the adjacency matrix is updated correctly after removal. |
| testDeleteEdgeException | AdjacentMatrixGraphtest | Checks whether appropriate exceptions are thrown when attempting to delete nonexistent edges or when edge vertices do not exist in the graph. |
| testDeleteEdgeInPseudoGraph | AdjacentMatrixGraphtest | Verifies whether edges are removed correctly in a pseudo graph and whether the other edges and vertices remain unchanged after deletion |

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| testBFSColor | AdjacentMatrixGraphtest | Tests whether the BFS algorithm correctly assigns colors to vertices in a graph after performing a width traverse from a given vertex |
| testBFSParents | AdjacentMatrixGraphtest | It tests whether the BFS algorithm correctly assigns parents tovertices in a graph after performing a width traverse from a given vertex. |
| testBFSDistance | AdjacentMatrixGraphtest | Test whether the BFS algorithm correctly assigns distances from a given vertex to all other vertices in the graph |

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| testDFSTime | AdjacentMatrixGraphtest | Test whether the DFS algorithm correctly assigns completion times to each vertex in the graph. The completion times represent the order in which the vertices are visited during the in-depth tour. |
| testDFSDistance | AdjacentMatrixGraphtest | It tests whether the DFS algorithm correctly assigns distances to each vertex in the graph. Distances represent the number of edges that are traversed from the source vertex to each vertex during the depth traverse. |
| testDFSParents | AdjacentMatrixGraphtest | Test whether the DFS algorithm correctly assigns the parents of each vertex in the graph. The parents represent the vertices from which each vertex was reached during the deep traversal. |

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| testFloydWarshall | AdjacentMatrixGraphtest | Test whether the Floyd-Warshall algorithm correctly calculates the shortest distances and path matrices in the graph. |
| testPrim | AdjacentMatrixGraphtest | Test whether Prim's algorithm correctly calculates the minimum spanning tree in the graph. The MST obtained must contain the edges that connect all the vertices of the graph so that the sum of the weights of the edges is minimal. |
| testKruskto | AdjacentMatrixGraphtest | It is verified that the edges in the MST connect the correct vertices and that the weights of the edges match those expected. |

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| testInsertVertex | AdjacentListGraphTest | 6 cities are added as vertices to check the insertion. |
| testAddRepeatedVertex | AdjacentListGraphTest | 4 cities are added as vertices, the test throws an error when a vertex is repeated. |
| testAddingMultipleVertex | AdjacentListGraphTest | 5 cities are added as vertices and the size is checked to see that they have been added correctly. |

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| testAddEdgeInSimpleGraph | AdjacentListGraphTest | Two vertices are added and a connection is created between them, then it is checked that it is not multiple and other cases. |
| testAddEdgeInDirectedGraph | AdjacentListGraphTest | Add 6 cities and add the edges between them in a directed graph. |
| testAddEdgeInPseudoGraph | AdjacentListGraphTest | Two cities are added and the connection is made through the edges so that a multigraph can be created. |

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| testDeleteVertexInDirectedGraph | AdjacentListGraphTest | A vertex of a directed graph is removed. You add 5 cities, create their connection with edges, and then delete two vertices. |
| testDeleteVertexInPseudoGraph | AdjacentListGraphTest | A vertex of a multidirected graph is removed. 5 cities are added, create their connection with edges and then two vertices are deleted. |
| testDeleteVertexInSimpleGraph | AdjacentListGraphTest | A vertex of a simple graph is removed. You add 3 cities, create their connection with edges, and then delete a vertex. |

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| testDeleteEdgeInSimpleGraph | AdjacentListGraphTest | An edge of a simple graph is removed. You add 4 cities, create their connection with edges, and then delete two connections. |
| testDeleteEdgeExceptions | AdjacentListGraphTest | 4 cities are added as vertices, connections are created with edges, and it is expected to throw 3 exceptions of its own. |
| testDeleteEdgeInPseudoGraph | AdjacentListGraphTest | An edge of a multi-directed graph is deleted. 5 cities are added, their connection with edges and then two connections are deleted. |

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| testBFSColor | AdjacentListGraphTest | 4 cities are added, connections are created with edges, the route is made and it is checked that its color has changed black after this. |
| testBFSParents | AdjacentListGraphTest | 4 cities are added, their connections are made with edges, the tour is made and the parents of each of the vertices after this are checked. |
| testBFSDistance | AdjacentListGraphTest | 6 cities are added, their connection with edges, a tour is made and its distance between each vertex after this is checked. |

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| testDFSTime | AdjacentListGraphTest | 6 cities are added, their connections are created with edges, a tour is made and their times are checked. |
| testDFSDistance | AdjacentListGraphTest | 5 cities are added, their connections are created with edges, their travel is made and their distances are checked. |
| testDFSParents | AdjacentListGraphTest | 7 cities are added, their connections are created with edges, a tour is made and their respective parents of each vertex are checked. |

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| testFloydWarshall | AdjacentListGraphTest | Check if the Floyd-Warshall algorithm correctly calculates the distances and shortest paths between the vertices of the graph |
| testPrim | AdjacentListGraphTest | It checks if Prim's algorithm correctly finds the minimum spanning tree in the graph and returns the expected vertices and weights. |
| testKruskal | AdjacentListGraphTest | Checks whether Kruskal's algorithm correctly finds the minimum spanning tree in the graph and returns the expected edges and weights. An exception is alsochecked if an exception is thrown when trying to access an invalid index in the peer list. |

**Test Case Design**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objective of the Test:** Test the functionalities related to the vertices. | | | | |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| AdjacentMatrixtGraphTest | addVertex | testInsertVertex | City-type objects | The insertion of vertices in the graph works correctly |
| AdjacentMatrixGraphTest | addVertex | testAddRepeatedVertex | City-type objects | The graph correctly handles the insertion of repeating vertices, and throws the expected exception when attempting to add a vertex that already exists in the graph. |
| AdjacentMatrixGraphTest | addVertex | testAddingMultipleVertex | City-type objects | The graph cancorrectly handle the addition of multiple vertices, and whether the size of the graph after adding the vertices is as expected. |
| AdjacentMatrixGraphtest | deleteVertex | testDeleteVertexInPseudoGraph | Objects to delete as vertices | The pseudograph correctly handles vertex removal, including detection of missing vertices and correct updating of the vertex list and adjacency matrix after deletion. |
| AdjacentMatrixGraphtest | deleteVertex | testDeleteCertexInSimpleGraph | Objects to delete as vertices | The simple graph correctly handles vertex removal, including updating the vertex list and adjacency matrix after deletion. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objective of the test:** Test the functionalities related to the edges. | | | | |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| AdjacentMatrixGraphTest | addVertex  addEdge | testAddEdgeInSimpleGraph | Objects to add as vertices | The graph correctly handles the addition of edges |
| AdjacentMatrixGraphTest | addVertex  addEdge | testAddEdgeInDirectedGraph | Objects to add as vertices | The pseudo graph correctly handles the addition of edges and whether the weights of the edges are correctly recorded in the adjacency matrix. |
| AdjacentMatrixGraphTest | addVertex  addEdge | testAddEdgand InPseudoGraph | Objects to add as vertices | The directed graph correctly handles the addition of edges and whether the edges weights are recorded correctly in the adjacency matrix |
| AdjacentMatrixGraphtest | deleteEdge | testDeleteEdgeInSimpleGraph | The two vertices that have the edge connection | The simple graph correctly handles edge removal, and whether the adjacency matrix is updated correctly after removal. |
| AdjacentMatrixGraphtest | deleteEdge | testDeleteEdgeException | The two vertices that have the edge connection | Appropriate exceptions are thrown when attempting to remove nonexistent edges or when edge vertices do not exist in the graph. |
| AdjacentMatrixGraphtest | deleteEdge | testDeleteEdgeInPseudoGraph | The two vertices that have the edge connection | Edges are correctly removed in a pseudo-graphic graph and if the other edges and vertices remain unchanged after deletion |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objective of the test:** Test the functionalities related to BFS | | | | |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| AdjacentListGraphTest | BFS | testBFSColor | Starting vertices | The BFS algorithm correctly assigns colors to vertices in a graph after performing a width traverse from a given vertex |
| AdjacentListGraphTest | BFS | testBFSParents | Starting vertices | The BFS algorithm correctly assigns parents tovertices in a graph after performing a width path from a given vertex. |
| AdjacentListGraphTest | BFS | testBFSDistance | Starting vertices | The BFS algorithm correctly assigns distances from a given vertex to all other vertices in the graph |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Objective:** Test DFS-related functionalities | | | | |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| AdjacentListGraphTest | DFS | testDFSTime |  | The DFS algorithm correctly assigns completion times to each vertex in the graph. |
| AdjacentListGraphTest | DFS | testDFSDistance |  | The DFS algorithm correctly assigns distances to each vertex in the graph. Distances represent the number of edges that are traversed from the source vertex to each vertex during the depth traverse. |
| AdjacentListGraphTest | DFS | testDFSParents |  | The DFS algorithm correctly assigns the parents of each vertex in the graph. The parents represent the vertices from which each vertex was reached during the deep traversal. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objective of the test:** To test the functionalities of the Floyd-Warshall, Prim and Kruskal algorithms. | | | | |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| AdjacentListGraphTest | FloyWarshall | testFloydWarshall | Pairs of vertices and edges | Correctly calculates the shortest distances and path matrices in the graph. |
| AdjacentListGraphTest | Prim | testPrim | Pairs of vertices and edges | Correctly calculates the minimum spanning tree in the graph. The MST obtained must contain the edges that connect all the vertices of the graph so that the sum of the weights of the edges is minimal. |
| AdjacentListGraphTest | Kruskal | testKruskal | Pairs of vertices and edges | It is verified that the edges in the MST connect the correct vertices and that the weights of the edges match those expected. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objective of the Test:** Test the functionalities related to the vertices. | | | | |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| AdjacentListGraphTest | addVertex | testInsertVertex | City-type objects | Cities are correctly added as vertices. |
| AdjacentListGraphTest | addVertex | testAddRepeatedVertex | City-type objects | A repeat vertex exception is thrown. |
| AdjacentListGraphTest | addVertex | testAddingMultipleVertex | City-type objects | The vertices are added correctly with the size. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objective of the test:** Test the functionalities related to the edges. | | | | |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| AdjacentListGraphTest | addVertex  addEdge | testAddEdgeInSimpleGraph | Objects to add as vertices | It does the aggregation of vertices and their connections and is expected not to throw proper exceptions given for graphs that are not simple. |
| AdjacentListGraphTest | addVertex  addEdge | testAddEdgeInDirectedGraph | Objects to add as vertices | Add the vertices and their direct connections to other vertices, and then verify that the connections are correct. |
| AdjacentListGraphTest | addVertex  addEdge | testAddEdgeInPseudoGraph | Objects to add asvertices | The aggregation of vertices and their connections is done and validation is expected to verify that the first vertex has an edge pointing to itself and that the second has no edge pointing to it. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective:** To test the functionality of removing vertices in different graphs. | | | | |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| AdjacentListGraphTest | deleteVertex | testDeleteVertexInDirectedGraph | Objects to delete as vertices | It is checked whether the vertices are correctly removed from the directed graph. |
| AdjacentListGraphTest | deleteVertex | testDeleteVertexInPseudoGraph | Objects to delete as vertices | It is checked if the vertices are correctly removed from the multi-directed graph and if the cases of repeated elimination of vertices are properly handled. |
| AdjacentListGraphTest | deleteVertex | testDeleteVertexInSimpleGraph | Objects to delete as vertices | Se checks whether vertices are correctly removed from the simple graph. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Objective:** To test the functionality of removing edges in different graphs. | | | | |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| AdjacentListGraphTest | removeEdge | testDeleteEdgeInSimpleGraph | The two vertices that have the edge connection | The edges of the simple graph are correctly removed and the adjacency lists and connections are updated appropriately. |
| AdjacentListGraphTest | removeEdge | testDeleteEdgeExceptions | The two vertices that have the edge connection | Throws exceptions when you cannot delete edges |
| AdjacentListGraphTest | removeEdge | testDeleteEdgeInPseudoGraph | The two vertices that havethe edge connection | Edges are successfully removed from the pseudograph and if adjacency lists and connections between vertices are properly updated |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective:** Test the BFS algorithm. | | | | |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| AdjacentListGraphTest | BFS | testBFSColor | Start vertex | After the tour all the colors of the vertices have gone to black |
| AdjacentListGraphTest | BFS | testBFSParents | Start vertex | After the tour check and match the parents of each vertex within the graph |
| AdjacentListGraphTest | BFS | testBFSDistance | Start vertex | After the tour check the distances between the vertices. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Objective:** Test the DFS algorithm. | | | | |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| AdjacentListGraphTest | DFS | testDFSTime |  | After the tour check the time between each vertex. |
| AdjacentListGraphTest | DFS | testDFSDistance |  | After the tour check the distance between each vertex. |
| AdjacentListGraphTest | DFS | testDFSParents |  | After the tour check and match the parents of each vertex within the graph |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Objective:** Test the Floyd-Warshall, Prim and Kruskal algorithms. | | | | |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| AdjacentListGraphTest | floydWarshall | testFloydWarshall | Pairs of vertices and edges | The Floyd-Warshall algorithm correctly calculates the shortest distances between all pairs of vertices in the graph, and whether it also correctly obtains the shortest paths between vertices |
| AdjacentListGraphTest | Prim | testPrim | Pairs of vertices and edges | Prim's algorithm correctly finds the minimum spanning tree in the graph, and whether the vertices and weights obtained are as expected. |
| AdjacentListGraphTest | Kruskal | testKruskal | Pairs of vertices and edges | Kruskal's algorithm correctly finds the minimum spanning tree in the graph, and whether the vertices and weights obtained are as expected. It also checks whether the exception is handled correctly when trying to access an invalid position in the edge list |