**TEST DESIGN**

**Configuration of the Scenarios**

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| **setUpStage1** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Adding vertices to the graph:**  **Vertex added: 1**  **Vertex added: 2** |
| **setUpStage2** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Finding vertices in the graph:**  **Vertex to find: 8**  **Vertex found: 8** |
| **setUpStage3** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Removing vertices from the graph:**  **Vertex removed: 3**  **Vertex not present in the graph after removal.** |
| **setUpStage4** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Adding edges for DFS traversal:**  **Vertices added: 1, 2, 3, 4**  **Edges added: (1->2), (1->3), (2->4), (3->4)** |
| **setUpStage5** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **BFS traversal scenario:**  **Vertices added: 1, 2, 3, 4**  **Edges added: (1->2), (1->3), (2->4), (3->4)** |
| **setUpStage6** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Dijkstra algorithm scenario:**  **Vertices added: A, B, C, D**  **Edges added: (A->B), (A->C), (B->D), (C->D)** |
| **setUpStage7** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Floyd-Warshall algorithm scenario:**  **Vertices added: A, B, C**  **Edges added: (A->B), (A->C), (B->C)** |
| **setUpStage8** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Prim's algorithm scenario:**  **Vertices added: A, B, C**  **Edges added: (A->B), (A->C), (B->C)** |
| **setUpStage9** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Kruskal's algorithm scenario:**  **Vertices added: A, B, C**  **Edges added: (A->B), (A->C), (B->C)** |
| **setUpStage10** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Vertex added: Integer.MAX\_VALUE**  **Vertex added: Integer.MIN\_VALUE** |
| **setUpStage11** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Vertex to find: Integer.MAX\_VALUE**  **Vertex found: Integer.MAX\_VALUE**  **Vertex to find: Integer.MIN\_VALUE**  **Vertex found: Integer.MIN\_VALUE** |
| **setUpStage912** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **ertex removed: Integer.MAX\_VALUE**  **Vertex not present in the graph after removal.** |
| **setUpStage13** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Vertices added: 1, 2, 3, 4**  **Edges added: (1->2), (1->3), (2->4), (3->4)** |
| **setUpStage14** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Vertices added: 1, 2, 3, 4**  **Edges added: (1->2) weight: 15, (1->3) weight: 20** |
| **setUpStage15** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Vertex added: Integer.MAX\_VALUE**  **Vertex added: Integer.MIN\_VALUE**  **Vertices involved: Integer.MAX\_VALUE, Integer.MIN\_VALUE**  **Edge weight: 15** |
| **setUpStage16** | **AdjacencyListGraph and AdjacencyMatrixGraph** | **Vertex added: 1**  **Vertex added: 2** |

***TEST CASES OF ADJACENCYMATRIXGRAPH AND ADJACENCYLISTGRAPH***

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To verify the functionality of the addVertex method in the AdjacencyListGraph and AdjacencyMatrixGraph class by adding a vertex and checking if it exists within the graph's vertices.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **addVertex** | **SetupStage1** | **Ensuring that the added vertex is contained within the graph's list of vertices using the assertTrue method** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testAddVertexLimit** | **SetupStage9** | **The vertex with the maximum Integer value is successfully found in the graph.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testAddVertexInteresting** | **Vertices added: 1, 2, 3, 4**  **Edges added: (1->2) weight: 15, (1->3) weight: 20** | **Both added vertices should exist in the graph.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the findVertex method in the AdjacencyListGraph and AdjacencyMatrixGraph class by adding a vertex with value 8, searching for it, and ensuring that the retrieved vertex matches the one added.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **findVertex** | **SetupStage2** | **Asserting that the retrieved vertex from findVertex matches the initially added vertex using assertEquals.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testFindVertexLimit** | **SetupStage11** | **The vertex with the minimum Integer value is successfully found in the graph.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testFindVertexInteresting** | **Vertex to find: 15**  **Vertex not found: 100** | **The added vertex is found in the graph, while the non-existent vertex should return null.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To verify the functionality of the removeVertex method in the AdjacencyListGraph and AdjacencyMatrixGraph class by adding a vertex with value 3, removing it, and confirming that it doesn't exist in the graph's vertices anymore.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **removeVertex** | **SetupStage3** | **Ensuring that the graph does not contain the removed vertex using Assertions.assertFalse(graph.getVertices().contains(vertex)).** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testRemoveVertexLimit** | **SetupStage12** | **The vertex with the maximum Integer value is successfully removed from the graph.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testRemoveVertexInteresting** | **Vertex removed: 50**  **Verifying that the vertex has been removed correctly and its edges have also been removed.** | **The removed vertex should not exist in the graph, and related edges should also be removed.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the functionality of the addEdge method in the AdjacencyListGraph and AdjacencyMatrixGraph class. It adds two vertices (10 and 20) to the graph, creates an edge between them with a weight of 7, and checks whether the edge exists and has the correct weight.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **addEdge** | **setUpStage4** | **The test asserts that the edge exists (assertNotNull(edge)) and that its weight matches the expected value (assertEquals(7, edge.getWeight())).** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testAddEdgeLimit** | **SetupStage15** | **The edge between vertices with extreme Integer values is successfully added to the graph.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testAddEdgeInteresting** | **Vertex added: 30**  **Vertex added: 40**  **Vertex added: 50**  **Vertices involved: 30, 40**  **Edge weight: 10**  **Vertices involved: 40, 50**  **Edge weight: 5** | **The added edges with their specified weights exist in the graph.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the functionality of the addEdge method in the AdjacencyListGraph and AdjacencyMatrixGraph class. It adds two vertices (10 and 20) to the graph, creates an edge between them with a weight of 7, and checks whether the edge exists and has the correct weight.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **addEdge** | **setUpStage4** | **The test asserts that the edge exists (assertNotNull(edge)) and that its weight matches the expected value (assertEquals(7, edge.getWeight())).** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testRemoveEdgeEdgeCases** | **SetupStage16** | **The removal of an edge in an empty graph scenario.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testAddEdgeInteresting** | **Vertex added:**  **"A"**  **Vertex added: "B"**  **Vertex added: "C"**  **Vertices involved: "A", "B"**  **Edge weight: 2**  **Vertices involved: "B", "C"**  **Edge weight: 4** | **The specified edge is successfully removed while other edges remain intact in the graph.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the dfs method in the AdjacencyListGraph and AdjacencyMatrixGraph class by performing a Depth-First Search (DFS) traversal on a constructed graph and ensuring that the traversal order matches the expected order.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **dfs** | **SetupStage4** | **The test checks whether the obtained DFS traversal order matches the expected order (1, 2, 4, 3).** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testDFSLimit** | **Empty graph** | **The DFS traversal on an empty graph results in an empty list.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testDFSInteresting** | **Vertex added: 1**  **Vertex added: 2**  **Vertex added: 3** | **Each vertex should be in a different component, and the traversal order should match the added vertices.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To verify the correctness of the bfs method in the AdjacencyListGraph and AdjacencyMatrixGraph class by performing a Breadth-First Search (BFS) traversal on a constructed graph and checking if the traversal order matches the expected order.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **bfs** | **SetupStage5** | **The test verifies whether the obtained BFS traversal order matches the expected order (1, 2, 3, 4).** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testBFSEdgeCases** | **Vertex added: 1** | **BFS traversal on a single-vertex graph results in a list with that single vertex.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testBFSInteresting** | **Vertex added: 1**  **Vertex added: 2**  **Vertex added: 3** | **Each vertex should be in a different component, verifying the traversal order matches the added vertices.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the dijkstra method in the AdjacencyListGraph and AdjacencyMatrixGraph class by finding the shortest path between two vertices (vertexA and vertexD) and ensuring that the path is correctly determined.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **dijkstra** | **SetupStage6** | **The test verifies whether the obtained shortest path matches the expected path (vertexD -> vertexB -> vertexA).** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testDijkstraEdgeCases** | **Empty graph** | **Dijkstra on an empty graph results in a list with a single vertex.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testDijkstraInterestingCase** | **Vertex added: 'S'**  **Vertex added: 'A'**  **Vertex added: 'B'**  **Edge 1: 'S' -> 'A' with weight 10**  **Edge 2: 'S' -> 'B' with weight 5**  **Edge 3: 'A' -> 'B' with weight -2** | **The shortest path between specified vertices in a graph with negative weights.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To verify the correctness of the floydWarshall method in the AdjacencyListGraph and AdjacencyMatrixGraph class by computing the shortest paths between vertices and ensuring that the distances are accurate.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **floydWarshall** | **SetupStage7** | **The test checks whether the computed shortest paths match the expected distances between vertices.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testFloydWarshallEdgeCases** | **Empty graph** | **Floyd-Warshall on an empty graph results in an empty 2D array.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testFloydWarshallInterestingCase** | **Vertex added: 'S'**  **Vertex added: 'A'**  **Vertex added: 'B'**  **Edge 1: 'S' -> 'A' with weight 5**  **Edge 2: 'A' -> 'B' with weight 3**  **Edge 3: 'B' -> 'A' with weight -7** | **The algorithm should detect a negative cycle in the graph.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the primAL method in the AdjacencyListGraph and AdjacencyMatrixGraph class by constructing a Minimum Spanning Tree (MST) and ensuring that it contains the expected number of edges and vertices.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **primAL** | **SetupStage8** | **The test verifies whether the MST contains the expected number of vertices and edges.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testPrimEdgeCases** | **Adding the minimum number of vertices (two) and connecting them** | **MST on a graph with two vertices, with a single edge.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testPrimInterestingCase** | **Vertex added: 'S'**  **Vertex added: 'A'**  **Vertex added: 'B'**  **Edge 1: 'S' -> 'A' with weight 5**  **Edge 2: 'S' -> 'B' with weight 7** | **The MST should have all vertices and a specific number of edges.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the kruskalAL method in the AdjacencyListGraph and AdjacencyMatrixGraph class by constructing a Minimum Spanning Tree (MST) using Kruskal's algorithm and ensuring that it contains the expected number of edges and vertices.** | | | |
| **Class** | **Mehod** | **Scenario** | **Expected result** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **kruskalAL** | **SetupStage9** | **The test verifies whether the MST contains the expected number of vertices and edges.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testKruskalEmptyGraph** | **Creating an empty graph** | **Kruskal on an empty graph results in an MST with no vertices or edges.** |
| **Adjacencymatrixgraph and Adjacencylistgraph** | **testKruskalInterestingCase** | **Vertex added: 'A'**  **Vertex added: 'B'**  **Vertex added: 'C'**  **Vertex added: 'D'**  **Edge 1: 'A' -> 'B' with weight 1**  **Edge 2: 'A' -> 'C' with weight 5**  **Edge 3: 'B' -> 'D' with weight 2**  **Edge 4: 'C' -> 'D' with weight 1** | **The MST should contain all vertices and a specific number of edges.** |