**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)** |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To verify the functionality of the addVertex method in the AdjacencyListGraph 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** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the findVertex method in the AdjacencyListGraph 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.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To verify the functionality of the removeVertex method in the AdjacencyListGraph 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)).** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the functionality of the addEdge method in the AdjacencyListGraph 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())).** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the dfs method in the AdjacencyListGraph 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).** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To verify the correctness of the bfs method in the AdjacencyListGraph 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).** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the dijkstra method in the AdjacencyListGraph 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).** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To verify the correctness of the floydWarshall method in the AdjacencyListGraph 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.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the primAL method in the AdjacencyListGraph 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.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective of the Test: To validate the kruskalAL method in the AdjacencyListGraph 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.** |