Design of use cases - Integrative task one.

Test cases, add vertex - edge.

Test objective: Test the functionality of adding a new vertex to the graph.				
Class	Method	Scenery	Input values	Expected result
GraphTest	addVertex()	One more vertex is added to a graph of 8 vertices.	New Key and Element.	The vertex is added successfully.

Test objective: Verify that a duplicate vertex cannot be added to the graph.				
Class	Method	Scenery	Input values	Expected result
GraphTest	addVertex(K key, T element)	The graph has 8 vertices.	Key = 1, element = "One"	The duplicate vertex should not be allowed to be added, and the graph should remain unchanged.

Test objective: Verify that an edge can be added successfully to the graph.				
Class	Method	Scenery	Input values	Expected result
GraphTest	addEdge(K sourceKey, K destinationKey, double weight)	The graph contains vertices with sourceId and targetId.	sourceId = 1, targetId = 4, weight = 2.5	An edge with weight 2.5 should be added between the vertices with sourceId and targetId.

Test cases, delete.

Test objective: Verify that a vertex and its associated edges can be successfully removed from the graph.				
Class	Method	Scenery	Input values	Expected result
GraphTest	deleteVertex(K key)	The graph is initialized with vertices and	key = 8	Vertex with key 8 should be removed, and

		edges.		the associated edges should no longer exist.

Test objective: Verify that trying to remove a nonexistent vertex does not affect the graph.				
Class	Method	Scenery	Input values	Expected result
GraphTest	deleteVertex(K key)	The graph is initialized with vertices and edges. (1 - 8)	key = 9	The nonexistent vertex should not affect the graph, and other vertices should remain unchanged.

Test objective: Verify that an edge can be successfully removed from the graph.				
Class	Method	Scenery	Input values	Expected result
GraphTest	deleteEdge(K sourceKey, K destinationKey)	The graph is initialized with vertices and edges.	sourceKey = 2, destinationKey = 4	The edge between vertices with IDs 2 and 4 should be removed from the graph.

Test cases, search.

Test objective: Verify that a vertex can be successfully searched in the graph.				
Class	Method	Scenery	Input values	Expected result
GraphTest	searchEdge(K sourceKey, K destinationKey)	The graph is initialized with vertices and edges.	key = 3	The vertex with Key 3 should be found, and its element should be "Three".

Test objective: Verify that searching for a nonexistent edge returns null.				
Class	Method	Scenery	Input values	Expected result
GraphTest	searchEdge(K	The graph is	key = 4, key = 6	The edge

	sourceKey, K destinationKey)	initialized with vertices and edges.		between vertices with Keys 4 and 6 does not exist, so searching for it should return null.

Test objective: Verify that searching for a nonexistent vertex in the graph returns null.				
Class	Method	Scenery	Input values	Expected result
GraphTest	searchEdge(K sourceKey, K destinationKey)	The graph is initialized with vertices and edges. (1 - 8)	key = 9	The vertex with Key 9 does not exist in the graph, so searching for it should return null.

Test cases, BFS.

Test objective:Verify that when applying the BFS algorithm all the vertices of the graph are black.				
Class	Method	Scenery	Input values	Expected result
GraphTest	public void BFS(K sourceElement)	The graph is initialized with vertices and edges. (1 - 8)	sourceElement = 1	The color of each vertex should be black at the end of the operation.

Test objective: Verify the correct functioning of the BFS algorithm in a graph.				
Class	Method	Scenery	Input values	Expected result
GraphTest	public void BFS(K sourceElement)	The graph is initialized with vertices and edges. (1 - 8)	None	Vertex predecessors are expected to follow the correct order according to the BFS algorithm.

Test objective: Verify the behavior of the BFS algorithm when it encounters a disconnected vertex.				
Class	Method	Scenery	Input values	Expected result
GraphTest	public void BFS(K sourceElement)	The graph is initialized with vertices and edges. (1 - 9)	None	The disconnected vertex (vertex 9) is expected to have color white, discovery time -1, and no predecessor assigned.

Test cases, DFS.

Test objective: Check if there is a backtracking edge in the graph after running the DFS algorithm.				
Class	Method	Scenery	Input values	Expected result
GraphTest	public void DFS()	The graph is initialized with vertices and edges, with a back edge added in the graph	None	No backtracking edge is expected to be found in the graph after running the DFS algorithm.

Test objective:Check if the vertex predecessors follow the correct order after running the DFS algorithm.				
Class	Method	Scenery	Input values	Expected result
GraphTest	public void DFS()	The graph is initialized with vertices and edges.	None	Vertex predecessors are expected to follow the correct order according to the DFS algorithm.

Test objective: Check if the vertex completion order is as expected after running the DFS algorithm.

Class	Method	Scenery	Input values	Expected result
GraphTest	public void DFS()	The graph is initialized with vertices and edges.	None	The vertex completion list is expected to be ordered in increasing order.

Test cases, Dijkstra.

Test objective: Verify the shortest path between two points using dijkstra's algorithm.				
Class	Method	Scenery	Input values	Expected result
GraphTest	public Path<K> dijkstra(K eSource, K eDestination)	Graph with the following nodes and edge weights: (1, 2, 3), (2, 5, 2), (5, 8, 1)	source = 1, destination = 8	The shortest path between 1 and 8 is [1, 2, 5, 8], with distance 3.0.

Test objective: Verify when the origin and destination are the same.				
Class	Method	Scenery	Input values	Expected result
GraphTest	public Path<K> dijkstra(K eSource, K eDestination)	Graph with any configuration of nodes and edge weights.	source = 1, destination = 1	The path must be [1], since the source and destination are the same, and the distance must be 0.0.

Test objective: Check when there is no path between source and destination.				
Class	Method	Scenery	Input values	Expected result
GraphTest	public Path<K> dijkstra(K eSource, K eDestination)	Unconnected graph between source and destination.	source = 1, destination = 9	The path variable (path) must be null, indicating that there is no valid path between the source and the destination.