Members:

-Pablo Pineda - A00395831
-Daniela Londoño - A00392917
-Isabella Huila - A00394751

Configuration of the Scenarios

Name	Class	Stage
setupStage1	GraphListTest	An object of type Undirected AdjacencyListGraph is created. 3 vertices are added to the graph with identifiers 1, 2 and 3. Two edges are then added to the graph: one edge connecting vertex 1 to vertex 2, and another edge connecting vertex 2 to vertex 3.
setupStage2	GraphListTest	An object of type Directed AdjacencyListGraph is created. 3 vertices are added to the graph with identifiers 1, 2 and 3. Two edges are then added to the graph: one edge going from vertex 1 to vertex 2, and another edge going from vertex 2 to vertex 3.
setupStage3	GraphListTest	An object of type Undirected AdjacencyListGraph is created. 2 vertices are added to the graph with identifiers 1 and 2.
setupStage4	GraphListTest	An object of type Undirected AdjacencyListGraph is created. 4 vertices are added to the graph with identifiers 1, 2, 3 and 4. The connections between the vertices and the associated weights are established: Vertex "1" has a neighbor "2" with a weight of 2. Vertex "1" has a neighbor "3" with a weight of 4. Vertex "2" has a neighbor "3" with a weight of 1. Vertex "2" has a neighbor "4" with a weight of 7. Vertex "3" has a neighbor "4" with a weight of 3.
setupStage5	GraphMatrxTest	An object of type directed AdjacencyMatrixGraph is created. 4 vertices are added to the graph with identifiers “a”, “b”, “c” and “d” The connections between the vertices and the associated weights are established: Vertex "a" has a neighbor "b" with a weight of 3. Vertex "a" has a neighbor "c" with a weight of 5. Vertex "b" has a neighbor "c" with a weight of 1. Vertex "b" has a neighbor "d" with a weight of 2. Vertex "c" has a neighbor "d" with a weight of 4.
setupStage6	GraphMatrixTest	An object of type directed AdjacencyMatrixGraph is created. 4 vertices are added to the graph with identifiers “a”, “b”, “c” and “d” The connections between the nodes and the associated weights are established: Vertex "a" has a neighbor "b" with a weight of 3. Vertex "a" has a neighbor "c" with a weight of 5. Vertex "b" has a neighbor "c" with a weight of -5. Vertex "b" has a neighbor "d" with a weight of 2. Vertex "c" has a neighbor "d" with a weight of 4.
setupStage7	GraphMatrixTes	An object of type directed AdjacencyMatrixGraph is created.

	t	4 vertices are added to the graph with identifiers “a”, “b”, “c” and “d” Vertex "a" has a neighbor "b" with a weight of 2. Vertex "b" has a neighbor "a" with a weight of 2. Vertex "c" has a neighbor "d" with a weight of 3. Vertex "d" has a neighbor "c" with a weight of 3.

Test Case Design

Objective of the Test: Verify that the graph works well, as well as its main methods of adding and removing both vertex and edge. Another objective is to verify the proper functioning of the algorithms that allow the search for shortest paths, the exhaustive search of graphs or the construction of minimum spanning trees.

Class	Method	Stage	Input Values	Expected result
GraphList	addVertice	setupStage 1	size=3 vertexValue=1 vertexValue=2 vertexValue=3	- Expected number of vertices: 3 - Value of the first expected vertex: 1 - Expected second vertex value: 2 - Value of the third expected vertex: 3
GraphList	addVertice	setupStage 1	newVertex=2	- Size of the graph before adding the vertex: 2 - Graph size after trying to add vertex 2: same size as before
GraphList	deleteVertice	setupStage 1	size=2 vertexValue=1 vertexValue=3	- Success in removing vertex 2 - Expected number of vertices after deleting: 2 - Value of the first vertex expected after deleting: 1 - Expected second vertex value after deleting: 3
GraphList	deleteVertice	setupStage 1	vertexToDelete=1 vertexToDelete=2 vertexToDelete=3 vertexToSearch=3	- Success in removing vertices 1, 2 and 3 - Null result when searching for vertex 3 after deleting it
GraphList	deleteVertice	setupStage 1	vertexToDelete=1 verticesSize=2	- Success in deleting vertex 1 - Expected number of vertices after deleting: 2
GraphList	addArista	setupStage 1	edgesSize=2 vertexEdgeSize=1, 2 (IndexVertex, edgesSize) vertexEdgeSize=2, 1	- Expected number of edges: 2 - Expected number of edges of vertex 2: 2 - First edge of vertex 2 not null - Expected number of edges of vertex 3: 1
GraphList	addArista	setupStage 2	vertexEdgeSize=1, 1 vertexEdgeSize=2, 0	- Expected number of edges of vertex 1: 1 - First edge of vertex 1 not zero - Expected number of edges of vertex 2: 0
GraphList	deleteArista	setupStage 1	elementA=2 elementB=3 edgesSize=1	- Success in deleting the edge between vertices 2 and 3 - Expected number of edges: 1 - Expected number of edges of the source

				element: 1 - Expected number of edges of the destination element: 2
GraphList	deleteArista	setupStage 2	vertexEdgeSize=1, 1 elementA=1 elementB=2 vertexEdgeSize=0, 0 vertexEdgeSize=1, 0	- Expected number of edges of vertex 1: 1 - First edge of vertex 1 not zero - Expected number of edges of vertex 2: 0 - Success in deleting the edge between vertices 1 and 2 - Number of edges of vertex 1 expected after deleting: 0 - Number of edges of vertex 2 expected after deleting: 0
GraphList	deleteArista	setupStage 1	elementA=1 elementB=2 edgesSize=1	- Success in deleting the edge between vertices 1 and 2 - Expected number of edges after deleting: 1
GraphList	searchVertice	setupStage 3	vertexToSearch=2	- Successful search for vertex 2 - vertex found not null - Value of the vertex found: 2
GraphList	searchVertice	setupStage 3	vertexToSearch=4	- Unsuccessful search for vertex 4 - Vertex not found (null)
GraphList	searchVertice	setupStage 3	vertexToSearch=10 vertexToSearch=1	- Unsuccessful search for vertex 10 - Vertex not found (null) - Successful search for vertex 1 - Value of the vertex found: 1
GraphList	BFS	setupStage 1	indexVertexToStart=0	- Successful BFS from vertex 1
GraphList	BFS	setupStage 2	newVertex=6 indexVertexToStart=4	- Failure to perform BFS from vertex 6 as it has no associated edge
GraphList	BFS	setupStage 2	indexVertexToStart=0 indexVertexToStart=2	- Failure when performing BFS from vertex 1 - Success when performing BFS from vertex 3
GraphList	DFS	setupStage 1	newVertex=4 newEdge=4,1	- Expected value when performing DFS: 1 - Add vertex 4 and edge between 4 and 1 - Expected value when performing DFS again: 2
GraphList	DFS	setupStage 2	newVertex=8 newVertex=9 newEdge=8, 9	- Add vertices 9 and 8, and edge between 8 and 9 - Expected value when performing DFS: 5
GraphList	DFS	setupStage 1	vertexToDelete=1 vertexToDelete=2 vertexToDelete=3	- Delete vertices 1, 2 and 3 - Expected value when performing DFS: 0
GraphList	dijkstra	setupStage 4	vertexOrigin=1 vertesDestination=4;	- The value of the shortest path from vertex 1 to vertex 4 is 6
GraphList	dijkstra	setStage4	vertexOrigin=1 vertesDestination=1;	The value of the shortest path from vertex 1 to vertex 1 is 0

GraphList	dijkstra	setupStage 4	vertexOrigin=1 vertesDestination=6;	- Must return a -1 because there is no valid path between vertices 1 and 6
GraphMatrix	floydWarshall	setupStage 5	…	The values added in the scenario were saved successfully - The result to throw must be null
	floydWarshall	setupStage 6	…	The values added in the scenario were saved successfully - The result to throw must be null
GraphMatrix	floydWarshall	setupStage 7	…	- "a" is not reachable from "c". - "a" is not reachable from "d". - "b" is not reachable from "c". - "b" is not reachable from "d". - "c" is not reachable from "a". - "c" is not reachable from "b". "is not reachable" is represented by: Integer.MAX_VALUE
GraphList	prim	setupStage 4	start=3	- The weight of the minimum spanning tree will be 6
GraphList	prim	setupStage 1	start=1	- The minimum spanning tree weight will be 3
GraphList	prim	setupStage 3	start=1	- The weight of the minimum spanning tree shall be 1
GraphList	kruskal	setupStage 4	GraphList<Integer> y ArrayList<Edge<Integer> >	The expected result is that the minimumSpanningTree variable contains the following edges in order: (1, 2, 1) (2, 4, 2) (1, 3, 3)
GraphList	kruskal	setupStage 1	…….	The expected result is that the minimumSpanningTree variable is empty, that is, it does not contain any edges.
GraphList	kruskal	setupStage 2	GraphList<Integer> y ArrayList<Edge<Integer> >	Since the edges of the graph form a cycle, it is not possible to construct a minimum spanning tree without forming cycles. Therefore, in this case, the expected result is that the algorithm returns an empty minimum spanning tree, which is verified by the assertion

Name	Class	Stage
setupStage1	MazeTest	An undirected GraphMatrix type object is created. {1, 0, 1} {1, 1, 1} {0, 0, 1}
setupStange 2	MazeTest	An undirected GraphMatrix type object is created. {1, 0, 1}, {1, 0, 1}, {1, 0, 1}

Test Case Design

Objective of the Test: Verify that the path traversed by the maze is as expected with the BFS and DFS algorithms. Also, verify that when the path is unreachable, it does not return any path.				
Class	**Method**	**Stage**	**Input Values**	**Expected result**
Maze	solveMazeWithBFS	setupStage1	start = {0, 0}; end = {2, 2}; useBFS = true;	The route would be as follows: - {0, 0}, - {1, 0}, - {1, 1}, - {1, 2}, - {2, 2}
Maze	solveMazeWithDFS	setupStange 1	start = {0, 0}; end = {2, 2}; useBFS = false;	The route would be as follows: - {0, 0}, - {1, 0}, - {1, 1}, - {1, 2}, - {2, 2}
Maze	testSolveMazeWithUnreachableEnd	setupStange 2	start={0,0}; end= {2, 2}; useBFS= true;	The expected result would be an empty list, since it could not reach the output: - {}