Jesús Daniel Villota
Natalia González

UNIT TESTING

Test 1: Verify if the method addVertex adds a vertex to the graph correctly.				
Class	**Method**	**Scenario**	**Input**	**Output**
Graph	addVertex	An empty graph	One single vertex	The graph now contains one vertex.
Graph	addVertex	A graph that contains four vertices with the next values: 1, 2, 3, 4	One vertex with a value of 5	The graph now contains five vertices and the vertex most recently added.
Graph	addVertex	A graph that contains four vertices with the next values: 1, 2, 3, 4	One vertex with a value of 1	The graph still contains four vertices and the vertex with value of 1 was not added because another vertex with the same value already exists.

Test 2: Verify if the method addEdge adds a weighted - directed edge to the graph correctly.				
Class	**Method**	**Scenario**	**Input**	**Output**
Graph	addEdge	A graph that contains four vertices with the next values: 1, 2, 3, 4	Vertex x = 1 Vertex y = 4 Weight = 5	The graph now contains an edge between the vertices 1 and 4 with a weight of 5.
Graph	addEdge	A graph that contains four vertices with the next values: 1, 2, 3, 4 And contains an edge from	Vertex x = 1 Vertex y = 4 Weight = 5	The graph already contains an edge from 1 to 4 with a weight of 5, so there are no changes

		1 to 4 with a weight of 5.		made in the graph.
Graph	addEdge	A graph that contains four vertices with the next values: 1, 2, 3, 4	Vertex x = 2 Vertex y = 2 Weight = 10	An edge from 2 to 2 (a loop) with a weight of 10 was added to the graph.
Graph	addEdge	A graph that contains four vertices with the next values: 1, 2, 3, 4 And the next edges: x, y, w [1, 2, 3] [2, 4, 6] [4, 2, 3] [3, 4, 5]	Vertex x = 2 Vertex y = 3 Weight = 5	An edge from 2 to 3 was added. Now vertices 4 and 3 are adjacent to vertex 2.

Test 3: Verify if the method removeVertex removes the specified vertex from the graph and every connection to it correctly.				
Class	**Method**	**Scenario**	**Input**	**Output**
Graph	removeVertex	A graph that contains four vertices with the next values: 1, 2, 3, 4 And the next edges: x, y, w [1, 2, 3] [2, 4, 6] [4, 2, 3] [3, 4, 5]	Vertex x = 2	The graph now contains 3 vertices: 2,3,4 And the next edge: x, y, w [3, 4, 5]
Graph	removeVertex	A graph that contains three vertices with the next values: 2, 3, 4	Vertex x = 1	The graph still contains three vertices with the next values: 2,3,4

		And the next edge: x, y, w [3, 4, 5]		And the next edge: x, y, w [3, 4, 5] No changes done.
Graph	removeVertex	A graph that contains four vertices with the next values: 1, 2, 3, 4 And the next edges: x, y, w [1, 3, 3] [1, 4, 6] [1, 3, 5]	Vertex x = 2	The graph now contains three vertices with the next values: 1,3,4 And the next edges: x, y, w [1, 3, 3] [1, 4, 6] [1, 3, 5]
Graph	removeVertex	An empty graph	Vertex x = 1	The graph is still empty. There aren't edges to remove.

Test 4: Verify if the method removeEdge removes the specified edge from the graph.				
Class	**Method**	**Scenario**	**Input**	**Output**
Graph	removeEdge	A graph that contains four vertices with the next values: 1, 2, 3, 4 And the next edges: x, y, w [1, 2, 3] [2, 4, 6] [4, 2, 3] [3, 1, 5]	Vertex x = 2 Vertex y = 4	The graph still has the same four vertices: 1, 2, 3, 4 And the remaining edges are: x, y, w [1, 2, 3] [4, 2, 3] [3, 1, 5]
Graph	removeEdge	A graph that contains four vertices with	Vertex x = 1 Vertex y = 3	The graph still has the same four vertices:

		the next values: 1, 2, 3, 4 And the next edges: x, y, w [1, 2, 3] [2, 4, 6] [4, 2, 3] [3, 1, 5]		1, 2, 3, 4 And no edge was removed because the specified edge doesn't exist in the actual graph. The remaining edges are: x, y, w [1, 2, 3] [2, 4, 6] [4, 2, 3] [3, 1, 5]
Graph	removeEdge	An empty graph	Vertex x = 1 Vertex y = 2	The graph is still empty and there aren't edges to remove.

Test 5: Verify that the method getVertex returns the vertex correspondent to the given value if and only if the graph contains the vertex.				
Class	**Method**	**Scenario**	**Input**	**Output**
Graph	getVertex	A graph that contains four vertices with the next values: 1, 2, 3, 4 And the next edges: x, y, w [1, 2, 3] [2, 4, 6] [4, 2, 3] [3, 1, 5]	Value x = 2	The method returns the vertex correspondent to the value of 2.
Graph	getVertex	A graph that contains four vertices with the next values: 1, 2, 3, 4	Value x = 5	The method returns a NIL value because the specified value doesn't have any related vertex

		And the next edges: x, y, w [1, 2, 3] [2, 4, 6] [4, 2, 3] [3, 1, 5]		in the actual graph.
Graph	getVertex	An empty graph	Value x = 1	The method returns a NIL value because it's empty.

Test 6: Verify that the method areAdjacent returns the correct Boolean value according to the correspondent given vertices and if the correct connection exists between them.				
Class	**Method**	**Scenario**	**Input**	**Output**
Graph	areAdjacent	A graph that contains four vertices with the next values: 1, 2, 3, 4 And the next edges: x, y, w [1, 2, 3] [2, 4, 6] [4, 2, 3] [3, 1, 5]	Vertex x = 2 Vertex y = 4	The method returns true, because there is an edge from vertex 2 to vertex 4 in the actual graph.
Graph	areAdjacent	A graph that contains four vertices with the next values: 1, 2, 3, 4 And the next edges: x, y, w [1, 2, 3] [2, 4, 6] [4, 2, 3] [3, 1, 5]	Vertex x = 1 Vertex y = 4	The method returns false, because there is not an edge from vertex 1 to vertex 4 in the actual graph.

Graph	areAdjacent	An empty graph	Vertex x = 1 Vertex y = 2	The method returns false because the given vertices don't even exist in the actual graph.

Test 7: Verify that the method bfs adjusts the information of the vertices in the given graph correctly, according to what the known algorithm is supposed to do.				
Class	**Method**	**Scenario**	**Input**	**Output**
Graph	bfs	A graph that contains four vertices with the next values: 1, 2, 3, 4, 5 And the next edges: x, y, w [1, 2, 3] [2, 4, 6] [4, 3, 3] [1, 3, 5] [4, 5, 2]	Vertex source = 1	The vertices in the actual graph now have the information assigned as it follows: 1.pred = NIL 2.pred = 1 3.pred = 1 4.pred = 2 5.pred = 4 1.dist = 0 2.dist = 3 3.dist = 5 4.dist = 9 5.dist = 11

Test 8: Verify that the method dfs adjusts the information of the vertices in the given graph correctly, according to what the known algorithm is supposed to do.				
Class	**Method**	**Scenario**	**Input**	**Output**
Graph	dfs	A graph that contains four vertices with the next values: 1, 2, 3, 4, 5 And the next edges:	Vertex source = 4	The vertices in the actual graph now have the information assigned as it follows: 1.pred = NIL

		x, y, w [1, 2, 3] [2, 4, 6] [4, 3, 3] [1, 3, 5] [4, 5, 2]		2.pred = NIL 3.pred = 4 4.pred = NIL 5.pred = 4 1.dist = INF 2.dist = INF 3.dist = 3 4.dist = 0 5.dist = 2

Test 9: Verify that the method prim assigns the correct values to the vertices so that it corresponds to forming a minimum spanning tree.				
Class	**Method**	**Scenario**	**Input**	**Output**
Graph	prim	A graph that contains four vertices with the next values: 1, 2, 3, 4, 5 And the next edges: x, y, w [1, 2, 2] [2, 1, 2] [1, 3, 12] [3, 1, 12] [2, 3, 7] [3, 2, 7] [2, 4, 15] [4, 2, 15] [3, 4, 3] [4, 3, 3] [2, 5, 4] [5, 2, 4] [4, 5, 6] [5, 4, 6]	None	The vertices in the actual graph now have the information assigned as it follows: 1.pred = NIL 2.pred = 1 3.pred = 4 4.pred = 5 5.pred = 2 1.dist = 0 2.dist = 2 3.dist = 15 4.dist = 12 5.dist = 6

Test 10: Verify that the method kruskal assigns the correct values to the vertices so that it corresponds to forming a minimum spanning tree.				
Class	**Method**	**Scenario**	**Input**	**Output**
Graph	kruskal	A graph that contains four	None	The method returns a list

		vertices with the next values: 1, 2, 3, 4, 5 And the next edges: x, y, w [1, 2, 2] [2, 1, 2] [1, 3, 12] [3, 1, 12] [2, 3, 7] [3, 2, 7] [2, 4, 15] [4, 2, 15] [3, 4, 3] [4, 3, 3] [2, 5, 4] [5, 2, 4] [4, 5, 6] [5, 4, 6]		with the edges that conforms the minimum spanning tree from the actual graph which are the following ones: { (1,2), (3,4), (2,5), (4,5) }

Test 11: Verify that the method Dijkstra assigns the correct values to the vertices so that it matches with the shortest path between the given vertices.				
Class	**Method**	**Scenario**	**Input**	**Output**
Graph	dijkstra	A graph that contains four vertices with the next values: 1, 2, 3, 4, 5 And the next edges: x, y, w [1, 2, 5] [1, 3, 1] [1, 4, 3] [2, 5, 8] [4, 2, 2] [4, 5, 5] [4, 3, 2]	Vertex source = 1 Vertex dest = 5	The vertices in the actual graph now have the information assigned as it follows: 1.pred = NIL 2.pred = 4 3.pred = 1 4.pred = 1 5.pred = 2 1.dist = 0 2.dist = 5 3.dist = 1 4.dist = 3 5.dist = 10

Test 12: Verify that the method Floyd Warshall finds the correct minimum distances from every vertex to any other vertex in the graph.				
Class	**Method**	**Scenario**	**Input**	**Output**
Graph	floydWarshall	A graph that contains four vertices with the next values: 1, 2, 3, 4 And the next edges: x, y, w [1, 2, 1] [2, 3, 2] [1, 4, 5] [3, 4, 1]	None	The method returns the next matrix of distances: 1 2 3 4 1 [0 1 3 4] 2 [∞ 0 2 3] 3 [∞ ∞ 0 1] 4 [∞ ∞ ∞ 0]