Test cases design
Computation and discrete structures

ADT test case design:

Graph:

Scenarios configuration:

Scenario	Class	Scenario
setUp1()	AdjacencyList	• No vertex or edges added to the graph.

setUp2()	AdjacencyList	graph.addVertex(0, 0);
		graph.addVertex(1, 1);
		graph.addVertex(2, 2);

setUp3()	AdjacencyList	graph.addVertex(0, 0);
		graph.addVertex(1, 1);
		graph.addVertex(2, 2);
		
		graph.addEdge(0, 1 , 10);
		graph.addEdge(1, 2 , 5);
		graph.addEdge(2, 0 , 20);

setUp4()	AdjacencyList	graph.addVertex(0, 0);
		graph.addVertex(1, 1);
		graph.addVertex(2, 2);
		
		graph.addEdge(1, 0 , 10);
		graph.addEdge(1, 2 , 5);

setUp5()	AdjacencyList	graph.addVertex(0, 0);
		graph.addVertex(1, 1);
		graph.addVertex(2, 2);
		
		graph.addEdge(0, 1 , 10);
		graph.addEdge(1, 2 , 5);
		graph.addEdge(2, 0, 20);

setUp6()	AdjacencyList	graph.addVertex(0, 0);
		graph.addVertex(1, 1);
		graph.addVertex(2, 2);
		graph.addVertex(3,3);
		graph.addVertex(4,4);
		
		graph.addEdge(0, 1 , 10);
		graph.addEdge(1, 2 , 5);
		graph.addEdge(2,0,20);
		graph.addEdge(3,4,1);

setUp7()	AdjacencyList	graph.addVertex(0, 0);
		graph.addVertex(1, 1);
		graph.addVertex(2, 2);
		
		graph.addEdge(0, 1 , 10);
		graph.addEdge(1, 2 , 5);
		graph.addEdge(2,0,20);
		graph.addEdge(2, 1 , 5);
		graph.addEdge(1, 0 , 10);
		graph.addEdge(0,2,20);
		graph.addEdge(0,1,2);

Tests:

Test Goal: Check the correct insertion of nodes in the graph				
Class	Method	Scenario	Input Values	Result
AdjacencyList	addVertex()	setUp1();	addVertex(0,0)	The graph should only have 1 node.
AdjacencyList	addVertex()	setUp1();	addVertex(1,1) graph.addVertex(1,4)	The program must show an exception because the user is trying to add two nodes with the same key
AdjacencyList	addVertex()	setUp1();	graph.addVertex(1,1) graph.addVertex(2,2)	The graph should have 3 nodes, and the

			graph.addVertex(3,3)	one with key 2 also has the value of 2.

Test Goal: Check the correct insertion of an edge in the graph				
Class	Method	Scenario	Input Values	Result
AdjacencyList	addEdge()	setUp2();	addEdge(2,0, 10)	The node of the graph with the key 2 should have in its adjacent nodes the node 0 with a weight of 10
AdjacencyList	addEdge()	setUp2();	addEdge(2, 0, 10) addEdge(2,0, 20)	The node of the graph with the key 2 should have in its adjacent nodes the node 0 two times with a weight of 10 and with the weight of 20.
AdjacencyList	addEdge()	setUp2();	addEdge(1, 0, -5)	The node of the graph with the key 1 should have in its adjacent nodes the node 0 with a weight of -5

Test Goal: Check the correct development of the BFS algorithm in the graph.				
Class	Method	Scenario	Input Values	Result
AdjacencyList	BFS()	setUp3();	BFS(0)	The method should return an ArrayList with the value of the nodes that are in the path of this algorithm. In this case the nodes are 0, 1, and 2.
AdjacencyList	BFS()	setUp4();	BFS(1)	The method should return an ArrayList with just one node because in the node that the algorithm begins can’t access another node.

AdjacencyList	BFS()	setUp4();	BFS(1)	The method should return an ArrayList with the value of the nodes that are in the path of this algorithm. In this case the order of the nodes are 1, 0, and 2.

Test Goal: Check the correct development of the DFS algorithm in the graph.				
Class	Method	Scenario	Input Values	Result
AdjacencyList	DFS()	setUp5();	None	The method should return the number of subtrees that are in the graph, in this case is 1.
AdjacencyList	DFS()	setUp6();	None	The method should return the number of subtrees that are in the graph, in this case is 2.
AdjacencyList	DFS()	setUp1	None	The method should return the number of subtrees that are in the graph, in this case is 0, because there aren't any nodes in this graph.

Test Goal: Check the correct development of the Dijkstra algorithm in the graph.				
Class	Method	Scenario	Input Values	Result
AdjacencyList	dijkstra()	setUp5();	dijkstra(1)	The method should return an arraylist with the weights of the shortest path between that node to each other. In this case the

				values of the arraylist should be: 0,10,15.
AdjacencyList	dijkstra()	setUp5();	dijkstra(2)	The method should return an arraylist with the weights of the shortest path between that node to each other. In this case the values of the arraylist are 25, 0, 5.
AdjacencyList	dijkstra()	setUp5();	dijkstra(3)	The method should return an arraylist with the weights of the shortest path between that node to each other. In this case the values of the arraylist are 20, 30, 0.

Test Goal: Check the correct development of the Floyd Warshall algorithm in the graph.				
Class	Method	Scenario	Input Values	Result
AdjacencyList	floydWarshall()	setUp5();	None	The method should return a matrix with the weight of the node i to another node j in the position [i][j]. In this case the diagonals must be all 0 because there aren't any cycles.
AdjacencyList	floydWarshall()	setUp5();	None	The value of the matrix in the coordinates [0][0] = 0, [0][1] = 10, [0][2] = 15
AdjacencyList	floydWarshall()	setUp6();	None	The value of the matrix in the coordinates [0][0] = 0, [0][1] = 10, [0][2] = 15 [0][3] == INF and [3][0] == INF

Test Goal: Check the correct development of the Prim algorithm in the graph.				
Class	Method	Scenario	Input Values	Result
AdjacencyList	prim()	setUp7();	prim(0)	The expected return value is an array with the weights of each edge of the tree. In this case the array should be equal to: 0 ,2 ,20
AdjacencyList	prim()	setUp7();	prim(1)	The expected value to the array in this test should be 10, 0, 5
AdjacencyList	prim()	setUp7();	prim(2)	The expected value to the array in this test should be 20, 5, 0

Test Goal: Check the correct development of the Kruskal algorithm in the graph.				
Class	Method	Scenario	Input Values	Result
AdjacencyList	kruskal()	setUp7()	kruskal(0)	The method return the minimum path for connected all trees, in this case the minimum distances for connected all vertex is 2,5
AdjacencyList	kruskal()	setUp5()	kruskal(1)	The minimum distances for connected all vertex are 5,10
AdjacencyList	kruskal()	setUp4()	kruskal(2)	The minimum distances for connected all vertex are 5,10

Solution of the problem unitary test design:

Scenarios configuration:

Scenario	Class	Scenario
setUp1()	AdjacentEdges	• No vertex or edges are added to the graph
setUp2()	AdjacentEdges	• All the data saved in the file Data is loaded to this setUp. This includes the 62 nodes and the 95 edges that represent the south of Cali on a Wednesday afternoon.

Tests:

Test Goal: Check the correct method of Dijkstra in graph of the solution				
Class	Method	Scenario	Input Values	Result
AdjacencyList	dijkstra()	setUp2();	dijkstra(1, 61)	The program must show the path that the user needs to follow that in this case is: 1 -> 7 ->12 ->20->21->29->30->43->39->47->51->53->61
AdjacencyList	dijkstra()	setUp2();	dijkstra(9, 27)	The program must show the path that the user needs to follow that in this case is: 9 -> 10 ->15 ->16->23->27
AdjacencyList	dijkstra()	setUp2();	dijkstra(36, 58)	The program must show the path that the user needs to follow that in

				this case is: 36 -> 29 ->30->43->39->47->51-> 53-> 58

Test Goal: Check the correct method of Prim in graph of the solution				
Class	Method	Scenario	Input Values	Result
AdjacencyList	prim()	setUp2();	prim(14)	The program should show the minimum spanning tree initiating in the node 14
AdjacencyList	prim()	setUp2();	prim(0)	The program should show the minimum spanning tree initiating in the node 14
AdjacencyList	prim()	setUp2();	prim(22)	The program should show the minimum spanning tree initiating in the node 22