|  |  |  |
| --- | --- | --- |
| Nombre | Clase | Escenario |
| setUpScenery1 | AdjacencyListTest |  |
| setUpScenery1 | AdjacencyMatrixTest |  |
| setUpScenery1 | Dijkstra |  |
| setUpScenery1 | FloydW |  |
| setUpScenery1 | Kruskal |  |
| setUpScenery1 | Prim |  |
| setUpScenery1 | BFS |  |
| setUpScenery1 | DFS |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clase | Metodo | Escenario | Valores de entrada | Resultado |
| Dijkstra | testShortestPath() | setUpScenery1 | int[][] matrix ={{0, 3, 20, 3, 0},  {3, 0, 0, 0, 0},  {20, 0, 0, 2, 3},  {3, 0, 2, 0, 0},  {0, 0, 3, 0, 0}, }; int nodes=3; int source=1; | Verifys that the algorithm travels the minimum distance of each vertex passing through all the edges |
| Dijkstra | testShortestPath1() | setUpScenery1 | int[][] matrix ={ {1,2,3},  {4,5,6},  {7,8,9}}; int nodes=3; int source=1; | Verifys that the algorithm travels the minimum distance of each vertex passing through all the edges |
| FloydW | testCalculate() | setUpScenery1 | dis = new int[6][6]; int[][] validate = { {0 , 5 , 3 , 4 , 18 , 19}, {5 , 0 , 2 , 1 , 13 , 14}, {3 , 2 , 0 , 1 , 15 , 16}, {4 , 1 , 1 , 0 , 14 , 15}, {18 , 13 , 15 , 14 , 0 , 1}, {19 , 14 , 16 , 15 , 1 , 0} }; | Calculate the minimum distance |
| Kruskal | kruskalAlgorithmTest1() | setUpScenery1 | matrix = new int[][] {  { 0, 2, 0, 6, 0 },  { 2, 0, 3, 8, 5 },  { 0, 3, 0, 0, 7 },  { 6, 8, 0, 0, 9 },  { 0, 5, 7, 9, 0 },  }; | Finds a minimum spanning forest of an undirected edge-weighted graph. |
| Kruskal | kruskalAlgorithmTest2() | setUpScenery2 | graph = new int[][] {  {0, 3, 20, 3, 0},  {3, 0, 0, 0, 0},  {20, 0, 0, 2, 3},  {3, 0, 2, 0, 0},  {0, 0, 3, 0, 0},  }; | Finds a minimum spanning forest of an undirected edge-weighted graph. |
| Prim | primAlgorithmTest1() | setUpScenery1 | graph = new int[][] {  { 0, 2, 0, 6, 0 },  { 2, 0, 3, 8, 5 },  { 0, 3, 0, 0, 7 },  { 6, 8, 0, 0, 9 },  { 0, 5, 7, 9, 0 },  }; | Finds a minimum spanning tree for a weighted undirected graph |
| AdjacencyListTest | testAddNode() | setUpScenery1 | n = "node0"; v = 0; | Adds node to AdjacencyList |
| AdjacencyListTest | testAddEdge() | setUpScenery2 | n = "node0"; n = "node1"; | Adds an edge between 2 nodes in the adjacency list |
| AdjacencyListTest | testCountNodes() | setUpScenery3 | graph.addNode("node0", 0); graph.addNode("node1", 1); graph.addNode("node2", 2); graph.addNode("node3", 3); graph.addNode("node4", 4);  graph.addEdge("node0", "node1"); graph.addEdge("node0", "node4"); graph.addEdge("node1", "node3"); graph.addEdge("node2", "node4"); | Counts the nodes in the adjacency list |
| AdjacencyListTest | testCountEdges() | setUpScenery3 | graph.addNode("node0", 0); graph.addNode("node1", 1); graph.addNode("node2", 2); graph.addNode("node3", 3); graph.addNode("node4", 4);  graph.addEdge("node0", "node1"); graph.addEdge("node0", "node4"); graph.addEdge("node1", "node3"); graph.addEdge("node2", "node4"); | Counts the edges in the adjacency list |
| AdjacencyListTest | testHasNode() | setUpScenery3 | n = "node3"; | Checks if a node is empty |
| AdjacencyListTest | testHasEdge() | setUpScenery3 | n = "node1"; n = "node3"; | Checks if theres an edge between 2 nodes |
| AdjacencyListTest | testGetNode() | setUpScenery3 | v = 2; | Returns the node with value 2, should be "node2" |
| AdjacencyMatrixTest | testAddNode() | setUpScenery1 | n = "node0"; v = 0; | Adds node to AdjacencyMatrix |
| AdjacencyMatrixTest | testAddEdge() | setUpScenery2 | n = "node0"; n = "node1"; v = 2; | Adds an edge between 2 nodes in the adjacency Matrix and adds a weight to it |
| AdjacencyMatrixTest | testCountNodes() | setUpScenery3 | graph.addNode("node0", 0); graph.addNode("node1", 1); graph.addNode("node2", 2); graph.addNode("node3", 3); graph.addNode("node4", 4);  graph.addEdge("node0", "node1"); graph.addEdge("node0", "node4"); graph.addEdge("node1", "node3"); graph.addEdge("node2", "node4"); | Counts the nodes in the adjacency Matrix |
| AdjacencyMatrixTest | testCountEdges() | setUpScenery3 | graph.addNode("node0", 0); graph.addNode("node1", 1); graph.addNode("node2", 2); graph.addNode("node3", 3); graph.addNode("node4", 4);  graph.addEdge("node0", "node1"); graph.addEdge("node0", "node4"); graph.addEdge("node1", "node3"); graph.addEdge("node2", "node4"); | Counts the edges in the adjacency list |
| AdjacencyMatrixTest | testHasNode() | setUpScenery3 | n = "node3"; | Checks if a node is empty |
| AdjacencyMatrixTest | testHasEdge() | setUpScenery3 | n = "node1"; n = "node3"; | Checks if theres an edge between 2 nodes |
| AdjacencyMatrixTest | testGetNode() | setUpScenery3 | v = 2; | Returns the node with value 2, should be "node2" |
| AdjacencyMatrixTest | testGetMatriz() | setUpScenery3 | int[][] validate = { {0 , 5 , 0 , 0 , 1}, {5 , 0 , 0 , 3 , 0}, {0 , 0 , 0 , 0 , 2}, {0 , 3 , 0 , 0 , 0}, {1 , 0 , 2 , 0 , 0}, };  int[][] result = graph.getMatrix(); |  |
| BFSTest | test() | setupScenery1 | v1= 'A';  v2= 'B';  v3= 'C';  v4= 'D';  v5= 'E';  v6= 'F'; | The method should return output based on BFS |
| DFSTest | test() | setupScenery1 | v1= 'A';  v2= 'B';  v3= 'C';  v4= 'D';  v5= 'E';  v6= 'F'; | Searches a vertex troughout the tree withoutcycles |