Scenery configuration for graph methods:

|  |  |  |
| --- | --- | --- |
| Name | Class | Scenarios (Examples) |
| settingUpVertexes | TestsMatrixGraph | *Initializes a graph using the adjacency matrix implementation* |
| settingUpVertexes | TestsMatrixGraph | *Adds a new vertex with its id and data associated to it.* |
| settingUpVertexes | TestsMatrixGraph | *Tries to add a new vertex that shares the id with an already existing vertex.* |
| settingUpVertexes | TestsMatrixGraph | *Tries to add a new vertex that contains null data associated to it.* |
| settingUpVertexes | TestsMatrixGraph | *Tries to add a new vertex that shares the same data with an already existing vertex* |
| settingUpEdges | TestMatrixGraph | *Initializes several vertexes in the graph and creates and edge between them.* |
| settingUpEdges | TestMatrixGraph | *Tries to create an edge between and existing and a nonexistent vertex.* |
| settingUpEdges | TestMatrixGraph | *Tries to create an edge between two existing edges in the graph, but with negative weight.* |
| settingUpEdges | TestMatrixGraph | *Tries to create an edge between one vertex and itself using a valid weight.* |
| setUpForDijkstra | TestMatrixGraph | *Initializes a graph with several vertexes that do not have any existing connection between them* |
| setUpForDijkstra1 | TestMatrixGraph | *Initializes a graph with several vertexes that do have an existing connection but it’s the same between them* |
| setUpForDijkstra1 | TestMatrixGraph | *Initializes a graph with several vertexes that do have an existing connection but it’s the same between them.* |
| setUpForDijkstra2 | TestMatrixGraph | *Initializes graph with several connections between them and multiple weights.* |
| settingUpForMatrix | TestMatrixGraph | *Initializes a graph that does not contain any graph, so the representative matrix is empty.* |
| settingUpForMatrix | TestMatrixGraph | *Initializes a graph that does contain vertexes and connections between them, so the matrix contains the information.* |
| settingUpForMatrix1 | TestMatrixGraph | *Initializes a graph that only contains a vertex pointing to nothing* |
| settingUpTrasversal | TestMatrixGraph | *Initializes a graph that contains multiple vertexes that contain multiple with different weights between them.* |
| settingUpTrasversal1 | TestMatrixGraph | *Initializes an empty graph with any path.* |
| settingUpTrasversal2 | TestMatrixGraph | *Initializes a graph with multiple vertexes but none of them are accessible between them.* |
| Setting List Graph | TestListGaph | *Initializes a graph implemented using adjacency lists and adds a vertex with (Key = A, Data = 1), then checks if the addition to the Hash table is done* |
| Setting Double Vertex | TestListGraph | *Initializes a graph implemented using adjacency lists and tries to add two vertexes that contain the same associated key. (Key =A, Data = 1) and Key =A, Data = 2), then checks that the second vertex was not added.* |
| Setting Multiple Vertexes | TestListGraph | *Initializes a graph implemented using adjacency lists and adds multiple vertexes containing non repeated data, ("A", 1); ("B", 2); ("C", 3); then verifies that they were added successfully. ("A", 1);("B", 2);*  *("C", 3l); then checks that the Hash contains them* |
| Setting Null Vertex | TestListGraph | *Initializes a graph implemented using adjacency lists and adds multiple vertexes containing non repeated data, ("A", 1); ("B", 2); ("C", 3); then verifies that they were added successfully. ("A", 1);("B", 2);*  *("C", null); then checks that the third vertex was not added.* |
| Setting Valid Edges | TestListGraph | *Initializes a graph implemented using adjacency lists, creates multiple vertexes with multiple data and creates edges containing a valid* |
| Setting Edge – Not Source | TestListGraph | *Initializes a graph implemented using adjacency lists, creates one Vertex. Then, verifies that an error is produced when trying to add an edge from a non-existent source to the created Vertex.* |
| Setting Edge - No Destination | TestListGraph | *Initializes a graph implemented using adjacency lists, creates one Vertex. Then, verifies that an error is produced when trying to add an edge from the created vertex to a non-existent destination.* |
| BFS - Linear Graph | TestListGraph | *Initializes a branched graph implemented using adjacency list, creates multiple vertexes and multiple edges with an assigned weight. Then verifies the result of the DFS* |
| BFS – Branched Graph | TestListGraph | *Initializes a branched graph implemented using adjacency list, creates a couple of vertexes and a couple of edges. Then verifies the result of the BFS* |
| BFS – Disconnected Graph | TestListGraph | *Initializes a disconnected graph implemented using adjacency list, creates a couple of vertexes and a couple of edges. Then verifies the result of the BFS* |
| DFS – Simple Graph | TestListGraph | *Initializes a simple graph implemented using adjacency list, creates multiple vertexes and multiple edges with an assigned weight. Then verifies the result of the DFS* |
| DFS – Branched Graph (With Weights) | TestListGraph | *Initializes a branched graph implemented using adjacency list, creates multiple vertexes and multiple edges with an assigned weight. Then verifies the result of the DFS.* |
| Dijkstra – Shortest Path Exists | TestListGraph | *Initializes a graph implemented using adjacency list, creates multiple vertexes and multiple edges with an assigned weight. Then verifies the shortest path from the source to the destination.* |
| Dijkstra – No Path Exists | TestListGraph | *Initializes a graph implemented using adjacency list, creates multiple vertexes and multiple edges with an assigned weight, there is no valid path from the source to the destination. Then verifies the result of the Dijkstra.* |
| Dijkstra – Disconnected Vertices | TestListGraph | *Initializes a graph implemented using adjacency list, creates a disconnected couple of vertexes and a couple of edges with an assigned weight. Then verifies the result of the Dijkstra.* |
| Get Adjacency list – Linear Graph | TestListGraph | *Initializes a linear graph implemented using adjacency list, creates couple of vertexes and couple of edges with an assigned weight. Then verifies that the adjacency list is the corresponding one* |
| Get Adjacency list – Branched Graph | TestListGraph | *Initializes a branched graph implemented using adjacency list, creates multiple vertexes and multiple edges with an assigned weight. Then verifies that the adjacency list is the corresponding one* |
| Get Adjacency list – Disconnected Graph | TestListGraph | *Initializes a disconnected graph implemented using adjacency list, creates multiple disconnected vertexes. Then verifies that isn’t an adjacency list for them.* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Scenery | Class | Method | Inputs | Result |
| Setting List Graph | TestListGraph | addVertex() | Vertex[“A”, 1] | The adjacency list must be no null. |
| Setting double vertex | TestListGraph | addVertex() | Vertex1[“A”, 1], Vertex2[“A”, 2] | The adjacency list must be of size one. |
| Setting multiple vertexes | TestListGraph | ddVertex() | Vertex1[“A”, 1], Vertex2[“A”, 2], Vertex3[“A”, 3] | The adjacency list must be of size three. |
| Setting null vertex | TestListGraph | addVetex() | Vertex[“A”, 1], Vertex2[“B”, 2], Vertex3[“C”, 3] | When obtaining the vertex with key "C", a null value is obtained. |
| Setting Valid Edges | TestListGraph | addEdge() | Vertex[“A”, 1], Vertex2[“B”, 2],  Edge1[“A”, “B”, 1], Edge2[“B”, “A”, 1], |  |
| Setting Edge – Not Source | TestListGraph | addEdge() | Vertex[“B”, ].  Edge[“A”, “B”, 5] | An error message occurs when trying to connect the edge. |
| Setting Edge – No Destination | TestListGraph | addEdge() | Vertex[“A”, 1]  Edge[“A”, “B”, 5] | An error message occurs when trying to connect the edge. |
| BFS – Linear Graph | TestListGraph | bfs() | Vertex[“A”, 1], Vertex2[“B”, 2],  Vertex3[“C”, 3], Vertex4[“D”, 4],  Edge1[“A”, “B”, 1], Edge2[“B”, “C”, 1],  Edge3[“C”, “D”, 1]  startVertexKey = “A” | The message informing that all vertices have been visited is obtained. |
| BFS – Branched Graph | TestListGraph | bfs() | Vertex[“A”, 1], Vertex2[“B”, 2],  Vertex3[“C”, 3], Vertex4[“D”, 4], Vertex5[“E”, 5], Vertex6[“F”, 6]  Edge1[“A”, “B”, 1], Edge2[“B”, “C”, 2],  Edge3[“A”, “D”, 3], Edge4[“B”, “E”, 4], Edge5[“B”, “F”, 5]  startVertexKey = “A” | The message informing that all vertices have been visited is obtained. |
| BFS – Disconnected Graph | TestListGraph | bfs() | Vertex[“A”, 1], Vertex2[“B”, 2],  Vertex3[“C”, 3], Vertex4[“D”, 4]  Edge1[“A”, “B”, 1], Edge2[“C”, “D”, 1],  startVertexKey = “A” | The message is obtained informing that the vertices with key "A" and "B" have been visited, since they are the ones that can be accessed starting from "A". |
| DFS – Simple Graph | TestListGraph | dfs() | Vertex[“A”, 1], Vertex2[“B”, 2],  Vertex3[“C”, 3], Vertex4[“D”, 4]  Edge1[“A”, “B”, 1], Edge2[“B”, “C”, 1],  Edge3[“C”, “D”, 1]  startVertexKey = “A” | The message informing that all vertices have been visited is obtained. |
| DFS – Branched Graph | TestListGraph | dfs() | Vertex[“A”, 1], Vertex2[“B”, 2],  Vertex3[“C”, 3], Vertex4[“D”, 4], Vertex5[“E”, 5], Vertex6[“F”, 6]  Edge1[“A”, “B”, 1], Edge2[“A”, “C”, 2],  Edge3[“A”, “D”, 3], Edge4[“A”, “E”, 4], Edge5[“A”, “F”, 5]  Edge6[“B”, “C”, 2]  Edge7[“B”, “E”, 4], Edge8[“B”, “F”, 5]  startVertexKey = “A” | The message informing that all vertices have been visited is obtained. |
| DFS – Disconnnected Graph | TestListGraph | dfs() | Vertex[“A”, 1], Vertex2[“B”, 2],  Vertex3[“C”, 3], Vertex4[“D”, 4]  Edge1[“A”, “B”, 1], Edge2[“C”, “D”,2],  startVertexKey = “A” | The message is obtained informing that the vertices with key "A" and "B" have been visited, since they are the ones that can be accessed starting from "A". |
| Djisktra – Shortest Path Exist | TestListGraph | djikstra() | Vertex[“A”, 1], Vertex2[“B”, 2],  Vertex3[“C”, 3], Vertex4[“D”, 4], Vertex5[“E”, 5]  Edge1[“A”, “B”, 1], Edge2[“B”, “C”, 2],  Edge3[“C”, “D”, 3], Edge4[“D”, “E”, 4]  startVertexKey = “A”  destinationVertexKey = “E” | The message informing the shortest path from the start vertex to the destination vertex is obtained. |
| Dijkstra – No Path Exist | TestListGraph | djikstra() | Vertex[“A”, 1], Vertex2[“B”, 2],  Vertex3[“C”, 3], Vertex4[“D”, 4]  Edge1[“A”, “B”, 2], Edge2[“C”, “D”, 1]  startVertexKey = “A”  destinationVertexKey = “D” | You get the message informing that there is no path from the start vertex to the destination vertex. |
| Dijkstra – Disconnected Vertices | TestListGraph | djikstra() | Vertex[“A”, 1], Vertex2[“B”, 2],  Vertex3[“C”, 3], Vertex4[“D”, 4]  Edge1[“A”, “B”, 2], Edge2[“C”, “D”, 1]  startVertexKey = “A”  destinationVertexKey = “C” | You get the message informing that there is no path from the start vertex to the destination vertex |