**Configuration of GraphAdjacencyListTest scenarios**

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenery** |
| setupScenary1 | GraphAdjacencyListTest |  |
| setupScenary2 | GraphAdjacencyListTest |  |
| setupScenary3 | GraphAdjacencyListTest |  |
| setupScenary4 | GraphAdjacencyListTest |  |
| setupScenary5 | GraphAdjacencyListTest |  |
| setupScenary6 | GraphAdjacencyListTest |  |
| setupScenary7 | GraphAdjacencyListTest |  |
| setupScenary8 | GraphAdjacencyListTest |  |
| setupScenary9 | GraphAdjacencyListTest |  |

**Test Cases Design**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective:** Test the correct operation of the GraphAdjacencyList class. | | | | |
| **Class** | **Method** | **Scenery** | **Inputs** | **Result** |
| GraphAdjacencyList | addVertex | setupScenary1 | vertex =  “New York City” | A new vertex with “New York City” as value is added to graph |
| GraphAdjacencyList | addVertex | setupScenary1 | vertex1 =  “New York City”  vertex2 =  “Los Angeles” | Two new vertices with “New York City” and “Los Angeles” as values are added to graph |
| GraphAdjacencyList | addVertex | setupScenary2 | vertex =  “New York City” | Vertex already exists exception is obtained |
| GraphAdjacencyList | addEdge | setupScenary3 | source = “NewYork City”  destination = “Los Angeles”  weight = 1 | A new edge is added between the New York City and Los Angeles vertices. |
| GraphAdjacencyList | addEdge | setupScenary4 | source = “NewYork City”  destination = “Los Angeles”  weight = 3  source = “NewYork City”  destination = “Chicago”  weight = 2 | Two new edges are added, one between the New York City and Los Angeles vertices and the other between the New York City and Chicago vertices. |
| GraphAdjacencyList | addEdge | setupScenary5 | source = “NewYork City”  destination = “Los Angeles”  weight = 1 | Edge already exists exception is obtained |
| GraphAdjacencyList | removeVertex | setupScenary2 | vertex =  “New York City” | Remove a vertex from the graph |
| GraphAdjacencyList | removeVertex | setupScenary3 | vertex1 =  “New York City”  vertex2 =  “Los Angeles” | Remove two vertices from the graph |
| GraphAdjacencyList | removeVertex | setupScenary2 | vertex =  “New York City” | Exception for trying to remove a vertex that doesn't exist |
| GraphAdjacencyList | removeEdge | setupScenary5 | source = “NewYork City”  destination = “Los Angeles”  weight = 5 | Remove a edge from the graph |
| GraphAdjacencyList | removeEdge | setupScenary4 | source = “NewYork City”  destination = “Los Angeles”  weight = 5  source = “NewYork City”  destination = “Chicago”  weight = 2 | Remove two edges from the graph |
| GraphAdjacencyList | removeEdge | setupScenary2 | vertex =  “New York City” | Exception for trying to remove a vertex that doesn't exist |
| GraphAdjacencyList | BFS | setupScenary6 | source = “NewYork City”  destination = “Los Angeles”  weight = 5  source = “NewYork City”  destination = “Chicago”  weight = 2  source = “Los Angeles”  destination = “Denver”  weight = 1  source = “Chicago”  destination = “Denver”  weight = 5  source = “Denver”  destination = “Miami”  weight = 3 | Verify that the implementation of BFS in the graph produces the expected distance from "New York City" in a graph |
| GraphAdjacencyList | BFS | setupScenary2 | vertex =  “New York City” | Verify that the BFS implementation properly handles the case of searching from a non-existent vertex and throws the appropriate exception in that scenario. |
| GraphAdjacencyList | BFS | setupScenary7 | source = “NewYork City”  destination = “Los Angeles”  weight = 5  source = “NewYork City”  destination = “Chicago”  weight = 2  source = “Los Angeles”  destination = “Denver”  weight = 1  source = “Chicago”  destination = “Denver”  weight = 5  source = “Denver”  destination = “Miami”  weight = 3  source = “Miami”  destination = “New York City”  weight = 3 | Verify that the implementation of BFS in the graph produces the expected distance from "New York City" in an acyclic graph. |
| GraphAdjacencyList | DFS | setupScenary6 | source = “NewYork City”  destination = “Los Angeles”  weight = 5  source = “NewYork City”  destination = “Chicago”  weight = 2  source = “Los Angeles”  destination = “Denver”  weight = 1  source = “Chicago”  destination = “Denver”  weight = 5  source = “Denver”  destination = “Miami”  weight = 3 | Verify that the implementation of DFS in the graph produces the expected distance from "New York City" in a graph |
| GraphAdjacencyList | DFS | setupScenary2 | vertex =  “New York City” | Verify that the DFS implementation properly handles the case of searching from a non-existent vertex and throws the appropriate exception in that scenario. |
| GraphAdjacencyList | DFS | setupScenary7 | source = “NewYork City”  destination = “Los Angeles”  weight = 5  source = “NewYork City”  destination = “Chicago”  weight = 2  source = “Los Angeles”  destination = “Denver”  weight = 1  source = “Chicago”  destination = “Denver”  weight = 5  source = “Denver”  destination = “Miami”  weight = 3  source = “Miami”  destination = “New York City”  weight = 3 | Verify that the implementation of DFS in the graph produces the expected distance from "New York City" in an acyclic graph. |
| GraphAdjacencyList | dijkstra | setupScenary6 | source = “NewYork City”  destination = “Los Angeles”  weight = 5  source = “NewYork City”  destination = “Chicago”  weight = 2  source = “Los Angeles”  destination = “Denver”  weight = 1  source = “Chicago”  destination = “Denver”  weight = 5  source = “Denver”  destination = “Miami”  weight = 3 | Verify that the implementation of Dijkstra in the graph produces the expected distance from "New York City" in a graph |
| GraphAdjacencyList | dijkstra | setupScenary2 | vertex =  “New York City” | Verify that the Dijkstra implementation properly handles the case of searching from a non-existent vertex and throws the appropriate exception in that scenario. |
| GraphAdjacencyList | dijkstra | setupScenary7 | source = “NewYork City”  destination = “Los Angeles”  weight = 5  source = “NewYork City”  destination = “Chicago”  weight = 2  source = “Los Angeles”  destination = “Denver”  weight = 1  source = “Chicago”  destination = “Denver”  weight = 5  source = “Denver”  destination = “Miami”  weight = 3  source = “Miami”  destination = “New York City”  weight = 3 | Verify that the implementation of DFS in the graph produces the expected distance from "New York City" in an acyclic graph. |
| GraphAdjacencyList | floydWarshall | SetupScenary6 | source = “NewYork City”  destination = “Los Angeles”  weight = 5  source = “NewYork City”  destination = “Chicago”  weight = 2  source = “Los Angeles”  destination = “Denver”  weight = 1  source = “Chicago”  destination = “Denver”  weight = 5  source = “Denver”  destination = “Miami”  weight = 3 | Verify that the implementation of the Floyd-Warshall algorithm in the graph produces the correct previous vertex in the shortest path from "New York City" to "Miami" |
| GraphAdjacencyList | floydWarshall | SetupScenary8 | source = “NewYork City”  destination = “Los Angeles”  weight = 5  source = “NewYork City”  destination = “Chicago”  weight = 2  source = “Los Angeles”  destination = “Denver”  weight = 1  source = “Chicago”  destination = “Denver”  weight = 5  source = “New York City”  destination = “Miami”  weight = -10 | Verify that the implementation of the Floyd-Warshall algorithm in the graph with negative-weighted edges produces the correct previous vertex in the shortest path. |
| GraphAdjacencyList | floydWarshall | SetupScenary2 | vertex =  “New York City” | Verify that the implementation of the Floyd-Warshall algorithm properly handles the case of a graph with a single vertex and sets the value of the previous vertex as null |
| GraphAdjacencyList | prim | SetupScenary9 | source = “NewYork City”  destination = “Los Angeles”  weight = 4  source = “NewYork City”  destination = “Chicago”  weight = 2  source = “Los Angeles”  destination = “Denver”  weight = 1  source = “Chicago”  destination = “Denver”  weight = 5  source = “Mami”  destination = “Los Angeles”  weight = 1 | The implementation of the Prim's algorithm in the graph finds a valid minimum spanning tree, where the selected vertices have distances equal to 'weight' |
| GraphAdjacencyList | prim | SetupScenary2 | vertex =  “New York City” | Verify that the Dijkstra implementation properly handles the case of searching from a non-existent vertex and throws the appropriate exception in that scenario. |
| GraphAdjacencyList | prim | SetupScenary10 | source = “NewYork City”  destination = “Los Angeles”  weight = -1  source = “NewYork City”  destination = “Chicago”  weight = 2  source = “Los Angeles”  destination = “Denver”  weight = 1  source = “Chicago”  destination = “Denver”  weight = 5  source = “Mami”  destination = “Los Angeles”  weight = -3 | Verify that the implementation of prim algorithm in the graph with negative-weighted edges produces the correct previous vertex in the shortest path |