|  |  |  |
| --- | --- | --- |
| Name | Class | Scenary |
| setupStage1 | Graph | An empty object of type graph is created |
| setupStage2 | Graph | A graph type object is created with three vertices and edges added |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Objective:** Verify the correct operation of the add method | | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| Graph | insert | setupScenary  1 | new Graph();  insert(vertex1: “Madrid”, vertex2: ”Barcelona” edge: 1000); | The vertices madrid and barcelona have been joined by an edge weighing 1000 |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Objective:** Verify that the method returns the minimum distance between one vertex and another | | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| Graph | dijkistra | setupScenary  1 | new Graph();  insert(vertex1: “Madrid”, vertex2: ”Barcelona” edge: 1000);  insert(vertex1: “Madrid”, vertex2: ”Valencia” edge: 800);  insert(vertex1: “Barcelona”, vertex2: ”Valencia” edge: 800);    dijkistra(“Barcelona”,”Valencia); | the minimum distance from barcelona to valencia is 800 |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Objective:** Verify that the method returns the minimum distance between one vertex and another | | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| Graph | floydWarshall | setupScenary  1 | new Graph();  insert(vertex1: “Madrid”, vertex2: ”Barcelona” edge: 1000);  insert(vertex1: “Madrid”, vertex2: ”Valencia” edge: 800);  insert(vertex1: “Barcelona”, vertex2: ”Valencia” edge: 800);    floydWarshall(int graph); | the minimum distance is 1600 |
|  |  |  |  |  |
|  |  |  |  |  |
| **Test Objective:** Verify that the method returns the minimum distance between one vertex and another | | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| Graph | bfs | setupScenary  1 | new Graph();  insert(vertex1: “Madrid”, vertex2: ”Barcelona” edge: 1000);  insert(vertex1: “Madrid”, vertex2: ”Valencia” edge: 800);  insert(vertex1: “Barcelona”, vertex2: ”Valencia” edge: 800);    bfs(“Madrid”); | From Madrid, the entire graph was traversed |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Objective:** Verify that the method returns the minimum distance between one vertex and another | | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| Graph | dfs | setupScenary  1 | new Graph();  insert(vertex1: “Madrid”, vertex2: ”Barcelona” edge: 1000);  insert(vertex1: “Madrid”, vertex2: ”Valencia” edge: 800);  insert(vertex1: “Barcelona”, vertex2: ”Valencia” edge: 800);    bfs(“Madrid”); | From Madrid, the entire graph was explored in depth |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Objective:** Verify that the algorithm travels the minimum distance of each vertex passing through all the edges | | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| Graph | primMST | setupScenary  1 | new Graph();  insert(vertex1: “Madrid”, vertex2: ”Barcelona” edge: 1000);  insert(vertex1: “Madrid”, vertex2: ”Valencia” edge: 800);  insert(vertex1: “Barcelona”, vertex2: ”Valencia” edge: 800);    primMST(“Barcelona”); | verify that the algorithm travels the minimum distance of each vertex passing through all the edges |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Objective:** Verify that the algorithm travels the minimum distance of each vertex passing through all the edges | | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| Graph | KruskalMST | setupScenary  1 | new Graph();  insert(vertex1: “Madrid”, vertex2: ”Barcelona” edge: 1000);  insert(vertex1: “Madrid”, vertex2: ”Valencia” edge: 800);  insert(vertex1: “Barcelona”, vertex2: ”Valencia” edge: 800);    primMST(“Barcelona”); | verify that the algorithm travels the minimum distance of each vertex passing through all the edges |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Objective:** Verify that the algorithm removes the desired vertex as well as the edge related to the vertex | | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| Graph | delete | setupScenary  1 | new Graph();  insert(vertex1: “Madrid”, vertex2: ”Barcelona” edge: 1000);  insert(vertex1: “Madrid”, vertex2: ”Valencia” edge: 800);  insert(vertex1: “Barcelona”, vertex2: ”Valencia” edge: 800);  delete(“Barcelona”); | The barcelona vertex and the edge that connected valencia to barcelona and madrid to barcelona have been eliminated |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Objective:** Check the correct operation of the setters and getters | | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| Graph | getters  &  setters | setupScenary  2 | get(“Barcelona”):  set(“Sevilla” | setters & getters work correctly |