**TEST DESIGN TAD**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 1** | **Objective**: Test that the program creates correctly different instances of the classes of model | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *Graph()* | Empty | Empty | An object of class Graph is created |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 2** | **Objective**: Test that the program creates correctly different instances of the classes of model | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| VertexTest | + *Vertex()* | Empty | obj | An object of class Vertex is created |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 3** | **Objective**: Test that the program creates correctly different instances of the classes of model | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| EdgeTest | + *Edge()* | Empty | v1,v2 | An object of class Edge is created |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 1** | **Objective**: Test that the program adds Vertices correctly | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *addVertex()* | Empty | obj | Vertex is added correctly. It is contained in Graph list |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 2** | **Objective**: Test that the program adds Vertices correctly | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *addVertex()* | Empty | obj=null | Vertex is added correctly. It is contained in Graph list |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 1** | **Objective**: Test that the program removes Vertices correctly | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *removeVertex ()* | Empty | obj | Vertex is removed correctly. It is not found on the adjacency list. |
| **Case N° 2** | **Objective**: Test that the program removes Vertices correctly | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *removeVertex ()* | Empty | obj=null | All vertices that contain a null object are removed. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 1** | **Objective**: Test that the program traverses the graph Correctly | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *breathFirstSearch ()* | Empty |  | It traverses and builds the bfs tree correctly |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 2** | **Objective**: Test that the program traverses the graph Correctly | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *breathFirstSearch ()* | Empty |  | Error. Cannot traverse empty graph. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 3** | **Objective**: Test that the algorithm returns as little path as possible | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *dijkstra ()* | Empty |  | Error. The minimum path cannot be found. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 4** | **Objective**: Test that the algorithm returns as little path as possible | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *dijkstra ()* | Empty |  | Find the minimum path possible correctly |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 5** | **Objective**: Test that the program traverses the graph Correctly | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *depthFirstSearch ()* | Empty |  | It traverses and builds the dfs tree correctly |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 6** | **Objective**: Test that the program traverses the graph Correctly | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *depthFirstSearch ()* | Empty |  | It traverses and builds the dfs tree correctly |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 7** | **Objective**: Test that the algorithm returns as little path as possible | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + floydWarshall *()* | Empty |  | Find the minimum path possible correctly |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 8** | **Objective**: Test that the algorithm returns as little path as possible | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + floydWarshall *()* | Empty |  | Error. The minimum path cannot be found. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 9** | **Objective**: Test that the algorithm returns as little path as possible | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *prim()* | Empty |  | Error. The minimum path cannot be found. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 10** | **Objective**: Test that the algorithm returns as little path as possible | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *prim()* | Empty |  | Find the minimum path possible correctly |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 11** | **Objective**: Test that the algorithm returns as little path as possible | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *kruskal ()* | Empty |  | Find the minimum path possible correctly |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case N° 12** | **Objective**: Test that the algorithm returns as little path as possible | | | |
| **Class** | **Method** | **Scenary** | **Input** | **Output** |
| GraphTest | + *kruskal ()* | Empty |  | Error. The minimum path cannot be found. |