**Test Cases Integrative Task 2**

**Carol**

**Samuel Navia Quiceno (A00405006)**

**Simón García (A00371828)**

**Test Cases for Model List of Adyacence structure:**

**Scenario configurations:**

| Name | Class | Scenario |
| --- | --- | --- |
| defaultTest() | ListGraphTest() | eventsList=new ArrayList<>();  mapLogic=new MapLogic(4,4); |

**Test Cases Design**

Test addVertex method

| Test’s objective: The objective is to evaluate and determine the limits to the add method, so that testing reacts in an expected way either when throwing an exception(when we break the limit of the method) or when we use a correct operation of this one. | | | | |
| --- | --- | --- | --- | --- |
| Class | Method | Scenario | Input | Expected Output |
| ListGraph | add() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4); Events event2= new Events(canvas,8,9,"number",4);  eventsList.add(event1);  eventsList.add(event2);  mapLogic.buildGraph(eventsList);  Vertex<Events> found=mapLogic.getGraphBFS().searchVertexValue(event1.getPosition());  int col=found.getValue().getCol(); | 7 |
| ListGraph | add() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,9,9,"number",4);  eventsList.add(event1);  eventsList.add(event2);  mapLogic.buildGraph(eventsList);  Vertex<Events> foundEvent2=mapLogic.getGraphBFS().searchVertexValue(event2.getPosition());  int foundCol=foundEvent2.getValue().getCol(); | 9 |
| ListGraph | add() | defaultTest() | Events event1= new Events(canvas,15,15,"number",4);  eventsList.add(event1);  mapLogic.buildGraph(eventsList);  Vertex<Events> foundEvent1=mapLogic.getGraphBFS().searchVertexValue(event1.getPosition());  int foundCol=foundEvent1.getValue().getCol(); | 15 |

Test addEdge method

| Test’s objective: The objective is to evaluate and determine the limits to the addEdge method, so that testing reacts in an expected way either when throwing an exception(when we break the limit of the method) and to verify the well connection between two vertex | | | | |
| --- | --- | --- | --- | --- |
| Class | Method | Scenario | Input | Expected Output |
| ListGraph | addEdge() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  eventsList.add(event1);  eventsList.add(event2);  mapLogic.buildGraph(eventsList);  mapLogic.getGraphBFS().addEdge(event1,event2,20);  mapLogic.getGraphBFS().addEdge(event2,event1,15);  Vertex<Events> foundEvent1=mapLogic.getGraphBFS().searchVertexValue(event1.getPosition());  Vertex<Events> foundEvent2=mapLogic.getGraphBFS().searchVertexValue(event2.getPosition());  Edge<Events> edge1=foundEvent1.searchEdge(foundEvent2);  int weight1=edge1.getWeight(); | 20 |
| ListGraph | addEdge() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  eventsList.add(event1);  mapLogic.buildGraph(eventsList);  mapLogic.getGraphBFS().addEdge(event1,event1,22); | “Loops are not allowed” |
| ListGraph | addEdge() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,0,0,null,0);  eventsList.add(event1);  mapLogic.buildGraph(eventsList);  mapLogic.getGraphBFS().addEdge(event1,event2,22); | “One or both vertex dont exists in the graph” |

Test bfs method

| Class | Method | Scenario | Input | Expected Output |
| --- | --- | --- | --- | --- |
| ListGraph | bfs() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  Events event3= new Events(canvas,9,9,"number",4);  eventsList.add(event1);  eventsList.add(event2);  eventsList.add(event3);  mapLogic.buildGraph(eventsList);  mapLogic.getGraphBFS().addEdge(event1,event2,10);  mapLogic.getGraphBFS().bFS(event1);  Vertex<Events> found=mapLogic.getGraphBFS().searchVertexValue(event1.getPosition()); | BLACK |
| ListGraph | bfs() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  Events event3= new Events(canvas,0,0,null,0);  eventsList.add(event1);  eventsList.add(event2);  mapLogic.buildGraph(eventsList);  mapLogic.getGraphBFS().addEdge(event1,event2,10);  mapLogic.getGraphBFS().bFS(event3); | “The vertex with the specified value was not found." |
| ListGraph | bfs() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  eventsList.add(event1);  eventsList.add(event2);  mapLogic.buildGraph(eventsList);  mapLogic.getGraphBFS().bFS(event1);  Vertex<Events> found=mapLogic.getGraphBFS().searchVertexValue(event2.getPosition()); | WHITE |

Test dijkstra method

| Test’s objective: The objective is to evaluate and determine the limits to the dijkstra method, when is applying to different situations which are planned about the limits of the method and to test the exceptions are working as expected | | | | |
| --- | --- | --- | --- | --- |
| Class | Method | Scenario | Input | Expected Output |
| ListGraph | dijkstra() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  Events event3= new Events(canvas,9,9,"number",4);  Events event4= new Events(canvas,10,10,"number",4);  eventsList.add(event1);  eventsList.add(event2);  eventsList.add(event3);  eventsList.add(event4);  mapLogic.buildGraph(eventsList);  mapLogic.getGraphBFS().addEdge(event1,event2,10);  mapLogic.getGraphBFS().addEdge(event2,event3,15);  mapLogic.getGraphBFS().addEdge(event3,event4,20);  mapLogic.getGraphBFS().addEdge(event2,event4,5);  mapLogic.getGraphBFS().dijkstra(event1);  Vertex<Events> found=mapLogic.getGraphBFS().searchVertexValue(event4.getPosition());  int searchVertexDistance=found.getDistance(); | 14 |
| ListGraph | dijkstra() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  Events event3= new Events(canvas,9,9,"number",4);  Events event4= new Events(canvas,10,10,"number",4);  eventsList.add(event1);  eventsList.add(event2);  eventsList.add(event3);  mapLogic.buildGraph(eventsList);  mapLogic.getGraphBFS().addEdge(event1,event2,10);  mapLogic.getGraphBFS().addEdge(event2,event3,15);  mapLogic.getGraphBFS().dijkstra(event4); | “The vertex with the specified value was not found." |
| ListGraph | dijkstra() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  eventsList.add(event1);  eventsList.add(event2);  mapLogic.buildGraph(eventsList);  mapLogic.getGraphBFS().dijkstra(event1);  Vertex<Events> found=mapLogic.getGraphBFS().searchVertexValue(event2.getPosition()); | null |

**Test Cases for Model Matrix of Adyacence structure:**

**Scenario configurations:**

| Name | Class | Scenario |
| --- | --- | --- |
| defaultTest() | MatrixGraphTest() | IGraph graph= new MatrixGraph(); |

**Test Cases Design**

Test addVertex method

| Test’s objective: The objective is to evaluate and determine the limits to the add method, so that testing reacts in an expected way either when throwing an exception(when we break the limit of the method) or when we use a correct operation of this one. | | | | |
| --- | --- | --- | --- | --- |
| Class | Method | Scenario | Input | Expected Output |
| MatrixGraph | add() | defaultTest() | graph.add(5) | 5 |
| MatrixGraph | add() | defaultTest() | graph.add(5)  graph.add(5) | null |
| MatrixGraph | add() | defaultTest() | graph.add(15) | 15 |

Test addEdge method

| Test’s objective: The objective is to evaluate and determine the limits to the addEdge method, so that testing reacts in an expected way either when throwing an exception(when we break the limit of the method) and to verify the well connection between two vertex | | | | |
| --- | --- | --- | --- | --- |
| Class | Method | Scenario | Input | Expected Output |
| MatrixGraph | addEdge() | defaultTest() | graph.add(5)  graph.add(7)  graph.addEdge(5,7,34) | (5,7,34) |
| MatrixGraph | addEdge() | defaultTest() | graph.add(5)  graph.add(5)  graph.addEdge(5,5,10) | “Loops are not allowed” |
| MatrixGraph | addEdge() | defaultTest() | graph.add(15)  graph.addEdge(15,12,34) | “One or both vertex dont exists in the graph” |

Test bfs method

| Class | Method | Scenario | Input | Expected Output |
| --- | --- | --- | --- | --- |
| MatrixGraph | bfs() | defaultTest() | graph.add(5)  graph.add(7)  graph.add(8)  graph.addEdge(5,7,10)  graph.addEdge(7,8,17)  graph.addEdge(5,8,15)  graph.bfs(5)  Vertex<Integer> vertexSearched=graph.searchValue(8)  vertexsearched.getPredecessor() | 5 |
| MatrixGraph | bfs() | defaultTest() | graph.addVertex(7)  graph.addVertex(8)  graph.addEdge(7,8,22)  graph.bfs(15) | “The vertex with the specified value was not found." |
| MatrixGraph | bfs() | defaultTest() | graph.add(5)  graph.add(7)  graph.bfs(5)  Vertex<Integer> vertexSearched=graph.searchValue(7)  vertexsearched.getColor() | WHITE |

Test dijkstra method

| Test’s objective: The objective is to evaluate and determine the limits to the dijkstra method, when is applying to different situations which are planned about the limits of the method and to test the exceptions are working as expected | | | | |
| --- | --- | --- | --- | --- |
| Class | Method | Scenario | Input | Expected Output |
| MatrixGraph | dijkstra() | defaultTest() | graph.add(5)  graph.add(7)  graph.add(8)  graph.add(9)  graph.addEdge(5,7,10)  graph.addEdge(7,8,17)  graph.addEdge(8,9,20)  graph.addEdge(7,9,5)  graph.dijkstra(5)  Vertex<Integer> vertexSearched=graph.searchValue(9)  vertexsearched.getDistance() | 15 |
| MatrixGraph | dijkstra() | defaultTest() | graph.addVertex(7)  graph.addVertex(8)  graph.addEdge(7,8,22)  graph.bfs(15) | “The vertex with the specified value was not found." |
| MatrixGraph | dijkstra() | defaultTest() | graph.add(5)  graph.add(7)  graph.bfs(5)  Vertex<Integer> vertexSearched=graph.searchValue(7)  vertexsearched.getPredecessor() | null |

**Test Cases for ListGraph structure:**

**Scenario configurations:**

| Name | Class | Scenario |
| --- | --- | --- |
| defaultTest() | ListGraphTest() | this.graph = new ListGraph(true,false,false); |

**Test Cases Design**

Test addVertex method

| Test’s objective: The objective is to evaluate and determine the limits to the add method, so that testing reacts in an expected way either when throwing an exception(when we break the limit of the method) or when we use a correct operation of this one. | | | | |
| --- | --- | --- | --- | --- |
| Class | Method | Scenario | Input | Expected Output |
| ListGraph | add() | defaultTest() | ;Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,9,"number",4);  graph.add(event1);  graph.add(event2); | 9 |
| ListGraph | add() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,7,7,"number",4);  graph.add(event1);  graph.add(event2); | 7 |
| ListGraph | add() | defaultTest() | Events event1= new Events(canvas,15,15,"number",4);  graph.add(event1); | 15 |

Test addEdge method

| Test’s objective: The objective is to evaluate and determine the limits to the addEdge method, so that testing reacts in an expected way either when throwing an exception(when we break the limit of the method) and to verify the well connection between two vertex | | | | |
| --- | --- | --- | --- | --- |
| Class | Method | Scenario | Input | Expected Output |
| ListGraph | addEdge() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  graph.add(event1);  graph.add(event2);  try{  graph.addEdge(event1,event2,35);  } catch (GraphException e){  e.printStackTrace();  } |  |
| ListGraph | addEdge() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  graph.add(event1);  //assert  String message="";  try{  graph.addEdge(event1,event1,10);  } catch (GraphException e){  message=e.getMessage();  } | “Loops are not allowed” |
| ListGraph | addEdge() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,0,0,null,0);  graph.add(event1);  //assert  String message="";  try{  graph.addEdge(event1,event2,34);  } catch (GraphException e){  message=e.getMessage();  } | “One or both vertex dont exists in the graph” |

Test bfs method

| Class | Method | Scenario | Input | Expected Output |
| --- | --- | --- | --- | --- |
| ListGraph | bfs() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  Events event3= new Events(canvas,9,9,"number",4);  graph.add(event1);  graph.add(event2);  graph.add(event3);  //Assert  VertexM<Events> vertex=graph.searchVertexValue(event2.getPosition());  String message = "";  try{  graph.addEdge(event1,event2,10);  graph.addEdge(event1,event3,15);  graph.addEdge(event2,event3,17);  graph.bFS(event1);  } catch (GraphException e){  message = e.getMessage();  } | BLACK |
| ListGraph | bfs() | defaultTest() | ;Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  Events event3= new Events(canvas,9,9,"number",4);  graph.add(event1);  graph.add(event2);  //assert  String message = "";  try {  graph.addEdge(event1,event2,10);  graph.addEdge(event1,event3,15);  graph.addEdge(event2,event3,17);  graph.bFS(event3);  } catch (GraphException e){  message = e.getMessage();  } | “One or both vertices do not exist in the graph.." |
| ListGraph | bfs() | defaultTest() | Events event1 = new Events(canvas, 7, 7, "number", 4);  Events event2 = new Events(canvas, 8, 8, "number", 4);  graph.add(event1);  graph.add(event2);  VertexM<Events> vertexL=graph.searchVertexValue(event2.getPosition());  //assert  try {  graph.bFS(event1);  }catch (GraphException e){  e.printStackTrace();  } | WHITE |

Test dijkstra method

| Test’s objective: The objective is to evaluate and determine the limits to the dijkstra method, when is applying to different situations which are planned about the limits of the method and to test the exceptions are working as expected | | | | |
| --- | --- | --- | --- | --- |
| Class | Method | Scenario | Input | Expected Output |
| ListGraph | dijkstra() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  Events event3= new Events(canvas,9,9,"number",4);  Events event4= new Events(canvas,10,10,"number",4);  graph.add(event1);  graph.add(event2);  graph.add(event3);  graph.add(event4);  //assert  VertexM<Events> vertexEvent2=graph.searchVertexValue(event2.getPosition());  try {  graph.addEdge(event1,event2,10);  graph.addEdge(event2,event3,17);  graph.addEdge(event3,event4,20);  graph.addEdge(event4,event2,5);  graph.dijkstra(event1);  }catch (GraphException e){  e.printStackTrace();  } | 10 |
| ListGraph | dijkstra() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  Events event3= new Events(canvas,9,9,"number",4);  Events event4= new Events(canvas,10,10,"number",4);  graph.add(event1);  graph.add(event2);  graph.add(event3);  graph.add(event4);  //assert  String message = "";  try {  graph.dijkstra(null);  }catch (GraphException e){  message = e.getMessage();  } | “A null event was searched.." |
| ListGraph | dijkstra() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  graph.add(event1);  graph.add(event2);  //assert  int predecessor=0;  VertexM<Events> vertexEvent2=graph.searchVertexValue(event2.getPosition());  VertexM<Events> vertexEvent1=graph.searchVertexValue(event1.getPosition());  try {  graph.addEdge(event1,event2,10);  graph.dijkstra(event1);  }catch (GraphException e){  e.printStackTrace();  }  if(vertexEvent2.getPredecessor().compareTo(vertexEvent1)==0){  predecessor=1;  } | 1 |

**Test Cases for MatrixGraph structure:**

**Scenario configurations:**

| Name | Class | Scenario |
| --- | --- | --- |
| defaultTest() | MatrixGraphTest() | this.graph = new MatrixGraph(50,true,false,false); |

**Test Cases Design**

Test addVertex method

| Test’s objective: The objective is to evaluate and determine the limits to the add method, so that testing reacts in an expected way either when throwing an exception(when we break the limit of the method) or when we use a correct operation of this one. | | | | |
| --- | --- | --- | --- | --- |
| Class | Method | Scenario | Input | Expected Output |
| ListGraph | add() | defaultTest() | ;Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,9,"number",4);  graph.add(event1);  graph.add(event2); | 9 |
| MatrixGraph | add() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,7,7,"number",4);  graph.add(event1);  graph.add(event2); | 7 |
| MatrixGraph | add() | defaultTest() | Events event1= new Events(canvas,15,15,"number",4);  graph.add(event1); | 15 |

Test addEdge method

| Test’s objective: The objective is to evaluate and determine the limits to the addEdge method, so that testing reacts in an expected way either when throwing an exception(when we break the limit of the method) and to verify the well connection between two vertex | | | | |
| --- | --- | --- | --- | --- |
| Class | Method | Scenario | Input | Expected Output |
| MatrixGraph | addEdge() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  graph.add(event1);  graph.add(event2);  try{  graph.addEdge(event1,event2,35);  } catch (GraphException e){  e.printStackTrace();  } |  |
| MatrixGraph | addEdge() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  graph.add(event1);  //assert  String message="";  try{  graph.addEdge(event1,event1,10);  } catch (GraphException e){  message=e.getMessage();  } | “Loops are not allowed” |
| MatrixGraph | addEdge() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,0,0,null,0);  graph.add(event1);  //assert  String message="";  try{  graph.addEdge(event1,event2,34);  } catch (GraphException e){  message=e.getMessage();  } | “One or both vertex dont exists in the graph” |

Test bfs method

| Class | Method | Scenario | Input | Expected Output |
| --- | --- | --- | --- | --- |
| MatrixGraph | bfs() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  Events event3= new Events(canvas,9,9,"number",4);  graph.add(event1);  graph.add(event2);  graph.add(event3);  //Assert  VertexM<Events> vertex=graph.searchVertexValue(event2.getPosition());  String message = "";  try{  graph.addEdge(event1,event2,10);  graph.addEdge(event1,event3,15);  graph.addEdge(event2,event3,17);  graph.bFS(event1);  } catch (GraphException e){  message = e.getMessage();  } | BLACK |
| MatrixGraph | bfs() | defaultTest() | ;Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  Events event3= new Events(canvas,9,9,"number",4);  graph.add(event1);  graph.add(event2);  //assert  String message = "";  try {  graph.addEdge(event1,event2,10);  graph.addEdge(event1,event3,15);  graph.addEdge(event2,event3,17);  graph.bFS(event3);  } catch (GraphException e){  message = e.getMessage();  } | “One or both vertices do not exist in the graph.." |
| MatrixGraph | bfs() | defaultTest() | Events event1 = new Events(canvas, 7, 7, "number", 4);  Events event2 = new Events(canvas, 8, 8, "number", 4);  graph.add(event1);  graph.add(event2);  VertexM<Events> vertexL=graph.searchVertexValue(event2.getPosition());  //assert  try {  graph.bFS(event1);  }catch (GraphException e){  e.printStackTrace();  } | WHITE |

Test dijkstra method

| Test’s objective: The objective is to evaluate and determine the limits to the dijkstra method, when is applying to different situations which are planned about the limits of the method and to test the exceptions are working as expected | | | | |
| --- | --- | --- | --- | --- |
| Class | Method | Scenario | Input | Expected Output |
| MatrixGraph | dijkstra() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  Events event3= new Events(canvas,9,9,"number",4);  Events event4= new Events(canvas,10,10,"number",4);  graph.add(event1);  graph.add(event2);  graph.add(event3);  graph.add(event4);  //assert  VertexM<Events> vertexEvent2=graph.searchVertexValue(event2.getPosition());  try {  graph.addEdge(event1,event2,10);  graph.addEdge(event2,event3,17);  graph.addEdge(event3,event4,20);  graph.addEdge(event4,event2,5);  graph.dijkstra(event1);  }catch (GraphException e){  e.printStackTrace();  } | 10 |
| MatrixGraph | dijkstra() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  Events event3= new Events(canvas,9,9,"number",4);  Events event4= new Events(canvas,10,10,"number",4);  graph.add(event1);  graph.add(event2);  graph.add(event3);  graph.add(event4);  //assert  String message = "";  try {  graph.dijkstra(null);  }catch (GraphException e){  message = e.getMessage();  } | “A null event was searched.." |
| MatrixGraph | dijkstra() | defaultTest() | Events event1= new Events(canvas,7,7,"number",4);  Events event2= new Events(canvas,8,8,"number",4);  graph.add(event1);  graph.add(event2);  //assert  int predecessor=0;  VertexM<Events> vertexEvent2=graph.searchVertexValue(event2.getPosition());  VertexM<Events> vertexEvent1=graph.searchVertexValue(event1.getPosition());  try {  graph.addEdge(event1,event2,10);  graph.dijkstra(event1);  }catch (GraphException e){  e.printStackTrace();  }  if(vertexEvent2.getPredecessor().compareTo(vertexEvent1)==0){  predecessor=1;  } | 1 |