**Diseño de Pruebas Unitarias**

**Escenarios:**

|  |  |  |
| --- | --- | --- |
| Nombre | Clase | Método |
| setUpStage1 | AdjListGraphTest | new AdjListGraph<String>(false) |
| setUpStage2 | AdjListGraphTest | new AdjListGraph<String>(false)  addVertex(“Pueblo Paleta”)  addVertex(“Ciudad Verde”)  addVertex(“Ciudad Plateada”) |
| setUpStage3 | AdjListGraphTest | new AdjListGraph<String>(false)  addVertex(“Pueblo Paleta”)  addVertex(“Ciudad Verde”)  addVertex(“Ciudad Plateada”)  alg.addEdge("Pueblo Paleta", "Ciudad Verde", 8)  alg.addEdge("Ciudad Verde", "Ciudad Plateada", 15) |

**Pruebas:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Entrada** | **Salida** |
| **AdjListGraphTest** | addVertexTest() | setUpStage1 | "Pueblo Paleta" | Correcto, porque "Pueblo Paleta" es el primer vértice del grafo. |
| **AdjListGraphTest** | addEdgeTest1() | setUpStage2 | ("Pueblo Paleta", "Ciudad Verde")  ("Ciudad Verde", "Ciudad Plateada") | Correcto, porque se las aristas que se agregan en la prueba conectan, "Pueblo Paleta" con "Ciudad Verde" y "Ciudad Verde" con "Ciudad Plateada" |
| **AdjListGraphTest** | addEdgeTest2() | setUpStage2 | ("Pueblo Paleta", "Ciudad Verde")  ("Ciudad Verde", "Ciudad Plateada") | Correcto, porque se la arista que se agrega en la prueba conecta "Pueblo Paleta" con "Ciudad Verde" y "Ciudad Verde", "Ciudad Plateada" no son adyacentes. |
| **AdjListGraphTest** | deleteVertexTest() | setUpStage2(); | ("Pueblo Paleta") | Correcto, porque en la prueba se elimina un vértice del escenario que al buscarlo obviamente retorna que no existe |
| **AdjListGraphTest** | deleteEdgeTest() | setUpStage3(); | ("Pueblo Paleta", "Ciudad Verde") | Correcto, porque en la prueba se elimina la arista entre las entradas y al verificar la matriz de pesos es igual al infinito al no existir conexión no existe peso. |
| **AdjListGraphTest** | searchTest1() | setUpStage1 | 0 | Correcto, porque la lista de adyacencia está vacía. |
| **AdjListGraphTest** | searchTest2() | setUpStage2() | ("Pueblo Paleta", (0))  ("Ciudad Plateada")  ("Hola") | Correcto, porque el primer vértice de la lista de adyacencia es "Pueblo Paleta" ya que es el primero agregado, “Ciudad Plateada” si se encuentra en el grafo y la última búsqueda es nula ya que no existe (“Hola”) |
| **AdjListGraphTest** | weightMatrixTest() | setUpStage3() | ("Pueblo Paleta", (0))  ("Ciudad Plateada")  ("Hola") | Correcto porque los vértices agregados representan una posición en la matriz, y esa matriz tienen el peso específico de las aristas agregadas como también las posiciones restantes no tendrán un peso específico entonces será infinito. |
| **AdjListGraphTest** | getIndexVTest() | setUpStage2() | (0, "Pueblo Paleta")  (1,"Ciudad Plateada") | Correcto, porque el primer vértice en el grafo es “Pueblo Paleta” entonces su índice será 0 como también el segundo vértice “Ciudad Plateada” será índice 1. |
| **AdjListGraphTest** | getVertexTest() | setUpStage2() | (3, alg.getVertex())  ("Ciudad Celeste")  (4, alg.getVertex()) | Correcto, porque en primera instancia el tamaño del grafo es igual a 3 vértices y después de agregar uno mas en el test completa la cantidad (4). |

**Escenarios:**

|  |  |  |
| --- | --- | --- |
| Nombre | Clase | Método |
| setUpStage1 | AdjMatrixGraphTest | new AdjListGraph<String>(false) |
| setUpStage2 | AdjMatrixGraphTest | new AdjListGraph<String>(false)  addVertex(“Pueblo Paleta”)  addVertex(“Ciudad Verde”)  addVertex(“Ciudad Plateada”) |
| setUpStage3 | AdjMatrixGraphTest | new AdjListGraph<String>(false)  addVertex(“Pueblo Paleta”)  addVertex(“Ciudad Verde”)  addVertex(“Ciudad Plateada”)  alg.addEdge("Pueblo Paleta", "Ciudad Verde", 8)  alg.addEdge("Ciudad Verde", "Ciudad Plateada", 15) |

**Pruebas:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre** | **Método** | **Escenario** | **Entrada** | **Salida** |
| AdjMatrixGraphTest | addVertexTest() | setUpStage1 | "Pueblo Paleta" | Correcto, porque "Pueblo Paleta" es el primer vértice del grafo y el primer agregado. |
| AdjMatrixGraphTest | addEdgeTest1() | setUpStage2 | ("Pueblo Paleta", "Ciudad Verde")  ("Ciudad Verde", "Ciudad Plateada") | Correcto, porque se las aristas que se agregan en la prueba conectan, "Pueblo Paleta" con "Ciudad Verde" y "Ciudad Verde" con "Ciudad Plateada" |
| AdjMatrixGraphTest | addEdgeTest2() | setUpStage2 | ("Pueblo Paleta", "Ciudad Verde")  ("Ciudad Verde", "Ciudad Plateada") | Correcto, porque se la arista que se agrega en la prueba conecta "Pueblo Paleta" con "Ciudad Verde" y "Ciudad Verde", "Ciudad Plateada" no son adyacentes. |
| AdjMatrixGraphTest | deleteVertexTest() | setUpStage2(); | ("Pueblo Paleta") |  |
| AdjMatrixGraphTest | deleteEdgeTest() | setUpStage3(); | ("Pueblo Paleta", "Ciudad Verde") | Correcto, porque en la prueba se elimina la arista entre las entradas y al verificar la matriz de pesos es igual al infinito al no existir conexión no existe peso. |
| AdjMatrixGraphTest | searchTest1() | setUpStage1 | 0 | Correcto, porque la lista de adyacencia está vacía. |
| AdjMatrixGraphTest | searchTest2() | setUpStage2() | ("Pueblo Paleta", (0))  ("Ciudad Plateada")  ("Hola") | Correcto, porque el primer vértice de la lista de adyacencia es "Pueblo Paleta" ya que es el primero agregado, “Ciudad Plateada” si se encuentra en el grafo y la última búsqueda es nula ya que no existe (“Hola”) |
| AdjMatrixGraphTest | weightMatrixTest() | setUpStage3() | ("Pueblo Paleta", (0))  ("Ciudad Plateada")  ("Hola") | Correcto porque los vértices agregados representan una posición en la matriz, y esa matriz tienen el peso específico de las aristas agregadas como también las posiciones restantes no tendrán un peso específico entonces será infinito. |
| AdjMatrixGraphTest | getIndexVTest() | setUpStage2() | (0, "Pueblo Paleta")  (1,"Ciudad Plateada") | Correcto, porque el primer vértice en el grafo es “Pueblo Paleta” entonces su índice será 0 como también el segundo vértice “Ciudad Plateada” será índice 1. |
| AdjMatrixGraphTest | getVertexTest() | setUpStage2() | (3, alg.getVertex())  ("Ciudad Celeste")  (4, alg.getVertex()) | Correcto, porque en primera instancia el tamaño del grafo es igual a 3 vértices y después de agregar uno mas en el test completa la cantidad (4). |

**Escenarios:**

|  |  |  |
| --- | --- | --- |
| Nombre | Clase | Método |
| setUpStage1 | GraphAlgorithmsAdjMatrixTest | new AdjMatrixGraph<String>(false)  ("Pueblo Paleta")  ("Ciudad Verde")  ("Ciudad Plateada")  ("Ciudad Celeste")  ("Pueblo Paleta", "Ciudad Verde", 8)  ("Ciudad Verde", "Ciudad Plateada", 15)  ("Pueblo Paleta", "Ciudad Celeste", 5)  ("Ciudad Celeste", "Ciudad Plateada", 6) |
| setUpStage2 | GraphAlgorithmsAdjMatrixTest | new AdjMatrixGraph<String>(false)  ("A")  ("B")  ("C")  ("D")  ("A", "B", 8)  ("A", "C", 5)  A", "D", 3)  ("B", "D", 6)  (C", "D", 1) |

**Pruebas:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre** | **Método** | **Escenario** | **Entrada** | **Salida** |
| GraphAlgorithmsAdjMatrixTest | bfsTest1() | setUpStage1 | ("Pueblo Paleta", b.get(0))  ("Ciudad Verde", b.get(1))  ("Ciudad Celeste", b.get(2))  ("Ciudad Plateada", b.get(3))  ("Ciudad Plateada", b.get(0))  ("Ciudad Verde", b.get(1))  ("Ciudad Celeste", b.get(2))  ("Pueblo Paleta", b.get(3)); | Correcto, los recorridos resultantes desde el vértice origen pasado por parámetro al método son iguales a los esperados en la prueba |
| GraphAlgorithmsAdjMatrixTest | bfsTest2() | setUpStage2 | ("A", b.get(0))  ("B", b.get(1))  ("C", b.get(2))  ("D", b.get(3))  ("D", b.get(0))  ("A", b.get(1))  ("B", b.get(2))  ("C", b.get(3)) | Correcto, los recorridos resultantes desde el vértice origen pasado por parámetro al método son iguales a los esperados en la prueba |
| GraphAlgorithmsAdjMatrixTest | dijkstraTest2() | setUpStage2 | (0, cost[0])  (8, cost[1])  (4, cost[2])  (3, cost[3])  (3,GraphAlgorithms.getPath()[2])  (3, cost[0])  (6, cost[1])  (1, cost[2])  (0, cost[3])  (3,GraphAlgorithms.getPath()[2]) |  |
| GraphAlgorithmsAdjMatrixTest | primTest1() | setUpStage1 | (19,GraphAlgorithms.prim("Pueblo Paleta",graph))  (13,GraphAlgorithms.prim("Ciudad Verde",graph))  (19,GraphAlgorithms.prim("Ciudad Celeste", graph))  (19,GraphAlgorithms.prim("Ciudad Plateada", graph)) |  |
| GraphAlgorithmsAdjMatrixTest | primTest2() | setUpStage2 | (10, raphAlgorithms.prim("A", graph))(7,GraphAlgorithms.prim("B",graph))  (10, GraphAlgorithms.prim("C", graph))  (10, GraphAlgorithms.prim("D", graph)) |  |
| GraphAlgorithmsAdjMatrixTest | kruskalTest1() | setUpStage1 | (e.getSource(),graph.search(graph.getIndexV("Pueblo Paleta")))  (e.getEnd(),graph.search(graph.getIndexV("Ciudad Verde")))  (e.getWeight(), 0); |  |

**Escenarios:**

|  |  |  |
| --- | --- | --- |
| Nombre | Clase | Método |
| setUpStage1 | GraphAlgorithmsAdjListTest | new AdjListGraph<String>(false)  ("Pueblo Paleta")  ("Ciudad Verde")  ("Ciudad Plateada")  ("Ciudad Celeste")  ("Pueblo Paleta", "Ciudad Verde", 8)  ("Ciudad Verde", "Ciudad Plateada", 15)  ("Pueblo Paleta", "Ciudad Celeste", 5)  ("Ciudad Celeste", "Ciudad Plateada", 6) |
| setUpStage2 | GraphAlgorithmsAdjListTest | new AdjListGraph<String>(false)  ("A")  ("B")  ("C")  ("D")  ("A", "B", 8)  ("A", "C", 5)  A", "D", 3)  ("B", "D", 6)  (C", "D", 1) |

**Pruebas:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre** | **Método** | **Escenario** | **Entrada** | **Salida** |
| GraphAlgorithmsAdjListTest | bfsTest1() | setUpStage1 | ("Pueblo Paleta", b.get(0))  ("Ciudad Verde", b.get(1))  ("Ciudad Celeste", b.get(2))  ("Ciudad Plateada", b.get(3))  ("Ciudad Plateada", b.get(0))  ("Ciudad Verde", b.get(1))  ("Ciudad Celeste", b.get(2))  ("Pueblo Paleta", b.get(3)); | Correcto, los recorridos resultantes desde el vértice origen pasado por parámetro al método son iguales a los esperados en la prueba |
| GraphAlgorithmsAdjListTest | bfsTest2() | setUpStage2 | ("A", b.get(0))  ("B", b.get(1))  ("C", b.get(2))  ("D", b.get(3))  ("D", b.get(0))  ("A", b.get(1))  ("B", b.get(2))  ("C", b.get(3)) | Correcto, los recorridos resultantes desde el vértice origen pasado por parámetro al método son iguales a los esperados en la prueba |
| GraphAlgorithmsAdjListTest | dijkstraTest2() | setUpStage2 | (0, cost[0])  (8, cost[1])  (4, cost[2])  (3, cost[3])  (3,GraphAlgorithms.getPath()[2])  (3, cost[0])  (6, cost[1])  (1, cost[2])  (0, cost[3])  (3,GraphAlgorithms.getPath()[2]) |  |
| GraphAlgorithmsAdjListTest | primTest1() | setUpStage1 | (19,GraphAlgorithms.prim("Pueblo Paleta",graph))  (13,GraphAlgorithms.prim("Ciudad Verde",graph))  (19,GraphAlgorithms.prim("Ciudad Celeste", graph))  (19,GraphAlgorithms.prim("Ciudad Plateada", graph)) |  |
| GraphAlgorithmsAdjListTest | floydTest | setUpStage2 | (8, m[1][0])  (8, m[0][1])  (4, m[2][0])  (4, m[0][2])  (3, m[3][0])  (3, m[0][3])  (8, m[1][0])  (8, m[0][1])  (4, m[2][0])  (4, m[0][2])  (3, m[3][0])  (3, m[0][3]) | Correcto porque los pesos de la matriz resultante del floyWarshal son iguales a los esperados en la prueba |
| GraphAlgorithmsAdjListTest | primTest2() | setUpStage2 | (10, raphAlgorithms.prim("A", graph))(7,GraphAlgorithms.prim("B",graph))  (10, GraphAlgorithms.prim("C", graph))  (10, GraphAlgorithms.prim("D", graph)) |  |
| GraphAlgorithmsAdjListTest | kruskalTest1() | setUpStage1 | (e.getSource(),graph.search(graph.getIndexV("Pueblo Paleta")))  (e.getEnd(),graph.search(graph.getIndexV("Ciudad Verde")))  (e.getWeight(), 0); |  |