|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No. 1 | **Objetivo de la prueba:** verificar que el método addNode agrega los valores al grafo  **Firma del método:** public void addNode(E node) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| Graph | addNode(E node) | graph = new Graph<>(4); | m = “re” + I;  AddNode(m):  Donde m es el objeto a agregar al grafo.  Donde 0≤i<4; siendo 4 el número de vértices en el grafo | Agrega un vértice con las características del elemento pasado como parámetro |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No. 2 | **Objetivo de la prueba:** verificar que el método isEmpty verifica el estado del grafo  **Firma del método:** public void isEmpty() | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| GraphM | isEmpty () | graph = new Graph<> (4);  String m = “re” +0;  AddNode(m):  m = “re” +1  AddNode(m);  m = “re” +2  AddNode(m);  m = “re” +3  AddNode(m); | No cuenta con valores de entrada. | El estado actual del grafo, si el grafo se encuentra vacío el método debe de retorno false, en caso contrario debe retornar true. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No. 3 | **Objetivo de la prueba:** verificar que el método searchNodeM retorna elemento buscado.  **Firma del método:** public NodeM searchNodeM(T key) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| GraphM | searchNodeM(T key) | graph = new Graph<> (4);  String m = “re” +0;  AddNode(m):  m = “re” +1  AddNode(m);  m = “re” +2  AddNode(m);  m = “re” +3  AddNode(m); | El elemento a buscar en el grafo.  m= “re” +i  searchNodeM(m)  donde 0≤i<4 | Se busca la  Llave o el elemento (m = “re” + i) en  el grafo y  se verifica que  sea el mismo  objeto.  Si no encuentra  la llave,  retornará null.  Donde 0≤i<4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No. 4 | **Objetivo de la prueba:** verificar que el método deleteEdge elimina correctamente la arista indicada de un vértice.  **Firma del método:** public void deleteEdge(T key, T key2) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| GraphM | deleteEdge(T key, T key2) | graph = new Graph<> (4);  String m = “re” +0;  AddNode(m):  m = “re” +1  AddNode(m);  m = “re” +2  AddNode(m);  m = “re” +3  AddNode(m); | La llave de los vértices con la arista a eliminar  m= “re” + i  Donde 0≤i<4 | **DeleteEdge(“re1”,”re4”)**  Al buscar la adyacencia entre las dos arista se retornara null |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No. 5 | **Objetivo de la prueba:** comprobar que el método adjacentNodeM verifica la existencia de una arista entre dos vértices.  **Firma del método:** public boolean adjacentNodeM(T key, T key2) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| GraphM | adjacentNodeM(T key, T key2) | graph = new Graph<> (4);  String m = “re” +0;  AddNode(m):  m = “re” +1  AddNode(m);  m = “re” +2  AddNode(m);  m = “re” +3  AddNode(m); | La llave de los vértices  A verificar  m= “re” + i  Donde 0≤i<4 | **adjacentNodeM(“re1”,”re3”)**  Se busca la adyacencia entre dos vértices, si los dos elementos tiene una adyacencia se retornara true, en caso contrario false |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No. 6 | **Objetivo de la prueba:** verificar que el método getDistance retorna la distancia correcta entre dos vértices.  **Firma del método:** public double getDistance(T key, T key2) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| GraphM | getDistance(T key, T key2) | graph = new Graph<> (4);  String m = “re” +0;  AddNode(m):  m = “re” +1  AddNode(m);  m = “re” +2  AddNode(m);  m = “re” +3  AddNode(m); | La llave de los elementos a verificar  m= “re” +i  getDistance(T key, T key2) | Se busca la lleves de los dos vértices, y mediante una comparación se verifica que el método este retornando la distancia correcta.  **m = graph.getDistance ("re2", "re4");**  **m2 = graph.getDistance ("re2", "re4");**  Si la distancia es la misma se retornará true, en caso contrario false |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No. 7 | **Objetivo de la prueba:** Verificar que el método dijkstra encuentre el peso mínimo para llevar de un vértice a todos los demás.  **Firma del método:** public double[] dijkstra(T key) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| GraphM | dijkstra(T key) | graph = new Graph<> (4);  String m = “re” +0;  AddNode(m):  m = “re” +1  AddNode(m);  m = “re” +2  AddNode(m);  m = “re” +3  AddNode(m); | graph.dijkstra("re0")  “re0” es la  llave que  identifica al  objeto | Se busca las llaves que entran por parámetro, y mediante una comparación se verifica que el método este retornando el arreglo correcto.  **arr = graph.dijkstra("re0")**  **arr2 = graph.dijkstra("re0");**  Si los arreglos son iguales se retornará true, en caso contrario false |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No. 8 | **Objetivo de la prueba:** Verificar que el método BFS realiza el correcto recorrido por un grafo  **Firma del método:** public Tree<T> BFS(T origin) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| GraphM | BFS(T origin) | graph = new Graph<> (4);  String m = “re” +0;  AddNode(m):  m = “re” +1  AddNode(m);  m = “re” +2  AddNode(m);  m = “re” +3  AddNode(m); | **Tree<String> m = graph.BFS("re0");** | **Tree<String> m = graph.BFS("re0");**  Se comprueba que el recorrido realizado en el grafo que se generó a partir de la matriz, coincida con recorrido esperado. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No. 9 | **Objetivo de la prueba:** Verificar que el método DFS realiza el correcto recorrido por un grafo  **Firma del método:** public ArrayList<T> DFS(T origin) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| GraphM | DFS(T origin) | graph = new Graph<> (4);  String m = “re” +0;  AddNode(m):  m = “re” +1  AddNode(m);  m = “re” +2  AddNode(m);  m = “re” +3  AddNode(m); | **ArrayList<String> m = graph.DFS("re0")** | **Tree<String> m = graph.DFS("re0");**  Se comprueba que el recorrido realizado en el grafo que se generó a partir de la matriz, coincida con recorrido esperado. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No. 10 | **Objetivo de la prueba:** comprobar que el método floydWarshall genera la matriz con los pesos del recorrido mínimo entre cada uno de los vértices.    **Firma del método:** public double[][] floydWarshall() | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| GraphM | floydWarshall() | graph = new Graph<> (4);  String m = “re” +0;  AddNode(m):  m = “re” +1  AddNode(m);  m = “re” +2  AddNode(m);  m = “re” +3  AddNode(m); | double[][] m = graph.floydWarshall();  double[][] g = graph.floydWarshall(); | Se comprueba que la matriz m coincide con la matriz g, que contiene los valores generados por el método floydWarshall. |