|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 1** | **Objetivo de la prueba:** Comprobar que el metodo insertVertex funciona correctamente y agrega un vértice al grafo  **Firma del método:**  public boolean insertVertex(V vert) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| ListGraphTest | InsertVertex | ListGraph<Integer,Integer> graphL=new ListGraph<Integer,Integer>(true); | listGraph.addInsertVertex(1) | Se comprueba que el vértice fue agregado al grafo correctamente |

Casos de prueba

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba #2** | **Objetivo de la prueba:** Comprobar que el InsertEdge funciona correctamente.  **Firma del método:**  public boolean insertEdge(E egde, V vertice1, V vertice2) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| ListGraphTest | insertEdge | ListGraph<Integer,Integer> graphL=new ListGraph<Integer,Integer>(true);  listGraph | listGraph.insertEdge(4, 2,0 ) | Se comprueba que la arista fue agregada al grafo correctamente |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 3** | **Objetivo de la prueba:** Comprobar que el metodo thereIsEdge funciona correctamente  **Firma del método:** public boolean thereIsEdge(V vertice1, V vertice2) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| ListGraphTest | thereIsEdge | ListGraph<Integer,Integer> graphL=new ListGraph<Integer,Integer>(true);  listGraph.insertEdge(4, 2,0 ) | thereIsEdge(4, 2) | Se comprueba que existe una arista entre los grafos seleccionados |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 4** | **Objetivo de la prueba:** Comprobar que el metodo getAmountVertices funciona correctamente  **Firma del método:** public int getAmountVertices() | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| ListGraphTest | getAmountVertices | ListGraph<Integer,Integer> graphL=new ListGraph<Integer,Integer>(true);  listGraph.insertEdge(4, 2,0 ) | getAmountVertices() | Retorna la cantidad de vértices existentes en el grafo |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 5** | **Objetivo de la prueba:** Comprobar que el metodo getValues funciona correctamente  **Firma del método:** public ArrayList<V> getValues() | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| ListGraphTest | getValues | ListGraph<Integer,Integer> graphL=new ListGraph<Integer,Integer>(true);  listGraph.insertEdge(4, 2,0 ) | getValues() | Retorna un arraylist con los valores del grafo |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 6** | **Objetivo de la prueba:** Comprobar que el metodo thereIsVertex funciona correctamente  **Firma del método:**  public boolean thereIsVertex(V vertice) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| ListGraphTest | thereIsVertex | ListGraph<Integer,Integer> graphL=new ListGraph<Integer,Integer>(true);  listGraph.insertEdge(4, 2,0 ) | thereIsVertex(1) | Se comprueba que existe un vértice en específico dentro del grafo |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 7** | **Objetivo de la prueba:** Comprobar que el metodo insertVertex funciona correctamente y agrega un vértice al grafo  **Firma del método:**  public boolean insertVertex(V vert) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| MatrixGraphTest | InsertVertex | MatrixGraph<Integer,Integer> graphM=new MatrixGraph<Integer,Integer>(true); | listGraph.addInsertVertex(1) | Se comprueba que el vértice fue agregado al grafo correctamente |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba #8** | **Objetivo de la prueba:** Comprobar que el InsertEdge funciona correctamente.  **Firma del método:**  public boolean insertEdge(E egde, V vertice1, V vertice2) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| MatrixGraphTest | insertEdge | MatrixGraph<Integer,Integer> graphM=new MatrixGraph<Integer,Integer>(true);  MatrixGraph | listGraph.insertEdge(4, 2,0 ) | Se comprueba que la arista fue agregada al grafo correctamente |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 9** | **Objetivo de la prueba:** Comprobar que el metodo thereIsEdge funciona correctamente  **Firma del método:** public boolean thereIsEdge(V vertice1, V vertice2) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| MatrixGraphTest | thereIsEdge | MatrixGraph<Integer,Integer> graphM=new MatrixGraph<Integer,Integer>(true);  MatrixGraph.insertEdge(4, 2,0 )  ) | thereIsEdge(4, 2) | Se comprueba que existe una arista entre los grafos seleccionados |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 10** | **Objetivo de la prueba:** Comprobar que el metodo getAmountVertices funciona correctamente  **Firma del método:** public int getAmountVertices() | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| MatrixGraphTest | getAmountVertices | MatrixGraph<Integer,Integer> graphM=new MatrixGraph<Integer,Integer>(true);  MatrixGraph.insertEdge(4, 2,0 ) | getAmountVertices() | Retorna la cantidad de vértices existentes en el grafo |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 11** | **Objetivo de la prueba:** Comprobar que el metodo getValues funciona correctamente  **Firma del método:** public ArrayList<V> getValues() | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| MatrixGraphTest | getValues | MatrixGraph<Integer,Integer> graphM=new MatrixGraph<Integer,Integer>(true);  MatrixGraph.insertEdge(4, 2,0 ) | getValues() | Retorna un arraylist con los valores del grafo |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 12** | **Objetivo de la prueba:** Comprobar que el metodo thereIsVertex funciona correctamente  **Firma del método:**  public boolean thereIsVertex(V vertice) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| MatrixGraphTest | thereIsVertex | MatrixGraph<Integer,Integer> graphM=new MatrixGraph<Integer,Integer>(true);  MatrixGraph.insertEdge(4, 2,0 ) | thereIsVertex(1) | Se comprueba que existe un vértice en específico dentro del grafo |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 13** | **Objetivo de la prueba:** Comprobar que el metodo dfs funciona correctamente.  **Firma del método:**  public ListGraph<V,E> dfs(IGraph<V, E> graph) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| GraphAlgorithms | dfs | ListGraph<Integer,Integer> graphL=new ListGraph<Integer,Integer>(true);  GraphAlgorithms<String, Integer> meths2= new GraphAlgorithms<Integer,Integer>(); | ListGraph<Integer, Integer> aux=meths.dfs(graphL); | Se comprueba que el método realiza correctamente un recorrido por profundidad en el grafo |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba #14** | **Objetivo de la prueba:** Comprobar que el metodo bfs funciona correctamente.  **Firma del método:**  public ListGraph<V,E> bfs(IGraph<V, E> graph) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| GraphAlgorithms | bfs | ListGraph<Integer,Integer> graphL=new ListGraph<Integer,Integer>(true);  GraphAlgorithms<String, Integer> meths2= new GraphAlgorithms<Integer,Integer>();  ListGraph.insertEdge(4, 2,12 )  )  ListGraph.insertEdge(3, 2,15)  )  ListGraph.insertEdge(9, 2,22 )  ) | ListGraph<Integer, Integer> aux=meths.bfs(graphL);) | Se comprueba que el método realiza correctamente un recorrido por anchura en el grafo |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 15** | **Objetivo de la prueba:** Comprobar que el metodo floydWarshall funciona correctamente  **Firma del método:**  public HashMap<ExchangePair<V,V>,Double> floydWarshall(IGraph<V, E> graph) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| GraphAlgorithms | floydWarshall | ListGraph<Integer,Integer> graphL=new ListGraph<Integer,Integer>(true);  GraphAlgorithms<String, Integer> meths2= new GraphAlgorithms<Integer,Integer>();  ListGraph.insertEdge(4, 2,12 )  )  ListGraph.insertEdge(3, 2,15)  )  ListGraph.insertEdge(9, 2,22 )  ) | HashMap<ExchangePair<Integer,Integer>,Double> aux=meths.floydWarshall(graphL); | Se comprueba que devuelve el camino de menor peso desde cada vértice hasta cada uno de los que componen el grafo |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 16** | **Objetivo de la prueba:** Comprobar que el metodo prim funciona correctamente  **Firma del método:**  public ListGraph<V,E> prim(IGraph<V, E> graph ,V vertex) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| GraphAlgorithms | prim | ListGraph<Integer,Integer> graphL=new ListGraph<Integer,Integer>(true);  GraphAlgorithms<String, Integer> meths2= new GraphAlgorithms<Integer,Integer>();  ListGraph.insertEdge(4, 2,12 )  )  ListGraph.insertEdge(3, 2,15)  )  ListGraph.insertEdge(9, 2,22 )  ) | ListGraph<Integer, Integer> aux=meths. prim (graphL, 4);) | Retorna el árbol de expansión mínima |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba # 17** | **Objetivo de la prueba:** Comprobar que el metodo kruskal funciona correctamente  **Firma del método:**  public ListGraph<V,E> kruskal(IGraph<V, E> graph) | | | |
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **resultado** |
| GraphAlgorithms | kruskal | ListGraph<Integer,Integer> graphL=new ListGraph<Integer,Integer>(true);  GraphAlgorithms<String, Integer> meths2= new GraphAlgorithms<Integer,Integer>();  ListGraph.insertEdge(4, 2,12 )  )  ListGraph.insertEdge(3, 2,15)  )  ListGraph.insertEdge(9, 2,22 )  ) | ListGraph<Integer, Integer> aux=meths. prim (graphL, 4);) | Retorna el árbol de expansión mínima |