|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba No. 1** | **Objetivo de la prueba:**Comprobar que el método addEdge(K keySource, K keyDestiny, K keyEdge, **double** weight) añade una arista.    **Firma del método:**public void  addEdge(K keySource, K keyDestiny, K keyEdge, **double** weight) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| **AdjacencyListUndirected** | addEdge(K keySource, K keyDestiny, K keyEdge, **double** weight)) | **generalScenary()** | **keySource:** Llave del nodo desde se quiere iniciar la arista  **keyDestiny:** Llave de la dirección de la arista  KeyEdge: llave de la arista  **Weight:** El peso de la arista | Se agrega una arista entre el nodo con llave keySource y keyDestiny, como es un grafo no dirigido, el nodo con llave **keySource** debe aparecer en las adyacencias de **keyDestiny** y de forma contraria también. |
| **AdjacencyList** | addEdge(K keySource, K keyDestiny, K keyEdge, **double** weight)) | **generalScenary()** | **keySource:** Llave del nodo desde se quiere iniciar la arista  **keyDestiny:** Llave de la dirección de la arista  KeyEdge: llave de la arista  **Weight:** El peso de la arista | Se agrega una arista entre el nodo con llave **keySource** y el nodo con llave **keyDestiny**. Como el grafo es dirigido, solo deberá aparecer el nodo con la llave **keyDestiny** en las adyacencias de el nodo con llave **keySource,** no se debe dar la situación contraria. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba No. 2** | **Objetivo de la prueba:**Comprobar que el método addNode(Node<V, K> newNode)    **Firma del método: public** **void** addNode(Node<V, K> newNode) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| **AdjacencyListUndirected** | addNode(Node<V, K> newNode) | **generalScenary()** | **newNode**: El nodo que se creó, y se desea agregar al grafo. | Se agrega a la hash de nodos el nodo **newNode.** |
| **AdjacencyList** | addNode(Node<V, K> newNode) | **generalScenary()** | **newNode:** El nodo que se creó, y se desea agregar al grafo. | Se agrega a la hash de nodos el nodo **newNode.** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba No. 3** | **Objetivo de la prueba:**Comprobar que el método bfs(Node<V,K> source)    **Firma del método: public** HashMap<K,K> bfs(Node<V,K> source) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| **AdjacencyListUndirected** | **bfs(Node<V,K> source)** | **generalScenary()** | **source**: Nodo desde se quiere iniciar el recorrido en anchura. | Un **HashMap<K,K>** donde estará almacenado cada nodo con su llave, y el valor será la llave del padre en la ruta. |
| **AdjacencyList** | **bfs(Node<V,K> source)** | **generalScenary()** |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba No. 4** | **Objetivo de la prueba:**Comprobar que el método dfs(Node<V,K> source)    **Firma del método: public** HashMap<K,K> dfs(Node<V,K> source) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| **AdjacencyListUndirected** | dfs(Node<V,K> source) | **generalScenary()** | **source:** Nodo desde se quiere iniciar el recorrido en profundidad. | Un **HashMap<K,K>** donde estará almacenado cada nodo con su llave, y el valor será la llave del padre en la ruta. |
| **AdjacencyList** | dfs(Node<V,K> source) | **generalScenary()** | **source:** Nodo desde se quiere iniciar el recorrido en profundidad. | Un **HashMap<K,K>** donde estará almacenado cada nodo con su llave, y el valor será la llave del padre en la ruta. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba No. 6** | **Objetivo de la prueba:**Comprobar que el método prim(Node<V,K> source) retorne el hash de padres de la ruta.  **Firma del método: public** HashMap<K,K> prim(Node<V,K> source) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| **AdjacencyListUndirected** | **prim(Node<V,K> source)** | **generalScenary()** | **source:** Nodo desde se quiere iniciar el recorrido en profundidad. | Un **HashMap<K,K>** donde estará almacenado cada nodo con su llave, y el valor será la llave del padre en la ruta. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba No. 7** | **Objetivo de la prueba:**Comprobar que el método kruscal() retorne el hash de padres de la ruta.  **Firma del método: public** HashMap<K,K> kruscal() | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| **AdjacencyListUndirected** | **kruscal()** | **generalScenary()** | **Ninguna** | Un **HashMap<K,K>** donde estará almacenado cada nodo con su llave, y el valor será la llave del padre en la ruta. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prueba No. 5** | **Objetivo de la prueba:**Comprobar que el método dijsktra(Node<V,K> source) retorne el hash con las mínimas distancias desde eso nodo hacia los demás.    **Firma del método: public** HashMap<K, Double> dijsktra(Node<V,K> source) | | | |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| **AdjacencyListUndirecte**d | **dijsktra(Node<V,K> source)** | **generalScenary()** | source: Nodo desde se quieren conocer las distancias mínimas. | Un **HashMap<K,Double**> donde estará almacenado cada nodo con su llave, y el valor será la distancia mínima que hay entre el nodo source y el nodo con la llave. |
| **AdjacencyList** | **dijsktra(Node<V,K> source)** | **generalScenary()** | source: Nodo desde se quieren conocer las distancias mínimas hacia los nodos a los cuales el nodo **source** pueda llegar. | Un **HashMap<K,Double**> donde estará almacenado cada nodo con su llave, y el valor será la distancia mínima que hay entre el nodo source y el nodo con la llave. |