**DISEÑO PRUEBAS UNITARIAS**

**PRUEBAS CLASE KruskalTest**

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Clase** | **Escenario** |
| setUp1 | KruskalTest | Un objeto Kruskal y una matriz de enteros con los valores {0,2,0,6,0,2,0,3,8,5,0,3,0,0,7,6,8,0,0,9,0,5,7,9,0} |
| setUp2 | KruskalTest | Un objeto Kruskal y una matriz de enteros con los valores {0,3,20,3,0,3,0,0,0,0,20,0,0,2,3,3,0,2,0,0,0,0,3,0,0} |

* **Objetivo de la prueba:** Determinar si el método ***kruskalMST*** encuentra el árbol de recubrimiento mínimo correctamente

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **KruskalTest** | **kruskalMSTTest1** | setUp1 |  | El método retorna correctamente el peso mínimo del árbol que se esperaba, en este caso 16 |

* **Objetivo de la prueba:** Determinar si el método ***kruskalMST*** encuentra el árbol de recubrimiento mínimo correctamente

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **KruskalTest** | **kruskalMSTTest2** | setUp2 |  | El método retorna correctamente el peso mínimo del árbol que se esperaba, en este caso 11 |

**PRUEBAS CLASE GraphTest**

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Clase** | **Escenario** |
| setUp1 | GraphTest | Un objeto Graph y 4 objetos User con los siguientes valores:  User a: “andrea”  User b: “danna”  User c: “escobar”  User d: “cordoba”  User e: “reyes”  donde :  a tiene conexión con d  a tiene conexión con c  d tiene conexión con c  c tiene conexión con e  a tiene conexión con b |

* **Objetivo de la prueba:** Determinar si el método ***dfs*** me devuelve el camino en profundidad desde un nodo de inicio hasta recorrer todo el grafo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **GraphTest** | **dfsTest1** | setUp1 | String expected= "andreacordobaescobarreyesdanna"  Nodo inicial= andrea | El método retorna correctamente el camino en profundidad desde el nodo andrea hasta llegar a danna, recorriendo todos los nodos del grafo. |

* **Objetivo de la prueba:** Determinar si el método ***dfs*** me devuelve el camino en profundidad desde un nodo de inicio hasta recorrer todo el grafo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **GraphTest** | **dfsTest2** | setUp1 | String expected= "dannaandreacordobaescobarreyes"  Nodo inicial= danna | El método retorna correctamente el camino en profundidad desde el nodo danna hasta llegar a reyes, recorriendo todos los nodos del grafo. |

* **Objetivo de la prueba:** Determinar si el método ***dfs*** me devuelve el camino en profundidad desde un nodo de inicio hasta recorrer todo el grafo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **GraphTest** | **dfsTest3** | setUp1 | String expected= "escobarandreacordobadannareyes"  Nodo inicial= escobar | El método retorna correctamente el camino en profundidad desde el nodo escobar hasta llegar a reyes, recorriendo todos los nodos del grafo. |

* **Objetivo de la prueba:** Determinar si el método ***bfs*** me devuelve el camino en amplitud desde un nodo de inicio hasta recorrer todo el grafo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **GraphTest** | **bfsTest1** | setUp1 | String expected= "andreadannacordobaescobarreyes"  Nodo inicial= andrea | El método retorna correctamente el camino en amplitud desde el nodo andrea hasta llegar a reyes, recorriendo todos los nodos del grafo. |

* **Objetivo de la prueba:** Determinar si el método ***bfs*** me devuelve el camino en amplitud desde un nodo de inicio hasta recorrer todo el grafo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **GraphTest** | **bfsTest2** | setUp1 | String expected= "dannaandreacordobaescobarreyes"  Nodo inicial= danna | El método retorna correctamente el camino en profundidad desde el nodo danna hasta llegar a reyes, recorriendo todos los nodos del grafo. |

* **Objetivo de la prueba:** Determinar si el método ***bfs*** me devuelve el camino en amplitud desde un nodo de inicio hasta recorrer todo el grafo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **GraphTest** | **bfsTest3** | setUp1 | String expected= "reyesescobarandreadannacordoba"  Nodo inicial= reyes | El método retorna correctamente el camino en profundidad desde el nodo reyes hasta llegar a cordoba, recorriendo todos los nodos del grafo. |

**PRUEBAS CLASE DijkstraAlgorithmForAdjacencyListTest**

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Clase** | **Escenario** |
| setUp1 | DijkstraAlgorithmForAdjacencyListTest | Un objeto DijsktraAlgorithmForAdjacencyList.  Un objeto Graph y 4 objetos User con los siguientes valores:  User a: “andrea”  User b: “danna”  User c: “escobar”  User d: “cordoba”  User e: “reyes”  donde :  a tiene conexión con d  a tiene conexión con c  d tiene conexión con c  c tiene conexión con e  a tiene conexión con b |

* **Objetivo de la prueba:** Determinar si el método dijkstra me devuelve el camino más corto entre un nodo origen y un nodo destino.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **DijkstraAlgortihm ForAdjacencyListTest** | **dijkstraTest** | setUp1 | String expected= "andreacordobaescobarreyes"  Nodo origen= andrea  Nodo destino= reyes | El método retorna correctamente el camino más corto entre los nodos andrea y reyes. |

* **Objetivo de la prueba:** Determinar si el método dijkstra me devuelve el camino más corto entre un nodo origen y un nodo destino.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **DijkstraAlgortihm ForAdjacencyListTest** | **dijkstraTest2** | setUp1 | String expected= "dannaandreacordobaescobar"  Nodo origen= danna  Nodo destino= escobar | El método retorna correctamente el camino más corto entre los nodos danna y escobar. |

* **Objetivo de la prueba:** Determinar si el método dijkstra me devuelve el camino más corto entre un nodo origen y un nodo destino.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **DijkstraAlgortihm ForAdjacencyListTest** | **dijkstraTest3** | setUp1 | String expected= "reyesescobarcordobaandrea"  Nodo origen= reyes  Nodo destino= andrea | El método retorna correctamente el camino más corto entre los nodos reyes y andrea. |

**PRUEBAS CLASE DijkstraAlgorithmForAdjacencyMatrixTest**

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Clase** | **Escenario** |
| setUp1 | DijkstraAlgorithmForAdjacencyMatrixTest | Un objeto DijsktraAlgorithmForAdjacencyMatrix y una matriz de enteros de costos con los valores: {0, 3, 20, 3, 0},  {3, 0, 0, 0, 0},  {20, 0, 0, 2, 3},  {3, 0, 2, 0, 0},  {0, 0, 3, 0, 0}, |

* **Objetivo de la prueba:** Determinar si el método dijkstra me devuelve el camino más corto entre un nodo origen y un nodo destino a partir de la matriz de costos del grafo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **DijkstraAlgortihm ForAdjacencyMatrixTest** | **dijkstraTest** | setUp1 | String expected= "0324"  Nodo origen=0  Nodo destino=4 | El método retorna correctamente el camino más corto entre los nodos andrea(0) y reyes(4). |

* **Objetivo de la prueba:** Determinar si el método dijkstra me devuelve el camino más corto entre un nodo origen y un nodo destino a partir de la matriz de costos del grafo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **DijkstraAlgortihm ForAdjacencyMatrixTest** | **dijkstraTest2** | setUp1 | String expected= "103"  Nodo origen=1  Nodo destino=3 | El método retorna correctamente el camino más corto entre los nodos danna(0) y escobar(3). |

* **Objetivo de la prueba:** Determinar si el método dijkstra me devuelve el camino más corto entre un nodo origen y un nodo destino a partir de la matriz de costos del grafo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **DijkstraAlgortihm ForAdjacencyMatrixTest** | **dijkstraTest3** | setUp1 | String expected= "42301"  Nodo origen=4  Nodo destino=1 | El método retorna correctamente el camino más corto entre los nodos reyes(4) y danna(1). |

**PRUEBAS CLASE PrimTest**

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Clase** | **Escenario** |
| setUp1 | PrimTest | Un objeto Prim y una matriz de enteros con los valores { 0, 2, 0, 6, 0 },  { 2, 0, 3, 8, 5 },  { 0, 3, 0, 0, 7 },  { 6, 8, 0, 0, 9 },  { 0, 5, 7, 9, 0 }, |
| setUp2 | PrimTest | Un objeto Prim y una matriz de enteros con los valores {0, 3, 20, 3, 0},  {3, 0, 0, 0, 0},  {20, 0, 0, 2, 3},  {3, 0, 2, 0, 0},  {0, 0, 3, 0, 0}, |

* **Objetivo de la prueba:** Determinar si el método ***primMST*** encuentra el árbol de recubrimiento mínimo correctamente

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **PrimTest** | **PrimMSTTest1** | setUp1 | 16 | El método retorna correctamente el peso mínimo del árbol que se esperaba, en este caso 16 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **PrimTest** | **PrimMSTTest2** | setUp2 | 27 | El método retorna correctamente el peso mínimo del árbol que se esperaba, en este caso 27 |

**PRUEBAS CLASE FloydWarshallTest**

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Clase** | **Escenario** |
| setUp1 | FloyWarshallTest | Un objeto FloydWarshall y una matriz de enteros con los valores { 0, 2, 0, 6, 0 },  { 2, 0, 3, 8, 5 },  { 0, 3, 0, 0, 7 },  { 6, 8, 0, 0, 9 },  { 0, 5, 7, 9, 0 }, |
| setUp2 | FloydWarshallTest | Un objeto FloydWarshall y una matriz de enteros con los valores {0, 3, 20, 3, 0},  {3, 0, 0, 0, 0},  {20, 0, 0, 2, 3},  {3, 0, 2, 0, 0},  {0, 0, 3, 0, 0}, |

* **Objetivo de la prueba:** Determinar si el método ***floydWarshall*** encuentra los caminos minimos entre todos los vértices que conforman al grafo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **FloydWarshallTest** | **floydWarshallTest1** | setUp1 | un lista de enteros con los valores {6, 3, 5, 3, 8, 3, 6, 8, 6, 11, 5, 8, 4, 2, 3, 3, 6, 2, 4, 5, 8, 11, 3, 5, 6} que representa los caminos minimos que hay entre cada par de vertices. | El método retorna correctamente los caminos mínimos entre cada para de vértices que están en el grafo |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| **FloydWarshallTest** | **floydWarshallTest1** | setUp2 | un lista de enteros con los valores {4, 2, 5, 6, 7, 2, 4, 3, 8, 5, 5, 3, 6, 11, 7, 6, 8, 11, 12, 9, 7, 5, 7, 9, 10} que representa los caminos minimos que hay entre cada par de vertices. | El método retorna correctamente los caminos mínimos entre cada para de vértices que están en el grafo |