DISEÑO DE PRUEBAS UNITARIAS

Representación: Matriz de adyacencias

Prueba N° 1	**Objetivo**: Probar el método de inserción en un grafo.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**
AdjacencyMatrixGraph	+ insertVertex(V): void	Insertar un vértice sin ninguna conexión con los otros vértices del grafo.	• Valor que almacena el vértice	El método insertó el nuevo vértice que no conecta con ningún otro vértice.

Prueba N° 2	**Objetivo**: Probar el método que agrega una arista entre dos vértices.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**
AdjacencyMatrixGraph	+ insertEdge(int, int, E): void	Agregar una arista entre dos vértices que no están conectados.	• Vértice origen. • Vértice destino. • Peso de la arista.	El método conectó a los dos vértices con una arista que va desde el vértice origen hasta el vértice destino con el peso pasado por parámetro.
AdjacencyMatrixGraph	+ insertEdge(int, int, E): void	Conectar un vértice consigo mismo.	• Vértice origen. • Vértice destino (el mismo que el vértice origen). • Peso de la arista.	El método conectó al vértice con él mismo creando un ciclo.

Prueba N° 3	**Objetivo**: Probar el método que elimina un vértice del grafo.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**
AdjacencyMatrixGraph	+ deleteVertex(int): void	Eliminar un vértice que no existe.	Índice del vértice.	El método lanzó una excepción de IndexOutOfBoundsException.
AdjacencyMatrixGraph	+ deleteVertex(int): void	Eliminar un vértice existente que no tenga conexión con otros vértices.	Índice del vértice.	El método eliminó el vértice sin alterar las conexiones de los otros.

Prueba N° 4	**Objetivo**: Probar el método que elimina una arista del grafo.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**
AdjacencyMatrixGraph	+ deleteEdge(int, int, E): void	Eliminar una arista que no existe.	• Índice del vértice origen. • Índice del vértice destino. • Peso de la arista.	El método lanzó una excepción de IndexOutOfBoundsExcpetion.
AdjacencyMatrixGraph	+ deleteEdge(int, int, E): void	Eliminar una arista entre dos vértices de un grafo simple.	• Índice del vértice origen. • Índice del vértice destino. • Peso de la arista.	El método eliminó la arista.
AdjacencyMatrixGraph	+ deleteEdge(int, int, E): void	Eliminar una arista cuyos vértices origen y destino sean iguales.	• Índice del vértice origen. • Índice del vértice destino. • Peso de la arista.	El método eliminó el ciclo.

Prueba N° 5	**Objetivo**: Probar el método que elimina todas las aristas entre dos vértices.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**
AdjacencyMatrixGraph	+ deleteAllEdge(int, int): boolean	Dos vértices conectados por una sola arista y que no tengan conexión con otros vértices.	•Índice del vértice origen. •Índice del vértice destino.	El método eliminó la arista entre esos dos vértices.
AdjacencyMatrixGraph	+ deleteAllEdge(int, int): boolean	Dos vértices conectados por una sola arista y que tengan conexión con otros vértices.	•Índice del vértice origen. •Índice del vértice destino.	El método eliminó la arista entre esos dos vértices sin eliminar las otras aristas.

Prueba N° 6	**Objetivo**: Probar el método BFS desde un vértice de partida.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**

AdjacencyMatrixGraph	+ BFS(int): ArrayList<Vertex<T>>	Un grafo de un solo vértices.	• Índice del vértice de origen.	El método devolvió un ArrayList de tamaño 1.
AdjacencyMatrixGraph	+ BFS(int): ArrayList<Vertex<T>>	Un grafo no conexo de n vértices.	• Índice del vértice de origen.	El método devolvió un ArrayList de tamaño n menos el número de vértices de los otros subgrafos.
AdjacencyMatrixGraph	+ BFS(int): ArrayList<Vertex<T>>	Un grafo conexo de n vértices.	• Índice del vértice de origen.	El método devolvió un ArrayList de tamaño n.
AdjacencyMatrixGraph	+ BFS(int): ArrayList<Vertex<T>>	Un grafo con un ciclo.	• Índice del vértice de origen.	El método devolvió un ArrayList de tamaño n.

Prueba N° 7	**Objetivo**: Probar el método BFS sin ningún vértice de partida.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**
AdjacencyMatrixGraph	+ BFS(): ArrayList<ArrayList<Vertex<T>>>	Un grafo de un solo vértices.	Ninguna.	El método devolvió un ArrayList de tamaño 1.
AdjacencyMatrixGraph	+ BFS(): ArrayList<ArrayList<Vertex<T>>>	Un grafo no conexo de n vértices.	Ninguna.	El método devolvió varios ArrayList.
AdjacencyMatrixGraph	+ BFS(): ArrayList<ArrayList<Vertex<T>>>	Un grafo conexo de n vértices.	Ninguna.	El método devolvió varios ArrayList.

AdjacencyMatrixGraph	+ BFS(): ArrayList<ArrayList<Vertex<T>>>	Un grafo con un ciclo.	Ninguna.	El método devolvió varios ArrayList.

Prueba N° 8	**Objetivo**: Probar el método DFS desde un vértice de partida.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**
AdjacencyMatrixGraph	+ DFS(int): ArrayList<Integer>	Recorrer un grafo de un solo vértice.	• Índice del vértice desde donde inicia el recorrido.	El método devolvió un árbol DFS con un solo índice.
AdjacencyMatrixGraph	+ DFS(int): ArrayList<Integer>	Recorrer un grafo conexo.	• Índice del vértice desde donde inicia el recorrido.	El método devolvió un árbol DFS con la misma cantidad de índices que de nodos en el grafo.
AdjacencyMatrixGraph	+ DFS(int): ArrayList<Integer>	Recorrer un grafo no conexo.	• Índice del vértice desde donde inicia el recorrido.	El método devolvió el árbol de expansión mínima correspondiente al índice de inicio.
AdjacencyMatrixGraph	+ DFS(int): ArrayList<Integer>	Recorrer un grafo con un ciclo.	• Índice del vértice desde donde inicia el recorrido.	El método no tuvo problemas y devolvió el árbol de expansión mínima correspondiente sin haber repetido la visita al vértice con el ciclo.

Prueba N° 8	**Objetivo**: Probar el método DFS que recorre todo el grafo sin ningún vértice de partida.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**

AdjacencyMatrixGraph	+ DFS(): ArrayList<ArrayList<Integer>>	Recorrer un grafo de un solo vértice.	Ninguna.	El método devolvió un árbol DFS con un solo índice.
AdjacencyMatrixGraph	+ DFS(): ArrayList<ArrayList<Integer>>	Recorrer un grafo conexo.	Ninguna.	El método devolvió un árbol DFS con la misma cantidad de índices que de nodos en el grafo.
AdjacencyMatrixGraph	+ DFS(): ArrayList<ArrayList<Integer>>	Recorrer un grafo no conexo.	Ninguna.	El método devolvió una cantidad de árboles de expansión mínima correspondiente al número de subgrafos conexos en el grafo no conexo.
AdjacencyMatrixGraph	+ DFS(): ArrayList<ArrayList<Integer>>	Recorrer un grafo con un ciclo.	Ninguna.	El método no tuvo problemas y devolvió el árbol o los árboles de expansión mínima correspondientes sin haber repetido la visita al vértice con el ciclo.

Prueba N° 9	**Objetivo**: Probar el método Prim.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**
AdjacencyMatrixGraph	+ Prim(int): ArrayList<Integer>	Recorrer un grafo conexo.	Índice del vértice desde donde se inicia el recorrido.	El método devolvió un árbol de expansión mínima con todos los vértices del grafo.
AdjacencyMatrixGraph	+ Prim(int): ArrayList<Integer>	Recorrer un grafo no conexo.	Índice del vértice desde donde se inicia el recorrido.	El método devolvió un árbol de expansión mínima pero solo del subgrafo al que pertenecía el vértice desde donde se comenzó el recorrido.

Prueba N° 10	**Objetivo**: Probar el método Dijkstra.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**

AdjacencyListGraph	+ Dijkstra(int): Object[]	Un grafo de un solo vértice.	Índice del vértice desde donde se inicia el recorrido.	El método devolvió el arreglo de distancias con respecto al nodo origen y el de predecesores ambos de tamaño 1.
AdjacencyListGraph	+ Dijkstra(int): Object[]	Recorrer un grafo conexo simple.	Índice del vértice desde donde se inicia el recorrido.	El método devolvió el arreglo de distancias con respecto al nodo origen y el de predecesores ambos de tamaño mayor a uno.
AdjacencyListGraph	+ Dijkstra(int): Object[]	Hacer el recorrido desde un vértice que no existe.	Índice del vértice desde donde se inicia el recorrido.	El método arrojó una excepción de IndexOutOfBoundsExceptio.