DISEÑO DE PRUEBAS UNITARIAS

Grafo no dirigido con listas de adyacencia

Prueba N° 1	**Objetivo**: Probar el método de inserción en un grafo.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**
AdjacencyListGraph	+ insertVertex(T, Edge): void	Insertar un vértice con una conexión al primer vértice del grafo.	• Valor que almacena el vértice. • Arista con el primer vértice del grafo.	El método insertó el nuevo vértice que conecta con el primer vértice.
AdjacencyListGraph	+ insertVertex (T, ArrayList<Edge>): void	Insertar un vértice que tenga conexión con los otros vértices que ya posea el grafo.	• Valor que almacena el vértice. • ArrayList con las aristas que conectan el vértice con los otros.	El método insertó el nuevo vértice que conecta con los otros vértices.
AdjacencyListGraph	+ insertVertex (T): 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**
AdjacencyListGraph	+ addEdge(Vertex, Vertex, int): 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 con el peso pasado por parámetro.
AdjacencyListGraph	+ addEdge(Vertex, Vertex, int): 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.
AdjacencyListGraph	+ addEdge(Vertex, Vertex, int): void	Agregar una arista entre dos vértices que ya están conectados por otra arista.	• Vértice origen. • Vértice destino. • Peso de la arista.	El método conectó a los dos vértices con una nueva arista, por lo cual ahora están unidos por dos aristas.

Prueba N° 3	**Objetivo**: Probar el método que elimina una arista del grafo.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**
AdjacencyListGraph	+ deleteEdge(Vertex, Vertex, int): void	Eliminar una arista que no existe.	• Vértice origen. • Índice del vértice destino.	El método lanzó una excepción de IndexOutOfBoundsExcpetion.
AdjacencyListGraph	+ deleteEdge(Vertex, Vertex, int): void	Eliminar una arista entre dos nodos.	• Vértice origen. • Índice del vértice destino.	El método eliminó la arista entre esos dos nodos sin interferir en las otras conexiones.
AdjacencyListGraph	+ deleteEdge(Vertex, Vertex, int): void	Eliminar una arista cuyos vértices origen y destino sean iguales.	• Vértice origen. • Índice del vértice destino (el mismo que el origen).	El método eliminó el ciclo.

Prueba N° 4	**Objetivo**: Probar el método que elimina un vértice del grafo.			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**
AdjacencyListGraph	+ deleteVertex(int): Vertex<T>	Eliminar un vértice que no existe.	Índice del vértice.	El método lanzó una excepción de IndexOutOfBoundsExcepti on.
AdjacencyListGraph	+ deleteVertex(int): Vertex<T>	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.
AdjacencyListGraph	+ deleteVertex(int): Vertex<T>	Eliminar un vértice hoja.	Índice del vértice.	El método eliminó el vértice junto con las aristas que incidían en él.
AdjacencyListGraph	+ deleteVertex(int): Vertex<T>	Eliminar un vértice con múltiples conexiones a otros vértices.	Índice del vértice.	El método eliminó el vértice junto con las aristas que incidían en él.

Prueba N° 5	**Objetivo**: Probar el método BFS (Breadth First Search).			
Clase	**Método**	**Esce nario**	**Entradas**	**Resultado**

AdjacencyListGraph	+ BFS(int, int): ArrayList<Vertex<T>>	Recorrer un camino que no existe entre dos vértices.	• Índice del vértice de origen. • Índice del vértice destino.	El método devolvió null ya que no hay un camino entre esos dos vértices.
AdjacencyListGraph	+ BFS(int, int): ArrayList<Vertex<T>>	Recorrer un camino entre el mismo vértice.	• Índice del vértice de origen. • Índice del vértice destino (el mismo que el de origen).	El método devolvió un ArrayList de tamaño 1.
AdjacencyListGraph	+ BFS(int, int): ArrayList<Vertex<T>>	Recorrer un camino entre dos vértices adyacentes.	• Índice del vértice de origen. • Índice del vértice destino (el mismo que el de origen).	El método devolvió un ArrayList de tamaño 2.
AdjacencyListGraph	+ BFS(int, int): ArrayList<Vertex<T>>	Recorrer un camino entre dos vértices que no son adyacentes entre sí.	• Índice del vértice de origen. • Índice del vértice destino (el mismo que el de origen).	El método devolvió un ArrayList de tamaño mayor a 2.

Prueba N° 6	**Objetivo**: Probar el método DFS (Depth First Search).			
Clase	**Método**	**Escenario**	**Entradas**	**Resultado**
AdjacencyListGraph	+ DFS(): ArrayList<ArrayList<Vertex<T>>>	Recorrer un grafo de un solo vértice.	Ninguna.	El método devolvió un árbol DFS con un solo nodo.
AdjacencyListGraph	+ DFS(): ArrayList<ArrayList<Vertex<T>>>	Recorrer un grafo conexo.	Ninguna.	El método devolvió un árbol DFS con la misma cantidad de nodos del grafo.

AdjacencyListGraph	+ DFS(): ArrayList<ArrayList<Vertex<T>>>	Recorrer un grafo no conexo.	Ninguna.	El método devolvió varios árboles DFS con varios nodos.