

# PROYECTO DE CURSO

## Enunciado

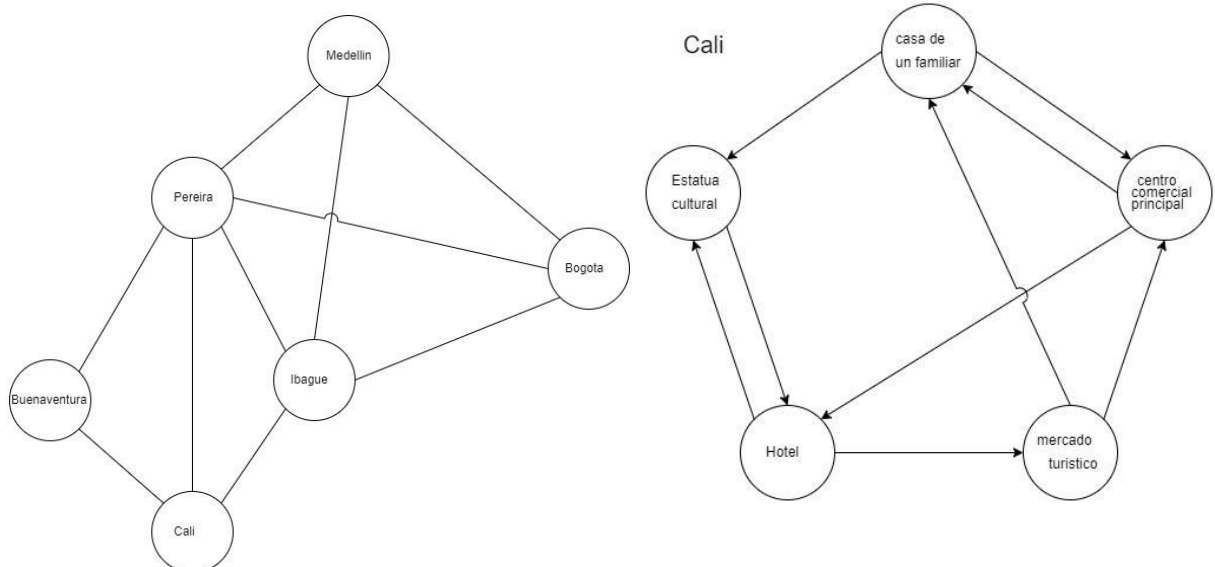
Tú y tu familia van a salir de vacaciones a diferentes ciudades de Colombia que no hayan visitado hasta el momento. El viaje se realizará en carro y debido a esto tienen la necesidad de utilizar un mapa completo de todas las ciudades para conocer como llegar a cada una a través de la otra, además de los mapas de cada ciudad con puntos de interés marcados.

Durante una discusión acerca del paseo se encontró el problema de que todos en tu familia son terribles leyendo mapas y si intentan llegar a un supermercado terminan en Siloé, por lo que te pidieron a ti, un ingeniero de sistemas de Icesi, que crearas un programa que pudiese representar todos estos mapas y que de una forma simple indique cuales son los caminos para llegar de una ciudad a otra, o de un punto de interés al otro, preferiblemente los más cortos.

Para implementar el programa cuentas con los mapas digitales de Colombia (más específicamente, de la zona donde se encuentran las ciudades a visitar) y de cada ciudad.

Los mapas tienen la información de las calles entre un punto al otro y sus longitudes en km.

Ej.:



\*Los mapas utilizados no son representaciones reales de las calles/carreteras.

El programa debe de permitir al copiloto obtener la siguiente información:

- El camino más corto de un punto de interés/ciudad al otro.
- Todos los puntos de interés o ciudades en el mapa.
- Cantidad de zonas inconexas (zonas a las que no se puede acceder por carro)
- El tiempo que se demora en llegar de un punto al otro teniendo en cuenta una velocidad introducida.

### Requerimientos funcionales

<b>Nombre</b>	R0: Conseguir el camino más corto de un punto de interés/ciudad al otro
<b>Resumen</b>	Se da a conocer a través de qué camino se llega de un punto de interés/ciudad a la otra
<b>Entrada</b>	Ciudad de inicio, ciudad a llegar
<b>Salida</b>	Camino mínimo entre los dos puntos

<b>Nombre</b>	R1: Mostrar los puntos de interés/ciudades de un mapa
<b>Resumen</b>	Imprime los puntos de interés/ciudades que se encuentran en el mapa
<b>Entrada</b>	Mapa a revisar
<b>Salida</b>	puntos de interés/ciudades del mapa

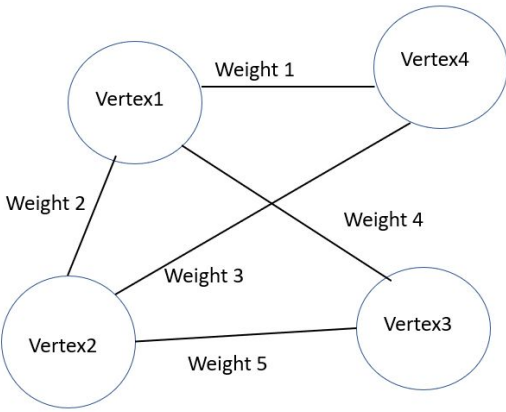
<b>Nombre</b>	R2: Mostrar la cantidad de zonas inaccesibles por carro
<b>Resumen</b>	Da a conocer la cantidad de zonas que existen en el mapa a las que no son posibles de acceder por medio de un carro.
<b>Entrada</b>	Mapa
<b>Salida</b>	Cantidad de zonas inconexas.

<b>Nombre</b>	R3: Dar a conocer el tiempo que se demora de una ciudad a otra al conocer la velocidad
<b>Resumen</b>	al usuario introducir la velocidad promedio, se calcula el tiempo de llegada de una ciudad a otra.
<b>Entrada</b>	Velocidad promedio, ciudad de inicio, ciudad a llegar
<b>Salida</b>	Tiempo estimado de llegada

<b>Nombre</b>	R4: Ilustrar el mapa elegido
<b>Resumen</b>	El usuario puede elegir un mapa para mostrar en pantalla.
<b>Entrada</b>	Mapa a mostrar
<b>Salida</b>	Ilustración del mapa

**TAD WeightedMatrixGraph:**

⊆

TAD: WeightedMatrixGraph		
 <pre> graph TD     V1((Vertex1)) --- Weight 1  V4((Vertex4))     V1 --- Weight 2  V2((Vertex2))     V1 --- Weight 3  V3((Vertex3))     V2 --- Weight 5  V3     V4 --- Weight 4  V3 </pre>		
Inv{any two vertex can only have one edge connecting them}		
•	<b>AddVertex: Vertex</b>	→ <b>Boolean</b>
•	<b>AddEdge: Vertex X Vertex X weight</b>	→ <b>Boolean</b>
•	<b>WeightedMatrixGraph:</b>	→ <b>Graph</b>
•	<b>SearchVertex: Texto</b>	→ <b>Vertex</b>
•	<b>DepthFirstSearch: Graph X Vertex</b>	→ <b>List</b>
•	<b>BreadthFirstSearch: Graph X Vertex</b>	→ <b>List</b>
•	<b>Dijkstra: Graph X Vertex</b>	→ <b>List</b>
•	<b>FloydWarshall: Graph</b>	→ <b>List</b>

- **RemoveEdge: Vertex X  
Vertex** → **Boolean**
- **RemoveVertex Vertex** → **Boolean**

**AddVertex(Vertex)**

**“Adds a vertex object to the graph”**

**{pre : Graph != null, Vertex  $\notin$  Graph }**

**{post Vertex added, Vertex  $\in$  Graph}**

**AddEdge(Vertex 1, Vertex 2, weight )**

**“Adds weighted Edge to the graph connecting two Vertices belonging to the graph”**

**{pre : Graph != null, Vertex1  $\wedge$  Vertex2  $\in$  Graph  $\wedge$  weight  $\in$  Double }**

**{post Edge added in between two vertices}**

**AddEdge(Vertex 1, Vertex 2)**

**“Adds non-weighted Edge to the graph connecting two Vertices belonging to the graph”**

**{pre : Graph != null, Vertex1  $\wedge$  Vertex2  $\in$  Graph}**

**{post Edge added in between two vertices}**

**WeightedMatrixGraph()**

**“Creates an empty graph, no vertices, no edges”**

**{pre : }**

**{post:empty Graph}**

**SearchVertex(name)**

**“from a text searches a vertex in the graph”**

**{pre : Vertex  $\in$  Graph ^ name  $\in$  Text }**

**{post returns Vertex}**

**DepthFirstSearch()**

**“Traverses the Graph in DepthFirst manner”**

**{pre : Graph != null}**

**{post returns list in order DFS}**

**BreadthFirstSearch()**

**“Traverses the Graph in BreadthFirst manner”**

**{pre :,Graph != null}**

**{post returns list in BFS}**

**Dijkstra(Vertex 1, Vertex 2)**

**“Searchs for the shortest distance between two Vertexs in the graph Graph ”**

**{pre :Vertex1 ^ Vertex2 ∈ Graph , Graph != null}**



**{post: returns a vertex list with the vertexes forming the shortest path between the two vertexes}**

**FloydWarshall()**

**“Searches for the shortest distance between all the vertexes in the graphs”**

**{pre : Graph != null}**

**{post: returns the list of all the vertex forming the shortest path}**

**RemoveEdge(Vertex 1, Vertex 2)**

**“Eliminates the edge between two vertexes”**

**{pre : Vertex1  $\wedge$  Vertex2  $\in$  Graph, Graph != null}**

**{post: returns a boolean value, true if it could be removed, false if the edge could not be removed}**

**RemoveVertex(name)**

**“Eliminates a vertex of the graph”**

**{Vertex  $\in$  Graph ^ name  $\in$  Text, Graph  $\neq$  null}**

**{post: returns the boolean value, true if the vertex was removed, false if the vertex was not}**

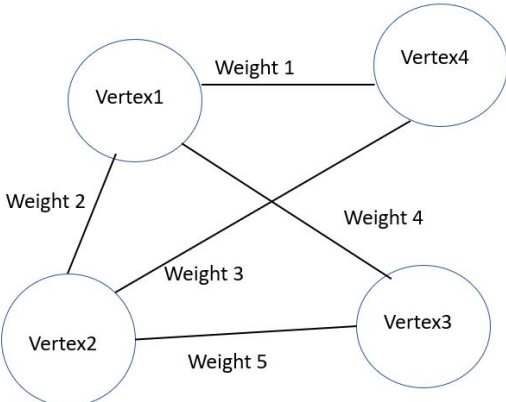
**areConnected(Vertex 1, Vertex 2)**

**“Checks if two vertices are connected”**

**{pre : Vertex1 ^ Vertex2  $\in$  Graph, Graph  $\neq$  null}**

**{post: returns the boolean value, true if the verticves}**

## TAD AdjacencyListGraph:

TAD: AdjacencyListGraph		
 <pre> graph TD     V1((Vertex1)) --- Weight 1  V4((Vertex4))     V1 --- Weight 2  V2((Vertex2))     V1 --- Weight 3  V3((Vertex3))     V2 --- Weight 5  V3     V2 --- Weight 4  V4         </pre>		
Inv{any two vertex can only have one edge connecting them}		
•	<b>AddVertex: Vertex</b>	→ <b>Boolean</b>
•	<b>AddEdge: Vertex X Vertex X weight</b>	→ <b>Boolean</b>
•	<b>AdjacencyListGraph:</b>	→ <b>Graph</b>
•	<b>SearchVertex: Texto</b>	→ <b>Vertex</b>
•	<b>DepthFirstSearch: Graph X Vertex</b>	→ <b>List</b>
•	<b>BreadthFirstSearch: Graph X Vertex</b>	→ <b>List</b>
•	<b>Dijkstra: Graph X Vertex</b>	→ <b>List</b>
•	<b>FloydWarshall: Graph</b>	→ <b>List</b>
•	<b>RemoveEdge: Vertex X Vertex</b>	→ <b>Boolean</b>

•      **RemoveVertex Vertex**       $\rightarrow$       **Boolean**

**AddVertex(Vertex)**

**“Adds a vertex object to the graph”**

**{pre : Graph != null, Vertex  $\notin$  Graph }**

**{post Vertex added, Vertex  $\in$  Graph}**

**AddEdge(Vertex 1, Vertex 2, weight )**

**“Adds weighted Edge to the graph connecting two Vertices belonging to the graph”**

**{pre : Graph != null, Vertex1  $\wedge$  Vertex2  $\in$  Graph  $\wedge$  weight  $\in$  Double }**

**{post Edge added in between two vertexes}**

**AddEdge(Vertex 1, Vertex 2)**

**“Adds non-weighted Edge to the graph connecting two Vertices belonging to the graph”**

**{pre : Graph != null, Vertex1  $\wedge$  Vertex2  $\in$  Graph}**

**{post Edge added in between two vertexes}**

**AdjacencyListGraph()**

**“Creates an empty graph, no vertices, no edges”**

**{pre : }**

**{post:empty Graph}**

**SearchVertex(name)**

**“from a text searches a vertex in the graph”**

**{pre : Vertex  $\in$  Graph ^ name  $\in$  Text }**

**{post returns Vertex}**

**DepthFirstSearch()**

**“Traverses the Graph in DepthFirst manner”**

**{pre : Graph != null}**

**{post returns list in order DFS}**

**BreadthFirstSearch()**

**“Traverses the Graph in BreadthFirst manner”**

**{pre :,Graph != null}**

**{post returns list in BFS}**

**Dijkstra(Vertex 1, Vertex 2)**

**“Searchs for the shortest distance between two Vertexs in the graph Graph ”**

**{pre :Vertex1 ^ Vertex2 ∈ Graph , Graph != null}**

**{post: returns a vertex list with the vertexes forming the shortest path between the two vertexes}**

**FloydWarshall()**

**“Searches for the shortest distance between all the vertexes in the graphs”**

**{pre : Graph != null}**

**{post: returns the list of all the vertex forming the shortest path}**

**RemoveEdge(Vertex 1, Vertex 2)**

**“Eliminates the edge between two vertexes”**

**{pre : Vertex1  $\wedge$  Vertex2  $\in$  Graph, Graph != null}**

**{post: returns a boolean value, true if it could be removed, false if the edge could not be removed}**



**RemoveVertex(name)**

**“Eliminates a vertex of the graph”**

**{Vertex  $\in$  Graph ^ name  $\in$  Text, Graph  $\neq$  null}**

**{post: returns the boolean value, true if the vertex was removed, false if the vertex was not}**

**areConnected(Vertex 1, Vertex 2)**

**“Checks if two vertices are connected”**

**{pre : Vertex1 ^ Vertex2  $\in$  Graph, Graph  $\neq$  null}**

**{post: returns the boolean value, true if the verticves}**

## **Diseño de pruebas unitarias:**

### **Diseño de pruebas unitarias**

**Escenarios:**

<b>nombre</b>	<b>clase</b>	<b>Escenario</b>
---------------	--------------	------------------

<b>SetupScenary()</b>	<b>WeightMatrixGraph</b>	<b>WeightMatrixGraph WG = new WeightMatrixGraph(false, 4)</b>
<b>SetupScenary1()</b>	<b>AdjencyListGraph</b>	<b>AdjencyListGraph AG = new AdjencyListGraph(False)</b>
<b>SetupScenary2()</b>	<b>GraphAlgorithims</b>	<b>WeightMatrixGraph WG = new WeightMatrixGraph(false, 4)</b>  <b>AdjencyListGraph AG = new AdjencyListGraph(False)</b>  <b>GraphAlgorithims = GA = new GraphAlgorithims()</b>

<b>Objetivo de la prueba : Comprobar que se creo el grafo Correctamente</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>WeightMatrixGraph</b>	<b>WeightMatrixGraph()</b>	<b>SetupScenary()</b>		<b>Se creo el grafo de manera adecuada</b>

<b>Objetivo de la prueba : Comprobar que se agrego un vertice de manera correcta</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>

<b>WeightMatrix Graph</b>	<b>AddVertex</b>	<b>SetupScenario()</b>	<b>V vertex V vertex1</b>	<b>Se Agrego El vértice de manera correcta</b>
-------------------------------	------------------	------------------------	-------------------------------	--

<b>Objetivo de la prueba : Comprobar que se agrego una arista al grafo de manera correcta</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>WeightMatrix Graph</b>	<b>AddEdge</b>	<b>SetupScenario()</b>	<b>V vertex V vertex1  double w = 5</b>	<b>Se agrego la arista de manera correcta al grafo</b>

<b>Objetivo de la prueba : Comprobar que se quito la arista de manera correcta</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>WeightMatrix Graph</b>	<b>Remove Edge</b>	<b>SetupScenario()</b>	<b>V vertex V vertex1</b>	<b>Se elimino la arista para el par de vertices</b>

<b>Objetivo de la prueba : Comprobar que se quito el vertice de manera correcta</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>WeightMatrix Graph</b>	<b>Remove Vertex</b>	<b>SetupScenar y()</b>	<b>V vertex</b>	<b>Se elimino correctamente el vertice</b>

<b>Objetivo de la prueba : Comprobar que el grafo se encuentre dirigido</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>WeightMatrix Graph</b>	<b>isDirected</b>	<b>SetupScenar y()</b>		<b>El grafo se encuentra de manera dirigida</b>

<b>Objetivo de la prueba : Comprobar que el tamaño del grafo sea el indicado</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>WeightMatrix Graph</b>	<b>Vertex Size()</b>	<b>SetupScenar y()</b>		<b>El tamaño del grafo es indicado</b>

**AdjencyList Graph :**

Objetivo de la prueba : Comprobar que se creo el grafo Correctamente				
Clase	Método	Escenario	Valores de Entrada	Resultado
AdjencyListGraph	AdjencyListGraph()	SetupScenariy1()		Se creo el grafo de manera adecuada

Objetivo de la prueba : Comprobar que se agrego un vertice de manera correcta				
Clase	Método	Escenario	Valores de Entrada	Resultado
AdjencyListGraph	AddVertex	SetupScenariy1()	V vertex V vertex1	Se Agrego El vértice de manera correcta

Objetivo de la prueba : Comprobar que se agrego una arista al grafo de manera correcta				
Clase	Método	Escenario	Valores de Entrada	Resultado

<b>AdjencyListGraph</b>	<b>AddEdge</b>	<b>SetupScenario()</b>	<b>V vertex</b> <b>V vertex1</b> <b>double w</b> <b>= 5</b>	<b>Se agrego la arista de manera correcta al grafo</b>
-------------------------	----------------	------------------------	--	--

<b>Objetivo de la prueba : Comprobar que se quito la arista de manera correcta</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>AdjencyListGraph</b>	<b>RemoveEdge</b>	<b>SetupScenario()</b>	<b>V vertex</b> <b>V vertex1</b>	<b>Se elimino la arista para el par de vertices</b>

<b>Objetivo de la prueba : Comprobar que se quito el vertice de manera correcta</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>AdjencyListGraph</b>	<b>RemoveVertex</b>	<b>SetupScenario()</b>	<b>V vertex</b>	<b>Se elimino correctamente el vertice</b>

Objetivo de la prueba : Comprobar que el grafo se encuentre dirigido				
Clase	Método	Escenario	Valores de Entrada	Resultado
AdjencyListGraph	isDirected	SetupScenary 1()		El grafo se encuentra de manera dirigida

Objetivo de la prueba : Comprobar que dos vértices estén conectados				
Clase	Método	Escenario	Valores de Entrada	Resultado
AdjencyListGraph	isConnected	SetupScenary 1()	V vertex V vertex1	Los dos vértices se encuentran conectados

Objetivo de la prueba : Comprobar que el tamaño del grafo sea el indicado				
Clase	Método	Escenario	Valores de Entrada	Resultado
AdjencyListGraph	Vertex Size()	SetupScenary 1()		El tamaño del grafo es indicado

### GraphAlgorithms:

Objetivo de la prueba : Comprobar que se recorrió de manera DFS el grafo				
Clase	Método	Escenario	Valores de Entrada	Resultado
GraphAlgorithms	DepthFirst Search	SetupScenario2()		Se recorrió de manera correcta el grafo

Objetivo de la prueba : Comprobar que se recorrió de manera BFS el grafo				
Clase	Método	Escenario	Valores de Entrada	Resultado
GraphAlgorithms	BreadthFirst Search	SetupScenario2()		Se recorrió de manera correcta el grafo

Objetivo de la prueba : Comprobar que se encontró de manera correcta el recorrido mas corto
---



Clase	Método	Escenario	Valores de Entrada	Resultado
GraphAlgorithms	Dijkstra()	SetupScenario2()	V vertex V vertex1 V vertex2 V vertex3	Se encontró de manera correcta el recorrido mas corto para los vertices

Objetivo de la prueba : Comprobar que se encontró de manera correcta el recorrido mas corto				
Clase	Método	Escenario	Valores de Entrada	Resultado
GraphAlgorithms	FloydWarshall	SetupScenario2()	V vertex V vertex1 V vertex2 V vertex3	Se encontró de manera correcta el recorrido mas corto para los vertices

<b>Objetivo de la prueba : Comprobar que se encontró el vertice indicado</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>GraphAlgorithms</b>	<b>Search()</b>	<b>SetupScenario2()</b>	<b>String name = "name1"</b>	<b>Se encontró el vertice indicado</b>

### **Diseño de pruebas unitarias**

#### **Escenarios:**

<b>nombre</b>	<b>clase</b>	<b>Escenario</b>
<b>SetupScenario()</b>	<b>WeightMatrixGraph</b>	<b>WeightMatrixGraph WG = new WeightMatrixGraph(false, 4)</b>
<b>SetupScenario1()</b>	<b>AdjacencyListGraph</b>	<b>AdjacencyListGraph AG = new AdjacencyListGraph(False)</b>
<b>SetupScenario2()</b>	<b>GraphAlgorithms</b>	<b>WeightMatrixGraph WG = new WeightMatrixGraph(false, 4)</b>  <b>AdjacencyListGraph AG = new AdjacencyListGraph(False)</b>  <b>GraphAlgorithms GA = new GraphAlgorithms()</b>

<b>Objetivo de la prueba : Comprobar que se creo el grafo Correctamente</b>
---

<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>WeightMatrix Graph</b>	<b>WeightMatrix Graph()</b>	<b>SetupScenariy()</b>		<b>Se creo el grafo de manera adecuada</b>

<b>Objetivo de la prueba : Comprobar que se agrego un vertice de manera correcta</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>WeightMatrix Graph</b>	<b>AddVertex</b>	<b>SetupScenariy()</b>	<b>V vertex V vertex1</b>	<b>Se Agrego El vértice de manera correcta</b>

<b>Objetivo de la prueba : Comprobar que se agrego una arista al grafo de manera correcta</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>WeightMatrix Graph</b>	<b>AddEdge</b>	<b>SetupScenariy()</b>	<b>V vertex V vertex1  double w = 5</b>	<b>Se agrego la arista de manera correcta al grafo</b>

Objetivo de la prueba : Comprobar que se quito la arista de manera correcta				
Clase	Método	Escenario	Valores de Entrada	Resultado
WeightMatrix Graph	Remove Edge	SetupScenar y()	V vertex V vertex1	Se elimino la arista para el par de vertices

Objetivo de la prueba : Comprobar que se quito el vertice de manera correcta				
Clase	Método	Escenario	Valores de Entrada	Resultado
WeightMatrix Graph	Remove Vertex	SetupScenar y()	V vertex	Se elimino correctamente el vertice

Objetivo de la prueba : Comprobar que el grafo se encuentre dirigido				
Clase	Método	Escenario	Valores de Entrada	Resultado

<b>WeightMatrix Graph</b>	<b>isDirected</b>	<b>SetupScenario()</b>		<b>El grafo se encuentra de manera dirigida</b>
---------------------------	-------------------	------------------------	--	---

<b>Objetivo de la prueba : Comprobar que el tamaño del grafo sea el indicado</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>WeightMatrix Graph</b>	<b>Vertex Size()</b>	<b>SetupScenario()</b>		<b>El tamaño del grafo es indicado</b>

**AdjacencyList Graph :**

<b>Objetivo de la prueba : Comprobar que se creo el grafo Correctamente</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>AdjacencyListGraph</b>	<b>AdjacencyListGraph()</b>	<b>SetupScenario1()</b>		<b>Se creo el grafo de manera adecuada</b>

<b>Objetivo de la prueba : Comprobar que se agrego un vertice de manera correcta</b>
--

Clase	Método	Escenario	Valores de Entrada	Resultado
AdjacencyListGraph	AddVertex	SetupScenario1()	V vertex V vertex1	Se Agrego El vértice de manera correcta

<b>Objetivo de la prueba : Comprobar que se agrego una arista al grafo de manera correcta</b>				
Clase	Método	Escenario	Valores de Entrada	Resultado
AdjacencyListGraph	AddEdge	SetupScenario()	V vertex V vertex1  double w = 5	Se agrego la arista de manera correcta al grafo

<b>Objetivo de la prueba : Comprobar que se quito la arista de manera correcta</b>				
Clase	Método	Escenario	Valores de Entrada	Resultado
AdjacencyListGraph	RemoveEdge	SetupScenario1()	V vertex  V vertex1	Se elimino la arista para el par de vertices

--	--	--	--	--

Objetivo de la prueba : Comprobar que se quito el vertice de manera correcta				
Clase	Método	Escenario	Valores de Entrada	Resultado
AdjencyListGraph	Remove Vertex	SetupScenariy1()	V vertex	Se elimino correctamente el vertice

Objetivo de la prueba : Comprobar que el grafo se encuentre dirigido				
Clase	Método	Escenario	Valores de Entrada	Resultado
AdjencyListGraph	isDirected	SetupScenariy1()		El grafo se encuentra de manera dirigida

Objetivo de la prueba : Comprobar que dos vértices estén conectados				
Clase	Método	Escenario	Valores de Entrada	Resultado

<b>AdjencyListGraph</b>	<b>isConnected</b>	<b>SetupScenary1()</b>	<b>V vertex V vertex1</b>	<b>Los dos vértices se encuentran conectados</b>
-------------------------	--------------------	------------------------	-------------------------------	--

<b>Objetivo de la prueba : Comprobar que el tamaño del grafo sea el indicado</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>AdjencyListGraph</b>	<b>Vertex Size()</b>	<b>SetupScenary1()</b>		<b>El tamaño del grafo es indicado</b>

### **GraphAlgorithims:**

<b>Objetivo de la prueba : Comprobar que se recorrió de manera DFS el grafo</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>GraphAlgorithims</b>	<b>DepthFirst Search</b>	<b>SetupScenariy2()</b>		<b>Se recorrió de manera correcta el grafo</b>



Objetivo de la prueba : Comprobar que se recorrió de manera BFS el grafo				
Clase	Método	Escenario	Valores de Entrada	Resultado
GraphAlgorithms	BreadthFirst Search	SetupScenario2()		Se recorrió de manera correcta el grafo

Objetivo de la prueba : Comprobar que se encontró de manera correcta el recorrido mas corto				
Clase	Método	Escenario	Valores de Entrada	Resultado
GraphAlgorithms	Dijkstra()	SetupScenario2()	V vertex V vertex1 V vertex2 V vertex3	Se encontró de manera correcta el recorrido mas corto para los vertices

<b>Objetivo de la prueba : Comprobar que se encontró de manera correcta el recorrido mas corto</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>GraphAlgorithms</b>	<b>FloydWarshall</b>	<b>SetupScenario2()</b>	<b>V vertex V vertex1 V vertex2 V vertex3</b>	<b>Se encontró de manera correcta el recorrido mas corto para los vertices</b>

<b>Objetivo de la prueba : Comprobar que se encontró el vertice indicado</b>				
<b>Clase</b>	<b>Método</b>	<b>Escenario</b>	<b>Valores de Entrada</b>	<b>Resultado</b>
<b>GraphAlgorithms</b>	<b>Search()</b>	<b>SetupScenario2()</b>	<b>String name = "name1"</b>	<b>Se encontró el vertice indicado</b>

**Diagrama de clases:**

