Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico de Ingeniería de Software.

Nombre de la Práctica Grafos.

Numero de Práctica 29.

Nombre de la carrera Ingeniería de Software.

Nombre de la materia Lab. Estructuras de Datos.

Nombre del alumno Jesús Manuel Juárez Pasillas.

Nombre del docente Aldonso Becerra Sánchez.

Fecha: 10/11/2021.

Práctica 29: Grafos.

Introducción:

Los grafos son estructuras de datos los cuales almacenan vértices o nodos y estos se unen con aristas las cuales representan la relación entre dos nodos. Para mostrar estas aristas es necesario de tener a la mano la matriz de adyacencia con la cual sabremos que nodos están relacionados entre sí.

Desarrollo:

Para el desarrollo de esta práctica se pide agregar funcionalidad a la clase de GrafoMtriz, en la cual se van agregar varios métodos de importancia los cuales nos darán información relevante sobre el grafo que se tiene.

El grafo matriz es un grafo en el cual los nodos se almacenan en un arreglo de vértices y las aristas se almacenan en una matriz la cual será la matriz de adyacencia del grafo. Esta matriz representa los nodos que están unidos por una arista y también representan su peso.

Los métodos agregados son:

- Elimina un vértice del grafo.
  `public boolean eliminarVertice(Object vertice).`
- Verifica si dos nodos son adyacentes.
  `public boolean esAdyacente(Object origen,Object destino).`
- Elimina una arista entre dos nodos.
  `public boolean eliminarArista(Object origen,Object destino).`
- Busca un nodo en un grafo.
  `public Vertice buscarVertice(Object vertice).`
- Verifica si el grafo contiene lazos o bucles.
  `public boolean esPseudografo().`
- Verifica si al menos dos de sus vértices están conectados entre sí por dos aristas paralelas.
  `public boolean esMultigrafo().`
- Obtiene el número de aristas de un vértice.
  `public int gradoVertice(Object vertice).`

- Determina si el grafo es dirigido o no.
  `public boolean esDirigido().`

- Lista todas aristas del grafo.
  `public void listarAristas().`

- Lista todas las aristas del vértice dado en formato (origen, destino, peso).
  `public void listarAristas(Object vertice).`

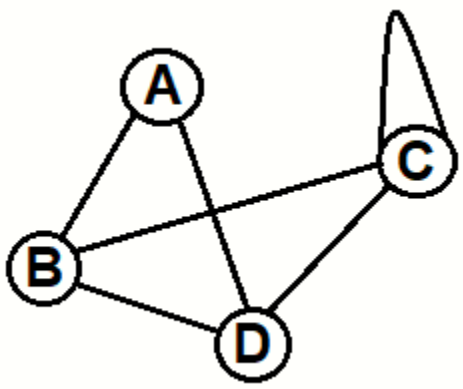
- Muestra los vértices del grafo.
  `public void listarVertices().`

Todos estos métodos utilizan métodos de las clases ArregloDatos y Tabla2D las cuales contienen los vértices y las aristas respectivamente. Utilizando estos métodos no es necesario construir un método tan grande ya que estas clases ya tienen algún método que hace lo que se requiere en un paso específico del método que se quiere hacer. Tal es el caso del método de eliminar vértice, este método tiene que eliminar el vértice del arreglo, pero también tiene que eliminar la fila y columna de las aristas. Quitar la fila y columna de las aristas es fácil teniendo los métodos de quitarFila y quitarColumna los cuales están en la clase Tabla2D y a los cuales solo es necesario pasar el índice de la fila y columna respectivamente a quitar.

Nota: Toda la documentación del proyecto esta agregada en la carpeta “doc” dentro de la carpeta del proyecto (“edylab_2021_29/doc”).

Capturas del programa funcionando:

Para realizar la prueba se hizo un solo grafo el cual quedaría así:



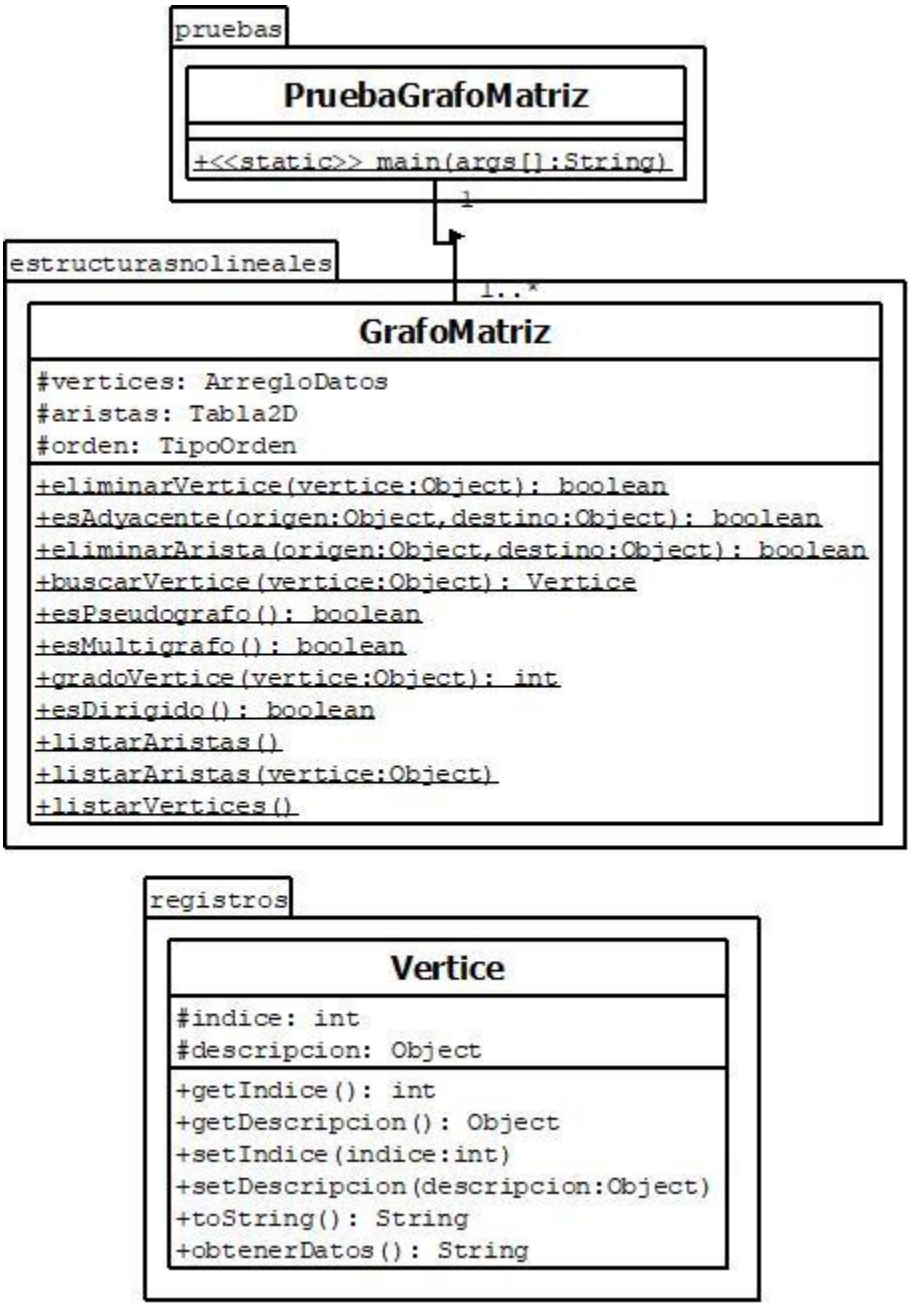
Como se puede observar es un grafo no dirigido, el cual contiene un lazo. Los grafos no dirigidos al eliminar una arista que no sea un lazo se convierten en un grafo dirigido. A este grafo se le elimina un vértice, pero vuelve a quedar igual al rehacerlo.

```
Listado de vértices:
A
B
C
D
La tabla de aristas:
0.0 1.0 0.0 1.0
1.0 0.0 1.0 1.0
0.0 1.0 0.0 1.0
1.0 1.0 1.0 0.0
Eliminar vertice (A):
Listado de vértices:
B
C
D
La tabla de aristas:
0.0 1.0 1.0
1.0 0.0 1.0
1.0 1.0 0.0

Es Adyacente (A,D): true
Es Adyacente (A,C): false
Listado de vértices:
A
B
C
D
La tabla de aristas:
0.0 1.0 0.0 1.0
1.0 0.0 1.0 1.0
0.0 1.0 0.0 1.0
1.0 1.0 1.0 0.0
Buscar vertice: C(2)
Es Pseudografo: false
Es multigrafo: true
Grado vertice (B): 3
Es dirigido: false
Listar Aristas:
[(A,B,1.0),(A,D,1.0), (B,A,1.0),(B,C,1.0),(B,D,1.0), (C,B,1.0),(C,D,1.0), (D,A,1.0),(D,B,1.0),(D,C,1.0),]
Listar Aristas (C):
(C,B,1.0),(C,D,1.0)
Listar Vertices:
A
B
C
D
```

Código agregado:

La única clase nueva es la prueba (PruebaGrafoMatriz), de ahí en mas no hay clases nuevas ni paquetes. Los únicos métodos agregados son los que aparecen en la clase GrafoMatriz los cuales están subrayados.



Pre-evaluación:

Pre-Evaluación para prácticas de Laboratorio de Estructuras de Datos	PRE-EVALUACIÓN DEL ALUMNO
CUMPLE CON LA FUNCIONALIDAD SOLICITADA.	No
DISPONE DE CÓDIGO AUTO-DOCUMENTADO.	Sí
DISPONE DE CÓDIGO DOCUMENTADO A NIVEL DE CLASE Y MÉTODO.	Sí
DISPONE DE INDENTACIÓN CORRECTA.	Sí
CUMPLE LA POO.	Sí
DISPONE DE UNA FORMA FÁCIL DE UTILIZAR EL PROGRAMA PARA EL USUARIO.	Sí
DISPONE DE UN REPORTE CON FORMATO IDC.	Sí
LA INFORMACIÓN DEL REPORTE ESTÁ LIBRE DE ERRORES DE ORTOGRAFÍA.	Sí
SE ENTREGÓ EN TIEMPO Y FORMA LA PRÁCTICA.	Sí
INCLUYE LA DOCUMENTACIÓN GENERADA CON JAVADOC.	Sí
INCLUYE EL CÓDIGO AGREGADO EN FORMATO UML.	Sí
INCLUYE LAS CAPTURAS DE PANTALLA DEL PROGRAMA FUNCIONANDO.	Sí
LA PRÁCTICA ESTÁ TOTALMENTE REALIZADA (ESPECIFIQUE EL PORCENTAJE COMPLETADO).	70%
Observaciones:	

Conclusión:

Los grafos necesitan una gran variedad de métodos con los cuales obtener información muy importante, ya que los grafos pueden representar una gran variedad de posibilidades las cuales tienen la particularidad que tienen elementos (nodos o vértices) que interactúan entre si (aristas). Sabiendo esto los grafos nos pueden proporcionar una gran variedad de información útil sobre los elementos y sus interacciones, tales como saber si dos vértices son adyacentes, si existen bucles o ciclos, etc.