

|  |
| --- |
|  |

**2020-1**

**ESTRUCTURA DE DATOS - GRAFOS**

* 1. CONSTRUYENDO CONOCIMIENTO EN LA PANDEMIA

**CIPA: CIPA I**

**Maira León Correa**

**Antonio Hurtado Aguas**

**Wilson Herrera Rincon**

**ASIGNATURA: ESTRUCTURA DE DATOS**

**DOCENTE : JOHN ARRIETA**

**TABLA DE CONTENDIOS**

1. OBJETIVOS
2. DESCRIPCIÓN DEL PROBLEMA. (Breve descripción de la situación problemática que se busca resolver)
3. MARCO CONCEPTUAL
4. METODOLOGÍA. (Fuentes y análisis de información)
5. PRESENTACIÓN DE RESULTADOS. (Análisis de información sobre el problema a la luz del enfoque teórico aplicado)
6. CONCLUSIONES (Síntesis de la solución o alternativas del problema planteado)

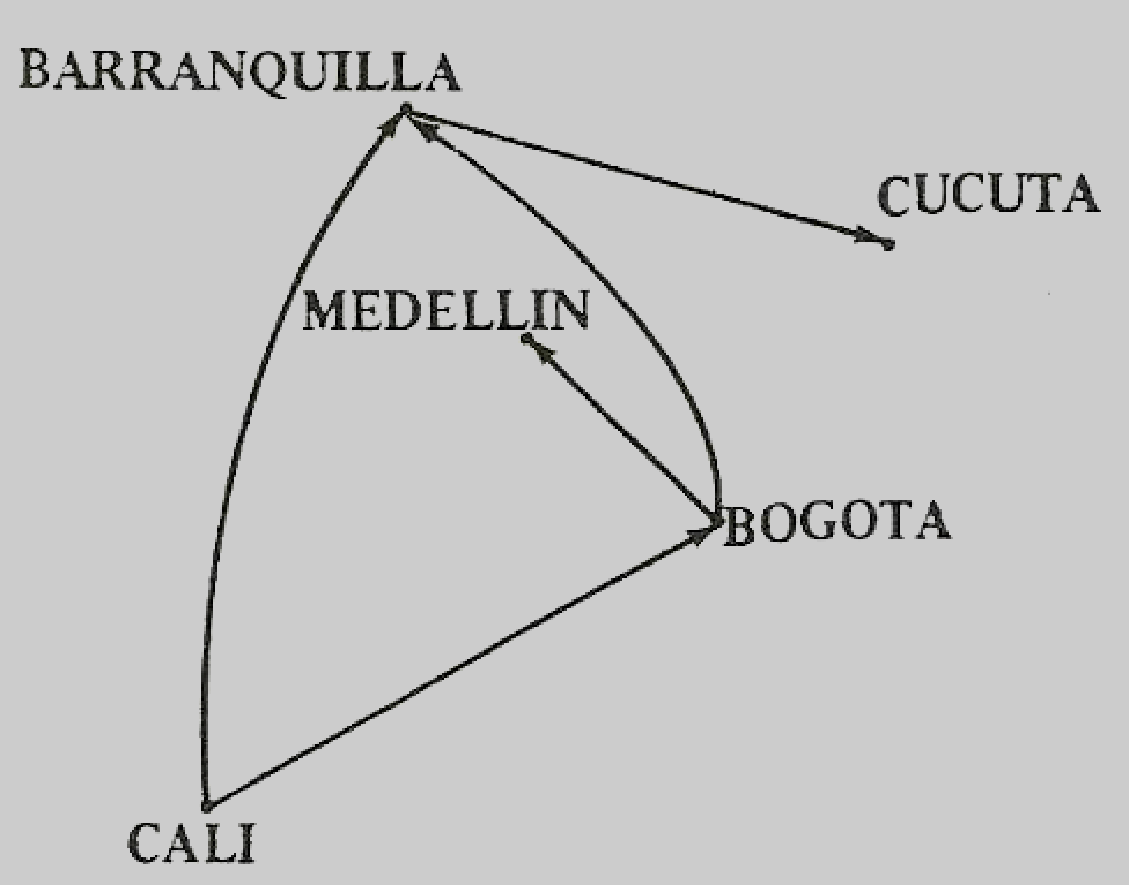
**1. OBJETIVOS**

El objetivo de la realización del presente trabajo es la aplicación del conocimiento adquirido en el curso de estructura de datos.

Utilizar el paradigma de programación orientada a objetos para implementar un software capaz de dibujar el grafo solicitado en el ejercicio correspondiente al grupo 1.

**2. DESCRIPCION DEL PROBLEMA**

Dado el siguiente grafo, Realizar un programa en Java que permita dibujar este grafo dirigido.



**3. MARCO TEORICO**

GRAFOS

4.1. DEFINICIONES

4.1.1. GRAFO

Un grafo es un conjunto, no vacío, de objetos llamados vértices conocidos también como NODOS y una selección de pares de vértices, llamados aristas que pueden ser orientados o no.

los grafos son una disciplina que en la actualidad presenta un desarrollo desenfrenado, debido a que tiene mucha importancia en el diseño de programas.

**Breve Historia de la teoría de Grafos**

El trabajo de Leonhard Euler, en 1736, sobre el problema de los puentes de Königsberg es considerado el primer resultado de la teoría de grafos.

En 1845 Gustav Kirchhoff publicó sus leyes de los circuitos para calcular el voltaje y la corriente en los circuitos eléctricos.

En 1852 Francis Guthrie planteó el problema de los cuatro colores que plantea si es posible, utilizando solamente cuatro colores, colorear cualquier mapa de países de tal forma que dos países vecinos nunca tengan el mismo color. Este problema, que no fue resuelto hasta un siglo después por Kenneth Appel y Wolfgang Haken, puede ser considerado como el nacimiento de la teoría de grafos. Al tratar de resolverlo, los matemáticos definieron términos y conceptos teóricos fundamentales de los grafos.

Inicialmente los grafos se empleaban para solventar las dificultades o problemas en diversas áreas. La teoría de los grafos puede ser utilizada para la creación de circuitos y sus diferentes estructuras, pero su mayor importancia radica en el área de sistemas o informática, ya que por medio de los grafos se brinda la posibilidad de estudiar las estructuras de redes de internet y neuronales, el manejo de las estructuras de datos y el análisis de los diversos algoritmos existentes.

El término “grafo” proviene de Grecia y se traduce al castellano como “imagen o dibujo”. Los grafos son una composición de un conjunto de objetos conocidos como “nodos o vértice”, que se relacionan con otros nodos a través de un conjunto de conexiones conocidas como “aristas”.

Los grafos permiten estudiar las relaciones que existen entre unidades que interactúan con otras.

En su totalidad, un grafo es un par de ordenado compuesto por vértices (V) y aristas (e); donde en la mayoría de los casos los vértices son de cuantificación finitas, el número de vértices que componen al grafo son conocidos como “orden”. También existe el concepto de “grado” que corresponde al número de arcos al que pertenecen de forma externa; en cuanto aristas, se maneja el concepto de “blue”, que no es más que una arista relacionada de diversas formas con el mismo nodo.

**Terminología Usada en la Teoría de Grafos**

**Vértice:** un vértice o nodo es la unidad fundamental de la que están formados los grafos está representado por los puntos convergentes del grafo.

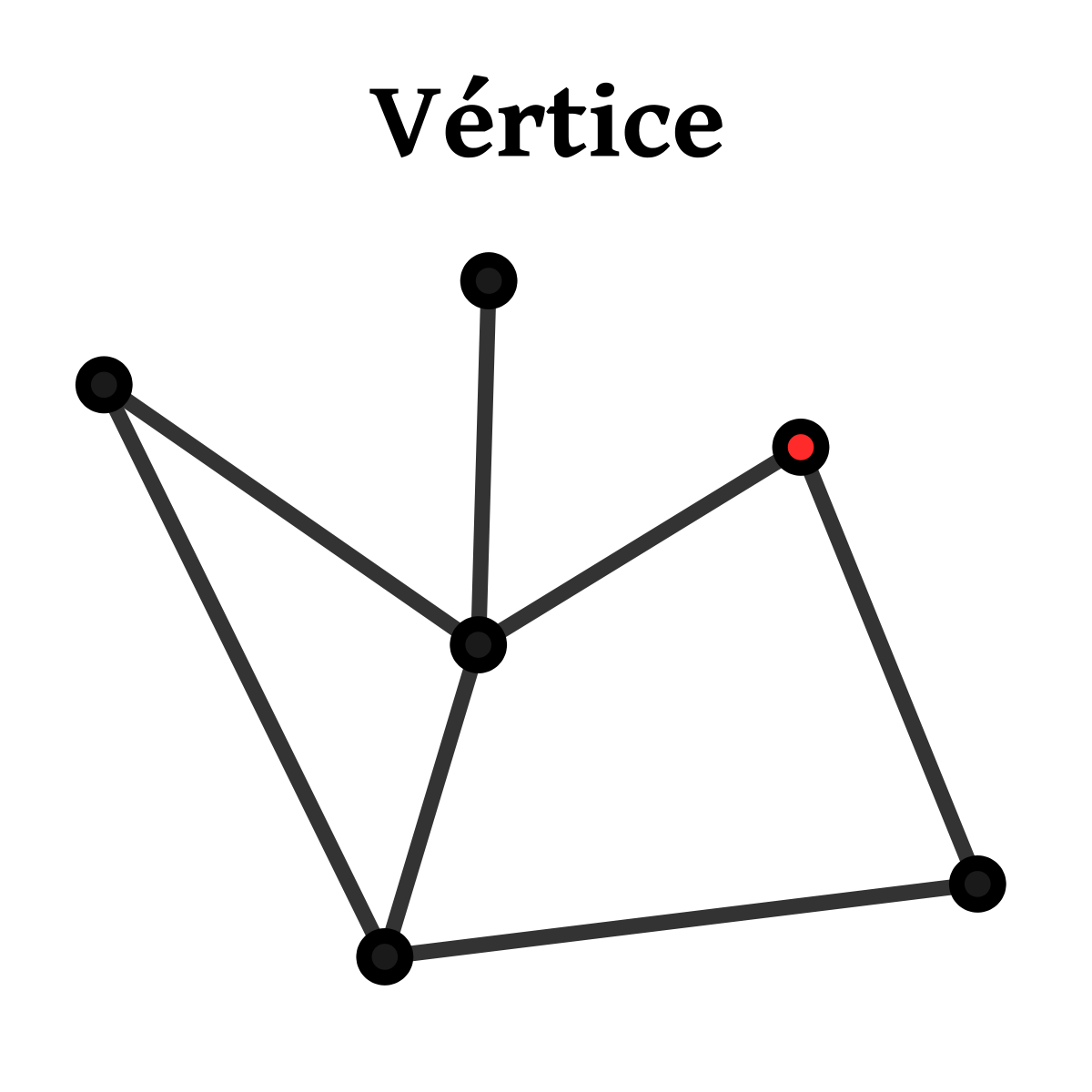


Figura 1: Vértice

**Aristas:** una arista corresponde a una relación entre dos vértices de un grafo en pocas palabras seria la línea que une dos vértices.

****

Figura 2: Arista

**Aristas Adyacentes:** la adyacencia de las aristas está determinada por si estas convergen sobre el mismo vértice.

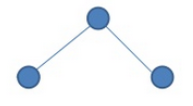


Figura 3: Aristas Adyacentes

**Aristas Paralelas o Múltiples:** Dos aristas son paralelas si tienen los mismos vértices en común.



Figura 4: Aristas múltiples

Lazo: Es una arista cuyos extremos convergen sobre el mismo vértice.



Figura 5: Lazo

**Vértice Aislado**: es un vértice sobre el cual no converge ninguna arista es decir es de grado cero.

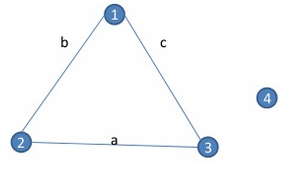


Figura 6: el vértice 4 es un vértice aislado.

**Vértice Pendiente:** es un vértice de grado uno que solo contiene una arista.



Figura 7: Vértice Pendiente Como se puede observar ambos vértices son de grado 1 es decir solo converge sobre ellos una sola arista.

**Grado o Valencia del Vértice:** es el número de aristas que inciden en el vértice.

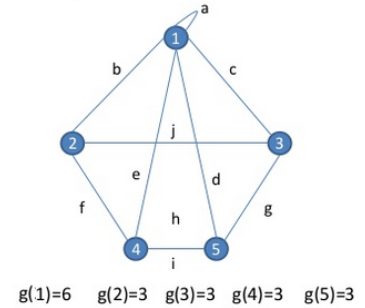


Figura 8: el pie de la figura representa el grado de valencia de cada vértice del grafo como podemos ver sobre el vértice 1 convergen 6 aristas ya que tiene 4 aristas que van desde otros vértices y un lazo.

**Grado Regular:** en un grafo simple G, se dice que tiene grado X-regular cuando todo vértice de G incide sobre X aristas y X es una constante. en síntesis si tiene el mismo número de aristas en cada vértice.

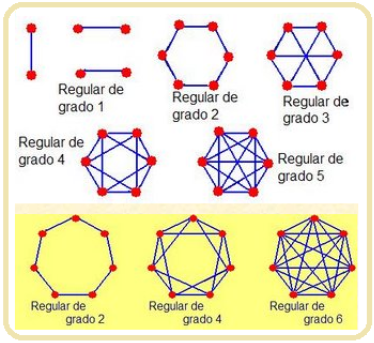


Figura 9: ejemplos de grafos regulares de diferentes grados.

4.1.2. GRAFO DIRIGIDO

Un grafo dirigido (o dígrafo) consta de un conjunto de vértices y un conjunto de arcos, a los vértices se le conoce también como nodos o puntos, y a los arcos aristas dirigidas. Los vértices de un dígrafo pueden usarse para representación de objetos.

Dentro del dígrafo se encuentra los grafos etiquetados, los cuales también dan representación a los dígrafos, en el que cada vértice o arco puede tener una etiqueta asociada. Y una etiqueta se representa con un nombre, costo o valor de algún tipo de dato dado.

A veces un dígrafo es denominado dígrafo simple para distinguirlo de los multígrafo dirigido, donde los arcos conforman un multiconjunto, en lugar de un conjunto.

Tipología básica: un arco e = (x, y) se considera dirigido desde x hasta y; y se denomina cabeza y x se denomina cola del arco, y también se denomina sucesor directo de x, se denomina a x un predecesor directo de y.

Un grafo dirigido G es llamado simétrico si, para cualquier arco que pertenece a G, el arco invertido correspondiente también pertenece a G. Un grafo dirigido simétrico y sin bucles es equivalente a un grafo no dirigido; basta con reemplazar cada par de arcos dirigidos por un solo arco no dirigido.

Una orientación de un grafo simple no dirigido se obtiene al asignar una orientación a cada uno de los arcos existentes. Un grafo dirigido construido de esta manera se denomina un grafo orientado. Una manera de distinguir entre un grafo simple dirigido y un grafo orientado es que si x e y son vértices, un grafo simple dirigido permite tanto (x, y) como (y,x) entre sus arcos, mientras que solo una de las dos posibilidades es admitida en un grafo orientado.

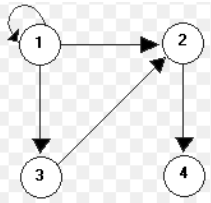


Figura 10: Grafo Dirigido

4.1.3. ADYACENCIA

Es una forma mucho más fácil para guardar los datos de los nodos, mediante la utilización de los vectores que indiquen los nodos; de tal manera que los arcos entre los nodos se observan como relaciones entre los índices.

La adyacencia es una matriz que muestra de una forma más precisa cómo está compuesto un grafo, esto es que donde se coloque un 1 se representa como una arista que une los dos nodos, y con 0 donde no hay ninguna unión. De esta manera se puede obtener un grafo de una matriz adyacencia o viceversa.

una de las ventajas de la matriz de adyacencia es que es simple, y para grafos pequeños es fácil ver qué nodos están conectados a otros nodos.

La matriz de adyacencia es una buena implementación para un grafo cuando el número de aristas es grande. Pero ¿qué entendemos por grande? ¿Cuántas aristas se necesitarían para llenar la matriz? Puesto que hay una fila y una columna para cada vértice en el grafo, el número de aristas requeridas para llenar la matriz es |V|2. Una matriz está llena cuando cada vértice está conectado a todos los otros vértices.

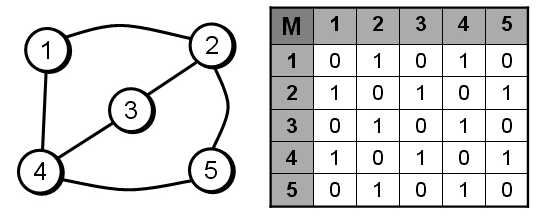


Figura 11: Grafo con su matriz de adyacencia.

4.1.4. INCIDENCIA

En los grafos dirigidos la incidencia se conoce como una matriz, se representa cuando dos aristas son incidentes si tienen un mismo vértice como extremo.

Ejemplo: si decimos que tenemos e = (x, y), entonces la matriz es incidente a los vértices “x” e “y”.

La matriz incidencia permite representar tanto a las aristas paralelas como a los lazos. Si la matriz de incidencia sólo contiene ceros y unos (matriz binaria). Como cada arista incide exactamente en dos vértices, cada columna tiene exactamente dos unos. La cantidad de unos que aparece en cada fila es igual al grado del vértice correspondiente. Una fila compuesta sólo por ceros corresponde a un vértice aislado.

Dado un grafo simple G = (V, E) con n=|V| vértices {v1,. .. , vn} y m=|E| aristas {e1, …, em}, su matriz de incidencia es la matriz B de orden nxm, B(G)=(bij), donde bij=1 si vi es incidente con ej y bij=0 en caso contrario.

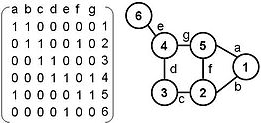


Figura 12: Grafo con su matriz de incidencia.

4.1.5. COMPONENTES CONECTADAS SEPARADAMENTE

En los grafos dirigidos la incidencia se conoce como una matriz, se representa cuando dos aristas son incidentes si tienen un mismo vértice como extremo.

Ejemplo: si decimos que tenemos e = (x,y), entonces la matriz es incidente a los vértices “x” y “y”.

La matriz incidencia permite representar tanto a las aristas paralelas como a los lazos. Si la matriz de incidencia sólo contiene ceros y unos (matriz binaria). Como cada arista incide exactamente en dos vértices, cada columna tiene exactamente dos unos. La cantidad de unos que aparece en cada fila es igual al grado del vértice correspondiente. Una fila compuesta sólo por ceros corresponde a un vértice aislado.

Dado un grafo simple G = (V, E) con n=|V| vértices {v1,. .. , vn} y m=|E| aristas {e1, …, em}, su matriz de incidencia es la matriz B de orden nxm, B(G)=(bij), donde bij=1 si vi es incidente con ej y bij=0 en caso contrario.

**4. METODOLOGIA**

En el desarrollo del presente proyecto utilizamos el entorno integrado de desarrollo NetBeans 8.2 utilizando el paradigma de programación orientada a objetos y valiéndonos de las siguientes clases del JDK.

* java.awt.Graphics
* java.awt.event.MouseListener
* java.awt.event.MouseEvent
* java.awt.Rectangle
* java.awt.Point
* javax.swing.JOptionPane
* javax.swing.JFrame;

en el presente proyecto utilizamos los pilares de la programación orientada a objetos como encapsulamiento, herencia, polimorfismo.

Se utiliza la clase Graphics para dibujar sobre un “lienzo” representado en un “Marco” JFrame de 600x600 px(Pixeles) . El cual en el método main tiene la cláusula EXIT\_ON\_CLOSE que sale automáticamente al cerrar el marco.

Nuestro proyecto cuenta con 4 clases Grafo, Lienzo, Nodo, Enlace

En la clase Grafo contamos con el método Main en el que se establece el tamaño del lienzo y la forma en que finaliza nuestro programa.

En la clase Nodo encontramos el método Pintar que dibuja un Ovalo sobre el lienzo para representar el nodo, imprime el dato centrado en ese ovalo y valiéndonos del clic derecho en los nodos los podemos relacionar mediante una línea.

La clase Lienzo contiene el evento MouseListener con los métodos para obtener los puntos donde se dibujará el nodo por medio del clic del mouse.

La clase enlace es donde ocurre la magia, esta contiene los métodos getter que toman las “Coordenadas” donde se va a dibujar el objeto y las fórmulas matemáticas y condicionales que establecen la manera en la que se dibujara el nodo.

**5. PRESENTACION DE RESULTADOS**

Presentaremos los detalles de las clases mencionadas en la metodología.

Clase Enlace

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. package Grafo;
8. import java.awt.Graphics;
10. /\*\*
11. \*
12. \* @author Antonio
13. \*/
14. public class Enlace {
16. private int x1, y1, x2, y2;
17. private String nombre;
19. public Enlace(int x1, int y1, int x2, int y2, String nombre) {
20. this.x1 = x1;
21. this.x2 = x2;
22. this.y1 = y1;
23. this.y2 = y2;
24. this.nombre = nombre;
25. }
27. public Enlace(int x1, int y1, int x2, int y2) {
28. this.x1 = x1;
29. this.y1 = y1;
30. this.x2 = x2;
31. this.y2 = y2;
32. this.nombre = "";
33. }
35. public void pintar(Graphics g) {
36. g.drawLine(x1, y1, x2, y2);
37. if ((x1 > x2) && (y1 > y2)) {
38. g.drawString(nombre, x1 - Math.abs((x1 - x2) / 2), y1 - Math.abs((y1 - y2) / 2));
39. }
40. if ((x1 < x2) && (y1 < y2)) {
41. g.drawString(nombre, x2 - Math.abs((x1 - x2) / 2), y2 - Math.abs((y1 - y2) / 2));
42. }
43. if ((x1 > x2) && (y1 < y2)) {
44. g.drawString(nombre, x1 - Math.abs((x1 - x2) / 2), y2 - Math.abs((y1 - y2) / 2));
45. }
46. if ((x1 < x2) && (y1 > y2)) {
47. g.drawString(nombre, x2 - Math.abs((x1 - x2) / 2), y1 - Math.abs((y1 - y2) / 2));
48. }
49. }
51. public int getX1() {
52. return x1;
53. }
55. public void setX1(int x1) {
56. this.x1 = x1;
57. }
59. public int getY1() {
60. return y1;
61. }
63. public void setY1(int y1) {
64. this.y1 = y1;
65. }
67. public int getX2() {
68. return x2;
69. }
71. public void setX2(int x2) {
72. this.x2 = x2;
73. }
75. public int getY2() {
76. return y2;
77. }
79. public void setY2(int y2) {
80. this.y2 = y2;
81. }
82. }

Clase Nodo

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. package Grafo;
8. import java.awt.Graphics;
10. /\*\*
11. \*
12. \* @author Antonio
13. \*/
14. public class Nodo {
16. private int x , y ;
17. private String nombre;
18. public static final int d=60;
20. public Nodo(int x , int y , String nombre) {
21. this.x = x;
22. this.y = y ;
23. this.nombre = nombre;
24. }
26. public void pintar (Graphics g){
27. g.drawOval(this.x - d/2, this.y - d/2 , d , d);
28. g.drawString(nombre, x, y);
30. }
32. public int getX() {
33. return x;
34. }
36. public void setX(int x) {
37. this.x = x;
38. }
40. public int getY() {
41. return y;
42. }
44. public void setY(int y) {
45. this.y = y;
46. }
48. }

Clase Grafo

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. package Grafo;
8. import javax.swing.JFrame;
10. /\*\*
11. \*
12. \* @author Antonio
13. \*/
14. public class Grafo {
16. /\*\*
17. \* @param args the command line arguments
18. \*/
19. public static void main(String[] args) {
20. // TODO code application logic here
22. JFrame ventana = new JFrame("GRAFO");
23. ventana.add(new Lienzo());
24. ventana.setSize(600,600);
25. ventana.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
26. ventana.setVisible(true);
28. }
30. }

Clase Lienzo

1. \* To change this license header, choose License Headers in Project Properties.
2. \* To change this template file, choose Tools | Templates
3. \* and open the template in the editor.
4. \*/
5. package Grafo;
7. import java.awt.Graphics;
8. import java.awt.Point;
9. import java.awt.Rectangle;
10. import java.awt.event.MouseEvent;
11. import java.awt.event.MouseListener;
12. import java.util.Stack;
13. import java.util.Vector;
14. import javax.swing.JOptionPane;
15. import javax.swing.JPanel;
17. /\*\*
18. \*
19. \* @author Antonio
20. \*/
21. public class Lienzo extends JPanel implements MouseListener{

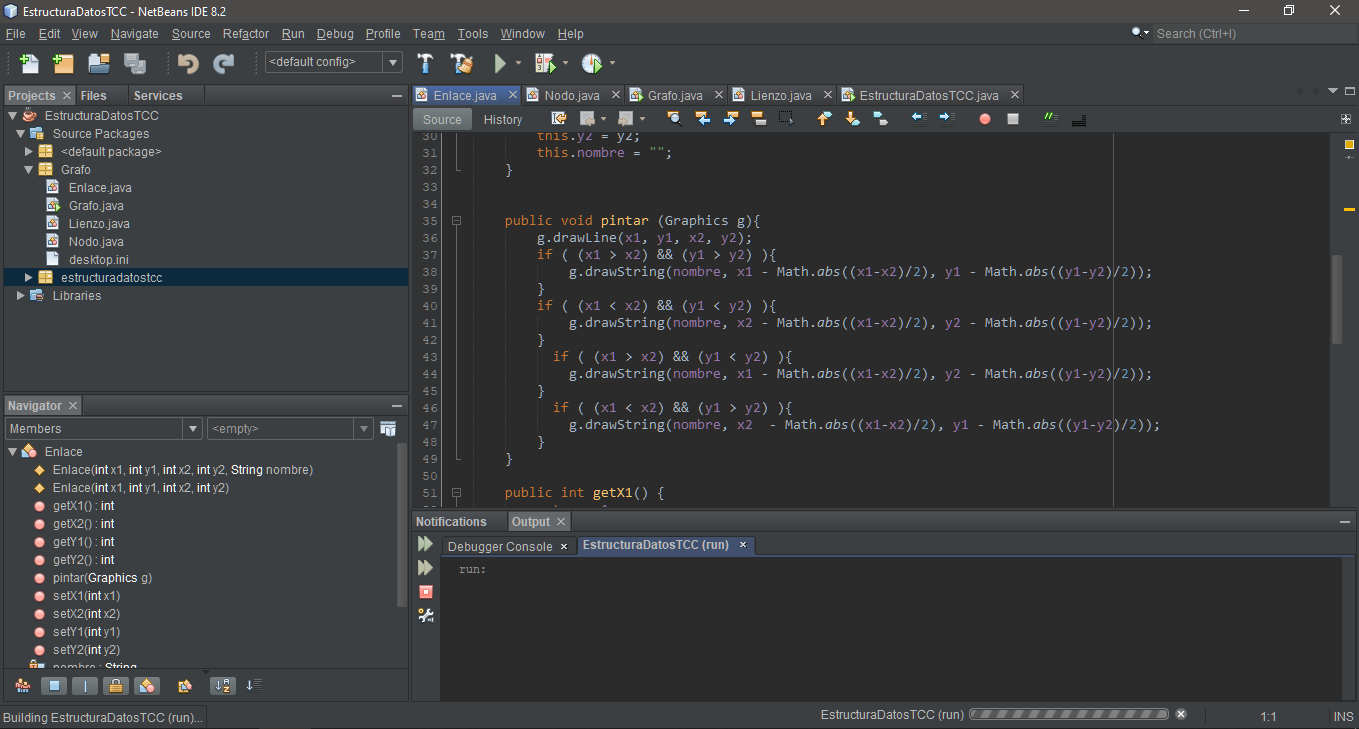
24. private Vector<Nodo> vectorNodos;
25. private Vector<Enlace> vectorEnlaces;
26. private Point p1,p2;
28. public Lienzo() {
29. this.vectorNodos = new Vector<>();
30. this.vectorEnlaces = new Vector<>();
31. this.addMouseListener(this);
32. }
34. @Override
35. public void paint(Graphics g){
36. for (Nodo nodos : vectorNodos) {
37. nodos.pintar(g);
38. }
39. for (Enlace enlaces : vectorEnlaces) {
40. enlaces.pintar(g);
41. }
43. }

46. @Override
47. public void mouseClicked(MouseEvent e) {
48. if (e.getButton() == MouseEvent.BUTTON1) {
49. String nombre = JOptionPane.showInputDialog("ingrese nombre al nodo");
50. this.vectorNodos.add(new Nodo(e.getX(), e.getY(), nombre));
51. repaint();
52. }
53. if (e.getButton() == MouseEvent.BUTTON3) {
54. for (Nodo nodo : vectorNodos) {
55. if(new Rectangle(nodo.getX() - Nodo.d/2, nodo.getY() - Nodo.d/2 , Nodo.d, Nodo.d).contains(e.getPoint())){
56. if (p1==null){
57. p1= new Point(nodo.getX(), nodo.getY());
58. }else{
59. p2 = new Point(nodo.getX(), nodo.getY());
60. // String nombre = JOptionPane.showInputDialog("ingrese peso al enlace");
61. this.vectorEnlaces.add(new Enlace(p1.x, p1.y, p2.x, p2.y));
62. repaint();
63. p1=null;
64. p2=null;
65. }
66. }
67. }
68. }

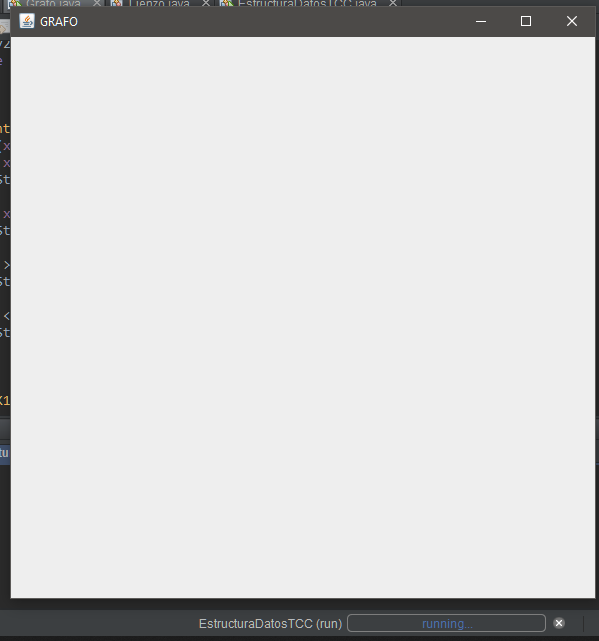

72. }
74. @Override
75. public void mousePressed(MouseEvent e) {
76. }
78. @Override
79. public void mouseReleased(MouseEvent e) {
80. }
82. @Override
83. public void mouseEntered(MouseEvent e) {
84. }
86. @Override
87. public void mouseExited(MouseEvent e) {
88. }
90. }

A continuación, pegaremos capturas de pantalla para evidenciar el funcionamiento de la aplicación

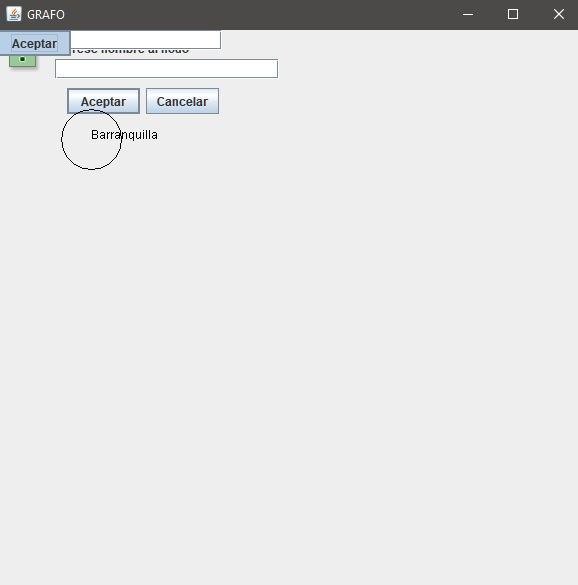
Iniciamos la compilacion



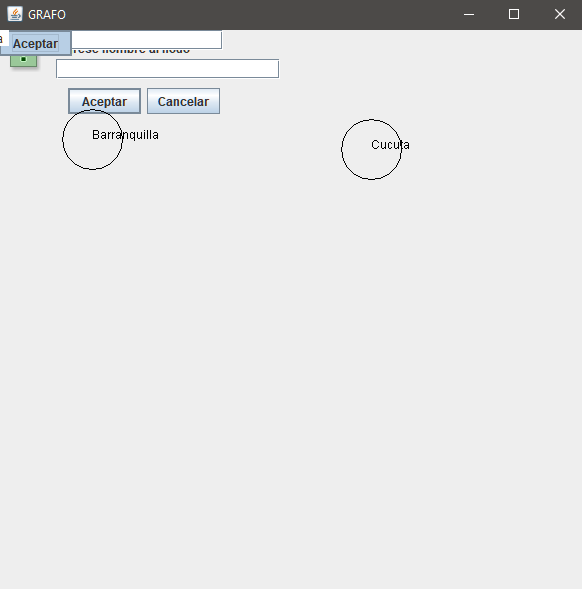
Obtenemos nuestro lienzo



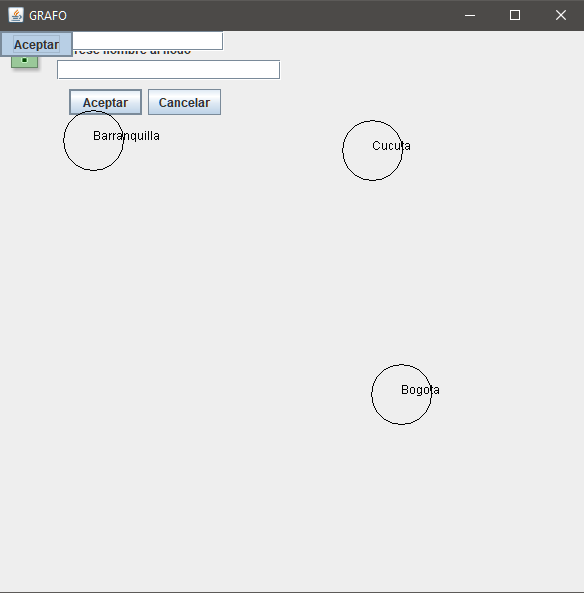
Dibujamos el primer nodo



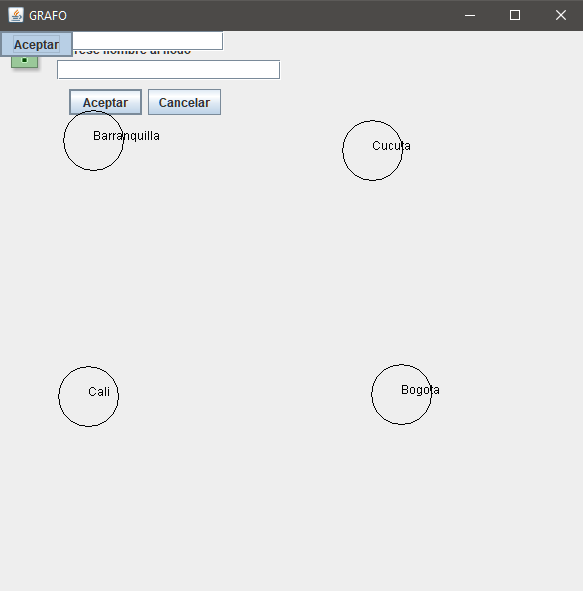
Segundo Nodo



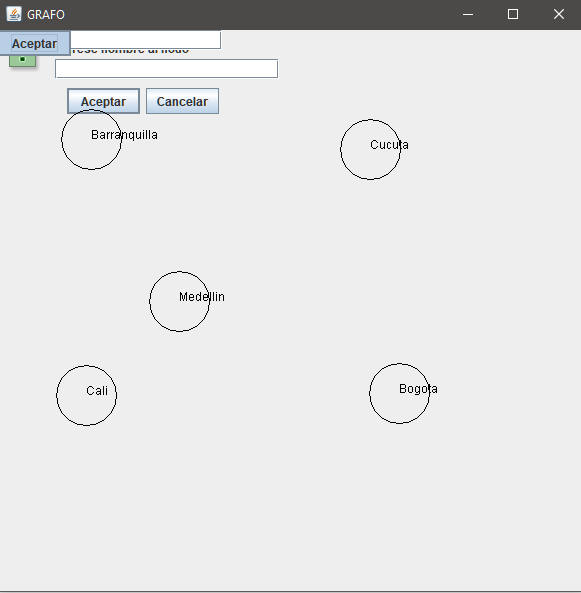
Tercer nodo



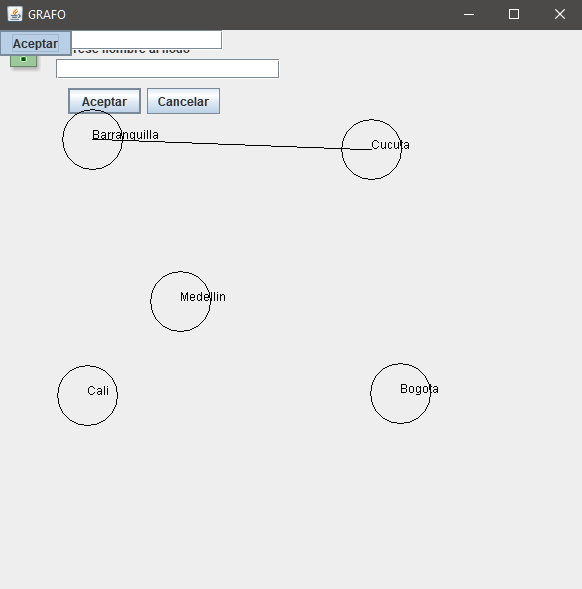
Cuarto Nodo



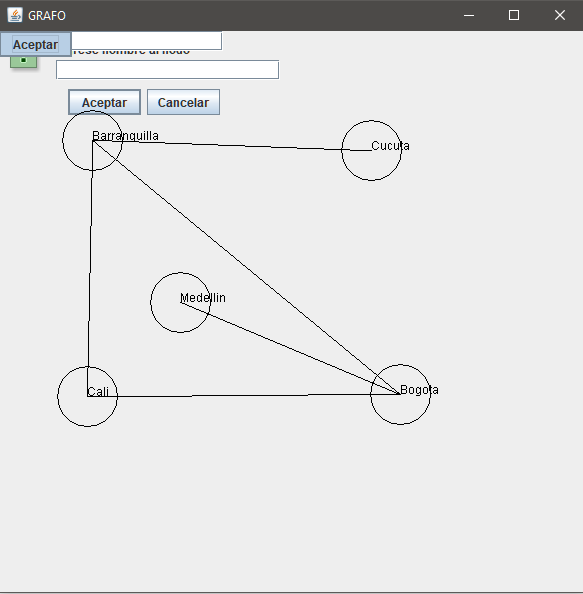
Quinto Nodo



Línea 1



Todas las Líneas



**6. CONCLUSIONES**

Fue muy gratificante la realización del presente trabajo ya que nos permito articularnos como grupo y ampliar nuestro conocimiento en la utilización de java al utilizar clases que no habíamos utilizado antes.

Además de lo anterior nos permitió ahondar en la teoría de grafos, reforzando conceptos como adyacencia e incidencia.

Agradecemos al profesor por colocar este tipo de trabajos ya que además de mejorar nuestras capacidades teóricas nos llevan más allá en el camino que hemos emprendido como futuros ingenieros de software.