Universidad de Guadalajara.
Centro Universitario de Ciencias Exactas e Ingenierías.

División de Electrónica y Computación.
Sem. de solución de problemas de algoritmia.

Sandoval Márquez Anthony Esteven
215660767

Planteamiento del problema.

Diseñar un sistema computacional con la capacidad de representar escenarios, utilizando grafos. Los escenarios son imágenes que cuentan con círculos negros bloques de otros colores. Para la construcción del grafo, los círculos representarán vértices, estos tienen la información del círculo, como su coordenada y radio. Las aristas entre vértices existirán solamente si desde el centro del circulo que representa el vértice “origen”, se puede trazar una línea recta hasta el centro del círculo que representa otro vértice “destino” sin ser obstruida por ninguna otra figura en la imagen. Los “obstáculos”, como otros círculos pueden hacer que un par de círculos no puedan representarse como un par de vértices con una arista que los une. El sistema debe poder encontrar el par de centros de círculos más cercanos, utilizando una estrategia de fuerza bruta para este problema.

Objetivos.

1. Comprender e implementar grafos, usando memoria dinámica.

2. Analizar e implementar algoritmos de fuerza bruta.

Marco teórico.

1. Grafos (teoría e implementación computacional).

Un grafo es una composición de un conjunto de objetos conocidos como nodos 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.

Podemos representar diversas situaciones o elementos con grafos. Estos son extraordinariamente útiles en situaciones complejas, es por esto, que es común conseguir la implementación de análisis de grafos en estudios de ciencias exactas, ciencias sociales y en aplicaciones informáticas.

Grafo dirigido

Un grafo dirigido conocido también como dígrafo consta de un conjunto de vértices y aristas donde cada arista se asocia de forma unidireccional a través de una flecha con otro.

Las aristas dependiendo de su salida o ingreso reciben la calificación de entrante o saliente, la condición común, es que siempre tienen un destino hacia un nodo.

Grafo no dirigido

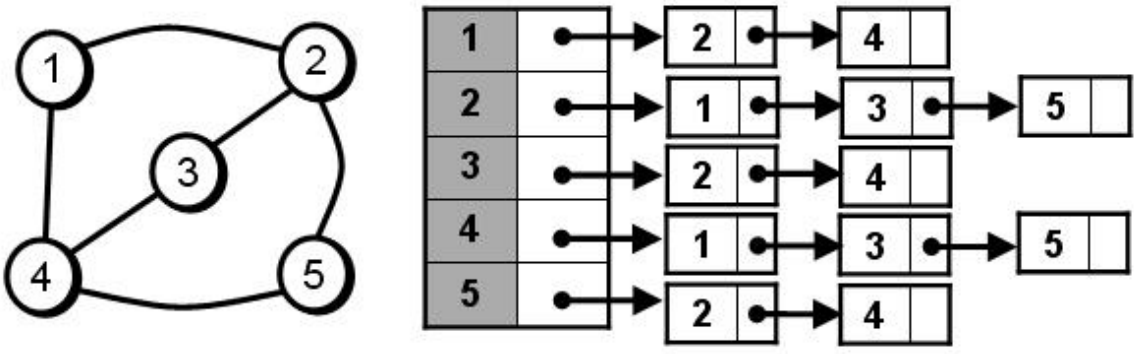
Los grafos no dirigidos son aquellos que constan un conjunto de vértices que están conectados a un conjunto de aristas de forma no direccional.

Esto significa que una arista puede indistintamente recorrerse desde cualquiera de sus puntos y en cualquier dirección.

Grafos etiquetados

Esta clasificación es denominada como grafos etiquetados o grafos dirigidos con pesos. Este tipo de grafos concentran aristas que pueden poseer información adicional donde podemos reflejar nombres, costos, valores u otros datos.

Ejemplo de implementación



2. Algoritmos de fuerza bruta.

Los algoritmos de Fuerza Bruta son capaces de encontrar la solución a cualquier problema por complicado que sea. Su fundamento es muy simple, probar todas las

posibles combinaciones, recorrer todos los caminos hasta dar con la situación que es igual que la solución. No le importa iniciar caminos malos o muy malos, al llegar a su final y ver que su destino no es la solución, se iniciará otro camino en busca del que conduzca a ella.

Closest Pair Of Points.

En geometría computacional, el problema del par de puntos más cercano es un problema clásico donde "Dados n puntos en un espacio métrico, se pide encontrar un par de puntos con la distancia más pequeña entre ellos". El problema del par de puntos más cercano en el plano euclidiano fue de los primeros problemas tratados en el estudio sistemático de la complejidad computacional de algoritmos geométricos.

```
minDist = infinito
para i = 1 a longitud(P) - 1
 para j = i + 1 a longitud(P)
  dist = distancia(P[i], P[j])
  si dist < minDist:
   minDist = dist
   closestPair = (i, j)
devolver closestPair
```

Bresenham's Line Algorithm

El algoritmo de línea de Bresenham es un algoritmo de dibujo de línea que determina los puntos de un ráster *n*- dimensional que deben seleccionarse para formar una aproximación cercana a una línea recta entre dos puntos . Se usa comúnmente para dibujar primitivas de línea en una imagen de mapa de bits (por ejemplo, en una pantalla de computadora), ya que solo usa suma, resta y desplazamiento de bits de enteros , todas las cuales son operaciones muy baratas en arquitecturas de computadora estándar . Es un algoritmo de error incremental . Es uno de los primeros algoritmos desarrollados en el campo de los gráficos por computadora.. Una extensión al algoritmo original se puede utilizar para dibujar círculos .

```
  Funcion LineaBresenham( X1, Y1, X2, Y2)
      // 0 - Distancias que se desplazan en cada eje
      dY = (Y2 - Y1)
      dX = (X2 - X1)

      // 1 - Incrementos para las secciones con avance inclinado
      Si (dY >= 0) luego
          IncYi = 1
```

```
      Sino
          dY = -dY
          IncYi = -1
      Fin si

      Si (dX >= 0) luego
          IncXi = 1
      Sino
          dX = -dX
          IncXi = -1
      Fin si

      // 2 - Incrementos para las secciones con avance recto:
      Si (dX >= dY) luego
          IncYr = 0
          IncXr = IncXi
      Sino
          IncXr = 0
          IncYr = IncYi

          // Cuando dy es mayor que dx, se intercambian, para
reutilizar el mismo bucle.
          // ver octantes blancos en la imagen encima del código
          k = dX: dX = dY: dY = k
      Fin si

      // 3  - Inicializar valores (y de error).
      X = X1: Y = Y1
      avR = (2 * dY)
      av = (avR - dX)
      avI = (av - dX)

      // 4  - Bucle para el trazado de las línea.
      Hacer
          DibujarPixel(X, Y, Color) // Como mínimo se dibujará siempre
1 píxel (punto).
          Mensaje(av + " ") // (debug) para ver los valores de error
global que van apareciendo.
          Si (av >= 0) luego
              X = (X + IncXi)     // X aumenta en inclinado.
              Y = (Y + IncYi)     // Y aumenta en inclinado.
              av = (av + avI)     // Avance Inclinado
          Sino
              X = (X + IncXr)     // X aumenta en recto.
              Y = (Y + IncYr)     // Y aumenta en recto.
              av = (av + avR)     // Avance Recto
          Fin si
      Repetir hasta que (X = X2) // NOTA: La condición de 'Repetir
Hasta', se debe cambiar si se elige 'Repetir Mientras'
   Fin funcion
```

DrawLine

Dibuja una línea que conecta los dos puntos especificados por los pares de coordenadas.

TextBox

Representa un control de cuadro de texto de Windows.

Bibliografía:

O. (2020, 10 marzo). *Qué son los grafos*. GraphEverywhere. https://www.grapheverywhere.com/que-son-los-grafos/

Algoritmo De Fuerza Bruta - 603 Palabras | Monografías Plus. (s. f.). Monografias. Recuperado 11 de abril de 2021, de https://www.monografias.com/docs/Algoritmo-De-Fuerza-Bruta-FKCKDRAZMY#:%7E:text=Los%20algoritmos%20de%20Fuerza%20Bruta,es%20igual%20que%20la%20soluci%C3%B3n.

colaboradores de Wikipedia. (2021, 31 marzo). *Algoritmo de Bresenham*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Algoritmo_de_Bresenham

D. (s. f.). *Graphics.DrawLine Method (System.Drawing)*. Microsoft Docs. Recuperado 11 de abril de 2021, de https://docs.microsoft.com/en-us/dotnet/api/system.drawing.graphics.drawline?view=net-5.0

D. (s. f.-b). *TextBox Class (System.Windows.Forms)*. Microsoft Docs. Recuperado 11 de abril de 2021, de https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.textbox?view=net-5.0

Desarrollo.

Para el desarrollo de esta actividad reutilice el proyecto realizado en la Actividad 1, al cual le agregue lo necesario para cumplir los requerimientos funcionales de esta actividad, la Actividad 2.

Primero decidí implementar el algoritmo de fuerza bruta “Closest Pair Of Poits”, que básicamente lo que hace es encontrar los dos puntos más cercanos en el bitmap, es este caso esos dos puntos son los centros en’ x’ e ‘y’ de los círculos encontrados. Para desarrollar este algoritmo primero investigue en internet el pseudocodigo y después lo adapte al lenguaje C# como una función que devuelve los índices de los círculos que están mas cerca, estos índices están relacionados a la lista de círculos que ya estaba creada en la actividad anterior. Al tener la función ya solo la utilizo cuando presiono un botón en la interfaz, además de colorearlos de color rojo en la imagen para hacerlos notar.

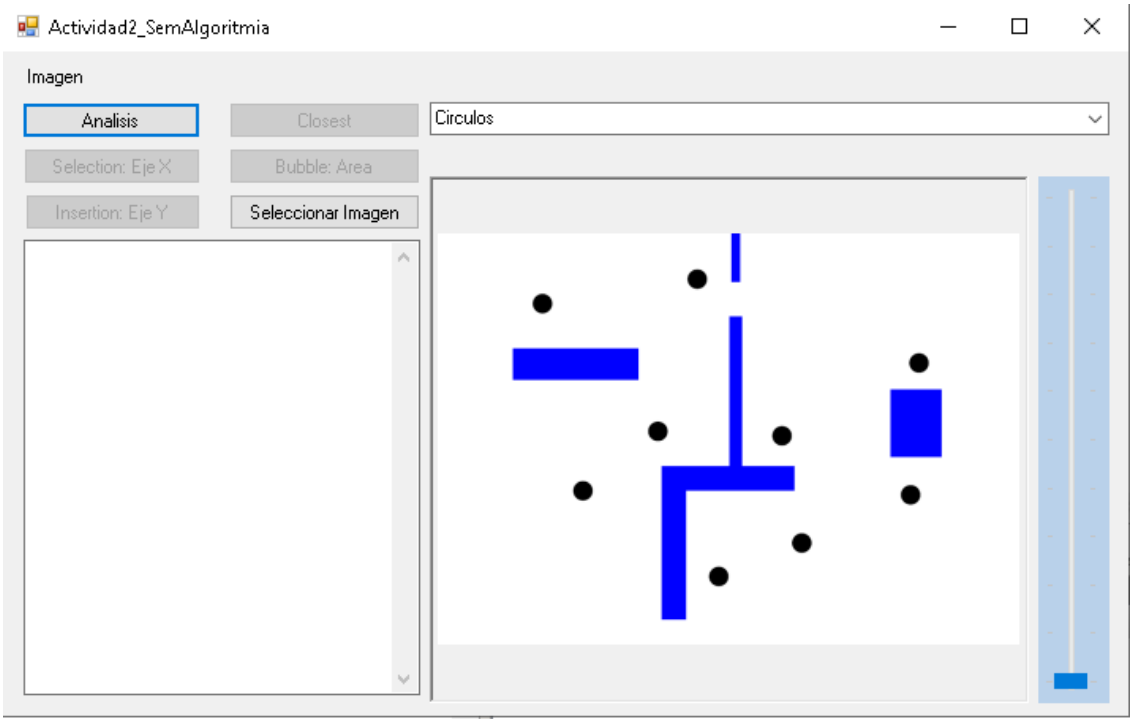
Después me puse a investigar una forma de dibujar una línea entre dos puntos, a lo cual encontré el método DrawLine, al cual se le pasa por parámetros un Pen con un color y los dos puntos en' x' e 'y' para dibujar una línea entre ellos. Me di cuenta que este método no me sería muy útil al momento de detectar si una línea choca con un obstáculo, entonces determine que lo que necesitaría seria hacer un método a mano que recorra el camino desde un punto a otro para ir comprobando si hay algún obstáculo o no. Para la creación de esta funcionalidad también me puse a investigar a lo cual obtuve información sobre un algoritmo llamado "Bresenham's Line Algorithm", el cual me permitiría recorrer las distancias entre dos puntos dados. Al implementar este algoritmo ya solo comprobé en cada pixel que recorría si este era igual al color de los obstáculos, en ese caso retornaba true y por lo tanto no se pintaba la línea ya que esto significaba que había un obstáculo. Estas comprobaciones se realizan mientras se recorre la lista de círculos encontrados.

Para la implementación del grafo dinámico necesite crear tres clases, Grafo, Arista y Vértice. La clase Vértice tendría un vértice siguiente, una arista adyacente y el circulo. La clase Arista, una arista siguiente y un vértice adyacente. Y para terminar la clase Grafo tendría un vértice inicial. En grafo es donde implemente todos los métodos relacionados al manejo del grafo, entre ellos esta uno para inicializar el grafo, para comprobar si esta vacío el grafo, obtener el tamaño del grafo, obtener un vértice a base de una referencia, insertar una arista, insertar un vértice y mostrar la lista de adyacencia. Al tener ya la implementación solamente lo que hago es crear un objeto de grafo, insertar los vértices que en este caso serían los círculos encontrados y después insertar las aristas respecto a los círculos que se conectan cuando no hay obstáculos.

Para mostrar la información del grafo utilice un TextBox, al cual le agregue la información en el método que está en el grafo, el que muestra la lista de adyacencia

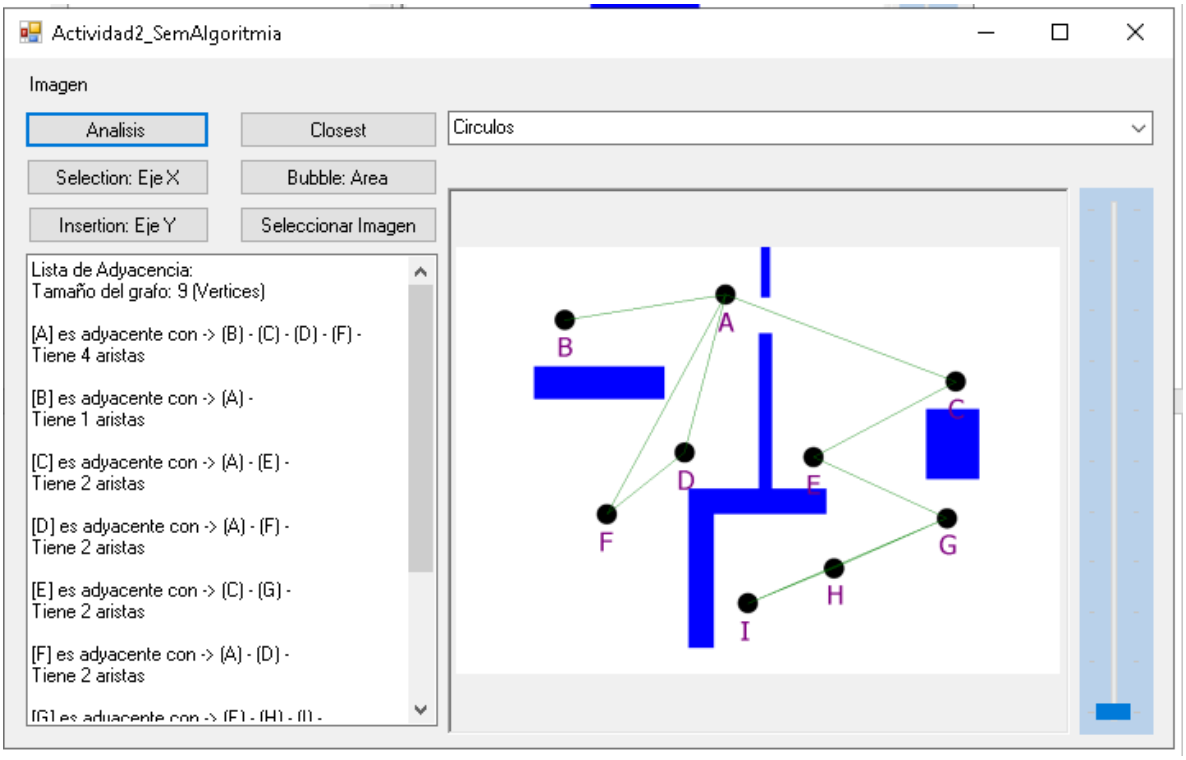
Pruebas y resultados.

Interfaz Principal

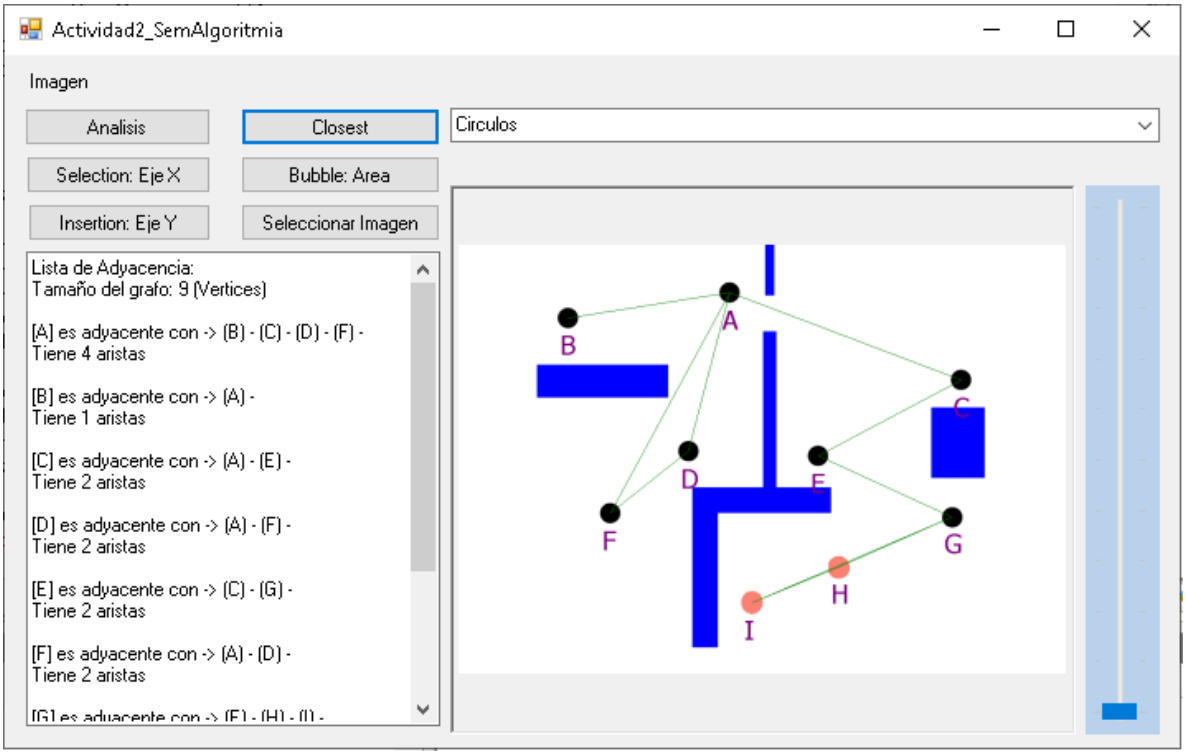


Cuando se le da clic a botón “Analisis” realiza lo siguiente:

- Busca los círculos y sus centros
- Guarda la información de los círculos en una lista de círculos
- Se les ponen etiquetas a los círculos
- Se carga la información de los círculos en el ComboBox
- Se recorre la lista de círculos y se comprueba si hay algún obstáculo entre todos los pares de círculos que existen. Si no hay algún obstáculo se dibuja una línea entre ellos.
- Se llena el grafo respecto a los círculos conectados con aristas
- Se muestra la información el grafo en el TextBox



Cuando se da clic en el botón “Closes” busca los pares de círculos más cercanos entre sí, se colorean:



Conclusiones.

En esta actividad descubrí dos algoritmos que de seguro me serán utilices en algún momento, hace tiempo me hubiera sido de utilidad uno de ellos, aunque supongo que hay otros mejores en cuanto a rendimiento. Me pareció entretenido e interesante realizar estas actividades ya que este tipo de temas los puedo aplicar en un futuro en algún proyecto importante.

También aprendí a hacer un grafo dinámico, pues en el semestre pasado no realice ninguno, entonces pues desconocía bastante la forma de hacerlo, pero investigando logre desarrollarlo

El mayor problema que se me presento fue el de recorrer los pixeles de un punto a otro, no logre comprender del todo el algoritmo que use, pero tengo una idea de cómo funciona

Comprendí un poco del funcionamiento del componente TextBox.

Apéndice(s).

Método para el algoritmo Closest Pair Of Points

```
Point getClosestPairOfPoints(){

        Point closestPoints = new Point(0, 0);
        double distance;
        double disMin = Double.PositiveInfinity;

        for(int i = 0; i < circulos.Count; i++){
           for(int j = i + 1; j < circulos.Count; j++){
              distance = Math.Sqrt(Math.Pow(circulos[i].getCentroX() - circulos[j].getCentroX(), 2)
                           + Math.Pow(circulos[i].getCentroY() - circulos[j].getCentroY(), 2));
              if(distance < disMin){
                 disMin = distance;
                 closestPoints = new Point(i, j);
              }

           }
        }

        return closestPoints;

     }
```

Boton “Closest”

```
void Btn_ClosestClick(object sender, EventArgs e)
     {
        Point p = getClosestPairOfPoints();
```

```
        bmpList  = new Bitmap(c1);
        imagen_PBox.Image = bmpList;

        colorearRespecto_X(circulos[p.X].getIniX(), circulos[p.X].getCentroY(), bmpList);
        colorearRespecto_X(circulos[p.Y].getIniX(), circulos[p.Y].getCentroY(), bmpList);

        etiquetasCirculos(36, 50, bmpList, 30);
        drawLines(bmpList);
    }
```

Método para el algoritmo Bresenham's Line Algorithm

```
bool isColision(Bitmap bmp, Circulo origen, Circulo destino){

        //Algoritmo: Bresenham's Line Algorithm

        int x = origen.getCentroX();
        int y = origen.getCentroY();
        int x2 = destino.getCentroX();
        int y2 = destino.getCentroY();
        Color c;

        int w = x2 - x ;
        int h = y2 - y ;
        int dx1 = 0, dy1 = 0, dx2 = 0, dy2 = 0 ;

        if (w<0) dx1 = -1 ; else if (w>0) dx1 = 1 ;
        if (h<0) dy1 = -1 ; else if (h>0) dy1 = 1 ;
        if (w<0) dx2 = -1 ; else if (w>0) dx2 = 1 ;
        int longest = Math.Abs(w) ;
        int shortest = Math.Abs(h) ;
        if (!(longest>shortest)) {
           longest = Math.Abs(h) ;
           shortest = Math.Abs(w) ;
           if (h<0) dy2 = -1 ; else if (h>0) dy2 = 1 ;
           dx2 = 0 ;
        }
        int numerator = longest >> 1 ;
        for (int i=0;i<=longest;i++) {

           c = bmp.GetPixel(x, y);

           if(c.R == 0 && c.G == 0 && c.B == 255){
                return true;
           }
           if(c.R == 255 && c.G == 0 && c.B == 0){
                return true;
           }
```

```
            numerator += shortest ;
            if (!(numerator<longest)) {
               numerator -= longest ;
               x += dx1 ;
               y += dy1 ;
            } else {
               x += dx2 ;
               y += dy2 ;
            }
         }

         return false;

      }
```

Métodos para dibujar líneas e insertar elementos en el grafo

```
//Recorrido de todos los circulos para dibujar lineas
      void drawLines(Bitmap bmp){

         //Se llena el grafo con los circulos, se introducen los vertices
         insertInGrafo();

         for(int i = 0; i < circulos.Count; i++){
            for(int j = 0; j < circulos.Count; j++){
               /*drawLine(bmp, circulos[i].getCentroX(), circulos[i].getCentroY(),
                     circulos[j].getCentroX(), circulos[j].getCentroY());*/

               if(circulos[i].getId() != circulos[j].getId()){

                  if(!isColision(bmp, circulos[i], circulos[j])){

                     drawLine(bmp, circulos[i].getCentroX(), circulos[i].getCentroY(),
                        circulos[j].getCentroX(), circulos[j].getCentroY());

                     //Se agregan las aristas
                     grafo.insertarArista(grafo.getVertice(circulos[i]), grafo.getVertice(circulos[j])
                                 , calculeDistance(circulos[i], circulos[j]));
                  }
               }



            }
         }
      }

      //Se dibuja una linea entre dos puntos
      void drawLine(Bitmap bmp, int x1, int y1, int x2, int y2){
```

```
            Graphics g = Graphics.FromImage(bmp);
            Pen p = new Pen(Color.Green);

            g.DrawLine(p, x1, y1, x2, y2);
        }

        //Insercion en el grafo
        void insertInGrafo(){

            for(int i = 0; i < circulos.Count; i++){

                //Insercion de vertice en el grafo
                grafo.insertarVertice(circulos[i]);
            }

        }


        double calculeDistance(Circulo origen, Circulo destino){
            return Math.Sqrt(Math.Pow(origen.getCentroX() - destino.getCentroX(), 2)
                        + Math.Pow(origen.getCentroY() - destino.getCentroY(), 2));
        }
```

Clase Vértice

```
using System;

namespace Actividad2_SemAlgoritmia
{

    public class Vertice
    {
        public Vertice sig;
        public Arista ady;
        public Circulo circulo;

        public Vertice()
        {

        }
    }
}
```

Clase Arista

```
using System;

namespace Actividad2_SemAlgoritmia
{
```

```
    public class Arista
    {
        public Arista sig;
        public Vertice ady;
        public double distance;

        public Arista()
        {
        }
    }
}
```

Clase Grafo

```
using System;
using System.Diagnostics;
using System.Windows.Forms;

namespace Actividad2_SemAlgoritmia
{

    public class Grafo
    {
        Vertice h;

        public Grafo()
        {
        }

        public void inicializar(){
            h = null;
        }
        public bool esVacio(){
            return h == null;
        }
        public int tam(){
            int cont = 0;
            Vertice aux;
            aux = h;
            while (aux != null) {
                cont++;
                aux = aux.sig;
            }
            return cont;
        }

        public Vertice getVertice(Circulo c){
            Vertice aux;
            aux = h;
```

```
    while (aux != null) {

      if(aux.circulo.getId() == c.getId()){
        return aux;
      }

      aux = aux.sig;
    }
    return null;
}

public void insertarArista(Vertice origen, Vertice destino, double distance){
    Arista nueva = new Arista();
    nueva.distance = distance;
    nueva.sig = null;
    nueva.ady = null;

    Arista aux;
    aux = origen.ady;

    if(aux == null){
      origen.ady = nueva;
      nueva.ady = destino;
    }else{
      while(aux.sig != null){
        aux = aux.sig;
      }
      aux.sig = nueva;
      nueva.ady = destino;
    }
}

public void insertarVertice(Circulo c){
    Vertice nuevo = new Vertice();
    nuevo.circulo = c;
    nuevo.sig = null;
    nuevo.ady = null;

    if(esVacio()){
      h = nuevo;
    }else{
      Vertice aux;
      aux = h;
      while(aux.sig != null){
        aux = aux.sig;
      }
      aux.sig = nuevo;
    }
```

```
    }

    public void listaAdyacencia(TextBox textB){

      Vertice vertAux;
      Arista arAux;
      int cantidadAristas = 0;

      vertAux = h;
      textB.Text += "Lista de Adyacencia:";
      textB.Text += Environment.NewLine;
      textB.Text += "Tamaño del grafo: " + tam() + " (Vertices)";
      textB.Text += Environment.NewLine;

      while(vertAux != null){
        //Debug.Write("(" + vertAux.circulo.getId() + ")" + "-->");
        textB.Text += Environment.NewLine;
        textB.Text += "[" + vertAux.circulo.getId() + "]" + " es adyacente con -> ";
        arAux = vertAux.ady;
        while (arAux != null) {
          cantidadAristas++;
          //Debug.Write("(" + arAux.ady.circulo.getId() + ")" + "[Distancia: " +
(float)arAux.distance + "px]" + " - ");
          textB.Text += "(" + arAux.ady.circulo.getId() + ") - ";
          arAux = arAux.sig;
        }
        vertAux = vertAux.sig;
        //Debug.Write("\n");
        textB.Text += Environment.NewLine;
        textB.Text += "Tiene " + cantidadAristas + " aristas";
        textB.Text += Environment.NewLine;
        cantidadAristas = 0;
      }
    }

    public void eliminarArista(Vertice origen, Vertice destino){
      Arista actual;
      Arista anterior = null;
      actual = origen.ady;
      int band = 0;

      if(actual == null){
        Debug.WriteLine("El origen no tiene aristas");
      }else if(actual.ady == destino){
        origen.ady = actual.sig;
        actual = null;
      }else{
        while (actual != null) {
```

```
            if(actual.ady == destino){
               anterior.sig = actual.sig;
               actual = null;
               band  = 1;
               break;
            }
            anterior = actual;
            actual = actual.sig;
         }
         if(band == 0){
            Debug.WriteLine("Vertices no conectados");
         }

      }
   }

   public void anular(){
      Vertice aux;

      while(h != null){
         aux = h;
         h = h.sig;
         aux = null;
      }
   }
  }
}
```