



UNIVERSIDAD DE
GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS
EXACTAS E INGENIERIAS

SEMINARIO DE SOLUCIÓN A PROBLEMAS DE ALGORITMIA

Actividad 2: Fuerza bruta

Presentado a:

David Alejandro Gómez Anaya
Departamento de Ciencias Computacionales

Presentado por:

Abraham Baltazar Murillo Sandoval, 218744368
Seccion D15, 2019 B

Contents

1	Diagrama de clases propuesto	2
2	Objetivo	2
3	Marco teórico	3
4	Desarrollo	4
4.1	Construcción del grafo	4
4.2	Par de puntos más cercanos	6
4.3	Organizacion de los circulos de mayor radio a menor radio	7
4.4	Eliminación de vértices	8
5	Códigos utilizados	10
6	Pruebas y resultados	14
7	Conclusiones	18
8	Apéndice(s)	18

1 Diagrama de clases propuesto

Grafo
<pre>+ HashMap<Integer, ArrayList<Integer>> listaAdyacencia; + ArrayList<Vertice> vertices; + ArrayList<Arista> aristas; + int nAristas; + int nVertices;</pre>
<pre>Grafo() + void vaciarGrafo() + Vertice quien + int numVertices() + int numAristas()</pre>

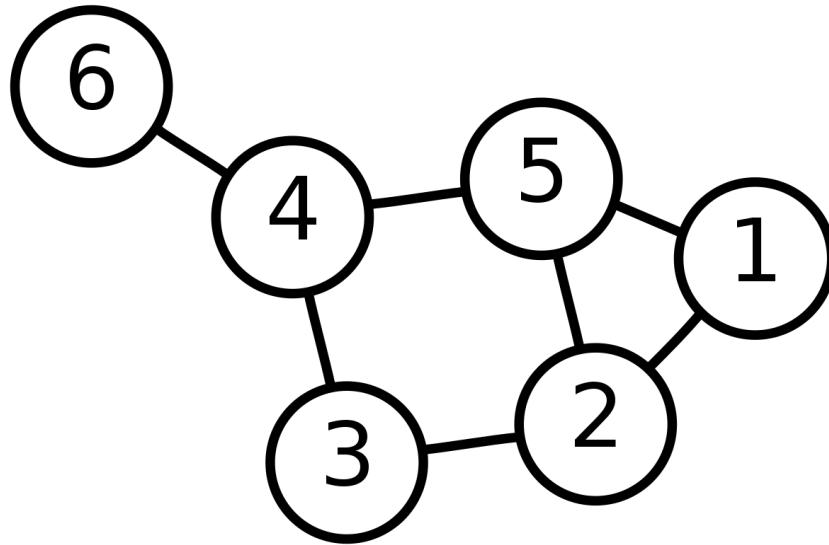
Vertice
<pre>+ Circulo cir; + boolean existe = true; + int id = -1;</pre>
<pre>Vertice() + int compareTo()</pre>

Arista
<pre>+ Vertice a, b; + ArrayList<Punto> linea; + int id = -1;</pre>
<pre>Arista()</pre>

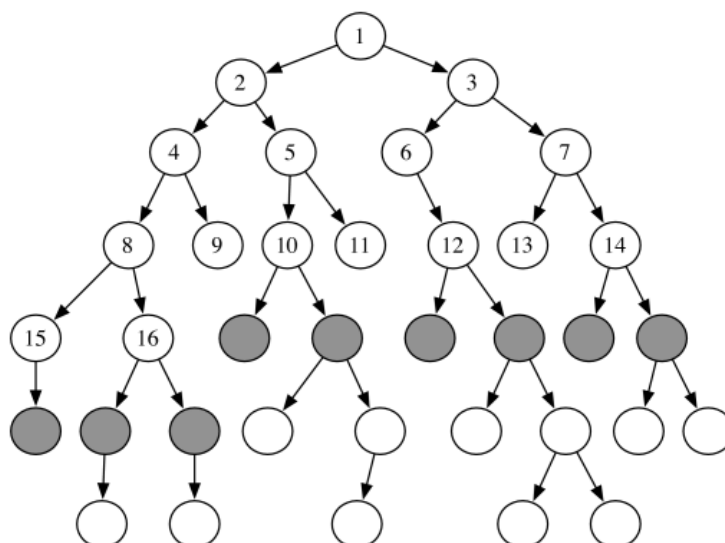
2 Objetivo

Diseñar un sistema en base el sistema anterior de detección de círculos capaz de unirlos mediante aristas para formar una estructura de relaciones binarias siempre y no haya objeto que le obstruya el paso, es decir, cada arista no debe de tocar un objeto de manera que choque con él. Además el sistema debe de ser capaz de eliminar círculos (vértices) y mantener las relaciones existentes.

Un grafo es una estructura de matemática de relaciones binarias que indican que un objeto A tiene un relación sobre un objeto B, esto mediante uniones (aristas).

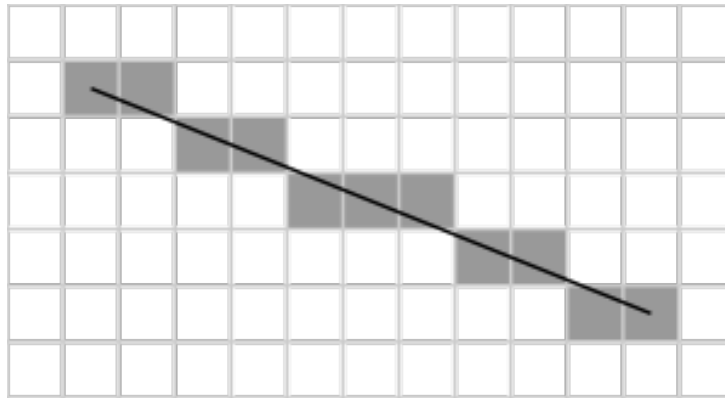


Es un algoritmo de teoría de grafos-árboles que nos permite desde uno o varios puntos de partida explorar los "vecinos" adyacentes a él, y generalmente encontrar la distancia mínima para alcanzarlo (puede extenderse a otras aplicaciones).



Algoritmo Bresenham

Es un algoritmo para el trazado de líneas rectas que determina los puntos que deberían de componer a la línea en un mapa n-dimensional. Este algoritmo selecciona un pixel aproximado en 'y' de acuerdo a la ecuación de la recta.

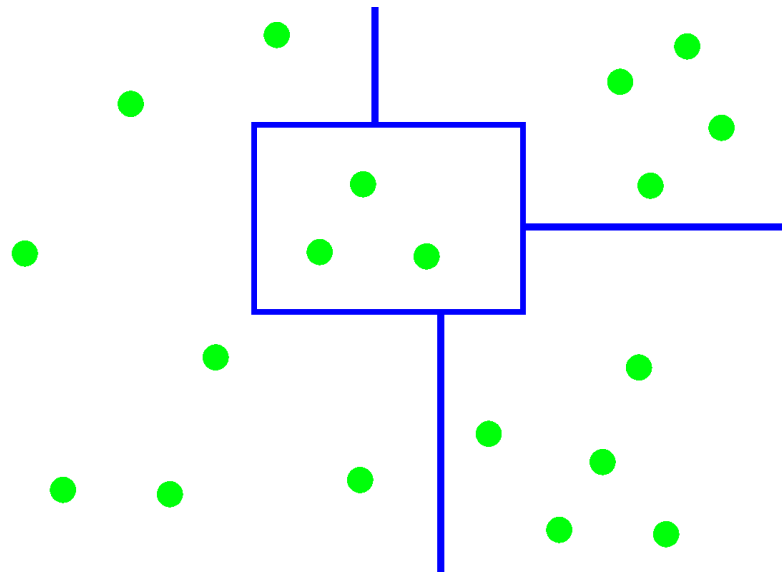


4 Desarrollo

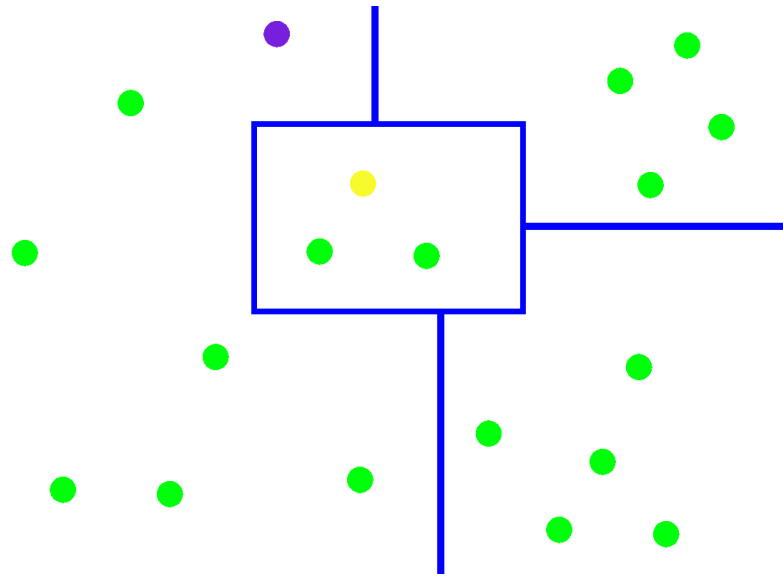
Para esta actividad hice uso de mucho análisis, imaginación y múltiples BFS para colorear conjuntos de píxeles (figuras, fondos). Además de reutilizar la actividad 1, haciendo el mismo procedimiento para detectar figuras válidas.

4.1 Construcción del grafo

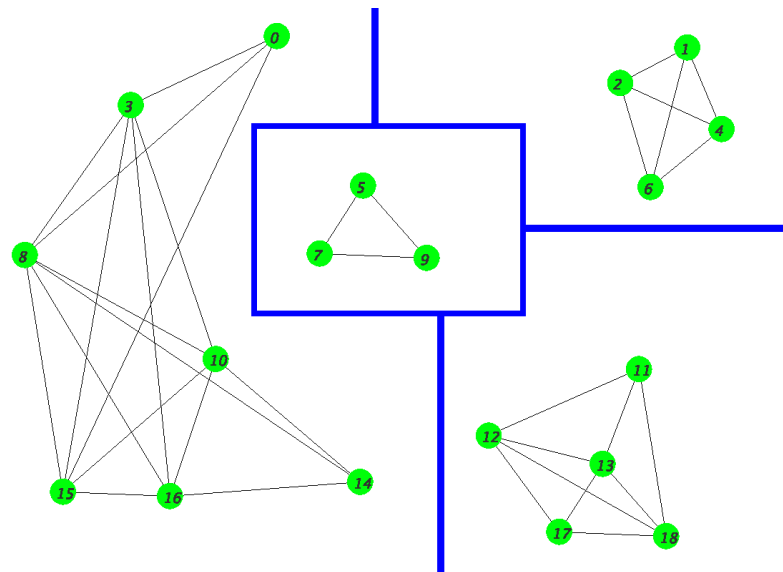
1. La clase grafo contiene una lista de vértices, ahí es donde cada uno de los círculos detectados quedará registrado para ser etiquetado posteriormente.



- Una vez detectado cada uno de los vértices, mediante el centro de cada uno y utilizando el algoritmo de Bresenham, trazo la recta en pixeles y detecto si intersecta alguna de las figuras o vértices que existen en la imagen, esto es, pintado el vertice de inicio de color morado y el vertice destino de color amarillo, entonces, si los pixeles que serán utilizados para trazar la línea que los une (arista), hay un pixel que es de color diferente, entonces choca con un objeto y se desecha esa línea, por tanto, esa posible arista; en caso de que el camino sea libre, agregamos la arista al grafo y la conexión respectivamente, además de regresar a su color original los dos vértices procesados, para seguir con el mismo proceso con todos los demás. Es por esto que la construcción del grafo es tiene una complejidad cuadrada puesto que cada uno de los vértices intenta conectarse con cada uno de los $n - 1$ vértices restantes.

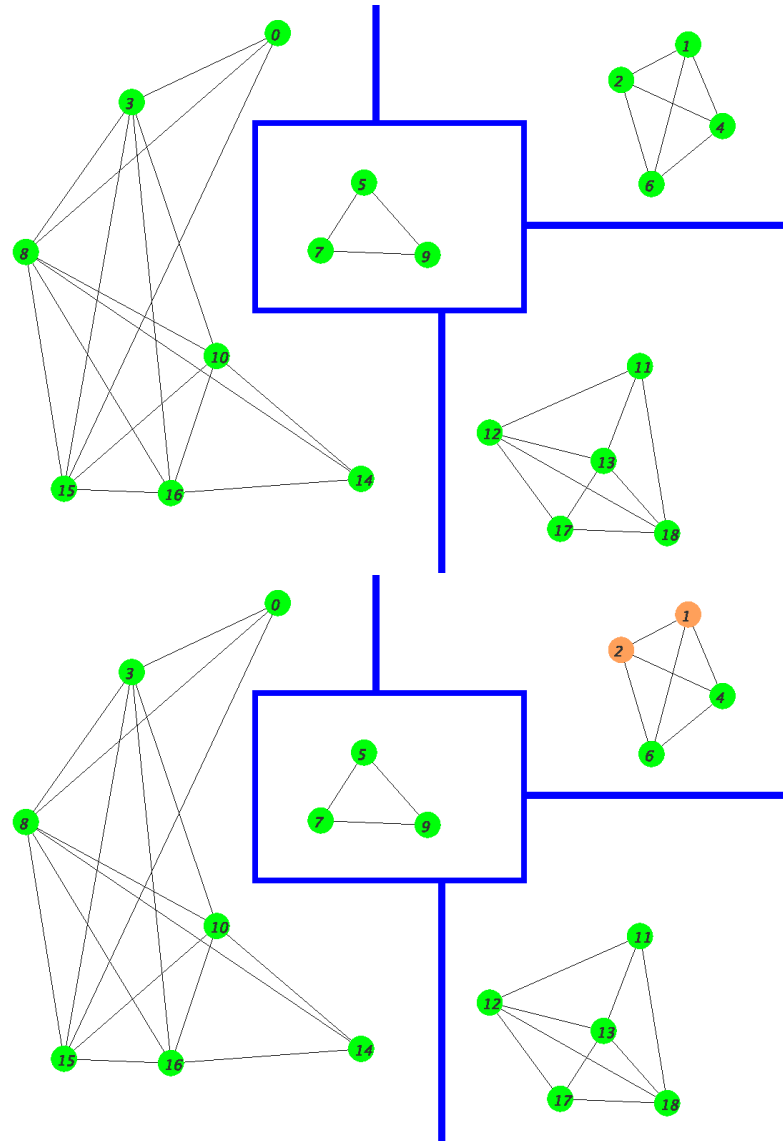


- Una vez realizado el procesamiento de las aristas, pinto cada uno de los pixeles existentes que guardo dentro de las propiedades de las aristas.



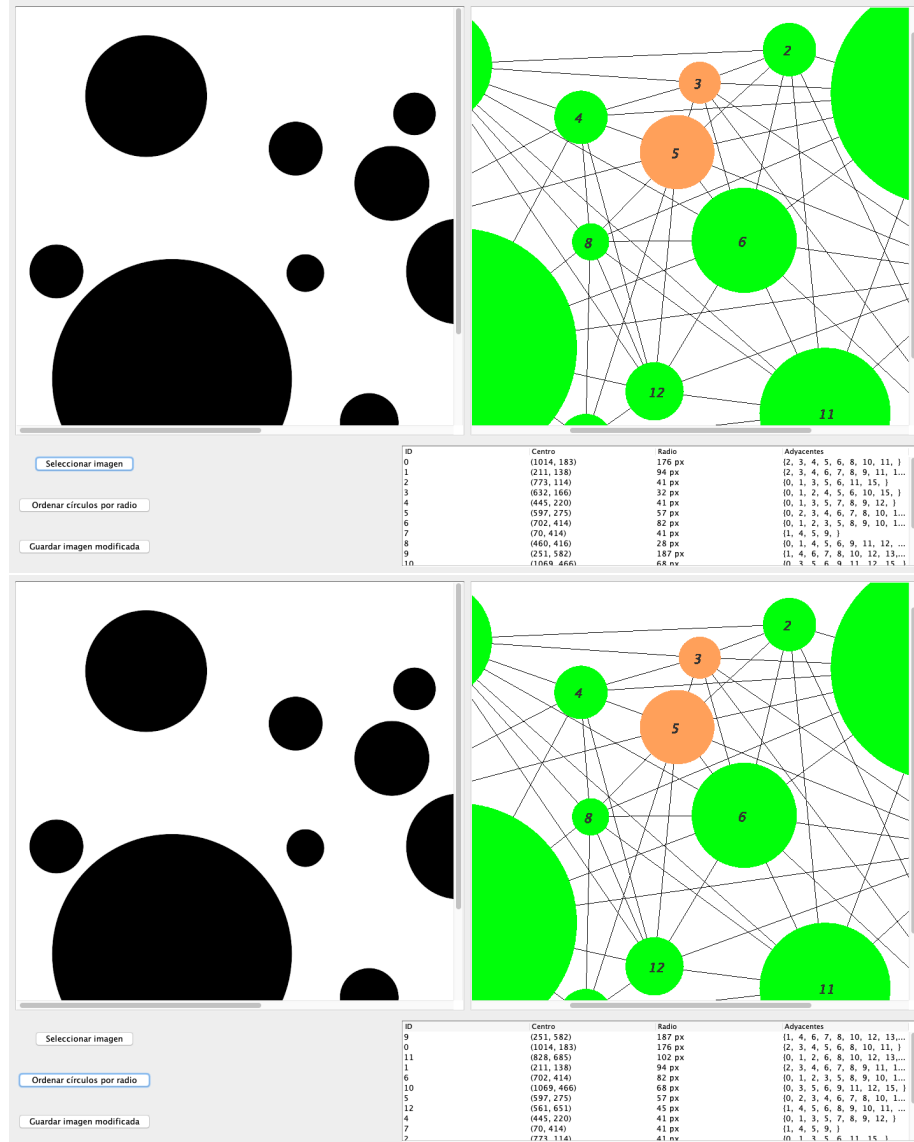
4.2 Par de puntos más cercanos

1. Para hacer esto, tomo un vertice A y B, calculo la distancia eucladiana entre los centros de ellos y si la nueva distancia calculada es mejor que la distancia que yo tengo como la mejor, entonces actualizo la distancia a esa nueva y tomo a los vértices A y B como los dos posibles candidatos a ser los más cercanos. Una vez reaizado pinta ambos vértices de color naranja para distinguirlos de los demás de color verde.



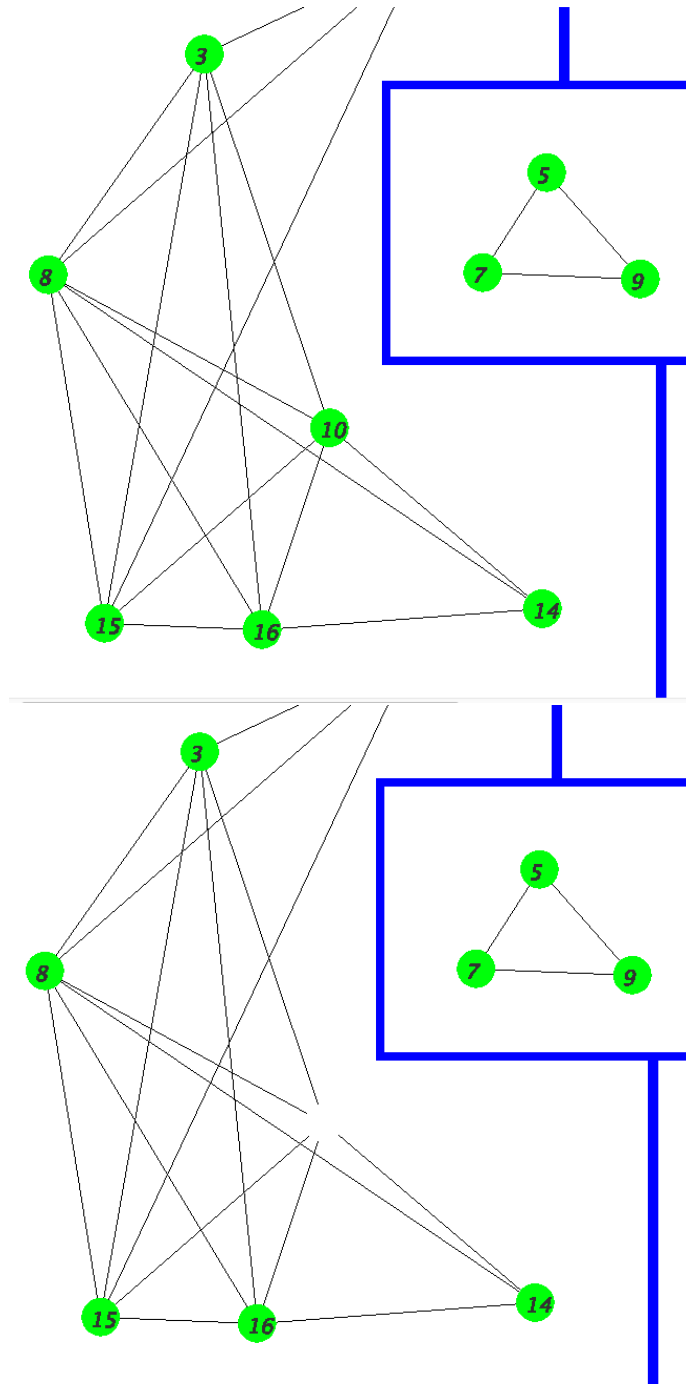
4.3 Organización de los círculos de mayor radio a menor radio

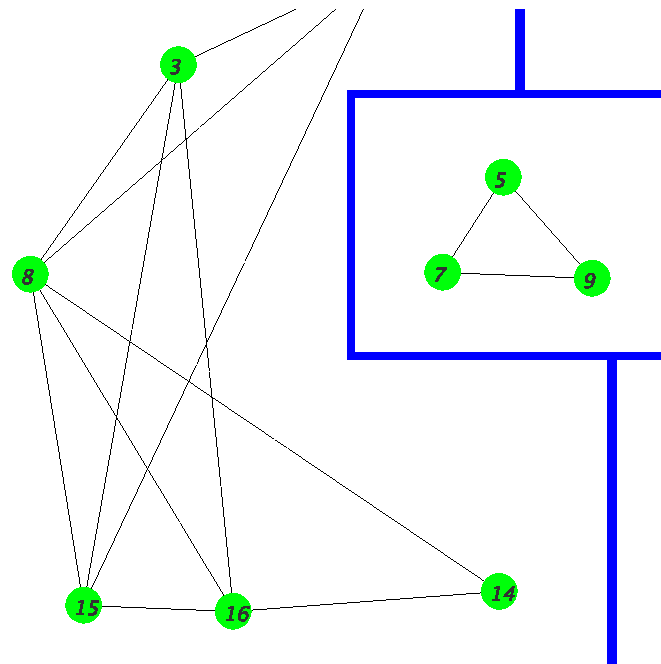
1. Para la organización de los vértices, calculo sus dimensiones en cuestión de radio y mediante un Bubble-sort los ordeno, comparando un elemento i con los siguientes en el rango (i, n) , llamándole a i "izquierda" y el siguiente "derecha", si el radio del de la derecha es mayor que el radio del de la izquierda, entonces los intercambio.



4.4 Eliminación de vértices

1. Para la eliminación de los vértices, hago uso de un MouseListener proporcionado por Java para obtener el pixel que mi mouse selecciona y así poder eliminar un vértice (pintar el objeto de color blanco, siempre y cuando sea un vértice). Después de borrarlo, para cada arista que le conecta, borro la línea que lo conectaba con los demás vértices.





5 Códigos utilizados

Se utilizó el siguiente código para la definición del grafo.

```
%class Vertice {
    public Circulo cir;
    public int id = -1;
}

class Arista {
    public Vertice a, b;
    public ArrayList<Punto> linea;
    public int id = -1;
}

public class Grafo {
    HashMap<Integer, ArrayList<Integer>> listaAdyacencia;
    ArrayList<Vertice> vertices;
    ArrayList<Arista> aristas;
    int nAristas, nVertices;
}
```

Se utilizó el siguiente código para la construcción del grafo.

```
%public void construirGrafo() {
    if (!hecho) {
        for (int i = 0; i < vertices.size(); i++) {
            Vertice a = vertices.get(i);
            pintarFigura(a, verde, morado);
            for (int j = i + 1; j < vertices.size(); j++) {
                Vertice b = vertices.get(j);
                pintarFigura(b, verde, amarillo);
                ArrayList<Punto> linea = buscarCamino(a, b, morado, amarillo);
                if( !linea.isEmpty() ){
                    anadirArista(a, b, linea);
                }
                pintarFigura(b, amarillo, verde);
            }
            pintarFigura(a, morado, verde);
        }
        System.out.println("Num de aristas: " + numAristas());
        System.out.println("Num de vertices: " + numVertices());
        for (Arista ari: aristas) {
            pintarLinea(ari.linea, negro);
        }
        calcularClosestPairOfPoints(verde, naranja, gris);
        agregarTodosLosIds(gris);
        hecho = true;
    }
}
```

Se utilizó el siguiente código basado en el algoritmo de Bresenham para el trazo de una línea (arista).

```
%private ArrayList<Punto> buscarCamino(Circulo a, Circulo b, Color
colorInicio, Color colorFinal) {
    // Bresenham's principles of integer incremental error
    ArrayList < Punto > linea = new ArrayList < > ();
    int x1 = a.centro.x, y1 = a.centro.y;
    int x2 = b.centro.x, y2 = b.centro.y;
    int dx = Math.abs(x2 - x1);
    int sx = x1 < x2 ? 1 : -1;
    int dy = -Math.abs(y2 - y1);
    int sy = y1 < y2 ? 1 : -1;
    int err = dx + dy;
    while (true) {
        if (imagenModificada.getRGB(x1, y1) == colorInicio.getRGB() ||
            imagenModificada.getRGB(x1, y1) == colorFinal.getRGB()) {
            // Estas sobre uno de los dos círculos
        } else if( imagenModificada.getRGB(x1, y1) == blanco.getRGB() ){
            // Es un pixel blanco por lo que puedo pintarlo
            linea.add(new Punto(x1, y1));
        } else {
            // Choca con algo, así que no me regreses la línea, regresame
            nada :c
            linea.clear();
            break;
        }
        if (x1 == x2 && y1 == y2) {
            break;
        }
        int err2 = 2 * err;
        if (err2 >= dy) {
            err += dy;
            x1 += sx;
        }
        if (err2 <= dx) {
            err += dx;
            y1 += sy;
        }
    }
    return linea;
}
```

Se utilizó el siguiente código para el calculo del par de puntos mas cercanos.

```
%public double calcularClosestPairOfPoints(Color colorInicio, Color
    colorFinal, Color colorLetras) {
    /*
        Inicialmente la peor distancia es dist = Double.POSITIVE_INFINITY
        Para cada par (i, j) calculamos la distancia que hay entre ellos y si
        es menor a nuestra peor distancia, los dos nuevos circulos
        mas cercanos son circulo(i) y circulo(j) que guardamos en a, b
        respectivamente.
    */
    double dist = Double.POSITIVE_INFINITY;
    Vertice a = new Vertice();
    Vertice b = new Vertice();
    for (int i = 0; i < vertices.size(); i++) {
        for (int j = i + 1; j < vertices.size(); j++) {
            if (vertices.get(i).cir.distancia(vertices.get(j).cir) < dist) {
                a = vertices.get(i);
                b = vertices.get(j);
                dist = a.cir.distancia(b.cir);
            }
        }
    }
    // Debo de tener al menos 2 circulos para poder calculara el par de
    puntos mas cercanos.
    if (vertices.size() >= 2) {
        anadirId(a, colorInicio);
        anadirId(b, colorInicio);
        pintarFigura(a, colorInicio, colorFinal);
        pintarFigura(b, colorInicio, colorFinal);
        anadirId(a, colorLetras);
        anadirId(b, colorLetras);
    }
    return dist;
}
```

Se utilizó el siguiente código para la organización de los círculos de mayor a menor radio.

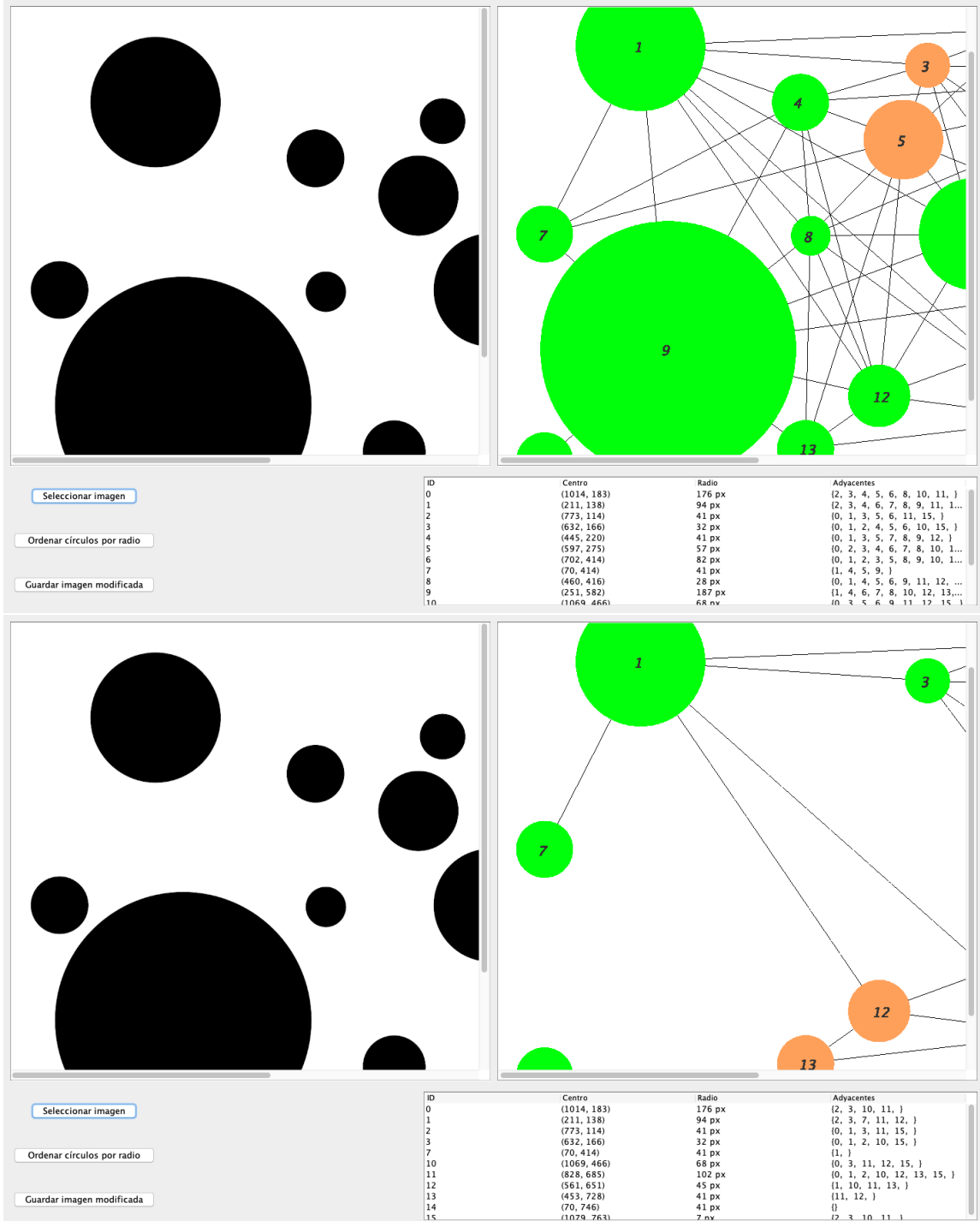
```
%public void ordenarCirculos() {
    // Revisamos cada par (i, j) y si el j.radio() (derecha) es mayor que
    i.radio(), quiere decir que j deberia de estar
    // donde se ecuentra el i y viceversa.
    for (int i = 0; i < vertices.size(); i++) {
        for (int j = i + 1; j < vertices.size(); j++) {
            if (vertices.get(j).cir.radio() > vertices.get(i).cir.radio())
            {
                Collections.swap(vertices, i, j);
            }
        }
    }
}
```

Se utilizó el siguiente código para la eliminación del vértices.

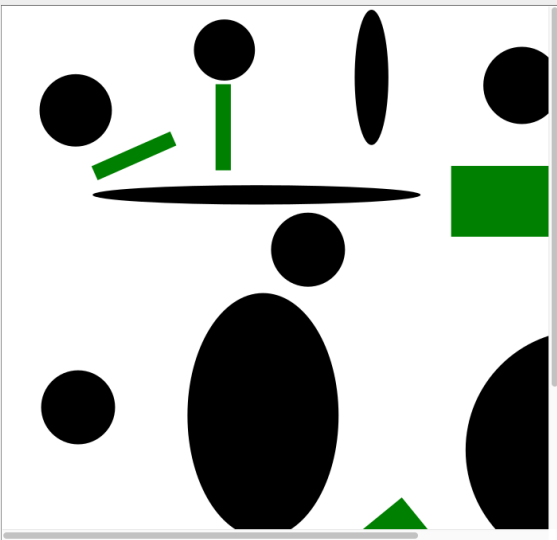
```
%public void eliminaVertice(int x, int y) {
    calcularClosestPairOfPoints(naranja, verde, verde);
    agregarTodosLosIds(verde);
    Circulo cir = buscarCirculo(x, y, verde, blanco);
    cir.calculaCentro();
    Vertice a = quien(cir);
    if( a == null ){
        System.out.println("Nodo ya eliminado!");
    }else{
        ArrayList<Integer> ady = listaAdyacencia.get(a.id);
        for (int i: ady) {
            Arista ari = aristas.get(i);
            Vertice b = a.compareTo(ari.a) == 0 ? ari.b: ari.a;
            listaAdyacencia.get(b.id).remove(Integer.valueOf(ari.id));
            pintarLinea(ari.linea, blanco);
            eliminaArista(ari);
        }
        listaAdyacencia.remove(a.id);
        vertices.remove(a);
        nVertices--;
    }
    calcularClosestPairOfPoints(verde, naranja, gris);
    agregarTodosLosIds(gris);
    System.out.println("Nodo " + a.id + " eliminado satisfactoriamente.");
}
```

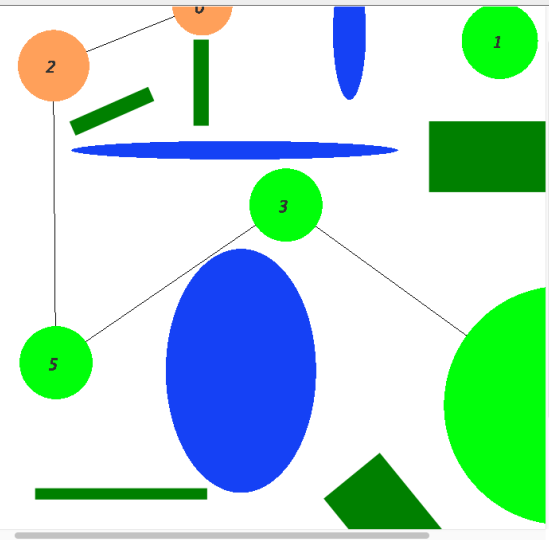
6 Pruebas y resultados

1. Prueba realizada en una imagen con únicamente círculos con diferentes tamaños.



2. Prueba realizada en la imagen 2-e1.



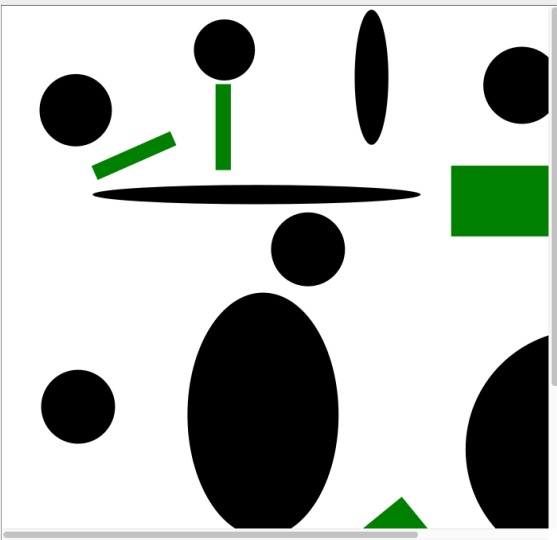


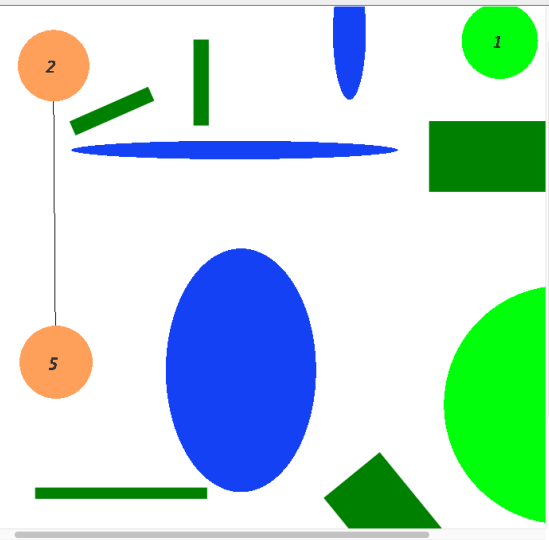
Seleccionar imagen

Ordenar círculos por radio

Guardar imagen modificada

ID	Centro	Radio	Adyacentes
0	(278, 54)	37 px	(2,)
1	(652, 98)	47 px	()
2	(91, 129)	44 px	(0, 5,)
3	(383, 304)	45 px	(4, 5,)
4	(731, 556)	149 px	(3,)
5	(94, 502)	45 px	(2, 3,)
6	(343, 801)	75 px	()





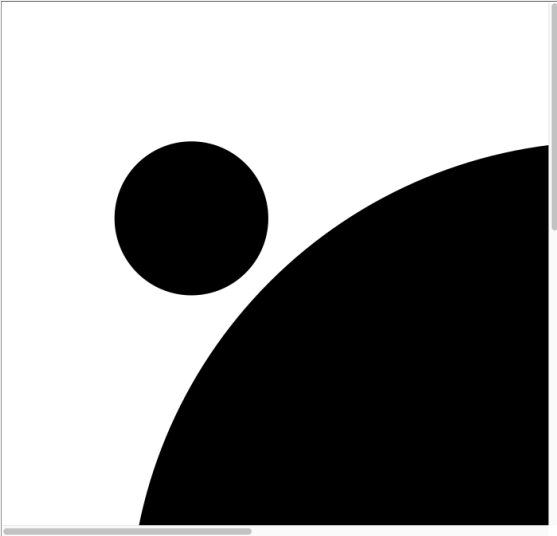
Seleccionar imagen

Ordenar círculos por radio

Guardar imagen modificada

ID	Centro	Radio	Adyacentes
1	(652, 98)	47 px	()
2	(91, 129)	44 px	(5,)
4	(731, 556)	149 px	()
5	(94, 502)	45 px	(2,)
6	(343, 801)	75 px	()

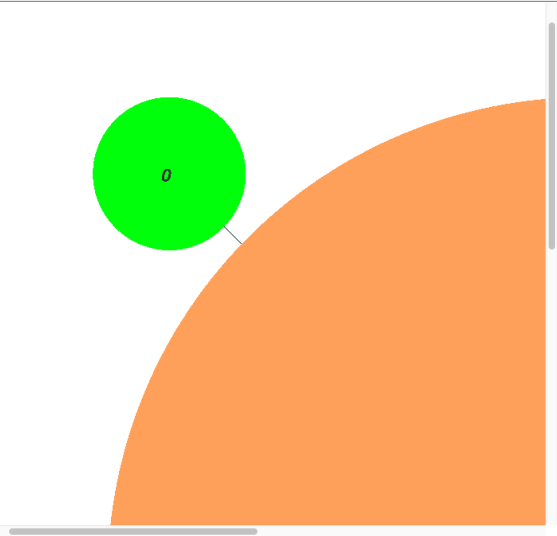
3. Prueba realizada en la imagen 2-e2.



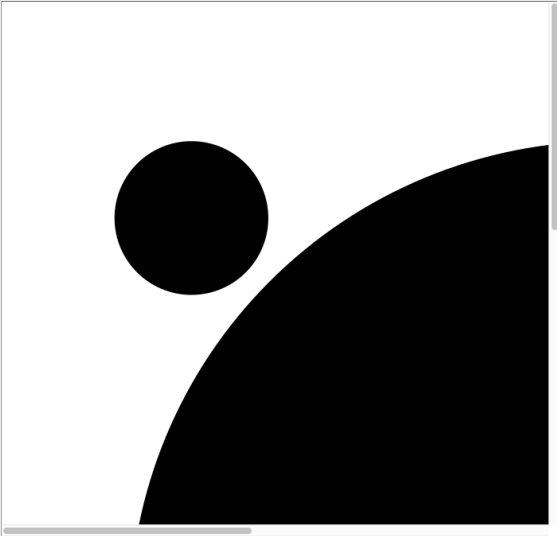
Seleccionar imagen

Ordenar círculos por radio

Guardar imagen modificada



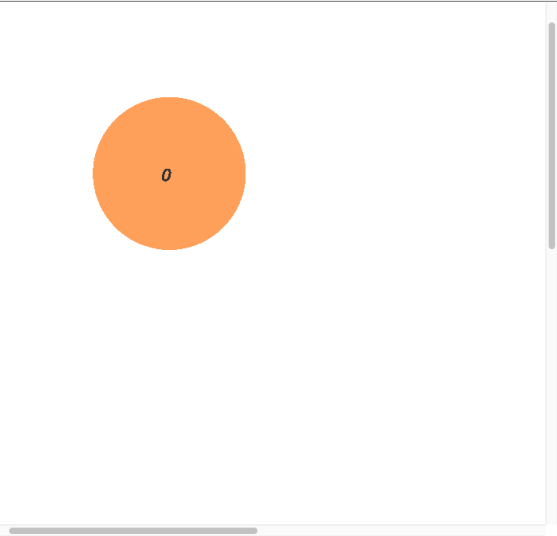
ID	Centro	Radio	Adyacentes
0	(236, 270)	95 px	(1,)
1	(761, 776)	601 px	(0, 2, 3, 4,)
2	(1284, 270)	95 px	(1,)
3	(236, 1281)	95 px	(1,)
4	(1284, 1281)	95 px	(1,)



Seleccionar imagen

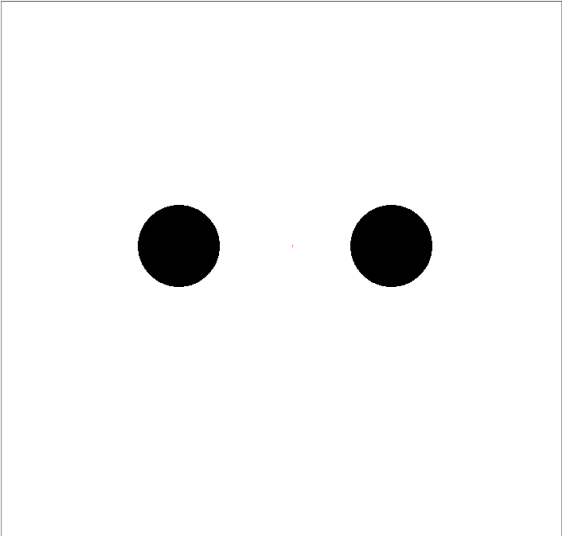
Ordenar círculos por radio

Guardar imagen modificada



ID	Centro	Radio	Adyacentes
0	(236, 270)	95 px	()
2	(1284, 270)	95 px	()
3	(236, 1281)	95 px	()
4	(1284, 1281)	95 px	()

4. Prueba realizada en la imagen 2-e3.

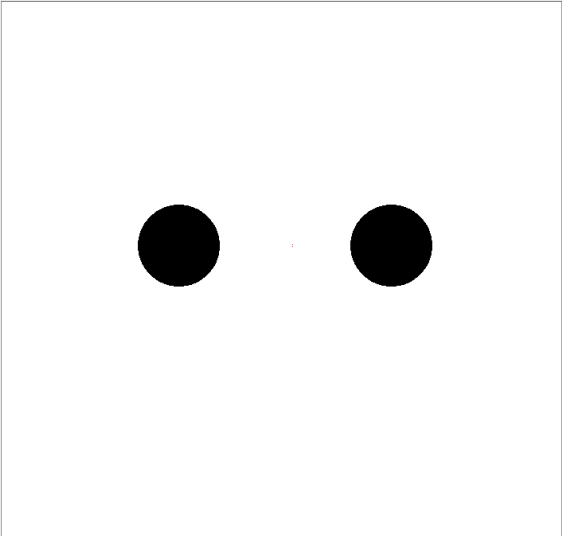


Seleccionar imagen

Ordenar círculos por radio

Guardar imagen modificada

ID	Centro	Radio	Adyacentes
0	(80, 152)	51 px	(1,)
1	(347, 152)	51 px	(0,)



Seleccionar imagen

Ordenar círculos por radio

Guardar imagen modificada

ID	Centro	Radio	Adyacentes
1	(347, 152)	51 px	()

7 Conclusiones

En esta actividad se puso a prueba la imaginación y las buenas ideas para no caer en una solución demasiado costosa en tiempo, puesto que podía haber imágenes muy grandes, y por tanto con una complejidad muy alta podría hacer que nuestro ordenador llegaría a trabarse.

Combinar estrategias de programación competitiva y recuerdos de la infancia (paint) hizo el proceso para resolverlo más sencillo y divertido. Una complejidad donde se procesa cada pixel exactamente una vez es la mejor en todos los aspectos, en este caso recurrir a una BFS para pintar e identificar lo que se está procesando.

Además de pensar cómo representar el grafo de tal manera que se evitara el calculo de ciertas cosas varias veces (el calculo de las líneas, de si existe un arista, ...)

8 Apéndice(s)

<https://www.geeksforgeeks.org/bresenham-line-generation-algorithm/> (algoritmo de Bresenham)

<https://docs.oracle.com/javase/tutorial/uiswing/events/mouselistener.html> (Detección del pixel seleccionado)

<https://www.hackerearth.com/practice/notes/graph-theory-part-i/> (Grafos)