Universidad Nacional de San Agustín de Arequipa

Escuela Profesional de Ciencia de la Computación
Estructura de Datos Avanzados

Laboratorio 4: QuadTree

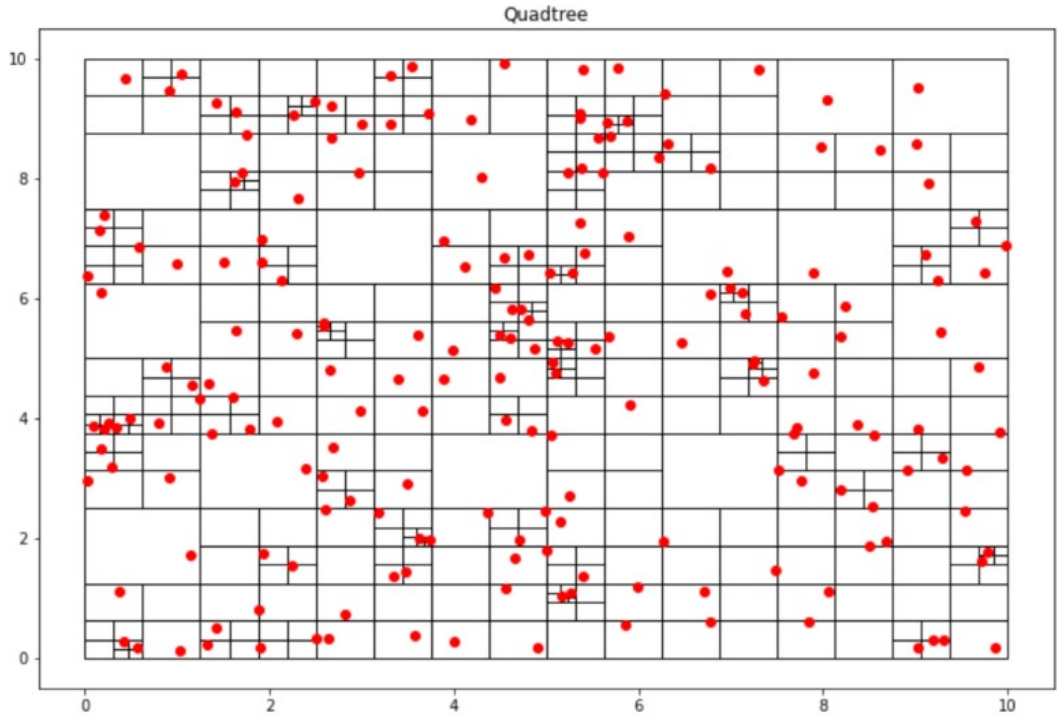
Presentado por:
Rushell Vanessa Zavalaga Orozco

Docente :
Rolando Jesús Cárdenas Talavera

QuadTree

Un `QuadTree` es una estructura de datos jerárquica que se utiliza para dividir un espacio bidimensional en subregiones más pequeñas, llamadas cuadrantes. Esta estructura es particularmente útil para gestionar de manera eficiente datos espaciales, como en gráficos por computadora, detección de colisiones, y sistemas de información geográfica.

Cada nodo en un `QuadTree` representa un rectángulo que puede contener un número limitado de puntos antes de dividirse en cuatro subcuadrantes (noroeste, noreste, suroeste y sureste). Esta división permite una rápida inserción, eliminación y búsqueda de puntos en el espacio, optimizando la complejidad de estas operaciones.



Implementación

Consta de varias clases y métodos implementados en Python, utilizando la biblioteca `pygame` para la visualización. La implementación principal incluye:

Clase Punto

La clase `Punto` representa un punto en el espacio 2D. Cada punto tiene coordenadas `x` e `y` y un método `draw` para representarlo gráficamente en la pantalla.

```
class Point():
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def draw(self, screen, color=(255,251, 80)):
        pygame.draw.circle(screen, color, (self.x, self.y), 4)
```

Listing 1: Código de la clase Punto

Clase QuadTree

La clase `QuadTree` es la estructura principal que divide el espacio en subregiones y permite insertar y eliminar puntos. Se define un rectángulo límite y se especifica una capacidad máxima de puntos antes de que se subdivida en cuadrantes.

```
class QuadTree():
    def __init__(self, x, y, w, h):
        self.x = x
        self.y = y
        self.width = w
        self.height = h
        self.points = []
        self.dividido = False
        self.capacity = 4
        self.NW = None
        self.NE = None
        self.SW = None
        self.SE = None

    def contiene(self, punto):
        return self.x <= punto.x <= self.x + self.width and self.y <=
   punto.y <= self.y + self.height

    def dividir(self):
        x,y, w, h = self.x, self.y, self.width, self.height

        self.NW = QuadTree(x, y, w/2, h/2)
        self.NE = QuadTree(x + w/2, y, w/2, h/2)
        self.SW = QuadTree(x, y + h/2, w/2, h/2)
        self.SE = QuadTree(x + w/2, y + h/2, w/2, h/2)

        self.dividido = True

        for point in self.points[:]:
            if not self.insertar(point):
                printd("Error al insertar el punto en el nodo dividido")

        self.points.clear()

    def insertar(self, punto):
        if not self.contiene(punto):
            return False
```

```
        if len(self.points) <= self.capacity-1  and not self.dividido:
            self.points.append(punto)
            return True

        if not self.dividido:
            self.dividir()

        if self.NW.insertar(punto):
            return True
        if self.NE.insertar(punto):
            return True
        if self.SW.insertar(punto):
            return True
        if self.SE.insertar(punto):
            return True

    def search_rango(self, x, y, w, h):
        puntos = []
        if self.x >= x + w or self.x + self.width <= x or self.y >= y +
h or self.y + self.height <= y:
            return puntos

        for point in self.points:
            if x <= point.x <= x + w and y <= point.y <= y + h:
                puntos.append(point)

        if self.dividido:
            puntos.extend(self.NW.search_rango(x, y, w, h))
            puntos.extend(self.NE.search_rango(x, y, w, h))
            puntos.extend(self.SW.search_rango(x, y, w, h))
            puntos.extend(self.SE.search_rango(x, y, w, h))

        return puntos

    def draw(self, screen):
        pygame.draw.rect(screen, (255, 255, 255), (self.x, self.y, self.
width, self.height), 1)
        if self.dividido:
            self.NW.draw(screen)
            self.NE.draw(screen)
            self.SW.draw(screen)
            self.SE.draw(screen)

        for punto in self.points:
            punto.draw(screen)
```

Listing 2: Código de la clase QuadTree

Inserción

El método de inserción verifica si el punto está dentro del límite del `QuadTree`, si no lo está retorna un booleano falso indicando que no se insertó. Por el contrario si está dentro del límite y hay espacio disponible, se agrega a la lista de puntos. Si no, se subdivide en

cuadrantes y se redistribuyen los puntos del cuadrante actual en sus hijos.

```
    def insertar(self, punto):
        if not self.contiene(punto):
            return False
        if len(self.points) <= self.capacity-1  and not self.dividido:
            self.points.append(punto)
            return True

        if not self.dividido:
            self.dividir()

        if self.NW.insertar(punto):
            return True
        if self.NE.insertar(punto):
            return True
        if self.SW.insertar(punto):
            return True
        if self.SE.insertar(punto):
            return True
```

Listing 3: Inserción de QuadTree

Búsqueda por Rango

La búsqueda por rango permite encontrar todos los puntos dentro de un área rectangular que el usuario especifíca. Esta función es la esencial para las aplicaciones del QuadTree. Se ingresa dos coordenadas, una del punto de inicio como otra del punto final. Tambien se utiliza un contenedor de puntos para coleccionar todos aquellos puntos pertenecientes al rango sin ser necesariamente del mismo cuadrante.

```
    def search_rango(self, x, y, w, h):
        puntos = []
        if self.x >= x + w or self.x + self.width <= x or self.y >= y +
    h or self.y + self.height <= y:
            return puntos

        for point in self.points:
            if x <= point.x <= x + w and y <= point.y <= y + h:
                puntos.append(point)

        if self.dividido:
            puntos.extend(self.NW.search_rango(x, y, w, h))
            puntos.extend(self.NE.search_rango(x, y, w, h))
            puntos.extend(self.SW.search_rango(x, y, w, h))
            puntos.extend(self.SE.search_rango(x, y, w, h))

        return puntos
```

Listing 4: Código de búsqueda por rango

Visualización

En cuanto a la visualización lo realice usando `pygame`, donde se dibujan los límites de los `QuadTree` y los puntos que contiene.

Complejidad Computacional

Complejidad de Inserción

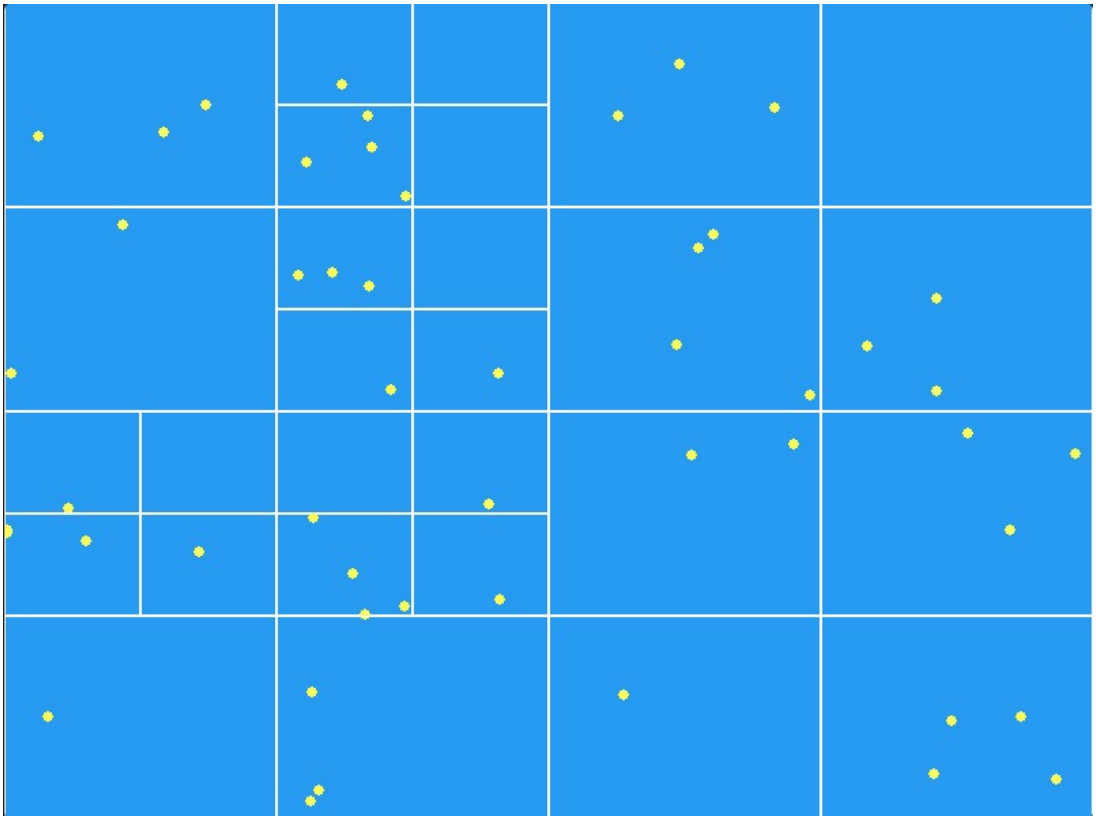
La inserción en el `QuadTree` tiene una complejidad en el peor caso de $O(n)$, donde n es el número total de puntos insertados. Este caso ocurre cuando todos los puntos se distribuyen en una línea o cuando el espacio está altamente desbalanceado, lo que impide una partición eficiente de los cuadrantes. Sin embargo, en condiciones ideales donde los puntos se distribuyen uniformemente en el espacio, la complejidad promedio de inserción es $O(\log n)$ debido a la subdivisión concurrente de los cuadrantes.

Complejidad de Búsqueda por Rango

La búsqueda por rango en el `QuadTree` tiene una complejidad promedio de $O(\sqrt{n} + k)$, donde k es el número de puntos reportados. Esta complejidad se debe a que, en promedio, el número de nodos visitados para realizar una consulta se aproxima a $O(\sqrt{n})$. En el peor de los casos, cuando el área de búsqueda abarca una gran parte del espacio (quiere decir todo el QuadTree) o se encuentra a un alto desbalanceo, la complejidad puede llegar a $O(n)$.

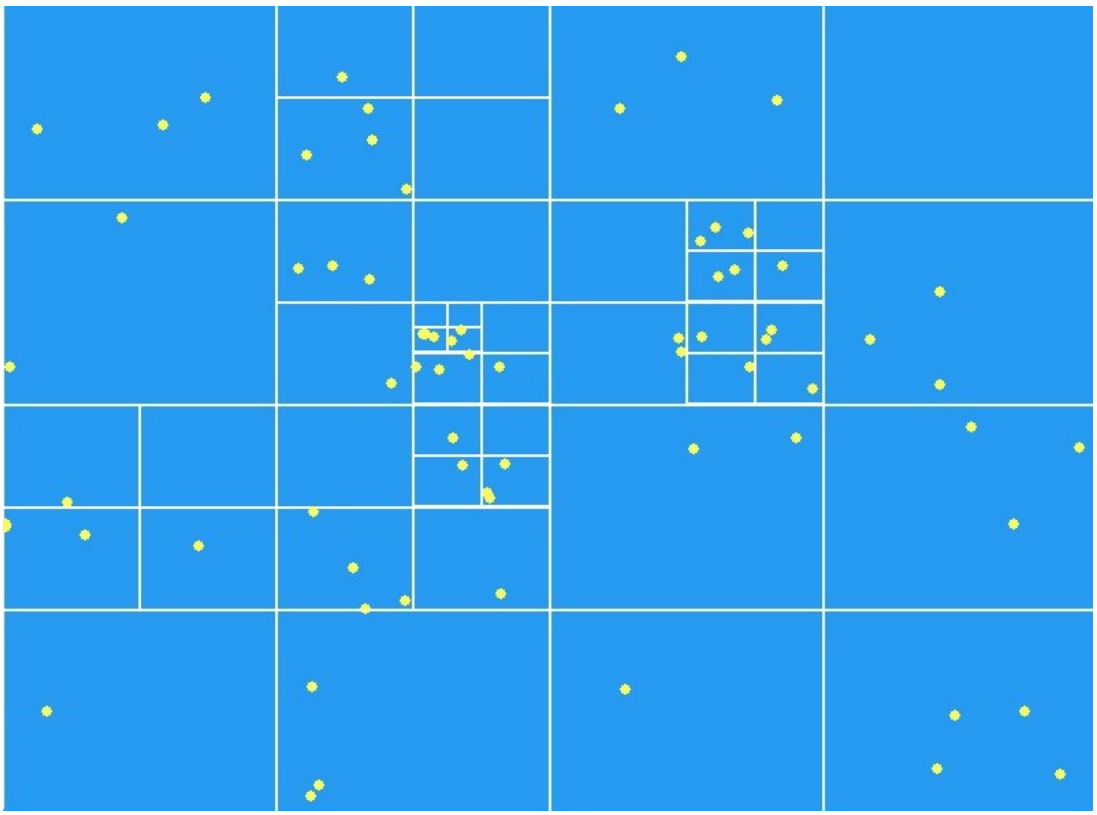
Resultados

Insertando 50 puntos aleatorios que se encuentre dentro de los limites (boundary) obtenemos lo siguiente:

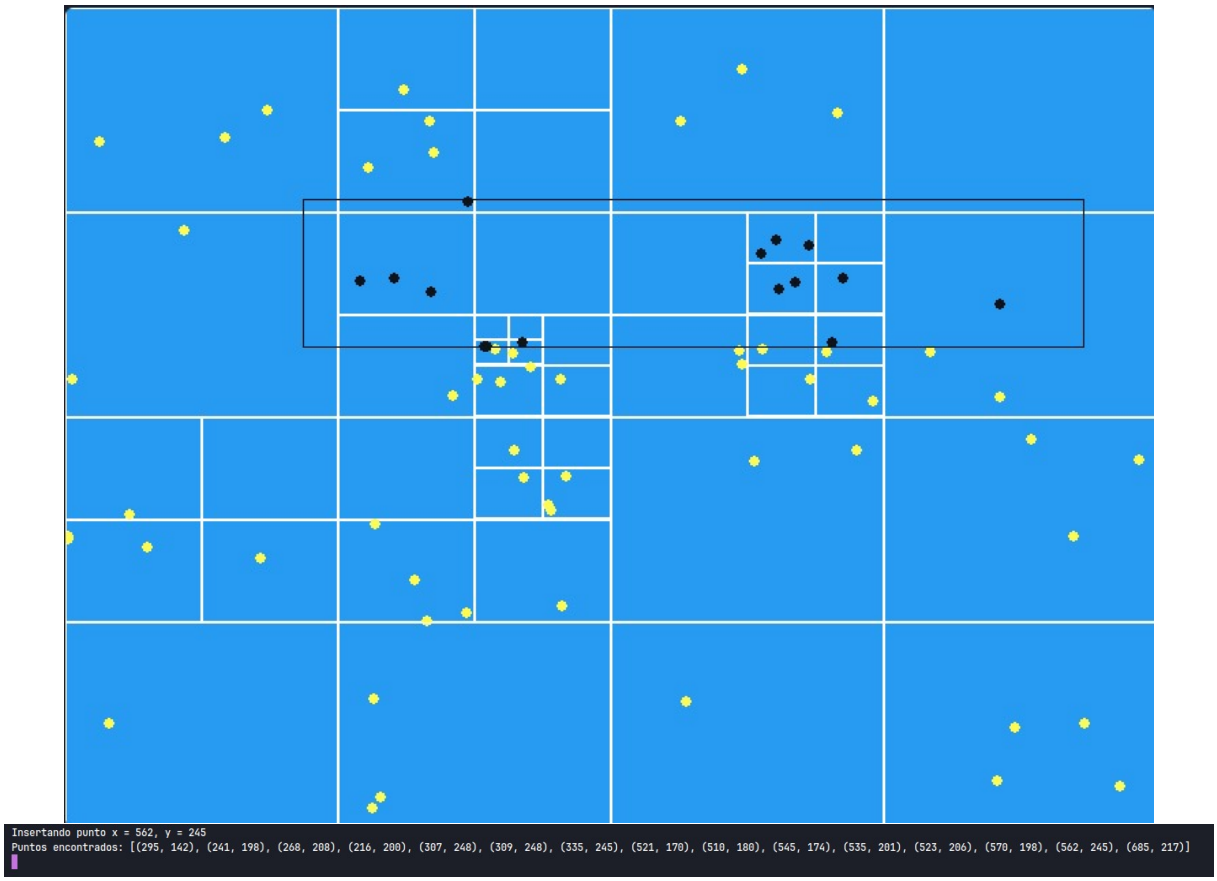


```
(myenv) rushh ~/Documentos/CODE/S6/Estructuras-de-Datos-Avanzados/La
pygame 2.6.1 (SDL 2.28.4, Python 3.12.7)
Hello from the pygame community. https://www.pygame.org/contribute.html
Generando e insertando 50 puntos aleatorios
Insertando punto x = 521, y = 170
Insertando punto x = 708, y = 316
Insertando punto x = 592, y = 288
Insertando punto x = 143, y = 403
Insertando punto x = 580, y = 324
Insertando punto x = 451, y = 83
Insertando punto x = 32, y = 524
Insertando punto x = 773, y = 570
Insertando punto x = 696, y = 527
Insertando punto x = 148, y = 75
Insertando punto x = 241, y = 198
Insertando punto x = 222, y = 117
Insertando punto x = 787, y = 331
Insertando punto x = 634, y = 252
Insertando punto x = 2, y = 389
Insertando punto x = 685, y = 217
Insertando punto x = 25, y = 98
Insertando punto x = 268, y = 208
Insertando punto x = 685, y = 285
Insertando punto x = 284, y = 284
Insertando punto x = 505, y = 332
Insertando punto x = 496, y = 45
Insertando punto x = 47, y = 371
Insertando punto x = 455, y = 508
Insertando punto x = 510, y = 180
Insertando punto x = 226, y = 506
Insertando punto x = 227, y = 378
Insertando punto x = 363, y = 272
Insertando punto x = 364, y = 438
Insertando punto x = 5, y = 272
Insertando punto x = 231, y = 578
Insertando punto x = 87, y = 163
Insertando punto x = 739, y = 387
Insertando punto x = 356, y = 368
Insertando punto x = 256, y = 419
Insertando punto x = 60, y = 395
Insertando punto x = 225, y = 586
Insertando punto x = 294, y = 443
Insertando punto x = 566, y = 77
Insertando punto x = 216, y = 200
Insertando punto x = 494, y = 251
Insertando punto x = 295, y = 142
Insertando punto x = 248, y = 60
Insertando punto x = 747, y = 524
Insertando punto x = 117, y = 95
Insertando punto x = 267, y = 83
Insertando punto x = 270, y = 106
Insertando punto x = 2, y = 387
Insertando punto x = 683, y = 566
Insertando punto x = 265, y = 449
```

Insertando ahora manualmente algunos otros puntos:

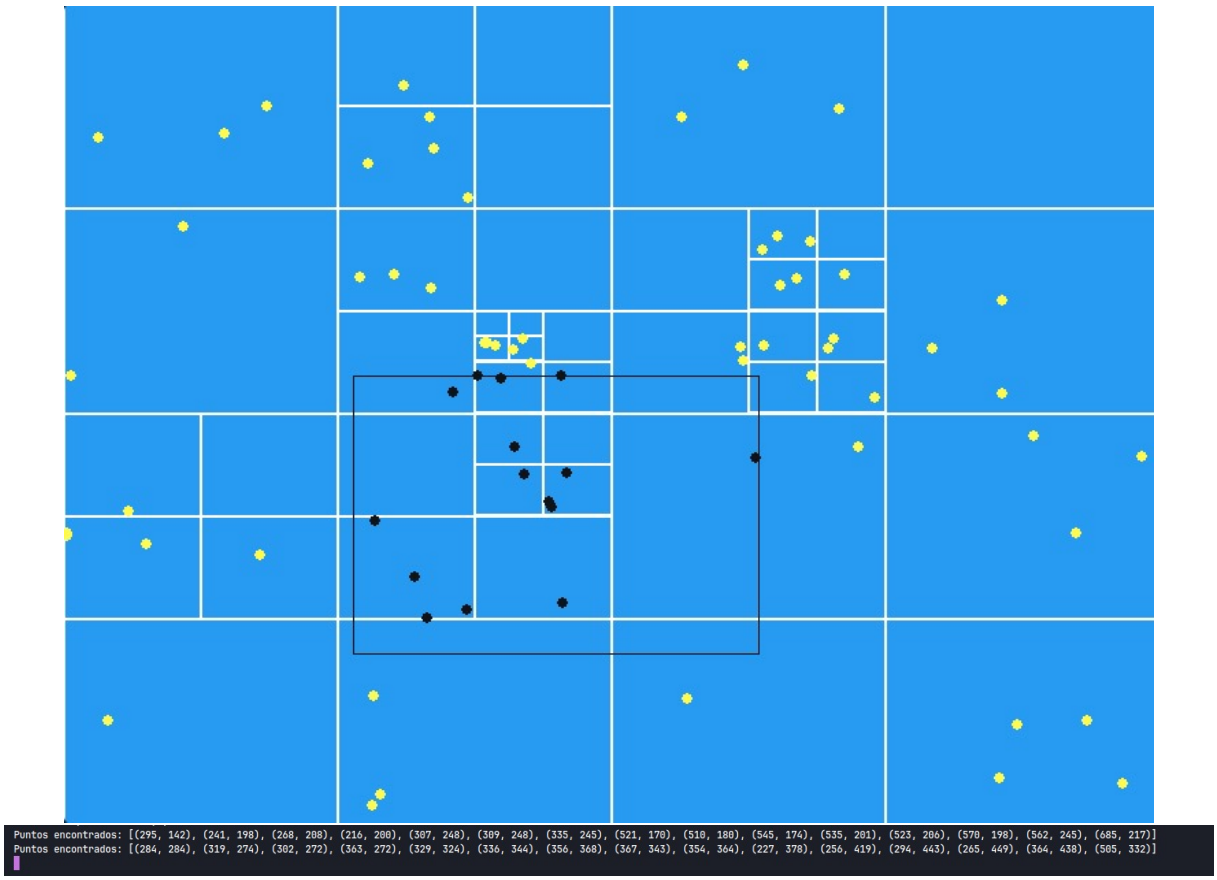


Buscando por rango:
Ejemplo 1:




```
Puntos encontrados: [(295, 142), (241, 198), (268, 208),
   (216, 200), (307, 248), (309, 248), (335, 245), (521, 170)
   , (510, 180), (545, 174), (535, 201), (523, 206), (570,
   198), (562, 245), (685, 217)]
```

Ejemplo 2:




```
Puntos encontrados: [(284, 284), (319, 274), (302, 272),
   (363, 272), (329, 324), (336, 344), (356, 368), (367, 343)
   , (354, 364), (227, 378), (256, 419), (294, 443), (265,
   449), (364, 438), (505, 332)]
```