UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

FACULTAD DE INGENIERIA DE PRODUCCION Y SERVICIOS

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

QuadTree

Curso:

Estructuras de datos Avanzadas

Docente:

Edson Luque

Alumno:

John Edson Sanchez Chilo

Arequipa – Perú

QuadTree

Marco Teórico

Un quadtree es una estructura de datos de árbol en la que cada nodo interno tiene exactamente cuatro hijos. Los quadtrees son el análogo bidimensional de octrees y se utilizan con mayor frecuencia para dividir un espacio bidimensional subdividiéndolo recursivamente en cuatro cuadrantes o regiones.

Las regiones subdivididas pueden ser cuadradas o rectangulares, o pueden tener formas arbitrarias. Esta estructura de datos fue nombrada quadtree por Raphael Finkel y J.L. Bentley en 1974. Todas las formas de quadtrees comparten algunas características comunes:

- Descomponen el espacio en células adaptables
- Cada celda (o cubeta) tiene una capacidad máxima. Cuando se alcanza la capacidad máxima, el balde se divide
- El directorio del árbol sigue la descomposición espacial del quadtree.

Metodología

Se va a realizar la implementación de un Quadtree (QTree) teniendo algunas clases base como nodo (Node), región (Boundary).

Además para su uso y la creación del gráfico se está tomando en cuenta la clase punto (Point)

Clase Node

```
class Node():
    def __init__(self, x0, y0, w, h, nodes):
        self.x0 = x0
        self.y0 = y0
        self.width = w
        self.height = h
        self.nodes = nodes
        self.children = []

    def get_nodes(self):
        return self.nodes
```

Clase Boundary

```
class Boundary():
    def __init__(self,xstart,xend,ystart,yend):
        self.xstart = xstart
        self.xend = xend
        self.ystart = ystart
```

```
        self.yend = yend
```

Clase Point

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

Clase QTree

```
class QTree():

    def __init__(self, boundary, array=[]):
        self.nodes = array
        self.root = Node(boundary.xstart, boundary.ystart, boundary.xend, boundary.yend, self.nodes)
        self.boundary = boundary

    def find_children(self,node):
        if not node.children:
            return [node]
        else:
            children = []
            for child in node.children:
                children += (self.find_children(child))
        return children

    def add_node(self, point):
        if(point.x < self.boundary.xend and point.x > self.boundary.xstart and point.y <
self.boundary.yend and point.y > self.boundary.ystart):
            self.nodes.append(point)

    def add_random_node(self):
        point=Point(random.uniform(self.boundary.xstart, self.boundary.xend),
random.uniform(self.boundary.ystart, self.boundary.yend))
        self.nodes.append(point)

    def get_nodes(self):
        return self.nodes

    def contains(self,x, y, w, h, nodes):
        pts = []
        for point in nodes:
            if point.x >= x and point.x <= x+w and point.y>=y and point.y<=y+h:
                pts.append(point)
        return pts
```

```
def recursive_subdivide(self,node):
    if len(node.nodes)<=1:
        return

    w_ = float(node.width/2)
    h_ = float(node.height/2)

    p = self.contains(node.x0, node.y0, w_, h_, node.nodes)
    x1 = Node(node.x0, node.y0, w_, h_, p)
    self.recursive_subdivide(x1)

    p = self.contains(node.x0, node.y0+h_, w_, h_, node.nodes)
    x2 = Node(node.x0, node.y0+h_, w_, h_, p)
    self.recursive_subdivide(x2)

    p = self.contains(node.x0+w_, node.y0, w_, h_, node.nodes)
    x3 = Node(node.x0 + w_, node.y0, w_, h_, p)
    self.recursive_subdivide(x3)

    p = self.contains(node.x0+w_, node.y0+h_, w_, h_, node.nodes)
    x4 = Node(node.x0+w_, node.y0+h_, w_, h_, p)
    self.recursive_subdivide(x4)

    node.children = [x1, x2, x3, x4]

def subdivide(self):
    self.recursive_subdivide(self.root)

def graph(self):
    self.subdivide()
    fig = plt.figure(figsize=(12, 8))
    plt.title("Quadtree")
    c = self.find_children(self.root)
    areas = set()
    for el in c:
        areas.add(el.width*el.height)
    for n in c:
        plt.gcf().gca().add_patch(patches.Rectangle((n.x0, n.y0), n.width, n.height, fill=False))
    x = [point.x for point in self.nodes]
    y = [point.y for point in self.nodes]
    plt.plot(x, y, 'ro',color='blue')
    plt.show()
```

Resultados

Se utilizo como experimento la creación de 50 valores aleatorios y se fueron insertando al quadtree, el resultado del gráfico es el mostrado a continuación