Point QuadTree

Alumno: Josue Samuel Philco Puma

1 de noviembre de 2024

Resumen

Este documento tendrá la explicación de la estructura llamada Point QuadTree junto con su implementación, prueba de funcionamiento y su complejidad computacional (Notación $Big-O$).

1. Introducción

La estructura QuadTree se basa en que cada nodo tiene 4 nodos hijos, este se asimila a un árbol binario con la diferencia que en vez de tener 2 ramas, el QuadTree tendrá hasta 4 ramas. En un QuadTree de puntos, el centro de una subdivisión está siempre en un punto. Si insertamos un elemento, este se va a dividir en un espacio de 4 cuadrantes. Al repetir este proceso, el cuadrante quedará dividiendo en 4 cuadrantes y así sucesivamente.

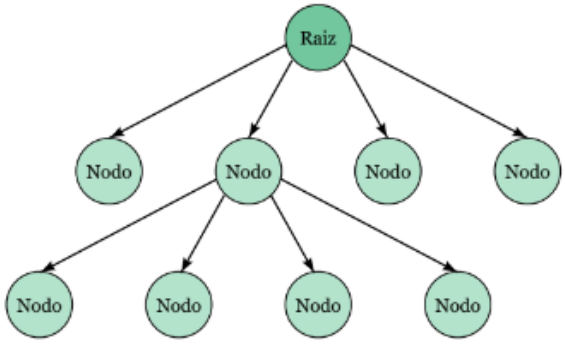

Figura 1: Ejemplo de árbol QuadTree

2. Implementación paso a paso

Ahora que vimos la definición del QuadTree, vamos a empezar con la implementación del Point QuadTree, este en diferencia a las variantes divide el espacio 2D en cuadrantes no perfectos, otras variantes como PR QuadTree o MX QuadTree dividen el espacion en cuadrantes perfectos y uniformes. Primero debemos ver como se estructura el nodo del **Point QuadTree**. Luego, vamos a implementar las funciones de **Inserción**, **Búsqueda por Rangos** y **Visualización** tanto del gráfico 2D como de la generación del árbol.

2.1. Nodo del QuadTree

Lo que hará es inicializar el nodo con las coordenadas (x, y) junto con su dato asociado y enlace a sus hijos en cada cuadrante (**NW, NE, SW, SE**).

```
class QuadTreeNode:
    def __init__(self, x, y, data):
        self.x = x
        self.y = y
        self.data = data
        self.NW = None
        self.NE = None
        self.SW = None
        self.SE = None
```

2.2. Función Insertion

Esta función se encarga simplemente de ir insertando los puntos en el QuadTree, esta función la vamos a dividir en dos partes:

- **Función insert:** Este método es público porque insertará el punto en el QuadTree y va a ir actualizando la estructura del QuadTree al seguir insertando más puntos.

```
def insert(self, x, y, data):
    if self.root is None:
        self.root = QuadTreeNode(x, y, data)
        self._add_to_graph(None, self.root, "Root")
    else:
        self._insert(self.root, x, y, data, self.x_min, self.x_max, self.
            y_min, self.y_max)

    self.ax.clear()
    self._draw(self.root, self.x_min, self.x_max, self.y_min, self.y_max)
    plt.pause(0.5)
```

- **Función _insert:** Este es un método recursivo privado, ya que se encarga de insertar en el cuadrante indicado y así poder actualizar el QuadTree. Si el cuadrante está vacío, creará un nuevo nodo. De caso contrario, se llamará recursivamente en el cuadrante hasta encontrar una posición vacía.

```
def _insert(self, node, x, y, data, x_min, x_max, y_min, y_max):
    if node is None:
        new_node = QuadTreeNode(x, y, data)
        self._add_to_graph(None, new_node, f"({x},{y})")
        return new_node

    if (x, y) == (node.x, node.y):
        node.data = data
        return node

    if x < node.x and y >= node.y:
        if node.NW is None:
            self._add_to_graph(node, QuadTreeNode(x, y, data), "NW")
        node.NW = self._insert(node.NW, x, y, data, x_min, node.x, node.y,
            y_max)
    elif x >= node.x and y >= node.y:
        if node.NE is None:
            self._add_to_graph(node, QuadTreeNode(x, y, data), "NE")
        node.NE = self._insert(node.NE, x, y, data, node.x, x_max, node.y,
            y_max)
    elif x < node.x and y < node.y:
        if node.SW is None:
            self._add_to_graph(node, QuadTreeNode(x, y, data), "SW")
        node.SW = self._insert(node.SW, x, y, data, x_min, node.x, y_min,
            node.y)
    else:
        if node.SE is None:
            self._add_to_graph(node, QuadTreeNode(x, y, data), "SE")
        node.SE = self._insert(node.SE, x, y, data, node.x, x_max, y_min,
            node.y)

    return node
```

2.3. Función Range Search

Esta función se encarga de buscar los puntos que se encuentre en un espacio determinado, es decir, en un rango de puntos establecido. Al igual que la función de **Insertion** lo dividiremos en dos partes:

- **Función range_search:** Esta función buscará los puntos dentro del rango especificado . Va a llamar a su método recursivo pasando por el nodo raíz y los límites del rango.

```
def range_search(self, x_min, x_max, y_min, y_max):
    result = []
    self._range_search(self.root, x_min, x_max, y_min, y_max, result)
    return result
```

- **Función _range_search:** Este método recursivo va a realizar la búsqueda por rango verificando si un nodo esta dentro del área, si lo cumple lo añade a la lista resultante. Luego, va a ir recorriendo recursivamente los cuadrantes del nodo actual que podrían contener más puntos, realizando búsquedas en los cuadrantes de forma selectiva para reducir la cantidad de nodos visitados.

```
def _range_search(self, node, x_min, x_max, y_min, y_max, result):
    if node is None:
        return

    if x_min <= node.x <= x_max and y_min <= node.y <= y_max:
        result.append((node.x, node.y, node.data))
        print(f"Punto dentro del rango: ({node.x}, {node.y}) con data: {
            node.data}")

    if node.x >= x_min and node.y <= y_max:
        print("Entrando al cuadrante NW")
        self._range_search(node.NW, x_min, x_max, y_min, y_max, result)
    if node.x <= x_max and node.y <= y_max:
        print("Entrando al cuadrante NE")
        self._range_search(node.NE, x_min, x_max, y_min, y_max, result)
    if node.x >= x_min and node.y >= y_min:
        print("Entrando al cuadrante SW")
        self._range_search(node.SW, x_min, x_max, y_min, y_max, result)
    if node.x <= x_max and node.y >= y_min:
        print("Entrando al cuadrante SE")
        self._range_search(node.SE, x_min, x_max, y_min, y_max, result)
```

2.4. Función Visualization

Ahora que ya vimos los métodos más importantes del **Point QuadTree**, vamos a ver las funciones que se van a encargar de que el gráfico 2D y el árbol se muestren para comprobar si la generación del gráfico y del árbol se están haciendo correctamente.

- **Función _draw:** Este método es recursivo que va a ir dibujando el QuadTree, va a dibujar los puntos y las divisiones. Si el nodo tiene subárboles (hijos), dibuja líneas de división en el gráfico para separar visualmente los cuadrantes y llama recursivamente a **_draw** en los hijos correspondientes.

```
def _draw(self, node, x_min, x_max, y_min, y_max):
    if node is None:
        return

    self.ax.plot(node.x, node.y, 'ro')

    self.ax.text(node.x + 1, node.y + 1, f'({node.x}, {node.y})\n{node.data
        }', fontsize=9, color='blue')

    if node.NW or node.NE or node.SW or node.SE:
        self.ax.plot([node.x, node.x], [y_min, y_max], 'k-')
        self.ax.plot([x_min, x_max], [node.y, node.y], 'k-')

    if node.NW:
        self._draw(node.NW, x_min, node.x, node.y, y_max)
    if node.NE:
        self._draw(node.NE, node.x, x_max, node.y, y_max)
```

```
        if node.SW:
            self._draw(node.SW, x_min, node.x, y_min, node.y)
        if node.SE:
            self._draw(node.SE, node.x, x_max, y_min, node.y)
```

- **Función _add_to_graph:** Esta función se va a encargar de construir el árbol para representarlo de forma visual. Este se va a realizar en **Graphviz**.

```
def _add_to_graph(self, parent, child, direction):
    child_label = f'({child.x}, {child.y})\n{child.data}'
    self.graph.node(child_label)
    if parent:
        parent_label = f'({parent.x}, {parent.y})\n{parent.data}'
        self.graph.edge(parent_label, child_label, label=direction)
```

- **Función render_graph:** Este va a generar el árbol en un archivo de imagen para mostrar las conexiones y como se estructuró el QuadTree.

```
def render_graph(self, filename="quadtree"):
    self.graph.render(filename, format='png', cleanup=False)
```

3. Implementación completa y ejecución

Ahora que vimos como es la implementación de cada función del **Point QuadTree**, vamos a proceder a poner toda la implementación completa y su ejecución para ver el resultado correspondiente:

- **Implementación completa:** Primero iremos por la implementación de toda la parte principal:

```
import random
import matplotlib.pyplot as plt
from graphviz import Digraph

class QuadTreeNode:
    def __init__(self, x, y, data):
        self.x = x
        self.y = y
        self.data = data
        self.NW = None
        self.NE = None
        self.SW = None
        self.SE = None

class PointQuadTree:
    def __init__(self, x_min, x_max, y_min, y_max):
        self.root = None
        self.x_min = x_min
        self.x_max = x_max
        self.y_min = y_min
        self.y_max = y_max
        self.fig, self.ax = plt.subplots()
        self.graph = Digraph()

    def insert(self, x, y, data):
        if self.root is None:
            self.root = QuadTreeNode(x, y, data)
            self._add_to_graph(None, self.root, "Root")
        else:
            self._insert(self.root, x, y, data, self.x_min, self.x_max,
                self.y_min, self.y_max)

        self.ax.clear()
```

```
33          self._draw(self.root, self.x_min, self.x_max, self.y_min, self.
                y_max)
34          plt.pause(0.5)
35
36      def _insert(self, node, x, y, data, x_min, x_max, y_min, y_max):
37          if node is None:
38              new_node = QuadTreeNode(x, y, data)
39              self._add_to_graph(None, new_node, f"({x},{y})")
40              return new_node
41
42          if (x, y) == (node.x, node.y):
43              node.data = data
44              return node
45
46          if x < node.x and y >= node.y:
47              if node.NW is None:
48                  self._add_to_graph(node, QuadTreeNode(x, y, data), "NW")
49              node.NW = self._insert(node.NW, x, y, data, x_min, node.x, node
                    .y, y_max)
50          elif x >= node.x and y >= node.y:
51              if node.NE is None:
52                  self._add_to_graph(node, QuadTreeNode(x, y, data), "NE")
53              node.NE = self._insert(node.NE, x, y, data, node.x, x_max, node
                    .y, y_max)
54          elif x < node.x and y < node.y:
55              if node.SW is None:
56                  self._add_to_graph(node, QuadTreeNode(x, y, data), "SW")
57              node.SW = self._insert(node.SW, x, y, data, x_min, node.x,
                    y_min, node.y)
58          else:
59              if node.SE is None:
60                  self._add_to_graph(node, QuadTreeNode(x, y, data), "SE")
61              node.SE = self._insert(node.SE, x, y, data, node.x, x_max,
                    y_min, node.y)
62
63          return node
64
65      def range_search(self, x_min, x_max, y_min, y_max):
66          result = []
67          self._range_search(self.root, x_min, x_max, y_min, y_max, result)
68          return result
69
70      def _range_search(self, node, x_min, x_max, y_min, y_max, result):
71          if node is None:
72              return
73
74          if x_min <= node.x <= x_max and y_min <= node.y <= y_max:
75              result.append((node.x, node.y, node.data))
76              print(f"Punto dentro del rango: ({node.x}, {node.y}) con data:
                    {node.data}")
77
78          if node.x >= x_min and node.y <= y_max:
79              print("Entrando al cuadrante NW")
80              self._range_search(node.NW, x_min, x_max, y_min, y_max, result)
81          if node.x <= x_max and node.y <= y_max:
82              print("Entrando al cuadrante NE")
83              self._range_search(node.NE, x_min, x_max, y_min, y_max, result)
84          if node.x >= x_min and node.y >= y_min:
85              print("Entrando al cuadrante SW")
86              self._range_search(node.SW, x_min, x_max, y_min, y_max, result)
87          if node.x <= x_max and node.y >= y_min:
88              print("Entrando al cuadrante SE")
89              self._range_search(node.SE, x_min, x_max, y_min, y_max, result)
```

```
90
91      def _draw(self, node, x_min, x_max, y_min, y_max):
92          if node is None:
93              return
94
95          self.ax.plot(node.x, node.y, 'ro')  # Dibuja el punto rojo
96
97          self.ax.text(node.x + 1, node.y + 1, f'({node.x}, {node.y})\n{node.
                data}', fontsize=9, color='blue')
98
99          if node.NW or node.NE or node.SW or node.SE:
100             self.ax.plot([node.x, node.x], [y_min, y_max], 'k-')
101             self.ax.plot([x_min, x_max], [node.y, node.y], 'k-')
102
103         if node.NW:
104             self._draw(node.NW, x_min, node.x, node.y, y_max)
105         if node.NE:
106             self._draw(node.NE, node.x, x_max, node.y, y_max)
107         if node.SW:
108             self._draw(node.SW, x_min, node.x, y_min, node.y)
109         if node.SE:
110             self._draw(node.SE, node.x, x_max, y_min, node.y)
111
112     def _add_to_graph(self, parent, child, direction):
113         child_label = f'({child.x}, {child.y})\n{child.data}'
114         self.graph.node(child_label)
115         if parent:
116             parent_label = f'({parent.x}, {parent.y})\n{parent.data}'
117             self.graph.edge(parent_label, child_label, label=direction)
118
119     def render_graph(self, filename="quadtree"):
120         self.graph.render(filename, format='png', cleanup=False)
121
122     def show(self):
123         plt.show()
```

Ahora que tenemos la parte principal, vamos a probar un caso de prueba para ver al **Point Quad-Tree** en acción:

```
1  tree = PointQuadTree(0, 100, 0, 100)
2
3  tree.insert(35, 42, "Chicago")
4  tree.insert(52, 10, "Mobile")
5  tree.insert(62, 77, "Toronto")
6  tree.insert(82, 65, "Buffalo")
7  tree.insert(5, 45, "Denver")
8  tree.insert(27, 35, "Omaha")
9  tree.insert(85, 15, "Atlanta")
10 tree.insert(90, 5, "Miami")
11
12 result = tree.range_search(25, 65, 5, 45)
13 print("Puntos encontrados en el rango (25,65) x (5,45): ", result)
14
15 tree.render_graph("quadtree_visualization")
16
17 tree.show()
```

- **Resultados en la ejecución para verificación:** Acá se muestran los resultados del primer testeo, decir que el resultado esta inspirado del libro ***Foundations of Multidimensional and Metric Data Structures , Hanan Samet (2006, Morgan Kaufmann)***:

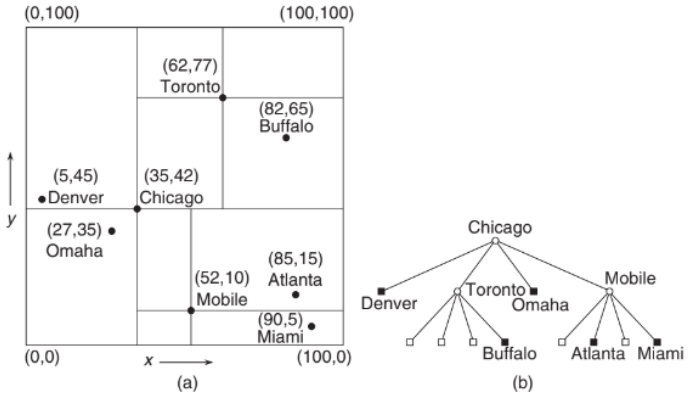

Figura 2: Ejemplo del Point QuadTree del libro

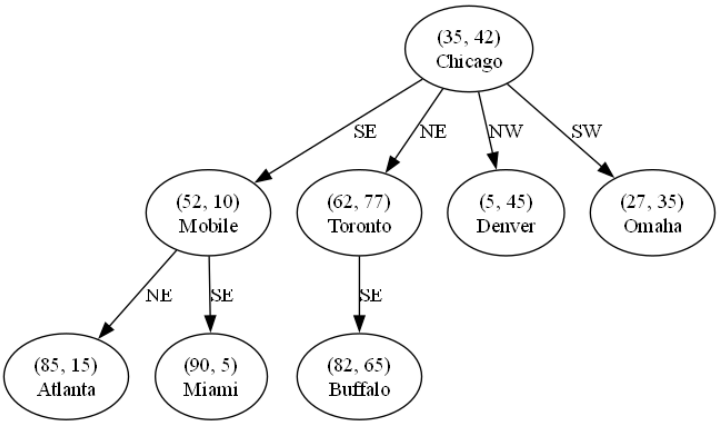

Figura 3: Visualización del árbol generado

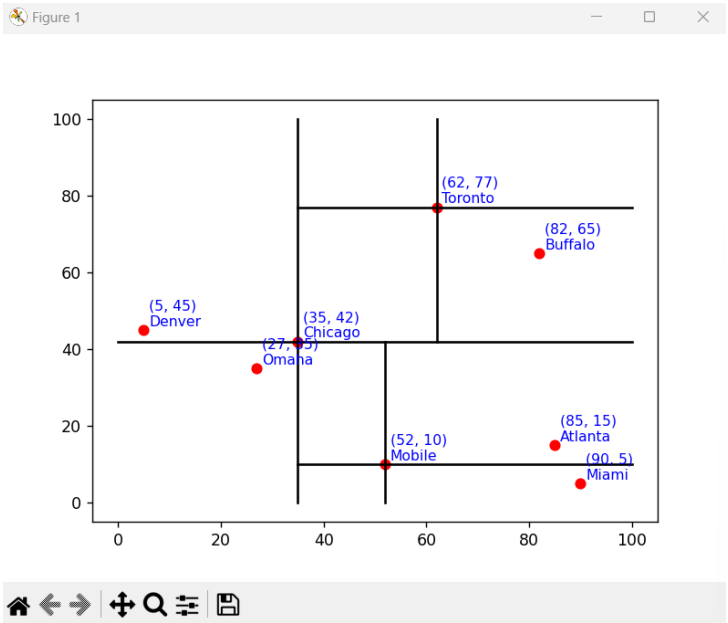

Figura 4: Gráfico 2D generado del Point QuadTree

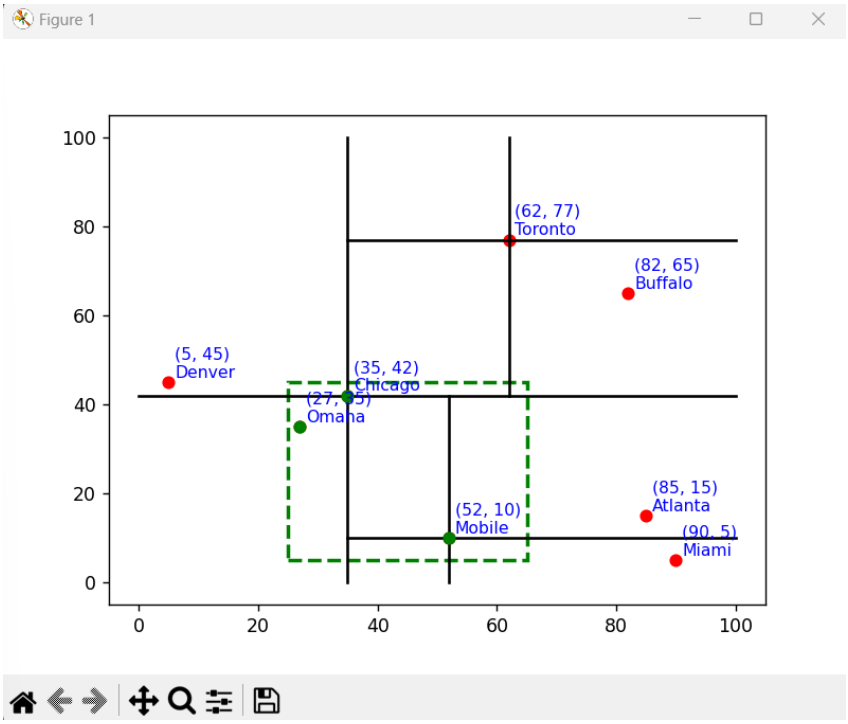

Figura 5: Gráfico 2D con el Range Search

- **Insertando 50 puntos aleatorios que se encuentre dentro de los límites y resultado:** Como última verificación vamos a insertar 50 puntos aleatorios que van a estar dentro de los límites y hacer una búsqueda por rango.

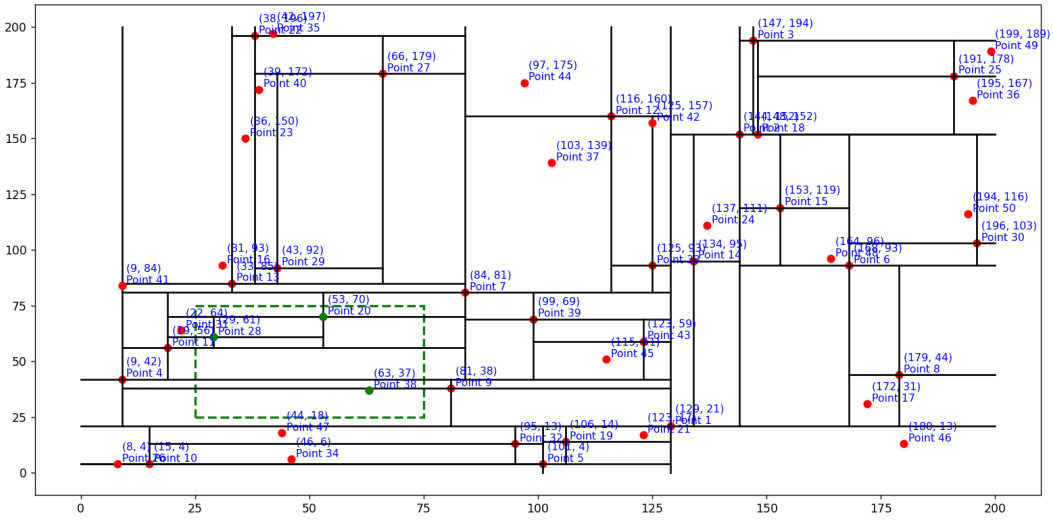

Figura 6: Insertando 50 puntos aleatorios y buscando en rango [25, 75]

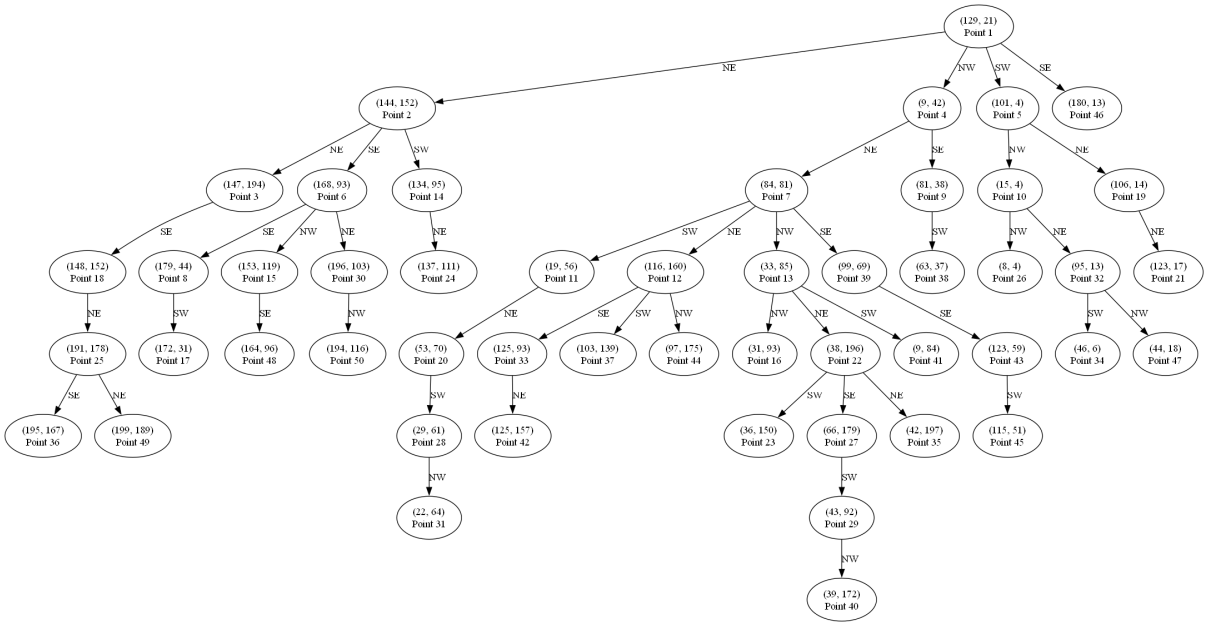

Figura 7: Árbol generado después de la inserción

4. Complejidad computacional

4.1. En la Función Insert

En esta función, en su caso promedio llega a ser $O(\log n)$ donde n será el número de puntos. Esto se debe porque en cada área de búsqueda se va a dividir en 4 subregiones, lo que permite localizar el cuadrante en donde se va a insertar en tiempo logarítmico. Sin embargo, si el árbol está muy desbalanceado, la complejidad puede llegar a ser $O(n)$ en caso de que todos los puntos estén en el mismo cuadrante.

En resumen, la función insert tiene una complejidad de $O(\log n)$ y si el árbol esta muy desbalanceado es $O(n)$.

4.2. En la Función Range Search

Su complejidad va a depender del área de búsqueda y de como están distribuidos los puntos. Si el árbol se encuentra balanceado, su complejidad es de $O(\sqrt{n}+k)$ donde k es el número de puntos que se encuentran en el rango. Sin embargo, si la distribución de los puntos es tal que el rango cubre la mayoría de los nodos, la complejidad puede llegar a ser $O(n)$.

En resumen, la función insert tiene una complejidad de $O(\sqrt{n}+k)$ y si el árbol esta muy desbalanceado y cubre la mayoría de nodos es $O(n)$.

Operación	Caso Promedio	Peor Caso
Insertar (`insert`)	$O(\log n)$	$O(n)$
Búsqueda por rango (`range_search`)	$O(\sqrt{n}+k)$	$O(n)$

Cuadro 1: Complejidad computacional de las operaciones en QuadTree