Laboratorio 03 EDA

Pablo Luis Carazas Barrios

October 2023

1 Introduction

El Rtree es una de las mejores de opciones a la hora de guardar de manera ordenada datos que suelen ser geograficos o geometricos, en el siguiente informe veremos la funcion Search y el grafico adaptado al codigo proporcionado en clase.

2 Cambios a la implementacion propuesta

En esta seccion veremos los cambios realizados al codigo dado.

2.1 Cambio Struct Branch

Este cambio fue igualar los child a un puntero nulo, con el fin de en un futuro poder realizar ciertos algoritmos.

```
struct Branch
{
        Rect m_rect;
        Node* m_child=nullptr;
        vector<pair<int, int>> m_data;
};
```

2.2 Cambio de Class a Struct en RTree

Este cambio fue igualar los child a un puntero nulo, con el fin de en un futuro poder realizar ciertos algoritmos.

```
struct RTree
{
        RTree();
        RTree(const RTree& other);
 ...
 };
```

3 Nuevas Funciones

Se implementaron nuevas funciones, algunas para graficar y facilitar ese proceso y otras para la funcion Search

3.1 Funcion para graficar Rectangulos

Esta funcion usa a la libreria SFML y toma como parametros un Rect, el tamaño de la ventana a graficar, el color del rectangulo y tam que es para el grosor de las lineas

```
int resolucion = 5;
sf::RectangleShape RecDraw(const Rect& R, int windowHeight = 500,
                           sf::Color c = sf::Color::Blue, float tam=1) {
        int xtam = R.m_max[0] - R.m_min[0];
        int ytam = R.m_max[1] - R.m_min[1];
        sf::RectangleShape r({ static_cast<float>(resolucion * xtam),
                               static_cast<float>(resolucion * ytam) });
        r.setPosition(sf::Vector2f(static_cast<float>(resolucion * R.m_min[0]),
                      static_cast<float>(windowHeight - resolucion * R.m_max[1])));
        sf::Color transparentColor = sf::Color(0, 0, 0, 0);
        r.setFillColor(transparentColor);
        sf::Color outlineColor = c;
        r.setOutlineColor(outlineColor);
        r.setOutlineThickness(tam);
        return r;
}
```

3.2 Funcion para graficar Rectangulos

Esta funcion usa a la libreria SFML y toma como parametros un Rect, el tamaño de la ventana a graficar, el color del rectangulo y tam que es para el grosor de las lineas

```
void graficar(Node* node) {
        if (!node) {
                return;
        }
        for (auto& branch : node->m_branch) {
                Rect rect = branch.m_rect;
                sf::Color c = sf::Color::Blue;
                int tam = 2;
                if (node->IsLeaf() == true) {
                        c = sf::Color::Green;
                        tam = 1;
                }
                sf::RectangleShape rectShape = RecDraw(rect, 500, c,tam);
                window.draw(rectShape);

                if (branch.m_child != nullptr) {
                        graficar(branch.m_child);
                }
                else {
                        for (const auto& point : branch.m_data) {
                                sf::CircleShape pointShape(3);
                                pointShape.setFillColor(sf::Color::Black);
                                pointShape.setPosition({ static_cast<float>(resolucion *
                                window.draw(pointShape);
                        }
                }
        }
}
```

usando esta logica de estructura

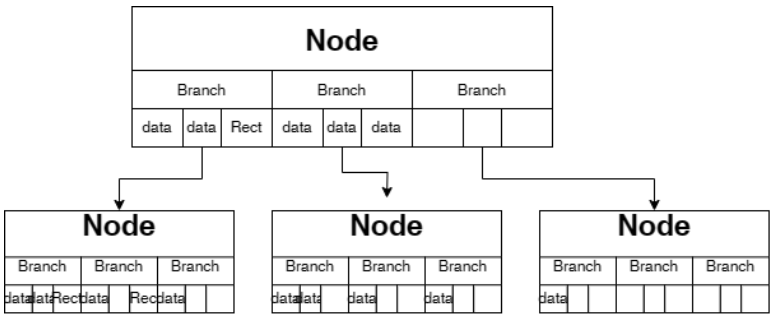


Figure 1: Estructura base del funcionamiento del Rtree

Lo que hace es Entrar a un Nodo, recorrer entre sus branch, si tiene hijo va por ahí, mientras grafica cada Rect, y cuando llega una hoja grafica su data que son puntos.

3.3 Funcion PartialOverlap

Lo unico que hace esta funcion es determinar si hay interseccion entre dos Rectangulos, si es que hay interseccion parcial o inclusion total entre los Rectangulso.

```
bool PartialOverlap(Rect* a_rectA, Rect* a_rectB)
{

        bool overlapX = (a_rectA->m_min[0] <= a_rectB->m_max[0] &&
            a_rectA->m_max[0] >= a_rectB->m_min[0]);

        bool overlapY = (a_rectA->m_min[1] <= a_rectB->m_max[1] &&
            a_rectA->m_max[1] >= a_rectB->m_min[1]);

        return overlapX && overlapY;
}
```

3.4 Funcion Search

Esta funcion usa el mismo algoritmo y logica que nos da el paper de Guttman, entra a un nodo, no es hoja y tiene overlap con el Rectangulo de entrada R ira por sus hijos, asi recursivamente yendo por todos las ramas que cumplan esas dos condiciones, al final todo será pusheado en un vector auxiliar que se da como parametro y listo.

```
void Search(Node* a_node, Rect& R, vector<Rect>& result)
{
        if (a_node->IsLeaf())
        {
                for (int i = 0; i < a_node->m_count; ++i)
                {
                        if (PartialOverlap(&R, &(a_node->m_branch[i].m_rect)))
                        {
                                result.push_back(a_node->m_branch[i].m_rect);
                        }
                }
        }
        else
        {
                for (int i = 0; i < a_node->m_count; ++i)
                {
                        if (PartialOverlap(&R,&(a_node->m_branch[i].m_rect)))
                        {
```

```
                                Search(a_node->m_branch[i].m_child, R, result);
                            }
                        }
                    }
}
```

3.5 Insersiones

Se modifico la forma en la que se hacia la insrcion de los puntos base y tambien se usó dos `for` para cuando se hagan los insert, en los ejemplos que veremos se inserta la misma cantidad de grupos de 2 y 3 puntos.

```
    vector<pair<int, int>> points = { {20, 59}, {20, 43}, {50, 58}, {48, 67},
                                      {105, 68}, {74, 64}, {83, 40}, {104, 54} };

        for (unsigned int i = 0; i < points.size(); i += 2) {
                vector<pair<int, int>> sub1(points.begin() + i,
                                            points.begin() + i + 2);
                vpoints.push_back(sub1);
        }
        vector<pair<int, int>> points2 = { {12, 28}, {19, 15}, {40, 29},
                                           {69, 25}, {70, 28}, {60, 15} };
        for (unsigned int i = 0; i < points2.size(); i += 3) {
                vector<pair<int, int>> sub1(points2.begin() + i,
                                            points2.begin() + i + 3);
                vpoints.push_back(sub1);
        }
        for (int i = 0; i < 0; ++i) {
                vector<pair<int, int>> pointPair(2);
                pointPair[0] = make_pair(10+rand() % 90, 10+rand() % 90);
                pointPair[1] = make_pair(10+rand() % 90, 10+rand() % 90);
                vpoints.push_back(pointPair);
        }

        for (int i = 0; i < 0; ++i) {
                vector<pair<int, int>> pointPair(3);

                pointPair[0] = make_pair(10 + rand() % 90, 10 + rand() % 90);
                pointPair[1] = make_pair(10 + rand() % 90, 10 + rand() % 90);
                pointPair[2] = make_pair(10 + rand() % 90, 10 + rand() % 90);

                vpoints.push_back(pointPair);
        }
```

4 Resultados

A continuacion veremos los resultados que tenemos de nuestro proyecto respecto a la parte grafica en conjuncion a la consulta que se hace mediante la funcion Search

4.1 Funcion Graficadora 1

Esta funcion trata de lograr por completo lo que se pidió, graficar de modo que los Rectangulos de las hojas queden en otro color (Ramas azules y hojas verdes), si se pudo hacer, con la funcion que fue mostrada instantes atras, el problema que tiene esto es que no se es capaz de graficar cuando se agregan más de 50 pares de puntos insertados con una capacidad máxima de 3 puntos por nodo, los resultados son los siguientes:

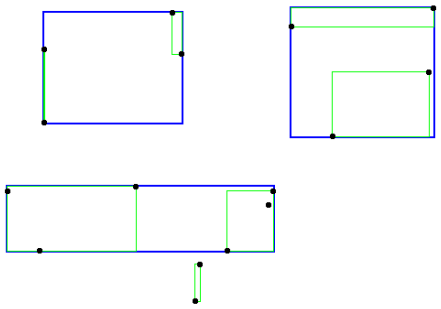
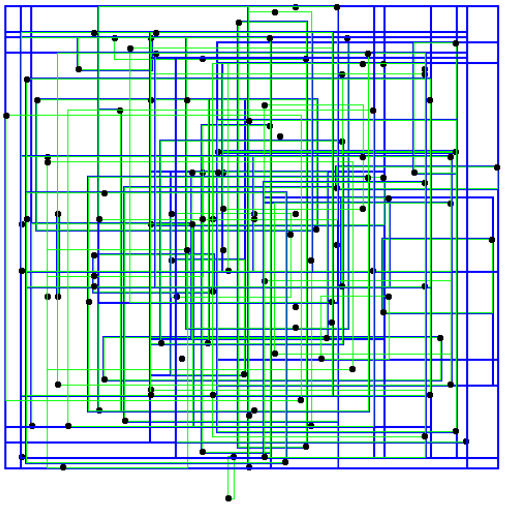
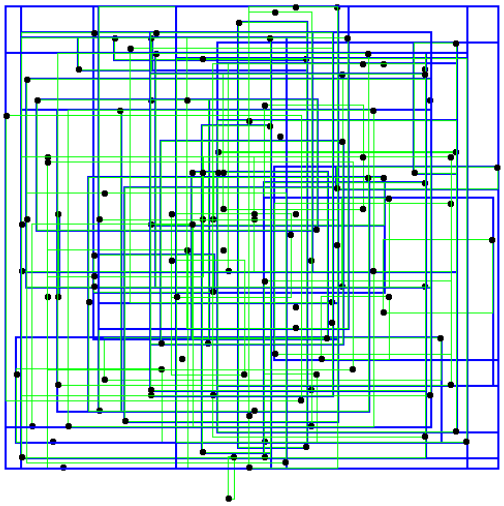


Figure 2: Grafico con los puntos base

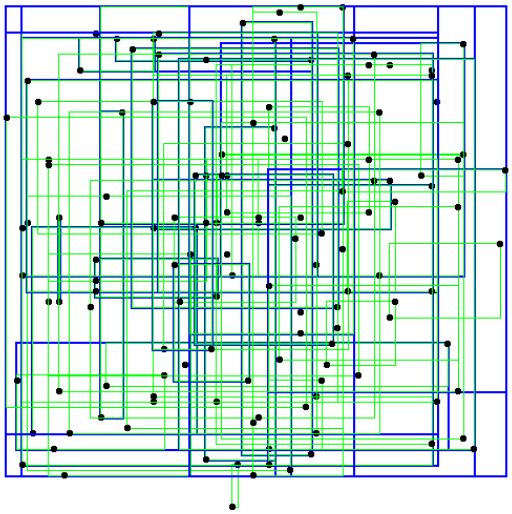
4.1.1 50 puntos



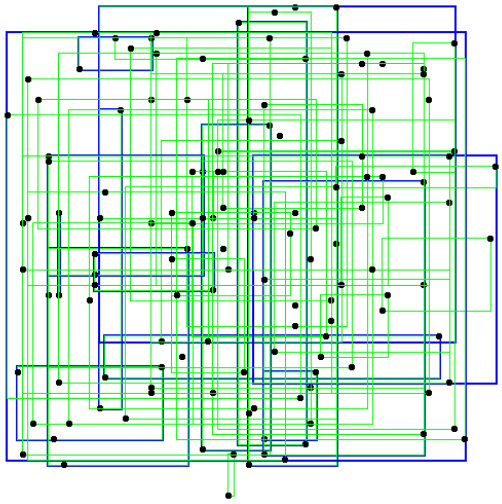
(a) 3 puntos por nodo



(b) 5 puntos por nodo

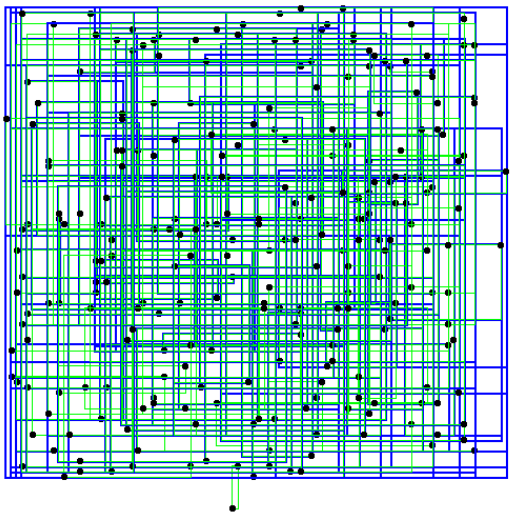


(c) 10 puntos por nodo

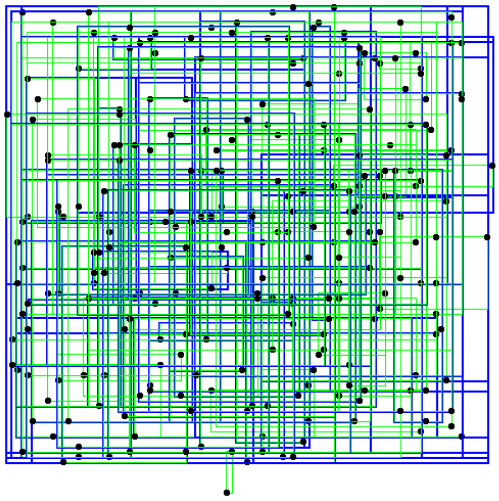


(d) 20 puntos por nodo

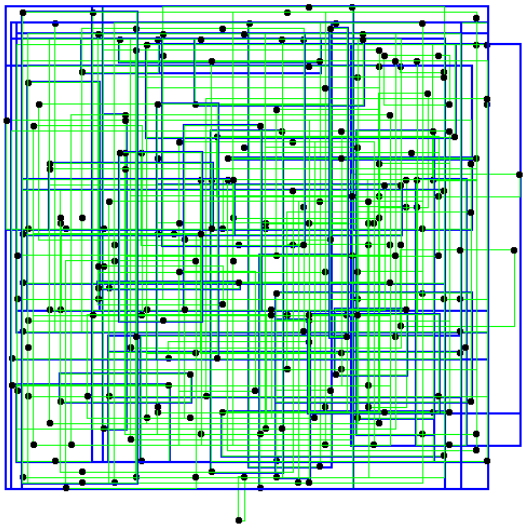
4.1.2 100 puntos



(a) 5 puntos por nodo

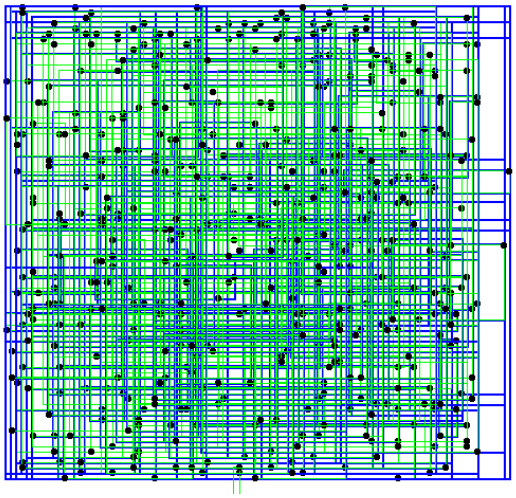


(b) 10 puntos por nodo

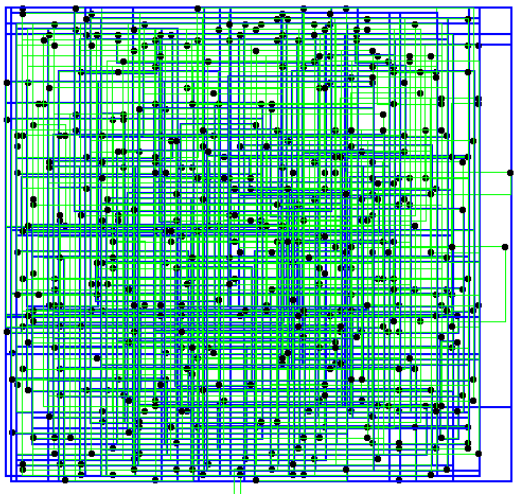


(c) 20 puntos por nodo

4.1.3 200 puntos

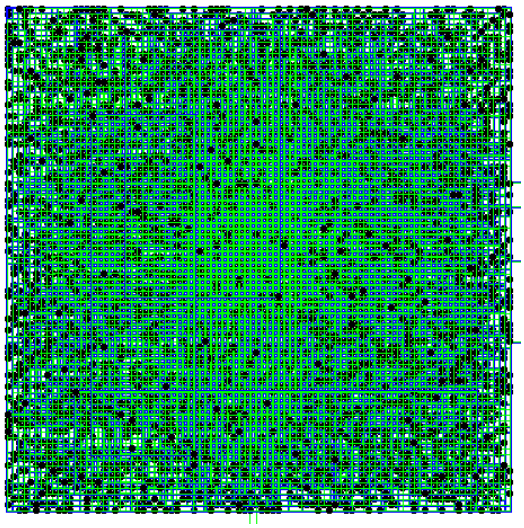


(a) 10 puntos por nodo



(b) 20 puntos por nodo

4.1.4 1000 puntos



(a) 800 puntos por nodo

5 Funcion Graficadora 2

La otra parte del codigo que tambien grafica es algo más simple, basicamente como el Rtree almacena todas las hojas en un vector de vectores llamado `mObjs`, entonces lo unico que se hace es recorrer el vector imprimiendo los cuadrados y puntos, lo bueno de este modo es que permite insertar cuantos nodos se quiera con cualquier restriccion, lo malo es que no se diferencia cuando se cambia la cantidad de rectangulos por nodo

```
for (auto& it : rtree.mObjs) {              //graficador hojas
        window.draw(RecDraw(Rect(rtree.MBR(it))));
        PointDraw(it);
}
```

dando este resultado para los nodos base que se dan.

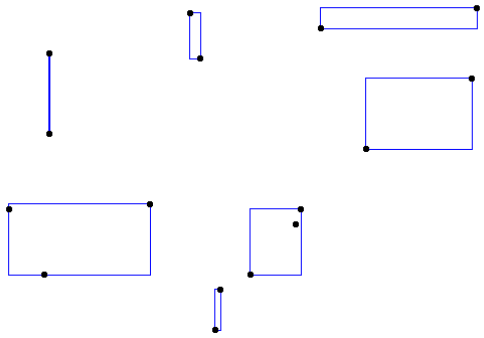
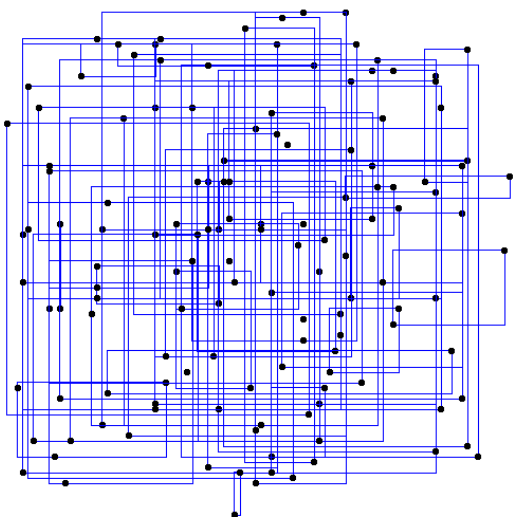


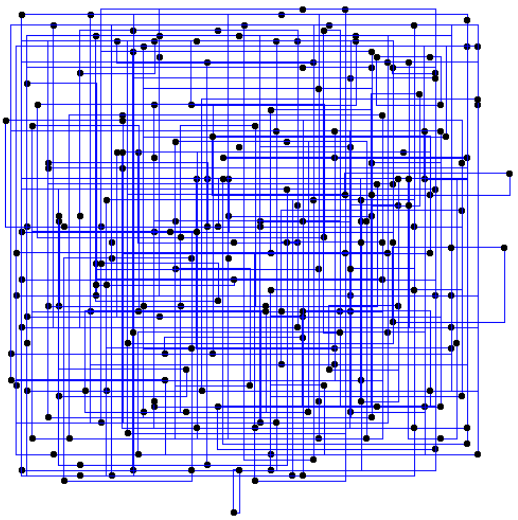
Figure 7: Graficado con puntos base

y dando este resultado para los siguientes casos

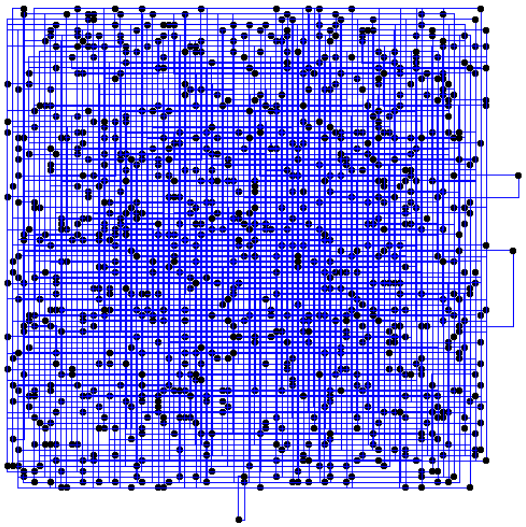
5.0.1 puntos por nodo



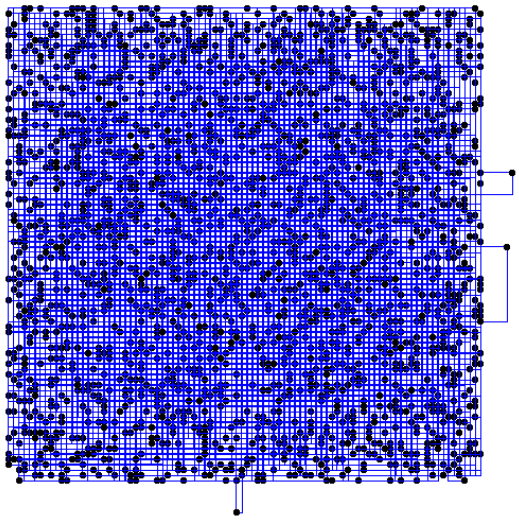
(a) 50 insersiones



(b) 100 insersiones



(c) 200 insersiones

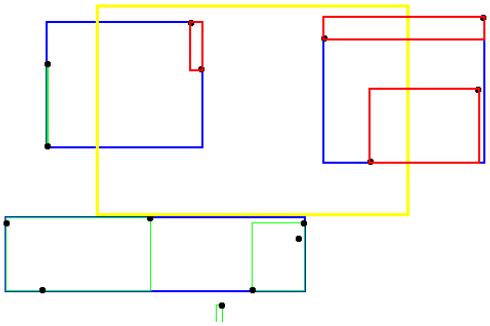


(d) 1000 insersiones

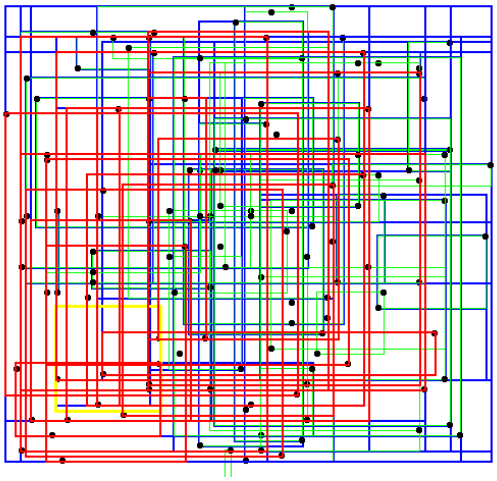
5.1 Consultas

Ahora mostraremos la implementacion de la funcion Search cuando se le da ciertas consultas, el rectangulo que se usará para consultar es de color amarillo, los rectangulos que cumplan la condicion y serán devueltos serán graficados de color rojo, veremos como funcionan las consultas para diferentes grupos de datos con las dos funciones graficadoras.

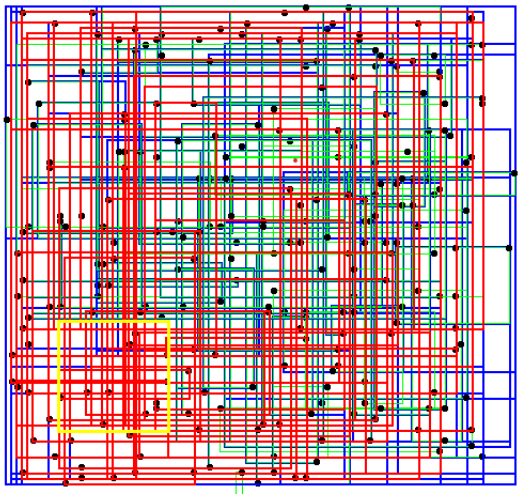
5.1.1 Consultas Graficador 1



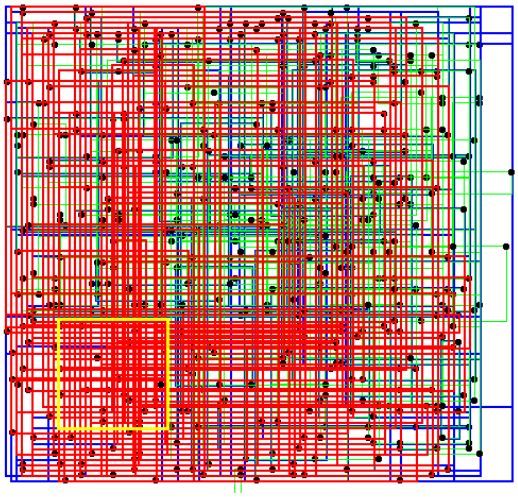
(a) Consulta con puntos base



(b) Consulta con 50 puntos

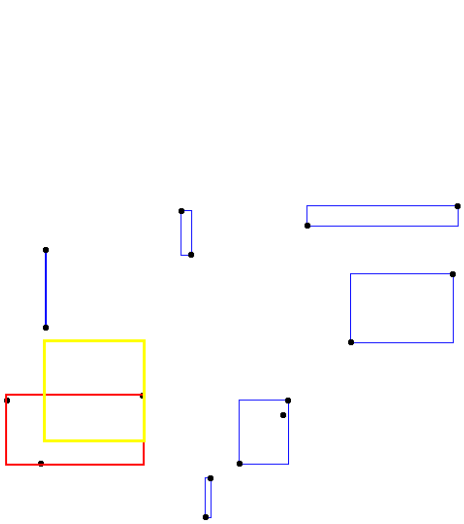


(c) Consulta con 100 puntos

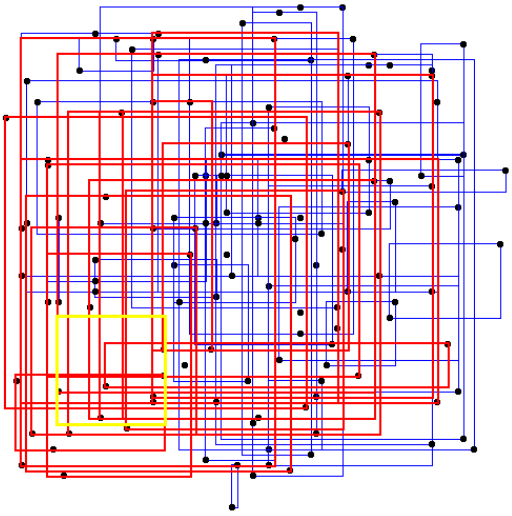


(d) Consulta con 200 puntos

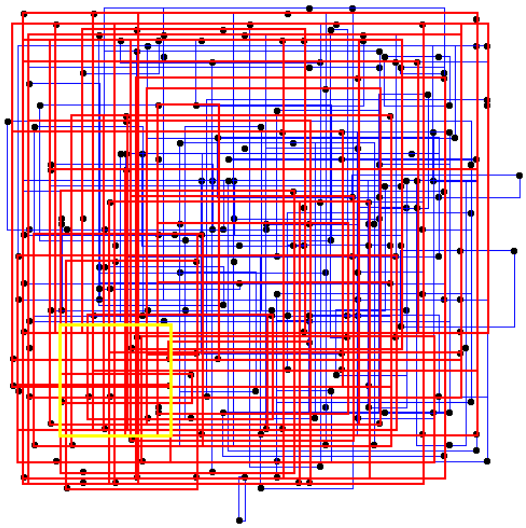
5.1.2 Consultas Graficador 2



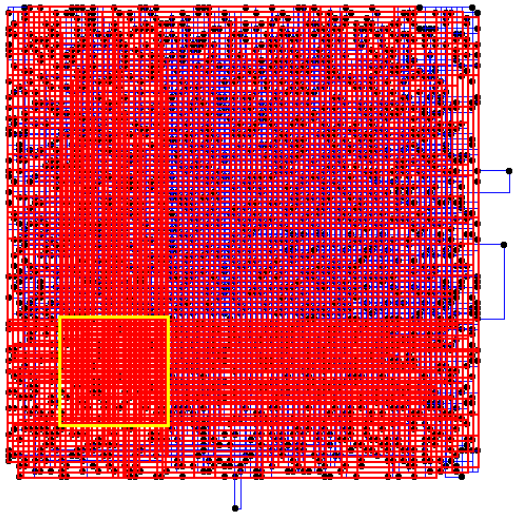
(a) Consulta con puntos base



(b) Consulta con 50 puntos



(c) Consulta con 100 puntos



(d) Consulta con 1000 puntos

6 Resultados

Los resultados respecto a la funcion que grafica detallado no son los esperados, no encuentro una explicacion del fallo pero parece que ser que tantas llamadas recursivas con tanto for terminan sofocando el espacio que se le da a visual en el stack asi sobrepasandolo, eso explica el error que da a la hora de hacer un debbug (no posible acceder a esa memoria) pero todo sigue funcionando normal cuando no se llama a graficar, pero como mencioné es una suposición y no logro determinar el origen del fallo.
De todos modos se logró graficar de 2 maneras diferentes los mismos arboles, se logro hacer las consultas como se esperaban y tambien graficarlas.

7 Comentarios

A opinion propia, una de las partes mas dificiles de realizar fue el entender el codigo para poder utilizarlo, desde mi perspectiva hay ciertas estructuras que se pueden obviar hasta cierto punto lo que facilitaría el acceso a ciertos lados de memoria evitando ciertos bucles, pero como mencionó es algo propio y tal vez a la hora de realizar el codigo mencionado me encuentre con fallas.

8 Enlaces

Github con el codigo fuente e informe