



Universidad Católica
San Pablo

GRUPO:

CCOMP6-1

CURSO:

ESTRUCTURA DE DATOS AVANZADOS

PROFESORA:

Rosa Yuliana Gabriela Paccotacya Yanque

INTEGRANTES:

- ANTHONY RODRIGUEZ PINTO

Arequipa - 2024

Laboratorio 3: RTree función search

Introducción

El R Tree es una estructura de datos muy estudiado por la manera ordenada que suele almacenar datos n-dimensionales, en el presente informe veremos la función Search y el gráfico para poder visualizar las intersecciones.

Funciones implementadas

- **Función Search**

En esta función sigue el pseudo código que nos da el paper de guttman. Si no es hoja va ir buscando donde exista overlap, y se vuelve a llamar a la función search. En caso que sea hoja mira donde exista overlap y se inserta en el vector para almacenar los resultados.

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

- **Función Overlapping**

Esta función nos ayudará a encontrar si existe overlap entre las consultas, en este caso modifique el overlap2 del código proporcionado, ya que me salía un error de datos incompatibles, por lo que hago que me devuelva si hay overlap en el eje X y en el eje Y.

```

bool Overlapping(const Rect* a_rectA, const 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;
}

```

FUNCIONES INSERCCION:

Antes de utilizar las funciones graficadoras lo que debemos hacer es insertar los puntos aleatorios solicitados, por lo que se instancia de la siguiente manera:

```

//Insteramos puntos aleatorios
int n_rand = 2;
for(int d=0;d<n_rand;d++)
{
    vector<pair<int, int>> pointPair(2);
    for (int i = 0; i < 2; ++i) {
        pointPair[i] = make_pair(rand() % 90, rand() % 100);
    }
    vpoints.push_back(pointPair);

    vector<pair<int, int>> pointPair2(3);
    for (int i = 0; i < 3; ++i) {
        pointPair2[i] = make_pair(rand() % 90, 10 + rand() % 80);
    }
    vpoints.push_back(pointPair2);
}

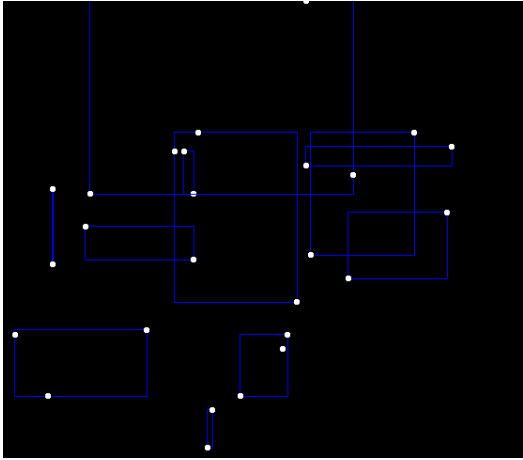
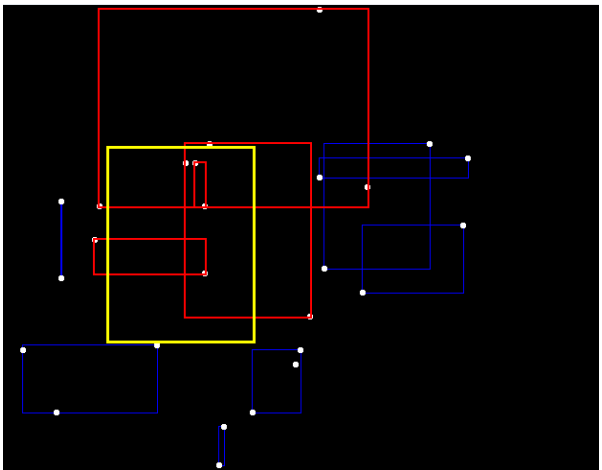
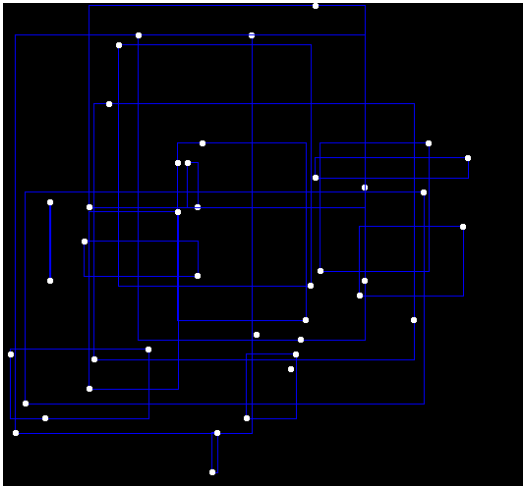
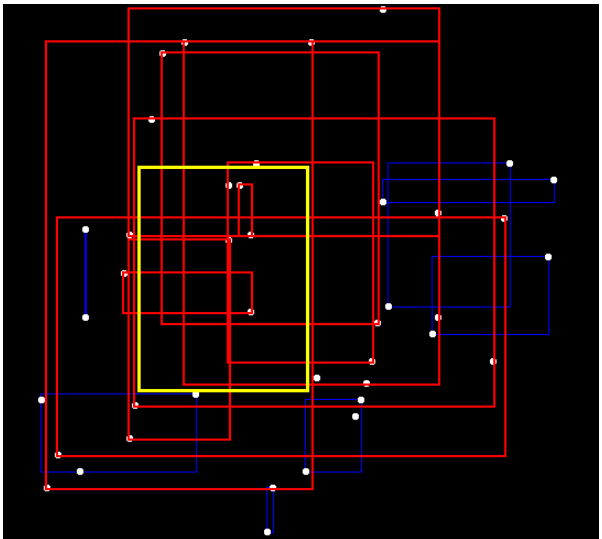
```

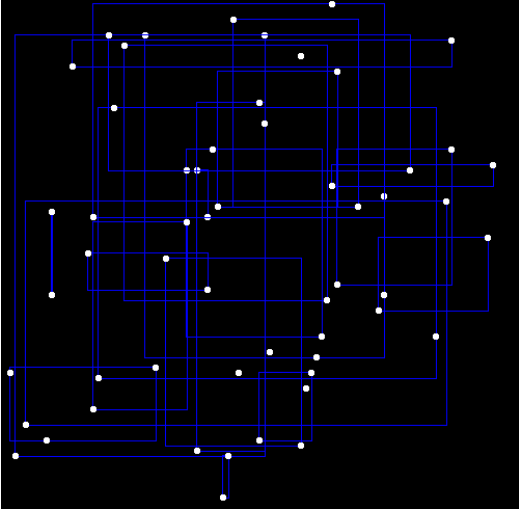
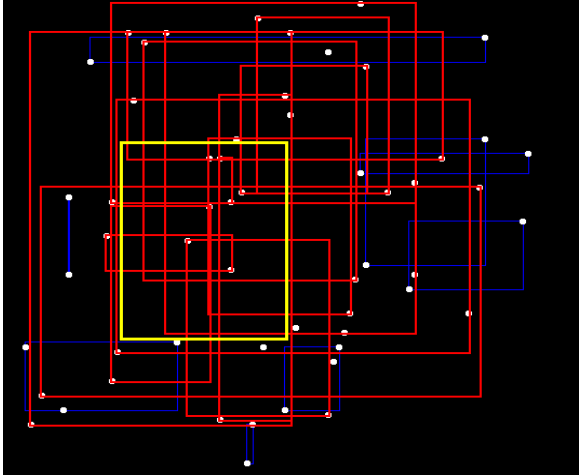
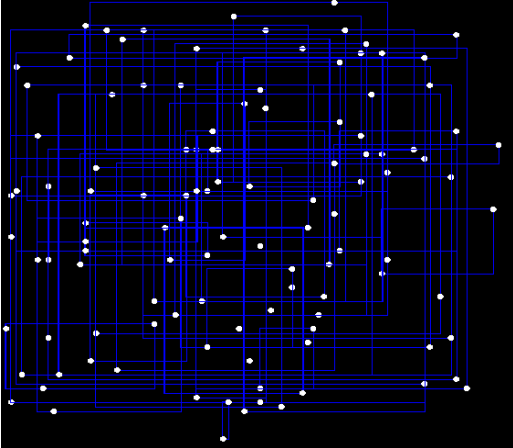
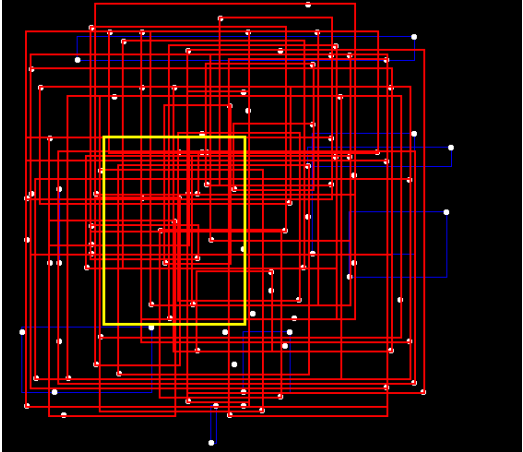
El n_rand es igual a 2, esto se modifica a la cantidad de puntos, ya que como se nos solicitó que se grupos de 2 puntos y 3 puntos, por lo que se distribuye en dos vectores una para 2 puntos, y otro para 3 puntos.

Resultados

Los resultados obtenidos se mostrarán de dos manera, en la primera se grafica sin la consulta search haciendo que solo se grafique los MBR's de los nodos en el arbol Rtree. En el segundo caso lo que realizamos es el grafico con la consulta search donde le pasamos un rectangulo que

esta de color amarillo y la respuesta se encontrara de color rojo, ya que estos son los que hacen overlap con nuestro rectangulo.

NÚMERO DE PUNTOS	GRÁFICO SIN CONSULTA SEARCH	GRÁFICO CON CONSULTA SEARCH
+10 PUNTOS ALEATORIOS		
+25 PUNTOS ALEATORIOS		

<p>+40 PUNTOS ALEATORIOS</p>		
<p>+100 PUNTOS ALEATORIOS</p>		

Conclusiones

Si bien la estructura Rtree es muy buena para almacenar varios datos, pero al no haberlo realizado desde cero se hizo una dificultad para entender cómo estaba organizado y el funcionamiento, y eso me llevó a modificar una que otra función para poder realizar la tarea. Además, la parte gráfica se me presenta como una dificultad muy cercana.

Enlaces

<https://github.com/Anthony-Rodriguez18/Search-Rtree>