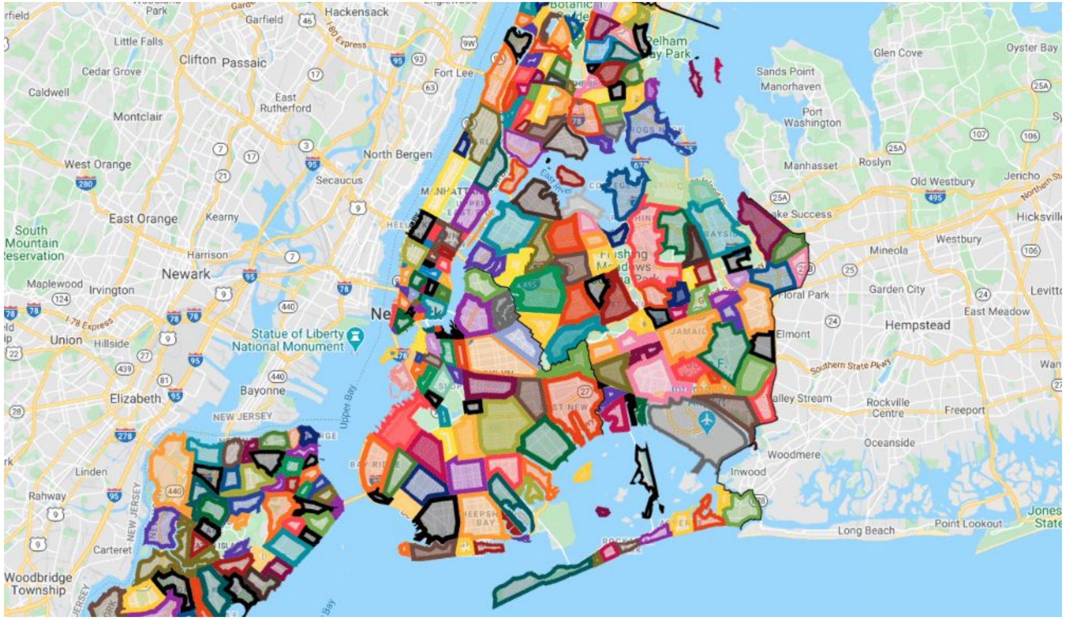
Proyecto Final

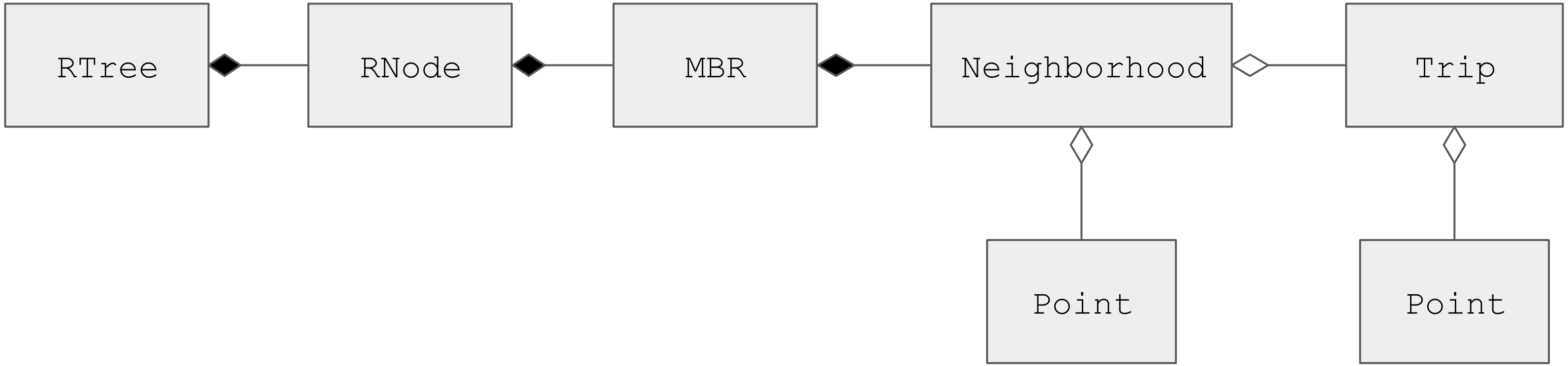
Estructuras de Datos Avanzadas

Alessia Yi, Maor Roizman, Macarena Oyague

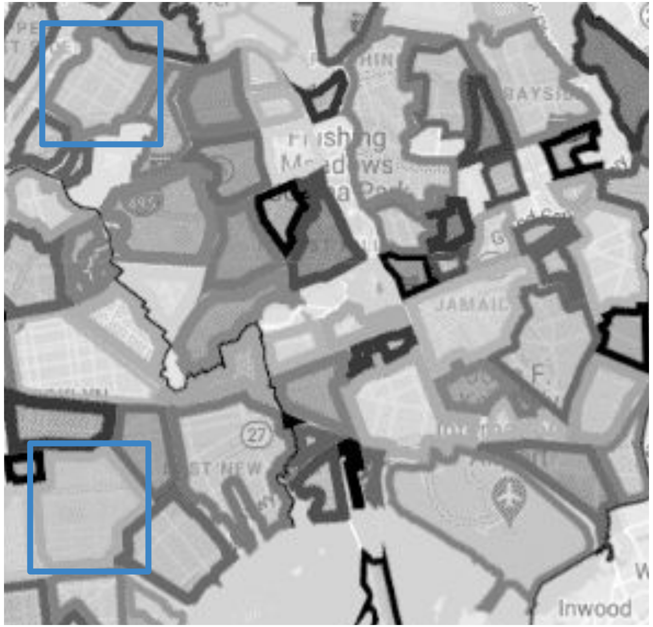
New York City



Estructura de Clases



Minimum Bounding Rectangle

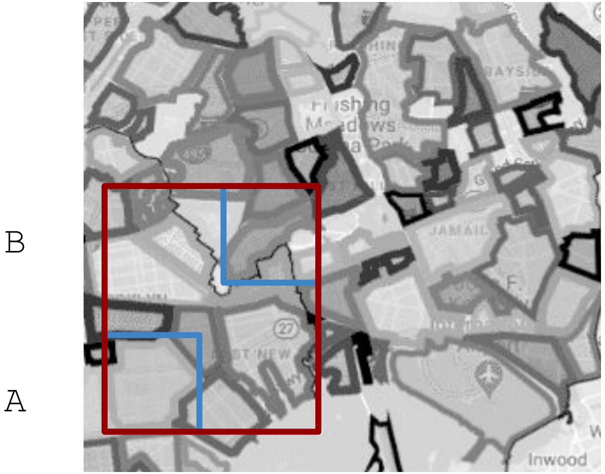


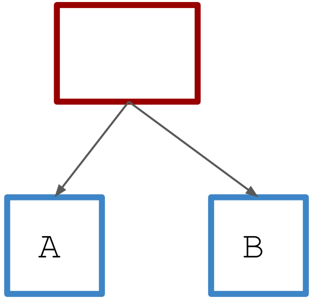
Barrio:
Coordenadas de un polígon

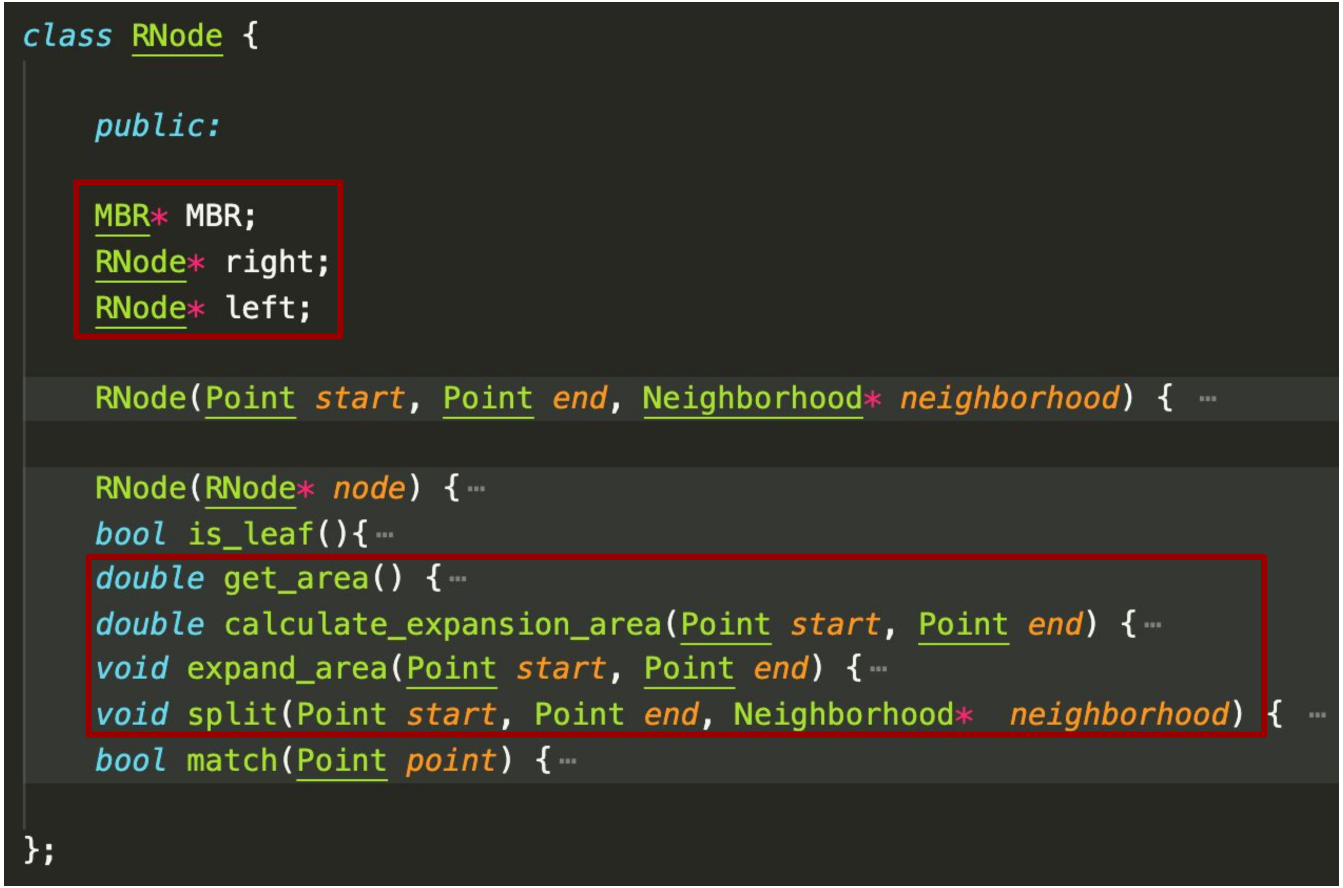
-> Simplificar:
Point min y max

Estructura de Datos Propuesta

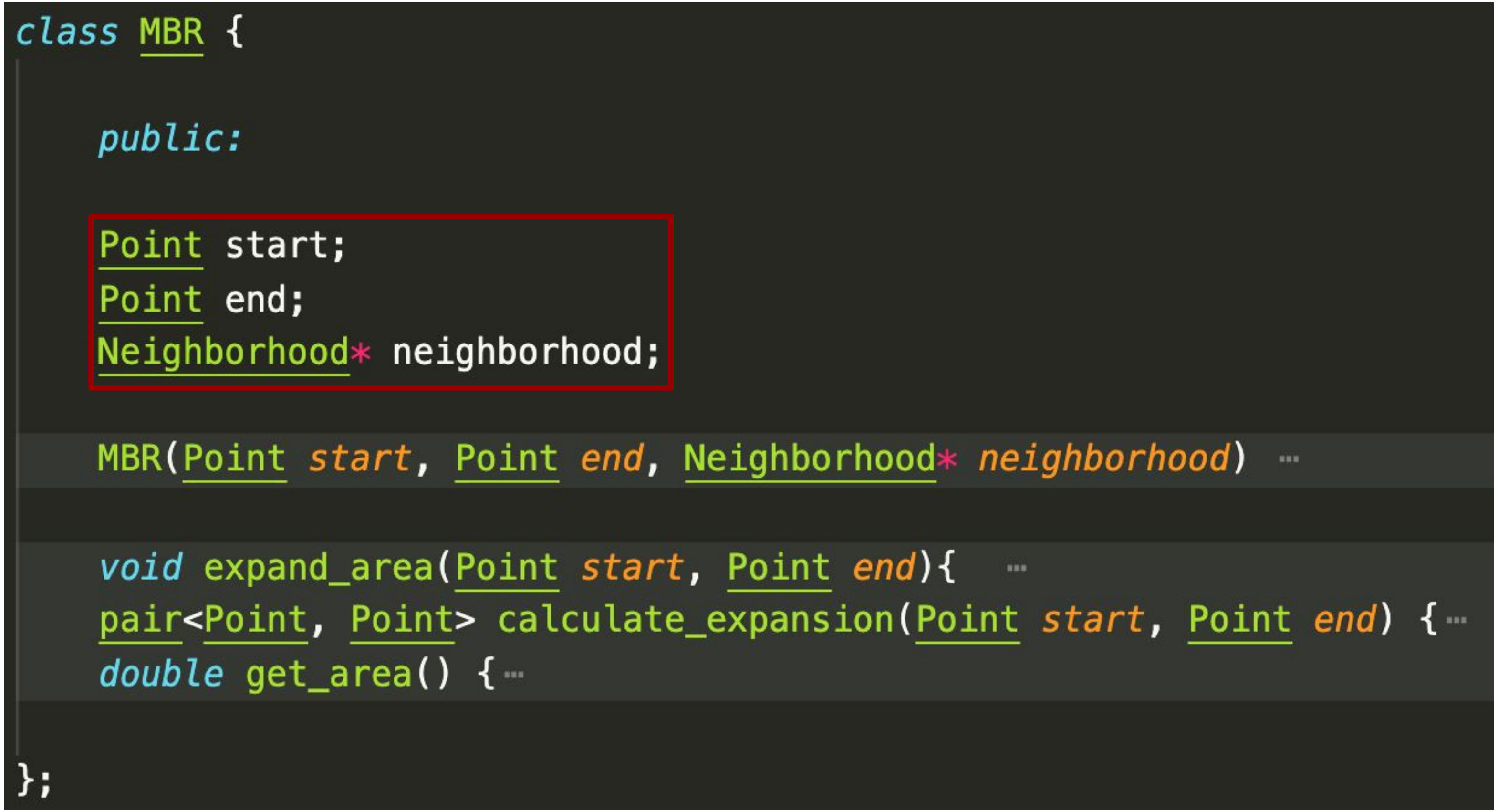
-> Reto: minimizar memoria utilizada





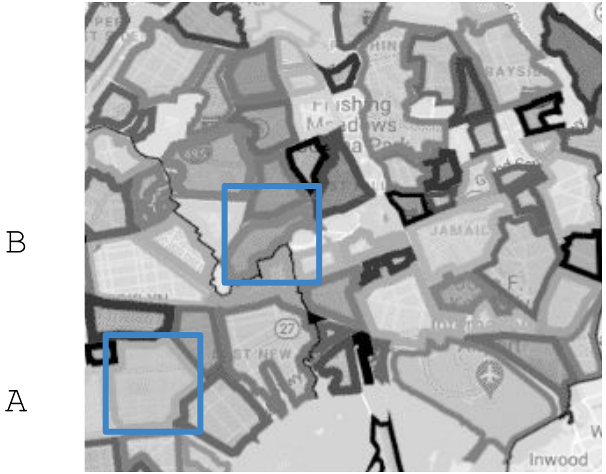


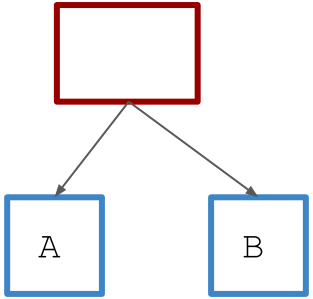
```cpp
class RNode {

    public:

    MBR* MBR;
    RNode* right;
    RNode* left;

    RNode(Point start, Point end, Neighborhood* neighborhood) { ⋯

    RNode(RNode* node) {⋯
    bool is_leaf(){⋯
    double get_area() {⋯
    double calculate_expansion_area(Point start, Point end) {⋯
    void expand_area(Point start, Point end) {⋯
    void split(Point start, Point end, Neighborhood*  neighborhood) { ⋯
    bool match(Point point) {⋯

};
```



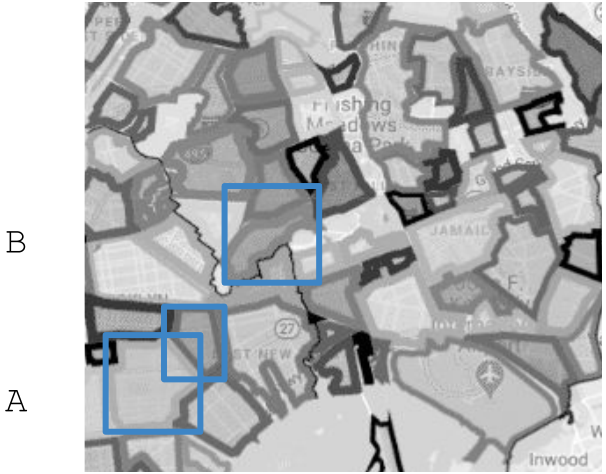
```
class MBR {

    public:

    Point start;
    Point end;
    Neighborhood* neighborhood;

    MBR(Point start, Point end, Neighborhood* neighborhood) ⋯

    void expand_area(Point start, Point end){  ⋯
    pair<Point, Point> calculate_expansion(Point start, Point end) {⋯
    double get_area() {⋯

};
```

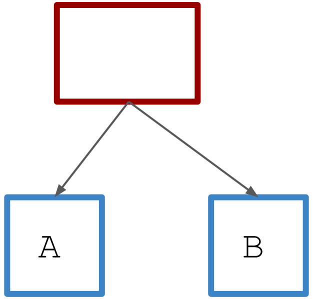
Estructura de Datos Propuesta



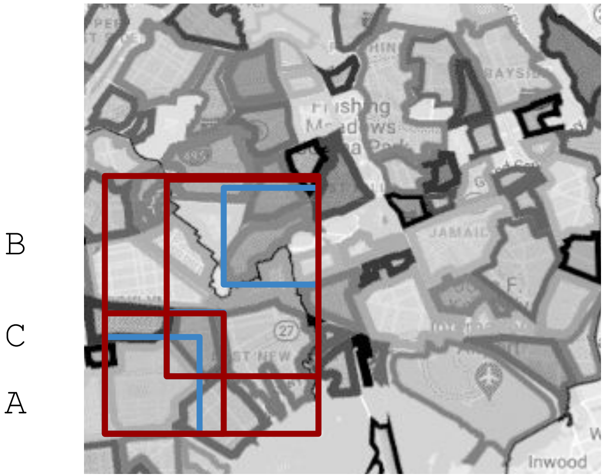


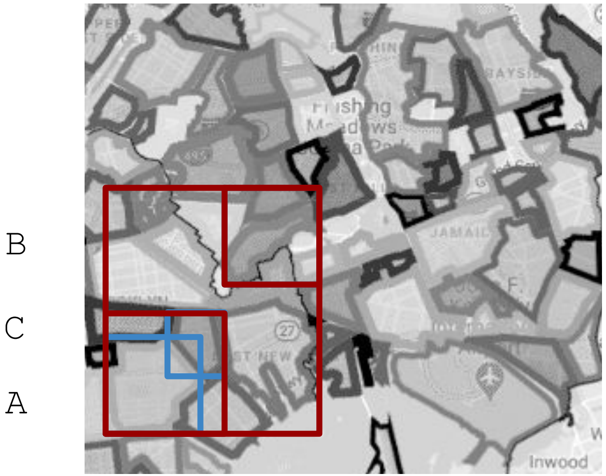
Estructura de Datos Propuesta



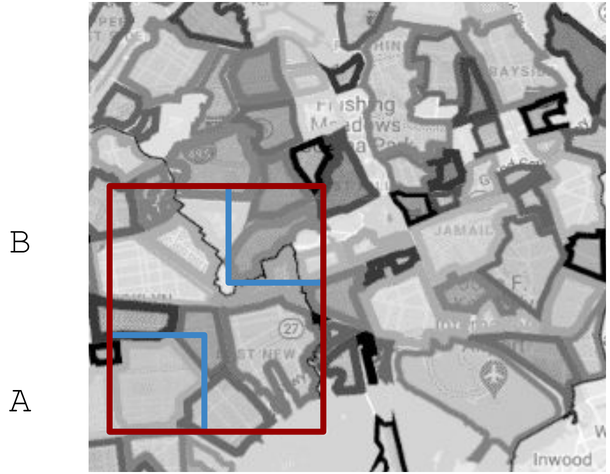


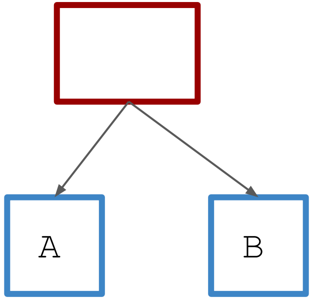
Estructura de Datos Propuesta



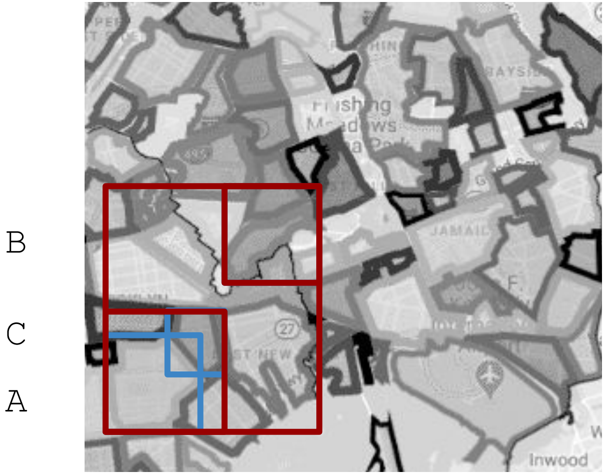


Estructura de Datos Propuesta

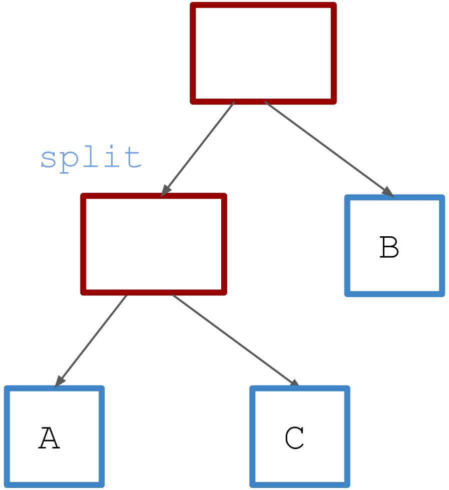


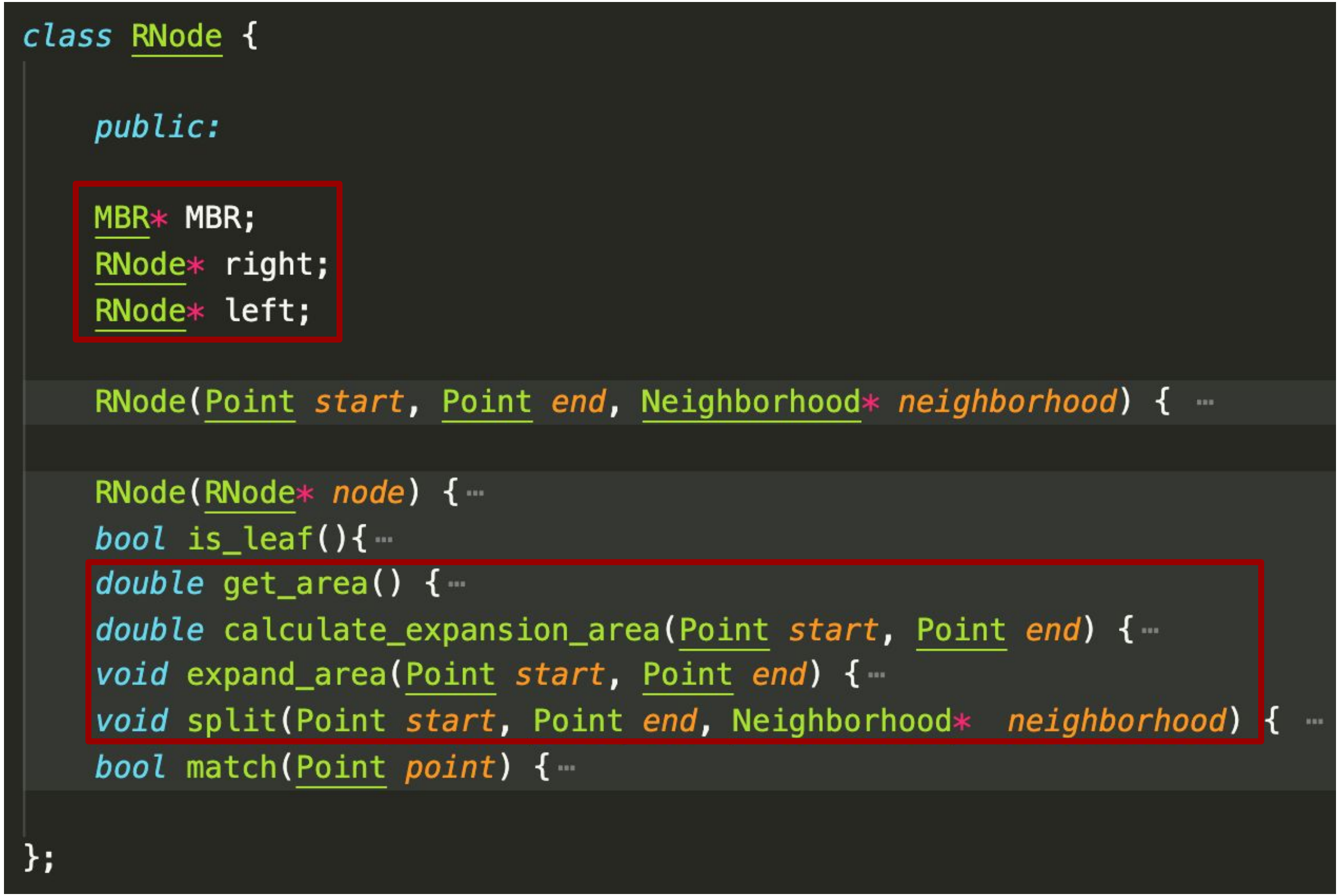


Estructura de Datos Propuesta



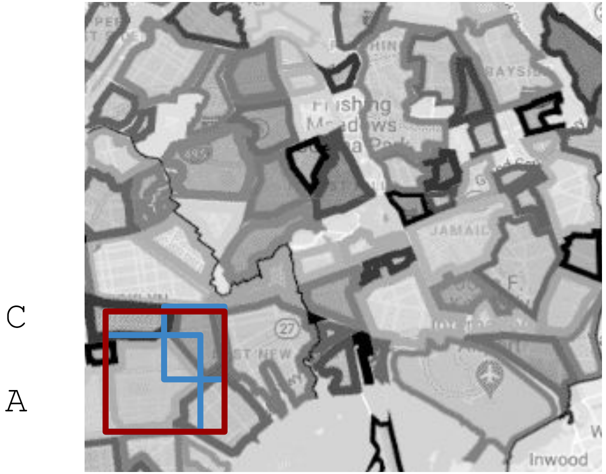
calculate_expansion_area

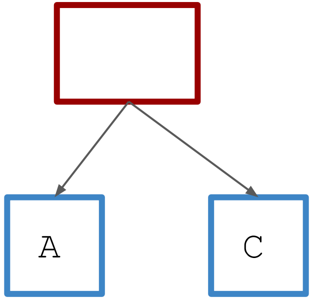




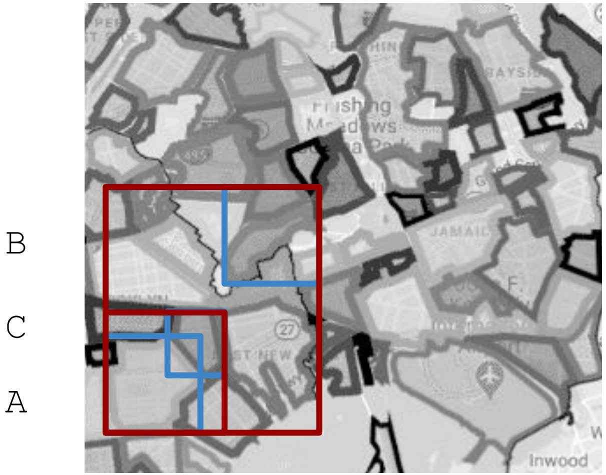
```
class RNode {

    public:

    MBR* MBR;
    RNode* right;
    RNode* left;

    RNode(Point start, Point end, Neighborhood* neighborhood) { ⋯

    RNode(RNode* node) {⋯
    bool is_leaf(){⋯
    double get_area() {⋯
    double calculate_expansion_area(Point start, Point end) {⋯
    void expand_area(Point start, Point end) {⋯
    void split(Point start, Point end, Neighborhood*  neighborhood) { ⋯
    bool match(Point point) {⋯

};
```

Estructura de Datos Propuesta

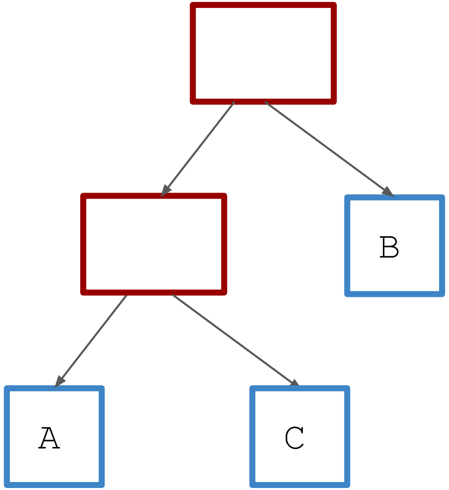


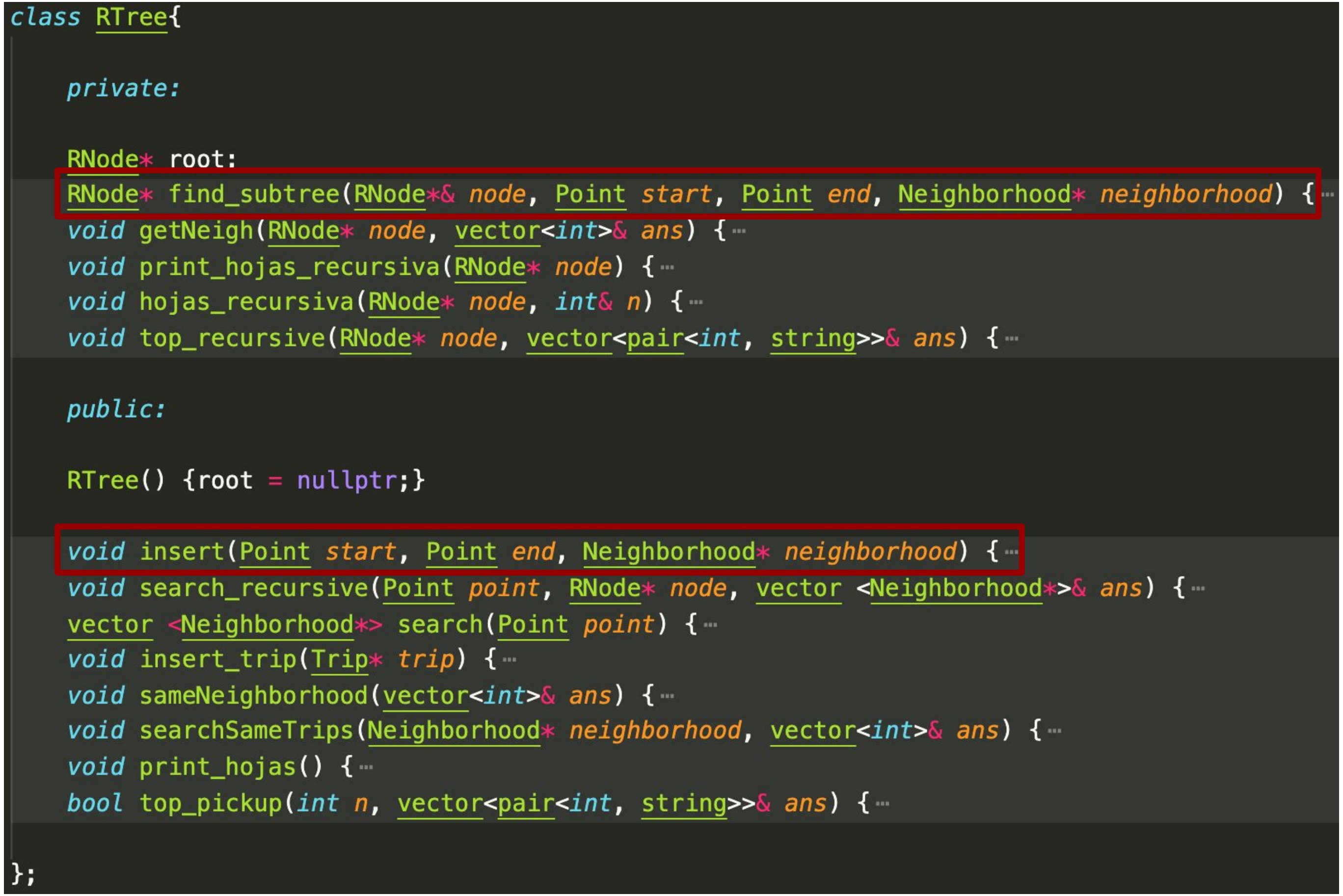


Estructura de Datos Propuesta

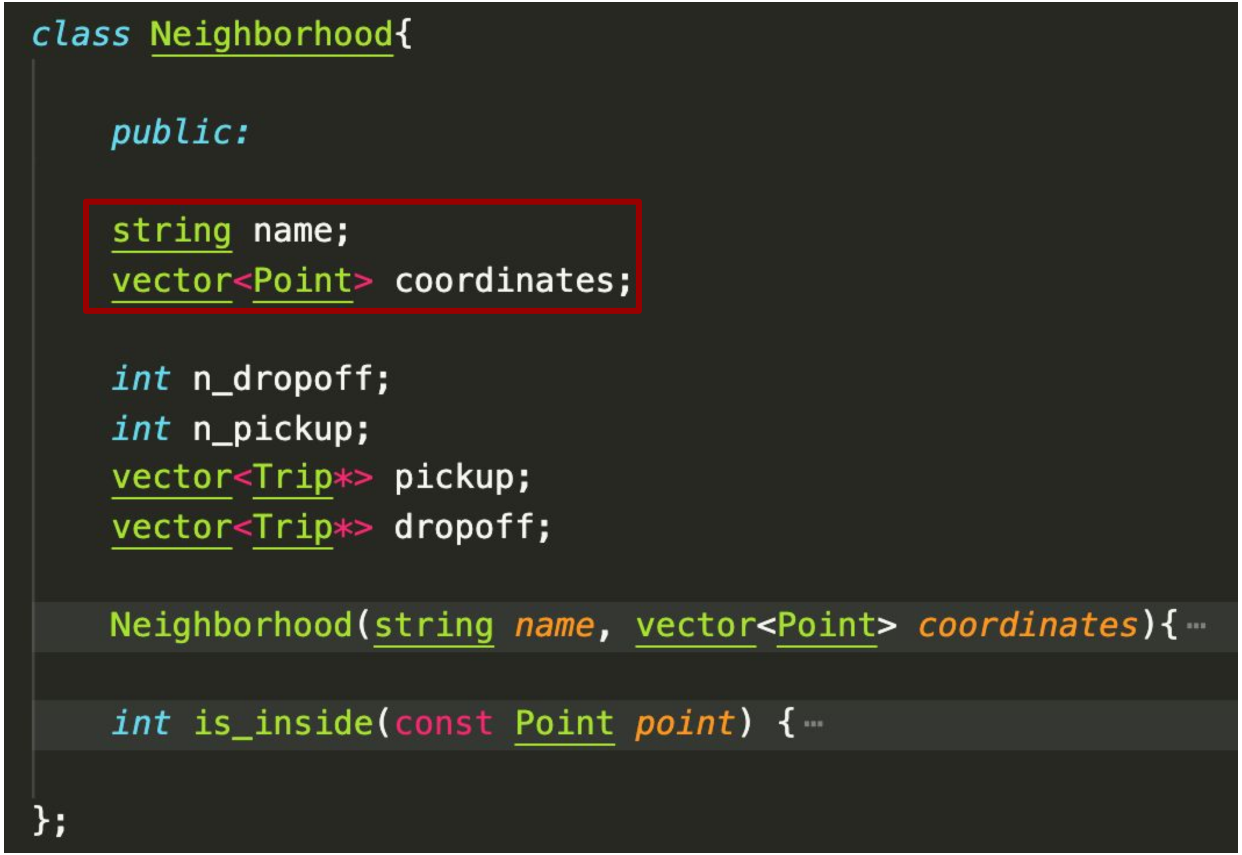


expand_area

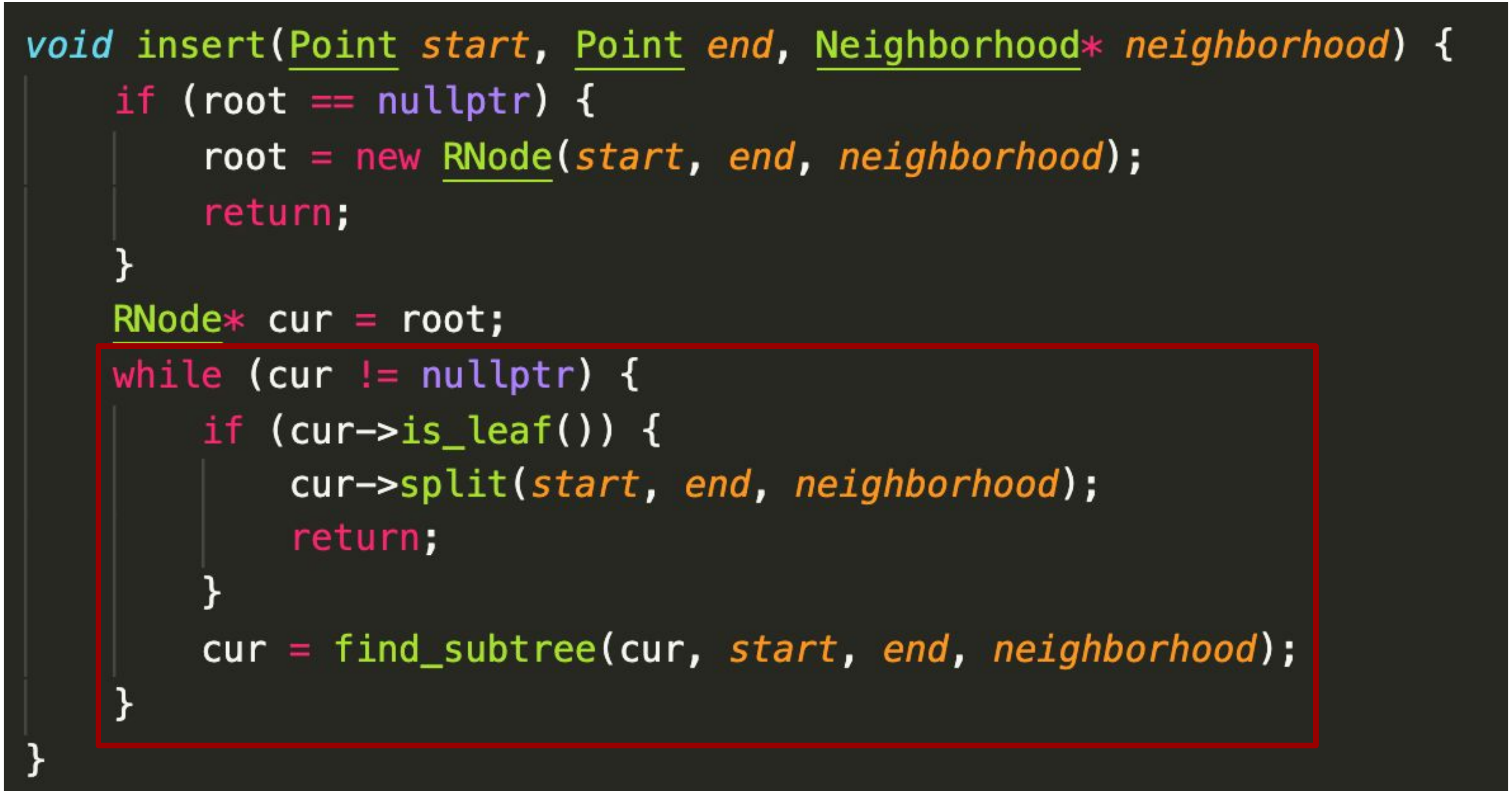




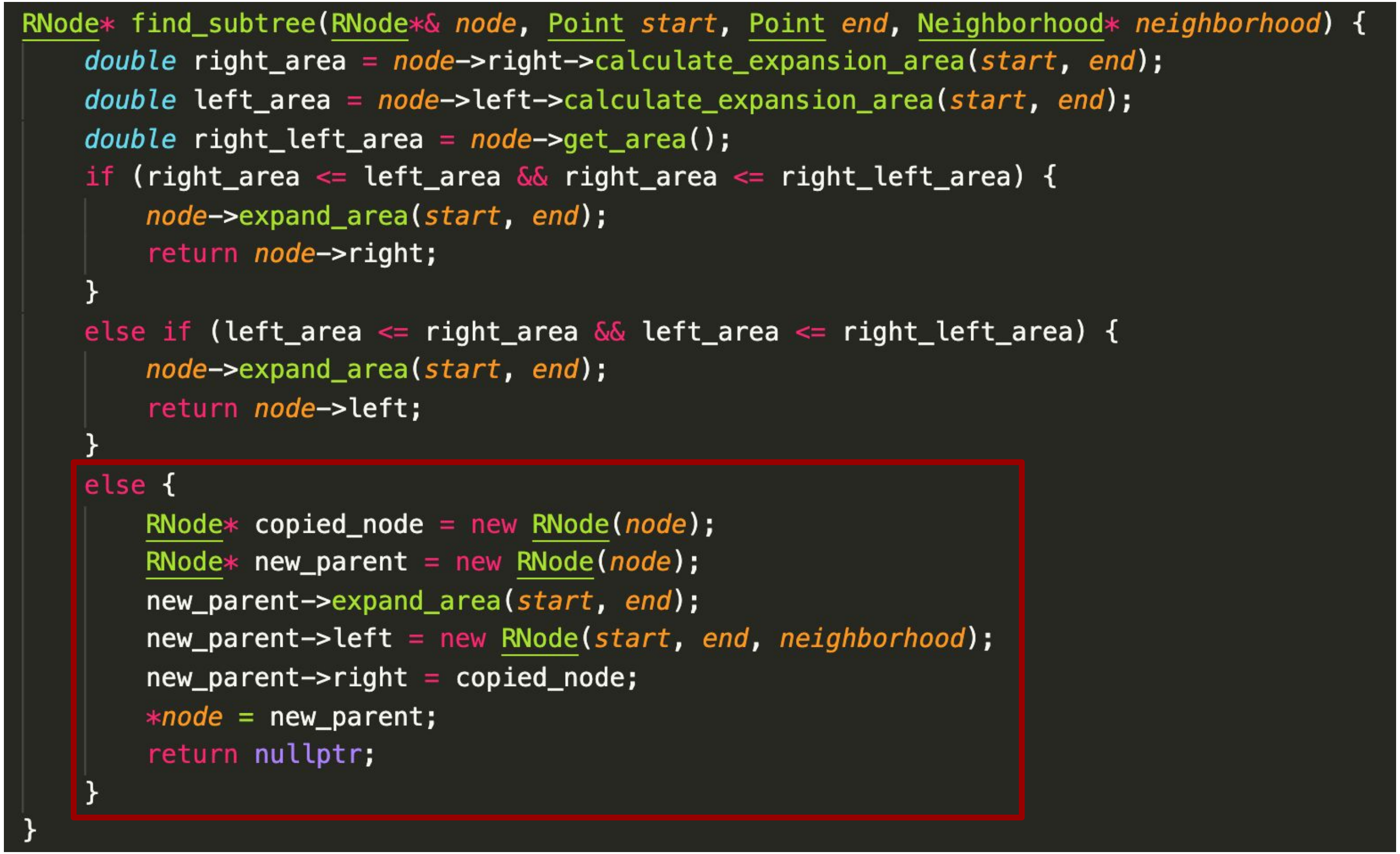
```cpp
class RTree{

    private:

    RNode* root:
    RNode* find_subtree(RNode*& node, Point start, Point end, Neighborhood* neighborhood) {⋯
    void getNeigh(RNode* node, vector<int>& ans) {⋯
    void print_hojas_recursiva(RNode* node) {⋯
    void hojas_recursiva(RNode* node, int& n) {⋯
    void top_recursive(RNode* node, vector<pair<int, string>>& ans) {⋯

    public:

    RTree() {root = nullptr;}

    void insert(Point start, Point end, Neighborhood* neighborhood) {⋯
    void search_recursive(Point point, RNode* node, vector <Neighborhood*>& ans) {⋯
    vector <Neighborhood*> search(Point point) {⋯
    void insert_trip(Trip* trip) {⋯
    void sameNeighborhood(vector<int>& ans) {⋯
    void searchSameTrips(Neighborhood* neighborhood, vector<int>& ans) {⋯
    void print_hojas() {⋯
    bool top_pickup(int n, vector<pair<int, string>>& ans) {⋯

};
```



```
class Neighborhood{

    public:

    string name;
    vector<Point> coordinates;

    int n_dropoff;
    int n_pickup;
    vector<Trip*> pickup;
    vector<Trip*> dropoff;

    Neighborhood(string name, vector<Point> coordinates){⋯

    int is_inside(const Point point) {⋯

};
```



```
void insert(Point start, Point end, Neighborhood* neighborhood) {
    if (root == nullptr) {
        root = new RNode(start, end, neighborhood);
        return;
    }
    RNode* cur = root;
    while (cur != nullptr) {
        if (cur->is_leaf()) {
            cur->split(start, end, neighborhood);
            return;
        }
        cur = find_subtree(cur, start, end, neighborhood);
    }
}
```



```cpp
RNode* find_subtree(RNode*& node, Point start, Point end, Neighborhood* neighborhood) {
    double right_area = node->right->calculate_expansion_area(start, end);
    double left_area = node->left->calculate_expansion_area(start, end);
    double right_left_area = node->get_area();
    if (right_area <= left_area && right_area <= right_left_area) {
        node->expand_area(start, end);
        return node->right;
    }
    else if (left_area <= right_area && left_area <= right_left_area) {
        node->expand_area(start, end);
        return node->left;
    }
    else {
        RNode* copied_node = new RNode(node);
        RNode* new_parent = new RNode(node);
        new_parent->expand_area(start, end);
        new_parent->left = new RNode(start, end, neighborhood);
        new_parent->right = copied_node;
        *node = new_parent;
        return nullptr;
    }
}
```

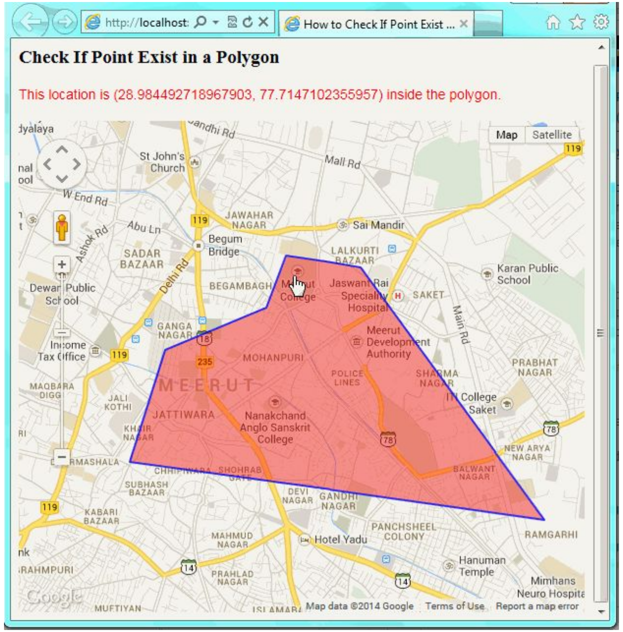
Estructura de Datos Propuesta

-> RTree Binario

- Sigue el principio de partición del RTree
- Tiene right y left
- No hay split cuadrático

Estructura de Datos Propuesta

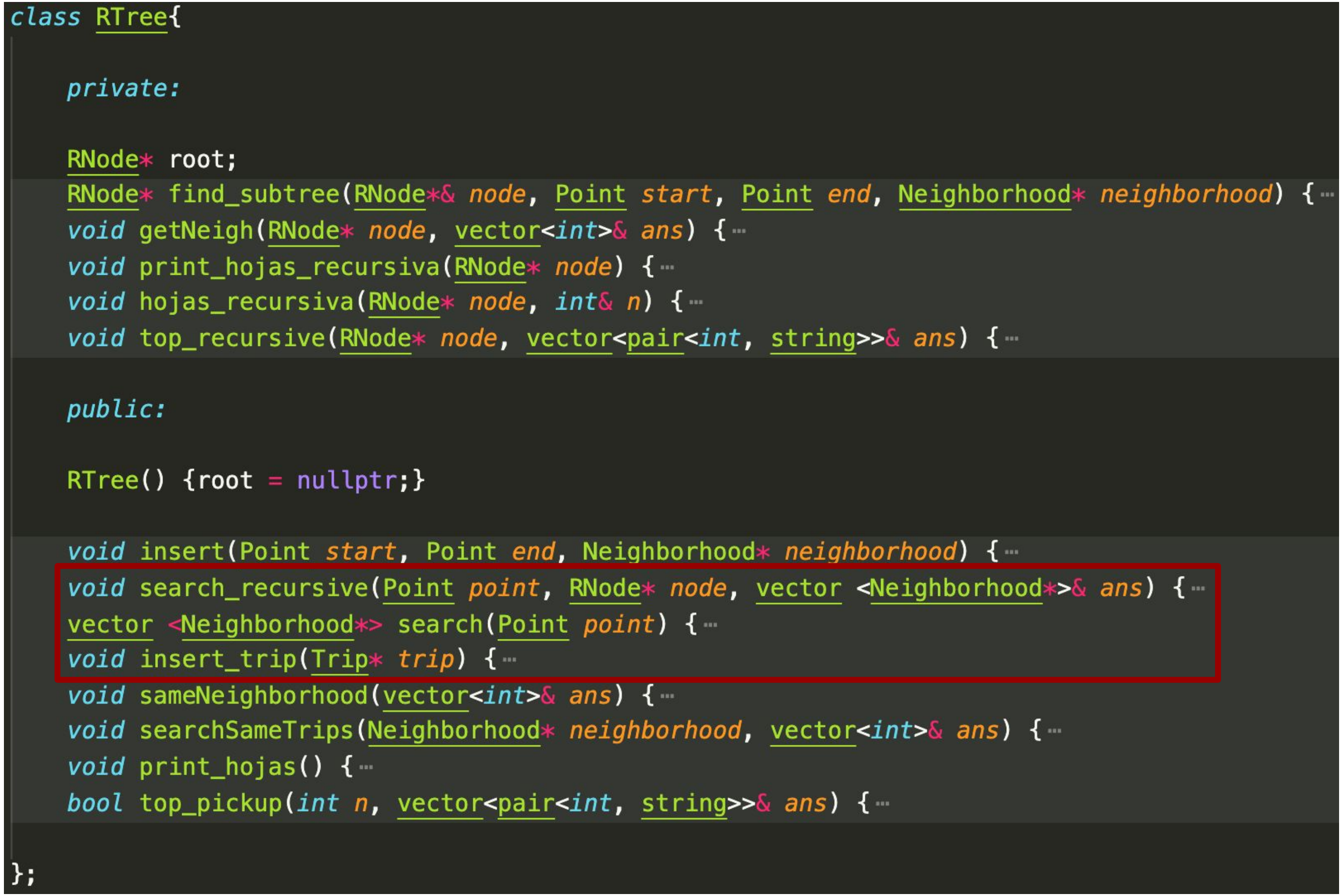
Flushing Meadows
Inwood



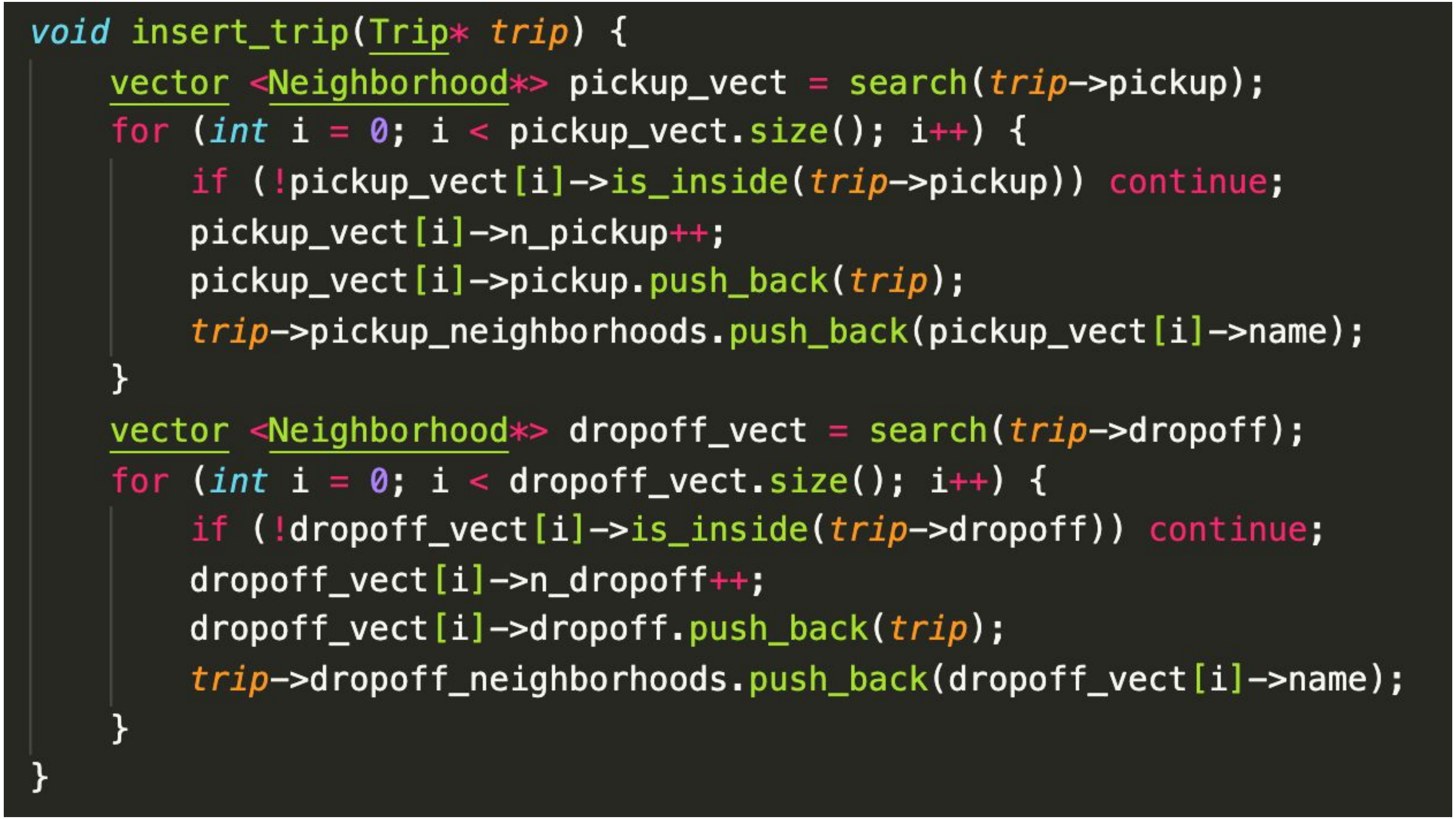
Check If Point Exist in a Polygon

This location is (28.984492718967903, 77.7147102355957) inside the polygon.

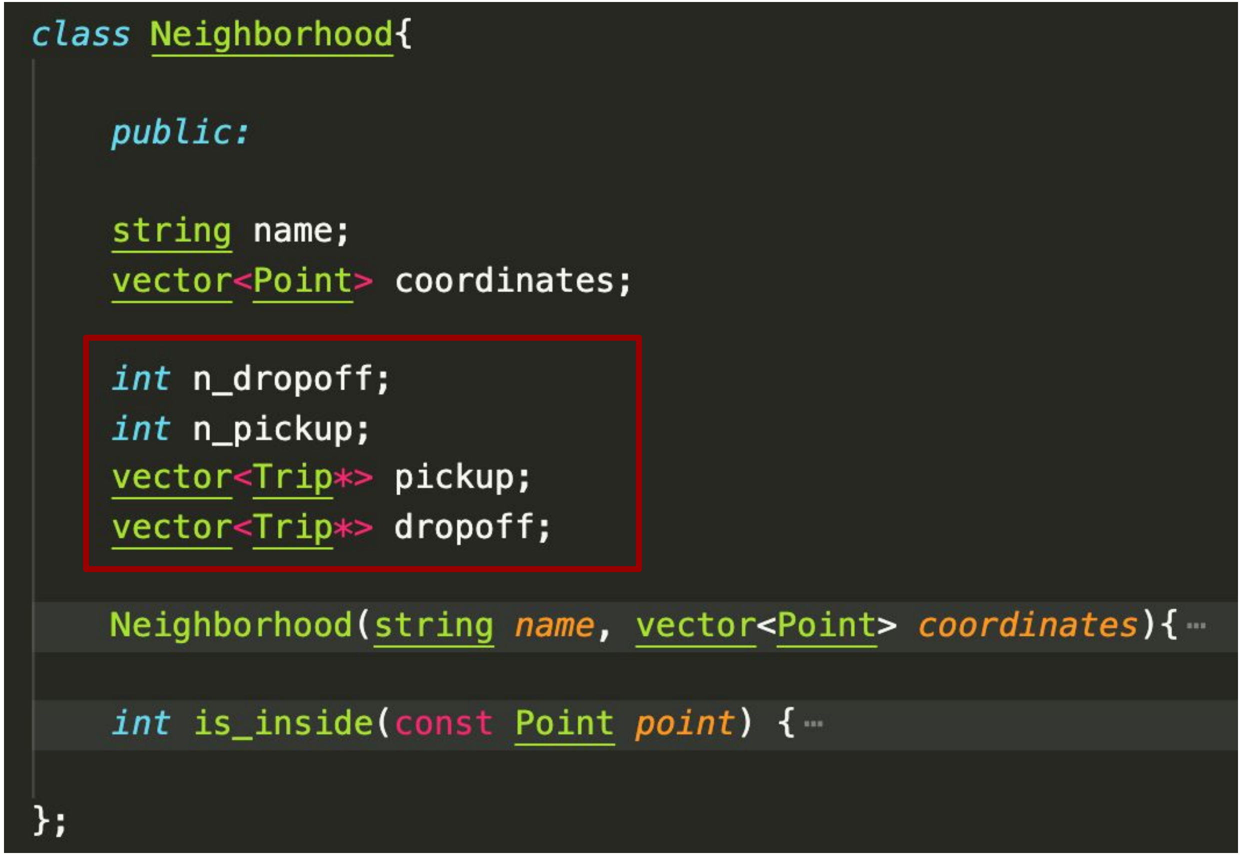
Problema



```
class RTree{

    private:

    RNode* root;
    RNode* find_subtree(RNode*& node, Point start, Point end, Neighborhood* neighborhood) {⋯
    void getNeigh(RNode* node, vector<int>& ans) {⋯
    void print_hojas_recursiva(RNode* node) {⋯
    void hojas_recursiva(RNode* node, int& n) {⋯
    void top_recursive(RNode* node, vector<pair<int, string>>& ans) {⋯

    public:

    RTree() {root = nullptr;}

    void insert(Point start, Point end, Neighborhood* neighborhood) {⋯
    void search_recursive(Point point, RNode* node, vector <Neighborhood*>& ans) {⋯
    vector <Neighborhood*> search(Point point) {⋯
    void insert_trip(Trip* trip) {⋯
    void sameNeighborhood(vector<int>& ans) {⋯
    void searchSameTrips(Neighborhood* neighborhood, vector<int>& ans) {⋯
    void print_hojas() {⋯
    bool top_pickup(int n, vector<pair<int, string>>& ans) {⋯

};
```



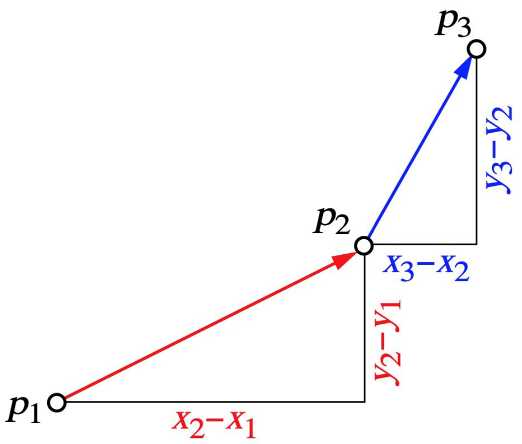
```
void insert_trip(Trip* trip) {
    vector <Neighborhood*> pickup_vect = search(trip->pickup);
    for (int i = 0; i < pickup_vect.size(); i++) {
        if (!pickup_vect[i]->is_inside(trip->pickup)) continue;
        pickup_vect[i]->n_pickup++;
        pickup_vect[i]->pickup.push_back(trip);
        trip->pickup_neighborhoods.push_back(pickup_vect[i]->name);
    }
    vector <Neighborhood*> dropoff_vect = search(trip->dropoff);
    for (int i = 0; i < dropoff_vect.size(); i++) {
        if (!dropoff_vect[i]->is_inside(trip->dropoff)) continue;
        dropoff_vect[i]->n_dropoff++;
        dropoff_vect[i]->dropoff.push_back(trip);
        trip->dropoff_neighborhoods.push_back(dropoff_vect[i]->name);
    }
}
```



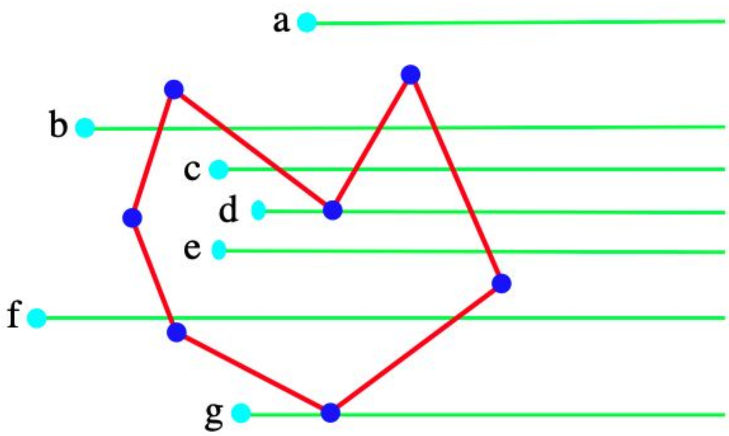
```
class Neighborhood{

    public:

    string name;
    vector<Point> coordinates;

    int n_dropoff;
    int n_pickup;
    vector<Trip*> pickup;
    vector<Trip*> dropoff;

    Neighborhood(string name, vector<Point> coordinates){⋯

    int is_inside(const Point point) {⋯

};
```

Point in Polygon

Interseción y Orientación



Inclusión del punto



Recuperada de:
http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf

Point in Polygon

```
is_inside(polygon, point):
     intersections = 0
     dy = 0
     dy2 = point.y - polygon[0].y
     for j in range(1, len(polygon)):
          dy = dy2
          dy2 = point.y - polygon[j]
          if (not line_is_above_bellow_right(dy, dy2)):
               if (not horizontal_line(dy, dy2)):
                    if (line_is_left_from_point(point.x, dy, dy2)):
                         intersections += 1
                    if (line_equal_point(point.x, dy, dy2))
                         return True
               else if(line_is_upper_peak_or_horizontal_line(point, dy, dy2)):
                    return True
return intersections%2
```

*considera el punto como parte del polígono incluso cuando está sobre el perímetro.

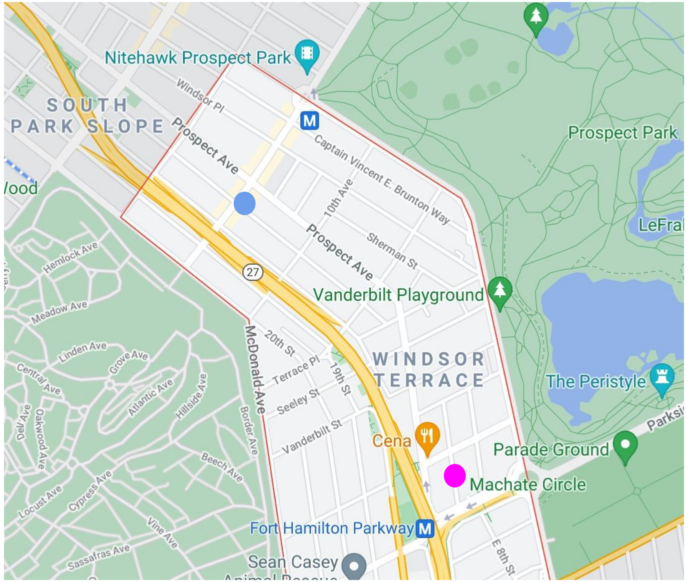
Recuperado de: https://github.com/sasamil/PointInPolygon/blob/master/point_inside.cpp

Consulta 1

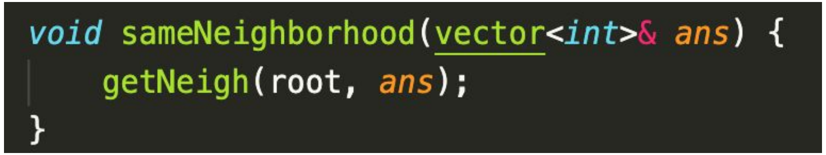
Recorrer todos los barrios (iterar hasta llegar a las hojas).

Dentro del neighborhood **buscar en los Trips de pickup.**

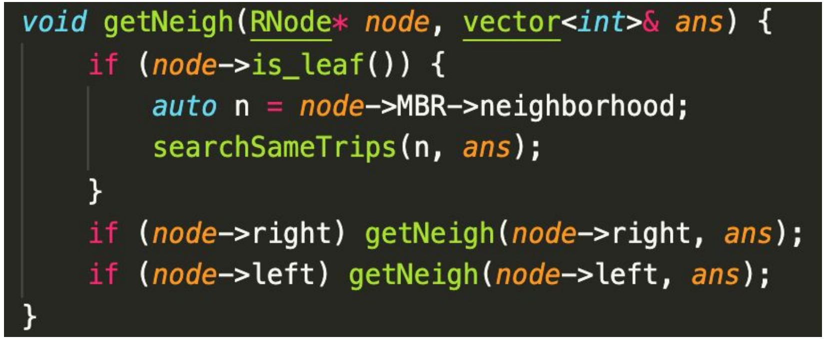
Con la función auxiliar same, se **compara el vector de string pickup contra los dropoff.**



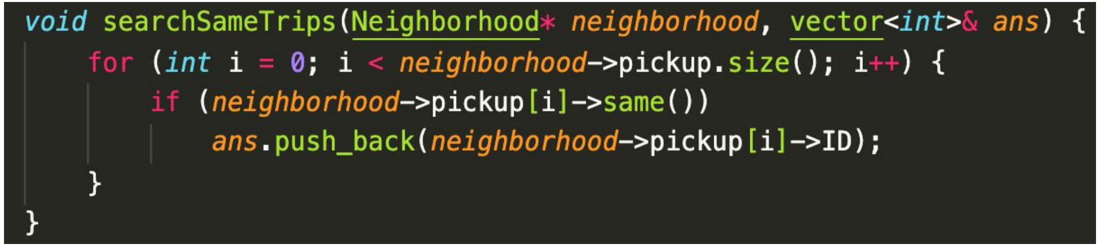
Consulta 1



```
void sameNeighborhood(vector<int>& ans) {
    getNeigh(root, ans);
}
```



```
void getNeigh(RNode* node, vector<int>& ans) {
    if (node->is_leaf()) {
        auto n = node->MBR->neighborhood;
        searchSameTrips(n, ans);
    }
    if (node->right) getNeigh(node->right, ans);
    if (node->left) getNeigh(node->left, ans);
}
```



```
void searchSameTrips(Neighborhood* neighborhood, vector<int>& ans) {
    for (int i = 0; i < neighborhood->pickup.size(); i++) {
        if (neighborhood->pickup[i]->same())
            ans.push_back(neighborhood->pickup[i]->ID);
    }
}
```

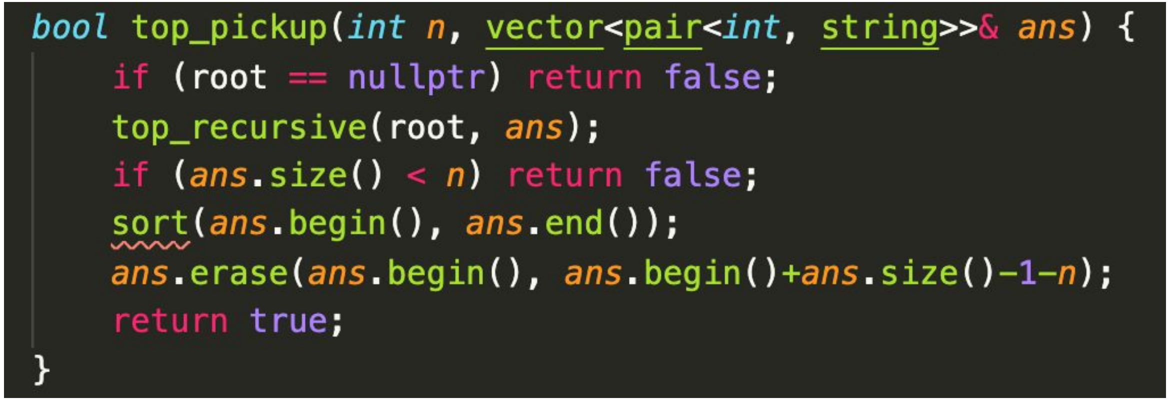
Consulta 2

En el insert se cuentan cuantos viajen están siendo insertados en un barrio en específico para luego, insertar en un vector un pair de cantidad de viaje y nombre del barrio

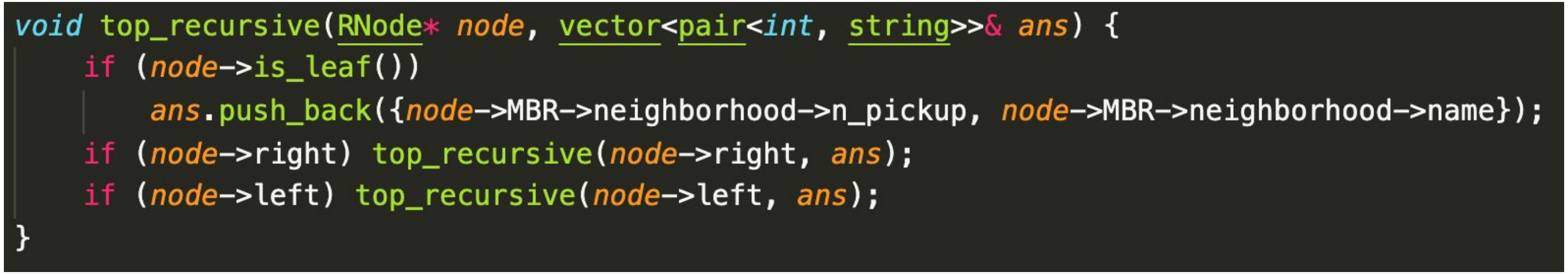
Se ordenan de menor a mayor el vector por criterio de viajes y **se obtienen los últimos 5** resultados que vendrían a ser los de más cantidad de viajes.

Línea Amarilla
Colegio Dos de Mayo
Alberto Gallardo
Vía Expresa
Enrique Meiggs
Av. Argentina
Parque Alameda Las Malvinas
UNFV
Plaza Dos de Mayo
Av. Oscar R. Benavides
Estación 2 de Mayo
Estadio Lolo Fernandez
Hospital Nacional Arzobispo Loayza
Instituto Nacional de Oftalmología
Jirón Gaspar Hernandez

Consulta 2

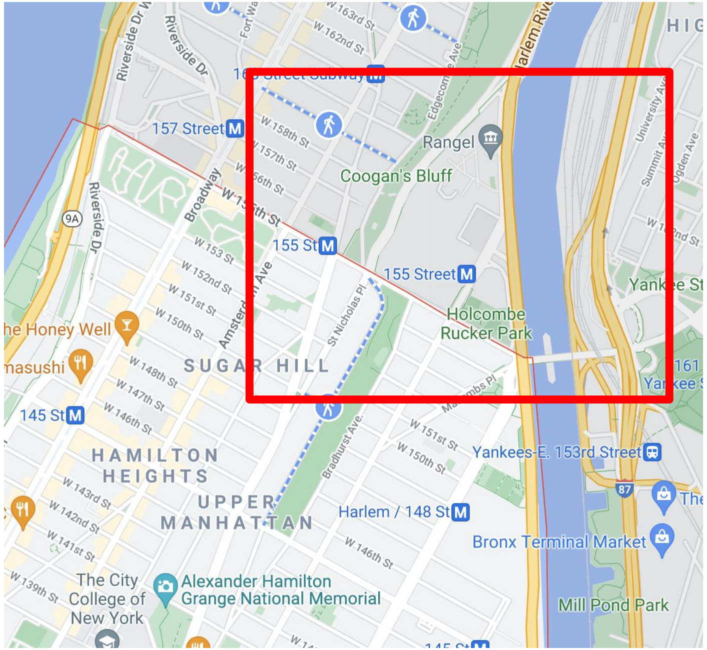


```cpp
bool top_pickup(int n, vector<pair<int, string>>& ans) {
    if (root == nullptr) return false;
    top_recursive(root, ans);
    if (ans.size() < n) return false;
    sort(ans.begin(), ans.end());
    ans.erase(ans.begin(), ans.begin()+ans.size()-1-n);
    return true;
}
```



```cpp
void top_recursive(RNode* node, vector<pair<int, string>>& ans) {
    if (node->is_leaf())
        ans.push_back({node->MBR->neighborhood->n_pickup, node->MBR->neighborhood->name});
    if (node->right) top_recursive(node->right, ans);
    if (node->left) top_recursive(node->left, ans);
}
```

Consulta 3

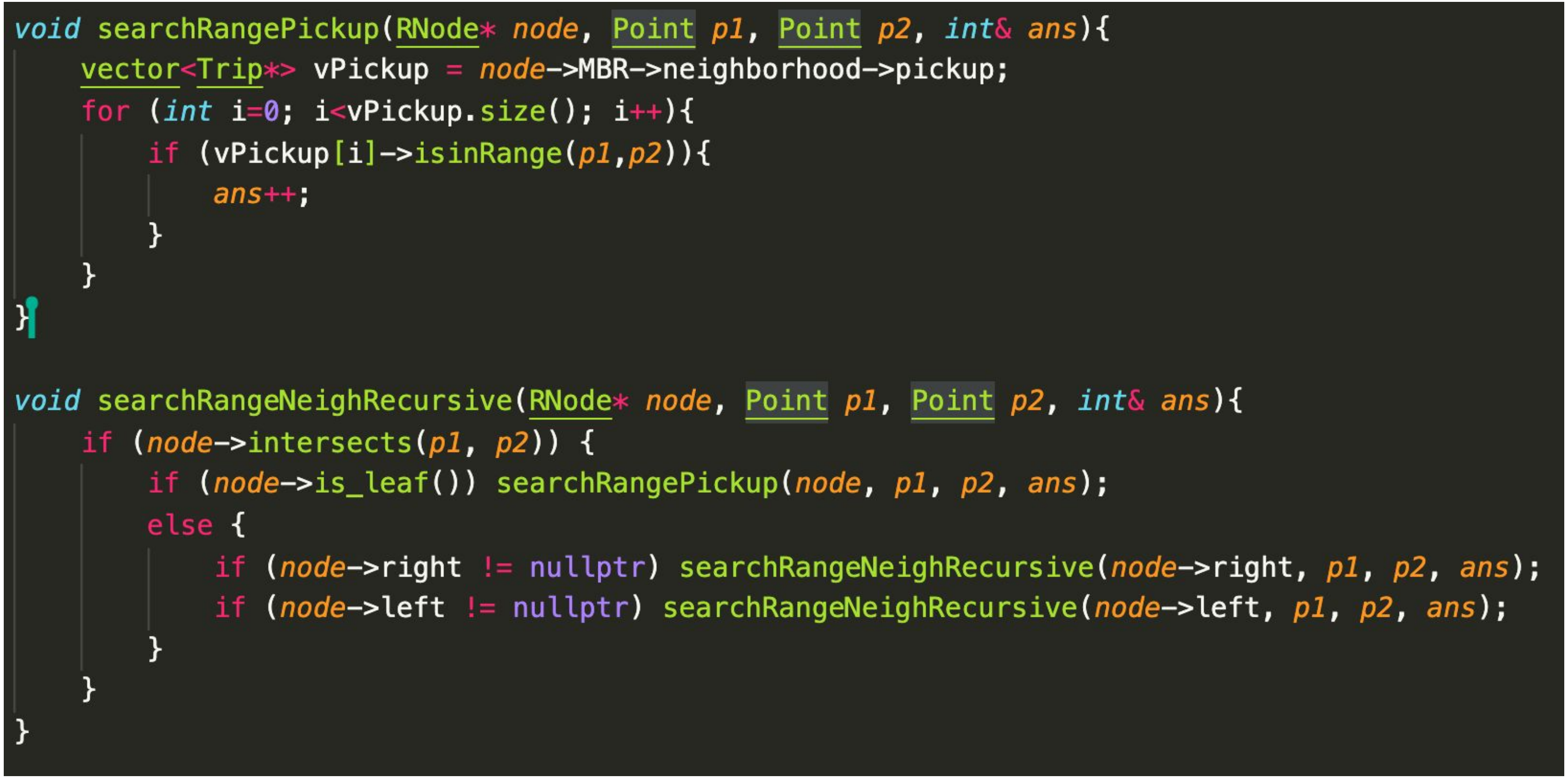


Barrio Harlem

Se busca por los MBRs si la región de consulta se **intersecta** con este (podando partes del árbol).

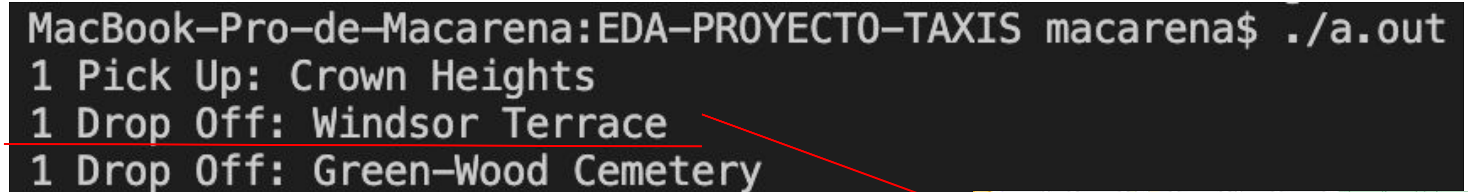
Si es así, se procede a **buscar sobre los MBRs->neighborhood->pickup** que han dado positivo con el mismo criterio de la búsqueda de MBRs (con la función auxiliar isinRange).

Cuando se encuentra una coincidencia de suma al contador.

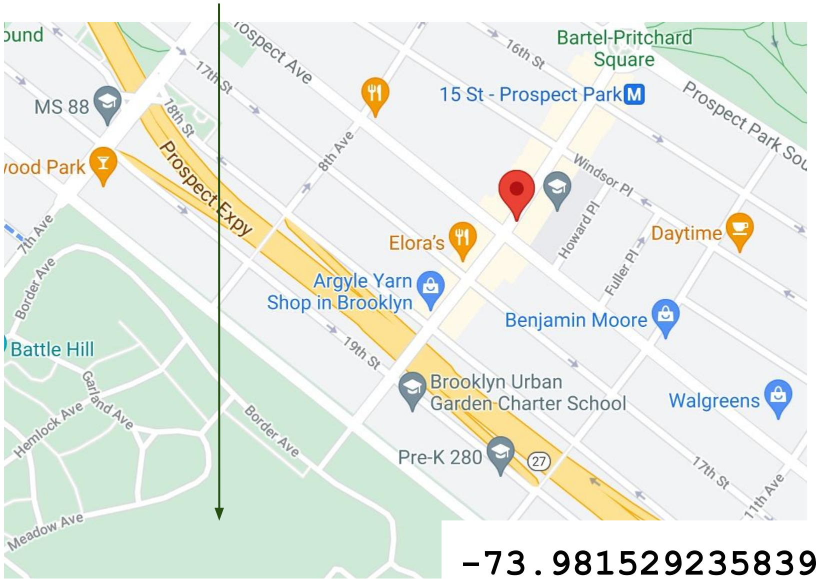


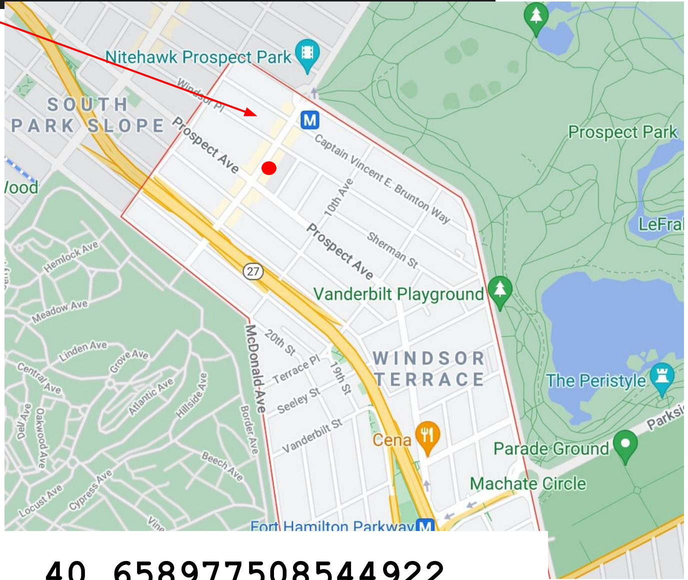
```
void searchRangePickup(RNode* node, Point p1, Point p2, int& ans){
    vector<Trip*> vPickup = node->MBR->neighborhood->pickup;
    for (int i=0; i<vPickup.size(); i++){
        if (vPickup[i]->isinRange(p1,p2)){
            ans++;
        }
    }
}

void searchRangeNeighRecursive(RNode* node, Point p1, Point p2, int& ans){
    if (node->intersects(p1, p2)) {
        if (node->is_leaf()) searchRangePickup(node, p1, p2, ans);
        else {
            if (node->right != nullptr) searchRangeNeighRecursive(node->right, p1, p2, ans);
            if (node->left != nullptr) searchRangeNeighRecursive(node->left, p1, p2, ans);
        }
    }
}
```

Identificando el problema



```
MacBook-Pro-de-Macarena:EDA-PROYECTO-TAXIS macarena$ ./a.out
1 Pick Up: Crown Heights
1 Drop Off: Windsor Terrace
1 Drop Off: Green-Wood Cemetery
```

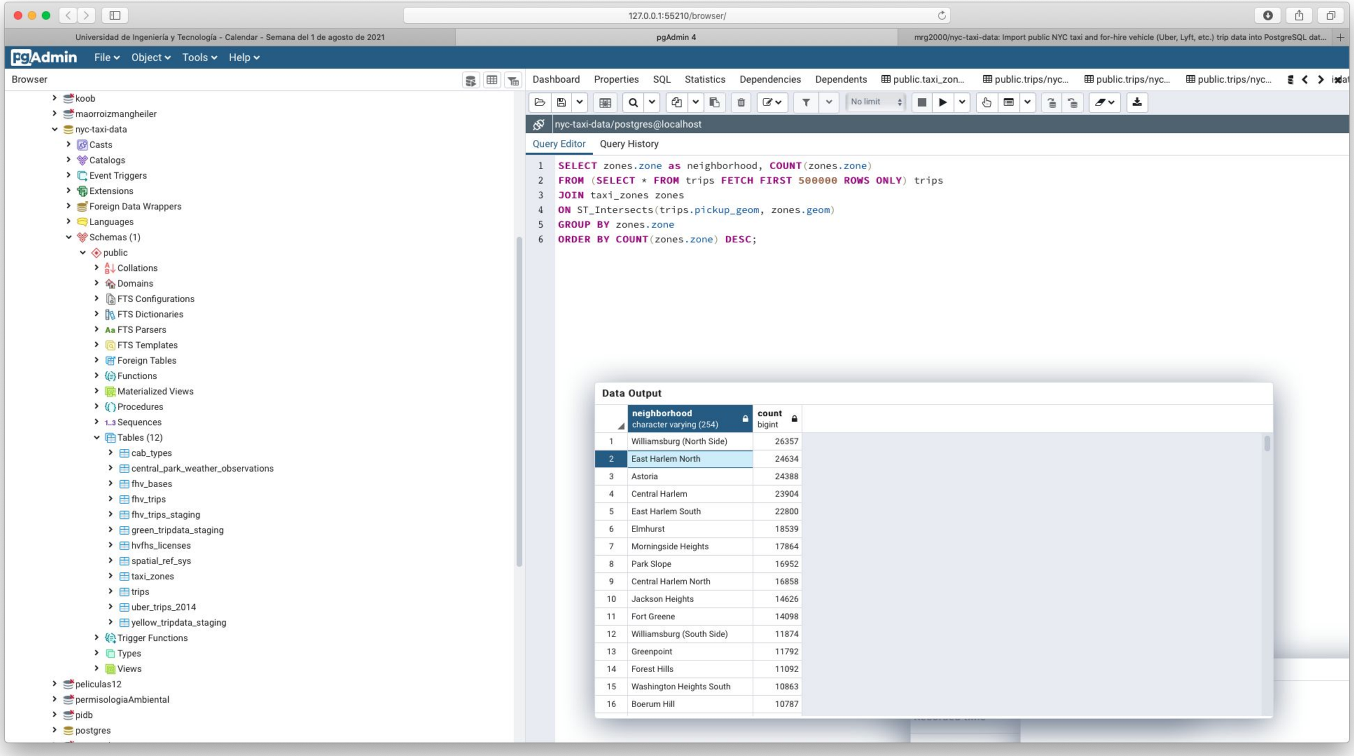




-73.981529235839844, 40.658977508544922

Testing y validación de datos

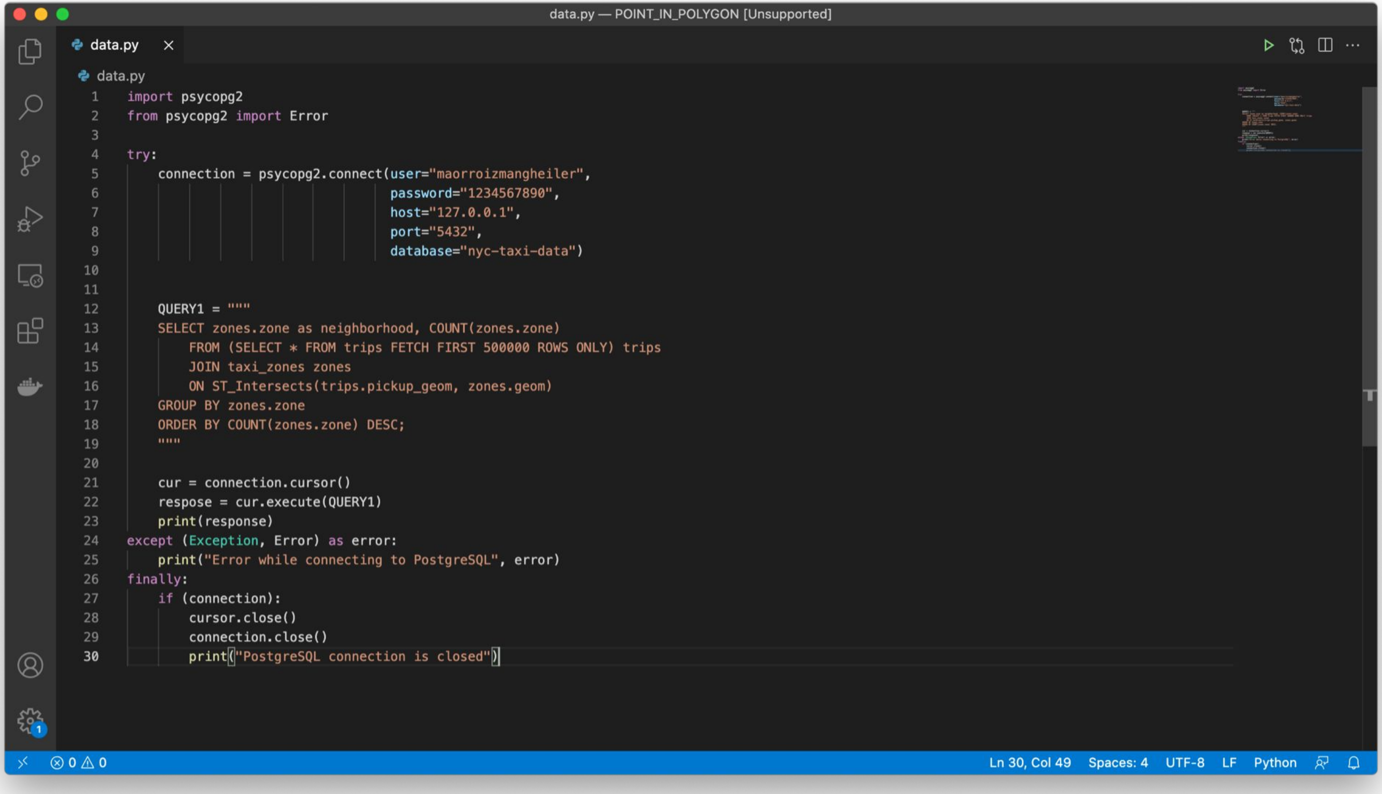
Repositorio: https://github.com/mrg2000/nyc-taxi-data



```
SELECT zones.zone as neighborhood, COUNT(zones.zone)
FROM (SELECT * FROM trips FETCH FIRST 500000 ROWS ONLY) trips
JOIN taxi_zones zones
ON ST_Intersects(trips.pickup_geom, zones.geom)
GROUP BY zones.zone
ORDER BY COUNT(zones.zone) DESC;
```

	neighborhood character varying (254)	count bigint
1	Williamsburg (North Side)	26357
2	East Harlem North	24634
3	Astoria	24388
4	Central Harlem	23904
5	East Harlem South	22800
6	Elmhurst	18539
7	Morningside Heights	17864
8	Park Slope	16952
9	Central Harlem North	16858
10	Jackson Heights	14626
11	Fort Greene	14098
12	Williamsburg (South Side)	11874
13	Greenpoint	11792
14	Forest Hills	11092
15	Washington Heights South	10863
16	Boerum Hill	10787

Consulta espacial para determinar el número de viajes por barrio. *solo tomando los primeros 500 mil registros de la tabla de trips.



```python
import psycopg2
from psycopg2 import Error

try:
    connection = psycopg2.connect(user="maorroizmangheiler",
                                  password="1234567890",
                                  host="127.0.0.1",
                                  port="5432",
                                  database="nyc-taxi-data")


    QUERY1 = """
    SELECT zones.zone as neighborhood, COUNT(zones.zone)
        FROM (SELECT * FROM trips FETCH FIRST 500000 ROWS ONLY) trips
        JOIN taxi_zones zones
        ON ST_Intersects(trips.pickup_geom, zones.geom)
    GROUP BY zones.zone
    ORDER BY COUNT(zones.zone) DESC;
    """

    cur = connection.cursor()
    respose = cur.execute(QUERY1)
    print(response)
except (Exception, Error) as error:
    print("Error while connecting to PostgreSQL", error)
finally:
    if (connection):
        cursor.close()
        connection.close()
        print("PostgreSQL connection is closed")
```

Referencias

- https://www.scitepress.org/Papers/2017/60408/60408.pdf
- http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf