Proyecto Semestral de Fundamentos de Estructuras de Datos y Algoritmos

Danilo Bonometti
Dept. Ingeniería Informática y Ciencias de la Computación
Universidad de Concepción
Concepción, Chile
danilobonometti@udec.cl

Diego Carrasco
Dept. Ingeniería Informática y Ciencias de la Computación
Universidad de Concepción
Concepción, Chile
dicarrascov@udec.cl

I. Introducción

En este trabajo se estudió e implementó la estructura de datos *Point Region Quad Tree* o *PR-QuadTree*. Está es una estructura diseñada para el trabajo y manejo de datos espaciales. Permite el uso eficiente en tiempo para almacenar y representar información de puntos en un plano en una jerarquía descrita mediante subdivisiones sucesivas de sus cuadrantes. Esta estructura pertenece a la familia de los *QuadTree* los cuales tienen una gran cantidad de aplicaciones [1]. Se han utilizado en descomposición de matrices dispersas como medio para disminuir el tamaño utilizado por estas. Se utilizan en computación grafica para la eliminación de líneas y superficies ocultas mediante la subdivisión sucesiva de la vista. También se utilizan en el desarrollo de codificadores de video comprimiendo la información subdividiendo áreas donde los pixeles alcanzan valores similares. En el caso del PR-QuadTree, es utilizado para la parcelación de información espacial de puntos en un plano de manera que estos solo pertenezcan a hojas del árbol. Como se verá en este trabajo, esto permite que el ingreso, acceso y eliminación de los puntos sea extremadamente rápido. En general los QuadTree deben cumplir con dos propiedades fundamentales: que la descomposición en particiones del plano sea infinitamente repetitiva y que las particiones puedan ser infinitamente descompuestas en particiones más pequeñas. El diseño de estas estructuras basado en estas propiedades, permiten que la complejidad temporal disminuya de gran manera para las operaciones de nodos, sin embargo, tienen la desventaja de que la ocupación de memoria se ve impactada fuertemente por los nodos hoja que no contienen información. Esta complejidad puede reducirse en términos de constantes usando representaciones sin referencia para estos nodos. En este trabajo se verá como la implementación cumple con estas condiciones de complejidad y se podrá observar que las propiedades del PR-QuadTree permiten obtener información que mediante otros métodos sería muy difícil de recopilar.

II. Estructuras de Datos

A. *Quad Tree*

El QuadTree corresponde a una estructura de tipo árbol en la que cada nodo posee 4 nodos hijo. Es el análogo bidimensional del Octree y es utilizado para particionar un espacio bidimensional de forma recursiva subdividiéndolo en 4 cuadrantes. Los datos pueden estar almacenados como estructuras los que se asocian a las hojas del árbol. Existen distintas clases de QuadTree, las que pueden poseer subdivisiones cuadradas o rectangulares dependiendo del tipo. En general las estructuras de tipo QuadTree cumplen con las siguientes propiedades:

- Subdividen el espacio en regiones referenciadas por nodos.
- Cada región o nodo posee una capacidad máxima de almacenamiento de puntos, el cual al ser sobrepasado fuerza a una subdivisión del espacio y a la creación de nuevos nodos.
- Existe una relación directa entre la topología del árbol creado y la relación espacial que existe entre los datos.

Los QuadTree se pueden clasificar en función del tipo de datos que representan, tales como puntos, líneas, curvas y áreas. Algunos tipos son el Region QuadTree, el Point QuadTree, Point Region QuadTree y Edge QuadTree.

B. *Point Region Quad Tree*

Esta estructura representa una colección de puntos en un espacio bidimensional los cuales se encuentran relacionados a datos. De esta manera, si una región contiene un solo punto o no contiene ningún punto, este espacio se puede representar en el árbol mediante una hoja. Cuando una región contiene más de un punto, es necesario subdividir el espacio en cuatro partes de forma recursiva hasta que estos puntos se encuentren en distintos nodos hoja. De esta forma, se tiene que el PR-QuadTree es un árbol cuyos nodos representan una región del espacio y también pueden contener subárboles de subregiones del espacio. A estos cuadrantes o subregiones se les denomina comúnmente como NW, NE, SW, SE.

A diferencia de estructuras como los árboles binarios, donde la topología del árbol no solo depende de los datos insertados, sino que también del orden en el que se insertan, la estructura que toma un PR-QuadTree depende solamente de los datos que contiene, sin importar el orden ya que esta estructura reorganiza las regiones donde se insertan o remueven nuevos puntos en función de la localización de estos.

Una de las principales razones para usar una estructura PR-QuadTree se debe a los tiempos de acceso a los datos. Datos que se encuentren bien distribuidos en el espacio resultarán en árboles de menor profundidad. Los datos se encuentran almacenados en las hojas del árbol por lo que resulta beneficioso tener un árbol de la menor profundidad posible. El QuadTree no es un árbol simétrico, ya que su topología depende directamente de la posición espacial de los datos, debido a esto es posible tener casos donde algunos puntos se encuentren muy cercanos en el espacio, lo que se vería reflejado en ramas muy profundas en el QuadTree, lo que resulta en tiempos de acceso mayores. En cierta forma la estructura PR-QuadTree se puede utilizar como una estructura de datos asociativos, donde se puede ocupar una coordenada bidimensional como llave al dato.

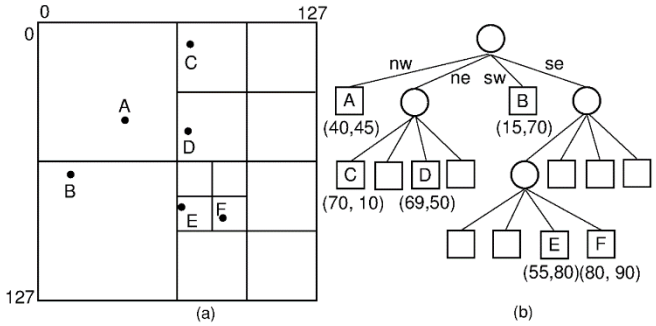

Fig. 1. Ejemplo de un PR-QuadTree.

Para implementar una estructura PR-QuadTree es necesario primero representar sus partes. Una forma simple de hacer esto es considerar distintos tipos de nodo, que se diferencien en la cantidad de puntos que contienen en el cuadrante que representan. De esta forma se tienen 3 tipos: los blancos, negros y grises. Los nodos blancos corresponden a aquellos nodos hoja que no contienen ningún punto en su interior, los nodos negros son aquellos nodos hoja que contienen un solo punto y los nodos grises actúan como ramas del árbol y por lo tanto son nodos intermedios (no hoja) que apuntan a 4 nodos hijo, los cuales a su vez pueden también ser blancos, negros o grises. La estructura PR-QuadTree al iniciar vacía consiste solo de un nodo blanco. Así, la estructura básica correspondería a:

```
PR-Quadtree:          Nodo:              Data:
    Nodo* root            Data* dato         Dato
                          Nodo* NE           pos x
                          Nodo* NW           pos y
                          Nodo* SE
                          Nodo* SW
```

El algoritmo de búsqueda en estructuras PR-QuadTree consiste en realizar un descenso recursivo por los nodos del árbol hasta llegar al nodo color negro que contiene los datos deseados. Se inicializa la búsqueda desde el nodo raíz de la estructura, si el nodo es gris, por definición de los nodos grises, este nodo no puede contener el dato, por lo tanto se debe bajar un nivel a los nodos hijos. Ahora, la eficiencia de la búsqueda radica en que no es necesario buscar en los 4 nodos hijos, ya que el punto solo puede estar en un cuadrante a la vez, de esta forma solo es necesario comprobar en qué cuadrante debiese estar contenido el punto en términos de su localización. Así, se comprueban las regiones que representan cada nodo hijo y se elige uno, luego la búsqueda continúa en este nodo repitiendo el proceso anterior. De esta forma, el proceso de búsqueda siempre avanza por la rama correcta, lo que lo hace un algoritmo rápido. Por definición el árbol no puede terminar en nodos grises, por lo que el proceso de búsqueda termina cuando se llega a un nodo blanco o un nodo negro. En el caso de un nodo blanco, no existe el dato en la estructura por lo que no se retorna nada. En el caso que el nodo sea negro se debe comparar el dato en el nodo con el dato buscado, ya que podría ser un dato distinto pero que ocupa un mismo cuadrante, de esta forma si el dato en el nodo negro no es igual al dato buscado se puede concluir que el dato no existe en la estructura y no se retorna nada. Si el punto del nodo negro es igual al punto del dato buscado se retorna el dato. El costo en tiempo es O(h) donde h corresponde a la profundidad de la rama que contiene el nodo buscado.

Algoritmo 1. BÚSQUEDA

```
Search (pos):
    node = root
    while node is grey:
        if: pos in node->NE->quad:
            node = node->NE
        if: pos in node->NW->quad:
            node = node->NW
        if: pos in node->SE->quad:
            node = node->SE
        if: pos in node->SW->quad:
            node = node->SW
    return node
```

La inserción de puntos en la estructura PR-QuadTree sigue un algoritmo simple, el primer dato en ser insertado se guarda en el nodo raíz de la estructura, el que corresponde a un nodo blanco que representa un cuadrante inicial que abarca la región máxima donde pueden estar los puntos. A partir de aquí, cada vez que se inserte un nuevo punto se deberá comprobar en qué cuadrante debiese ir. Si el nodo que contiene a esta subregión del espacio está vacío, se puede guardar el dato en este nodo y se termina el proceso. Si el nodo es gris eso quiere decir que no podemos guardar el dato en este nodo ya que habría más de un punto en un mismo cuadrante, debido a esto es necesario descender en el árbol a un sub cuadrante mediante los nodos hijo. Este proceso de descender se repite cada vez que se encuentre un nodo gris hasta llegar a un nodo blanco o negro. En el caso que el nodo sea negro, se tiene lo que se llama una colisión, esto implica que el cuadrante ya contiene un solo punto, por lo que no se puede insertar el dato en este nodo, pero además los datos siempre tienen que estar en hojas diferentes del árbol, debido a esto el nodo actual tendrá que considerarse como un nodo gris de aquí en adelante y se tienen que mover ambos puntos a los nodos hijos. Este proceso de subdivisión se repite hasta que ambos puntos correspondan a cuadrantes diferentes.

Algoritmo 2. INSERCIÓN

```
Insert (pos, data):
    node = search(pos)
    if node exist:
        delete data
        end
    else:
        if node is white:
            node->data = data
            node->color = black
        else if node is black:
            node->color = grey
            old_data = node->data
            old_pos = node->pos
            while pos in same position with old_pos:
                subdivide(quadrants)
            set pos and data in correct quadrant
            set old_pos and old_data in correct
quadrant
```

En la eliminación de datos del PR-QuadTree se tienen distintos casos. Considerando el algoritmo de inserción de datos, se puede observar que una invariante de las estructuras PR-QuadTree, es de que no pueden existir nodos negros que no tengan al menos un vecino negro o gris. Lo primero es encontrar el nodo que contiene el dato a eliminar. Para esto se realiza un descenso por los nodos grises hasta este nodo mediante una búsqueda del dato de la misma manera que para la inserción. Una vez que se encuentra el nodo con el dato a eliminar se elimina el dato y el nodo se considera como un nodo blanco. Como los datos siempre se encuentran en los

nodos negros que son nodos hoja, no es necesario preocuparse de eliminar nodos hijo de este. En el momento en que el nodo se transforma a nodo blanco, se pueden presentar distintos casos. Para el caso en que este nodo y sus 3 nodos hermanos sean todos blancos, se tiene que no existe ningún punto en ninguno de estos 4 nodos, por lo tanto el nodo superior o padre representa una región que no posee ningún punto, de esta forma se pueden eliminar los 4 nodos y cambiar al nodo padre de gris a blanco. Otro caso posible es que al eliminar el nodo, aún exista un nodo hermano negro y los otros dos nodos sean blancos, en este caso no se cumpliría la invariante de que no puede existir un solo nodo negro con 3 nodos hermanos blancos, por lo que se deben mover los datos del nodo negro sobrante al nodo padre y eliminar sus 4 nodos hijos. Cualquier otro caso consideraría que existe por lo menos 1 nodo gris o al menos 2 negros al momento de eliminar el nodo y estos casos sí cumplen con la invariante. Los casos anteriores se ejecutan de forma repetida hasta cumplir con la invariante. A este proceso se le llama colapso.

Algoritmo 3. ELIMINACIÓN

```
Remove(pos):
    node = search(pos)
    father = null
    if node does not exist:
       return
    node->color = white
    if node == root:
       node->data = null
       return
    father = node->father
    while node != null && node->father != null
                 father = node->father
       g = father->count_colors(grey)
       w = father->count_colors(white)
       b = father->count_colors(black)
       if w == 4:
          node->data = null
                        delete father->children
          father->color = white
          node = father
       else if b == 1 && w == 3:
          father->color = black
          father->data = node->data
          delete father->children
          node = father
       else
          break
    return
```

Analizar los costos de búsqueda, inserción y eliminación de nodos en un QuadTree depende directamente de los nodos ya existentes en la estructura. Estos costos son directamente proporcionales a la máxima profundidad del árbol, la que está también relacionada a la mínima distancia existente entre todos los puntos de la región. Si se tiene una región cuadrada con lados de largo s, y la distancia mínima entre todos los puntos es d, entonces se puede obtener que la profundidad máxima h de un PR-QuadTree está dada por $h = log_2\left(\frac{s}{d}\right)\sqrt{2}$. Con este valor es posible deducir los costos de buscar, insertar y remover elementos de la estructura.

Para el caso de la búsqueda de un nodo se tiene que en el peor de los casos habrá que descender por una rama hasta un nodo que se encuentre en la máxima profundidad h. Es decir, el costo de búsqueda para el QuadTree se encuentra acotado por la profundidad h de la estructura, así la complejidad temporal es $O(h)$.

Para el caso de la inserción, la distancia mínima entre todos los puntos de un conjunto de datos puede conocerse desde antes de insertarlos a la estructura, por lo que se tiene algo similar a la búsqueda, ya que el peor de los casos de inserción se dará cuando se inserten los dos puntos que tienen distancia mínima entre ellos, y para este caso se tendrá que recorrer hasta la profundidad h para resolver la colisión de puntos. Es decir, el costo para inserción está dado por $O(h)$.

En el mejor caso la inserción llena un nodo blanco cada vez, por lo que debe llenar 4^n nodos en cada nivel. Esto da una complejidad temporal de $O(\log_4 n)$ o $\frac{1}{2}O(\log_2 n)$.

Para el caso de eliminación de un nodo, se tiene que considerar el descenso desde la raíz hasta el nodo a eliminar, esto es equivalente a realizar una búsqueda del nodo, por lo que se comienza con un costo mínimo de $O(h)$, pero además es posible que eliminar el nodo concluya con un colapso, que en el caso más lento puede resultar en colapsar nodos hasta la raíz del árbol, por lo que se suma el costo de ascender que también es $O(h)$. De esta forma el costo de eliminar un nodo en el peor caso posible es de $O(h)$.

Para búsquedas por región el costo del peor caso se da al buscar en todos los puntos de una región, que es equivalente a $O(F + 2^h)$, donde F es el número de puntos encontrados y h es la profundidad máxima de los nodos. Es decir, depende de la densidad de puntos en el espacio.

III. IMPLEMENTACIÓN

La implementación final se basó en dos implementaciones paralelas que se utilizaron para comparar que métodos daban mejores resultados en términos de utilidad y costo. En general el trabajo tomó como referencia los algoritmos y diseño descritos en [1], con algunas diferencias que se explican en esta sección.

Se consideraron los nodos blancos como punteros a *NULL* para reducir de gran manera las constantes asociadas a la complejidad espacial que producen.

Los nodos consideran el dato que almacenan, un dato para su color, punteros a los nodos hijos y un puntero al nodo padre. Para el alcance del proyecto solo se considera la población de los datos de entrada.

Para el plano se consideró como inicio el punto *(0,0)* y las dimensiones en los ejes *x* e *y* varían desde -90 a 90 y -180 a 180 respectivamente para coincidir con las medidas de latitud y longitud. También es posible considerar medidas de coordenadas de forma arbitraria.

Los cuadrantes se definen en términos de un centro y radio para *x* e *y*. Esto facilita ciertas operaciones como revisar si un cuadrante interseca con otro o si está contenido. Estos cuadrantes se propagan como variables por el árbol con su nodo correspondiente al momento de hacer una consulta.

También se consideró que los puntos ya ingresados son descartados. Por lo que en solo entran en el árbol un porcentaje de los datos de entrada.

A. *Inserción*

Para la inserción se implementó el algoritmo de referencia. Se consulta por el nodo al que correspondería el dato a ingresar, se retorna a una función controladora que revisa si hay datos en el nodo y si es que corresponde al mismo dato

que se desea ingresar. Si el dato es ingresado exitosamente se retorna *verdadero* de lo contrario se retorna *falso*.

B. *Eliminación*

Para la eliminación de un nodo se utilizó una referencia a los nodos padre. De esta manera se puede encontrar el siguiente nivel a colapsar mediante un recorrido hacia arriba en el árbol a través de los nodos padre. Esto aumenta ligeramente las constantes asociadas al consumo de memoria, pero reduce de gran manera las constantes asociadas al colapso de los nodos que no cumplen la invariante, con respecto al diseño de referencia.

C. *Búsqueda*

Las consultas de búsqueda fueron diseñadas de manera que puedan recibir funciones como argumentos. Esto permite una gran capacidad para realizar distintas consultas ya que se pueden definir de forma independiente del algoritmo de búsqueda, lo que permite implementar muchos tipos de consulta distintos de forma encapsulada y modular.

Para las consultas por punto se implementó una búsqueda voraz, donde se busca el cuadrante al que pertenece el punto en cada nivel hasta llegar al que le corresponde. Si el nodo encontrado es gris o negro, se le aplica una función. Si el nodo es negro se retorna verdadero. Si el nodo encontrado es blanco o gris, retorna falso. Esta información puede ser utilizada para controlar el comportamiento al momento de encontrar un nodo de cierto tipo.

Para las consultas por región se implementaron algoritmos de búsqueda por anchura y por profundidad que recorren solo los nodos grises. Esta decisión es controlada por una función que retorna verdadero para los nodos grises y falso para los nodos blancos y negros. La función a su vez recibe como argumento una función para aplicar a los nodos encontrados por la búsqueda. Esto permite implementar distintos tipos de consulta solo en términos de lo que se desea hacer con un cierto tipo de nodo. En general solo se desea trabajar con los nodos negros, pero se implementó la función de búsqueda de manera que puede aplicar funciones a cada tipo de nodo, lo que permite una gran versatilidad de consultas.

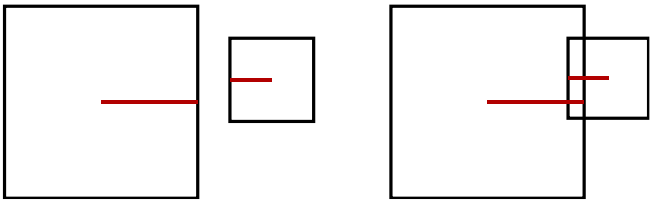

Fig. 2. Colisión de áreas.

La búsqueda por región requiere saber que cuadrantes colisionan con la región de interés, por lo que se necesita una función que maneje estos casos. Si se consideran *C1* y *r1* como centro y radio del cuadrante y *C2* con *r2* como el centro y radio de la región de búsqueda, es posible comprobar si las áreas de ambos cuadrados colisionan en algún lugar con la siguiente fórmula:

$$abs(C1_{x,y} - C2_{x,y}) \leq r1_{x,y} + r2_{x,y} \rightarrow colisión$$

Es decir, si la distancia entre los centros de ambos cuadrados es menor a la suma de sus radios, existe una colisión de áreas entre los dos cuadrados, como se observa en la figura Fig. 2.

Adicionalmente se implementó una búsqueda por barrido de regiones para obtener información muestreada del plano la que se utiliza para generar histogramas bidimensionales de la región como se ve en las figuras**Error! Reference source not found.**-Fig. 12. Esta búsqueda requiere de teselar el plano con sub cuadrantes de dimensiones arbitrarias. Estas subregiones pueden entenderse como se ve en la figura.

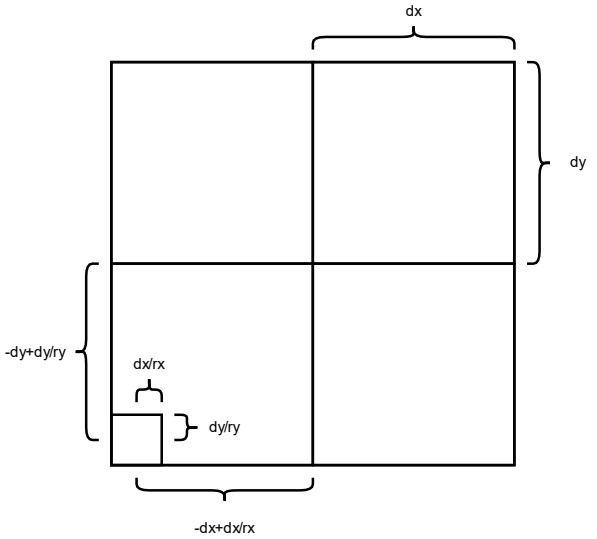

Fig. 3. Región de teselación para búqueda por barrido.

Donde r_x y r_y son las subdivisiones del plano para cada dimensión correspondientemente. De esto se obtiene que los centros de cada sub cuadrante están dados por:

$$C_r = \left(\frac{2d_x}{r_x}i - d_x + \frac{d_x}{r_x}, \frac{2d_y}{r_y}j - d_y + \frac{d_y}{r_y}\right)$$

Donde $i \in \{0,1,\dots,r_x\}, j \in \{0,1,\dots,r_y\}$, son todas las posiciones que pueden tomar los sub cuadrantes.

IV. Resultados

A. *Consultas por Punto*

Para las consultas por punto se consideró una ventana de 20000 elementos ponderados, ya que la velocidad de la consulta es demasiado rápida para medirla individualmente y entrega principalmente ruido. De esta forma se limpian un poco los datos.

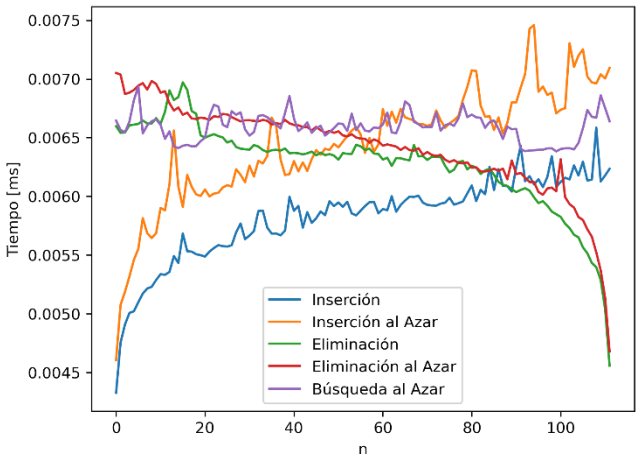

Fig. 4. Tiempo de consultas por punto.

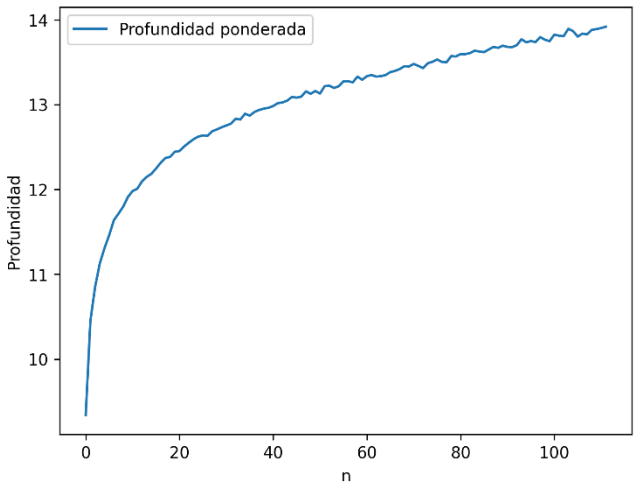

Fig. 5. Profundidad de los nodos al ser insertados.

Se puede ver de las figuras Fig. 4 y Fig. 5 claramente como el tiempo de inserción y profundidad promedio de los datos tiene un comportamiento asintótico logarítmico. Esto corresponde a que a medida que se llena el árbol, este aumenta de tamaño proporcionando mayor distribución de nodos blancos que pueden captar datos. Por lo tanto, mientras más grande sea el árbol menos necesita aumentar su profundidad, lo que se refleja claramente en las figuras.

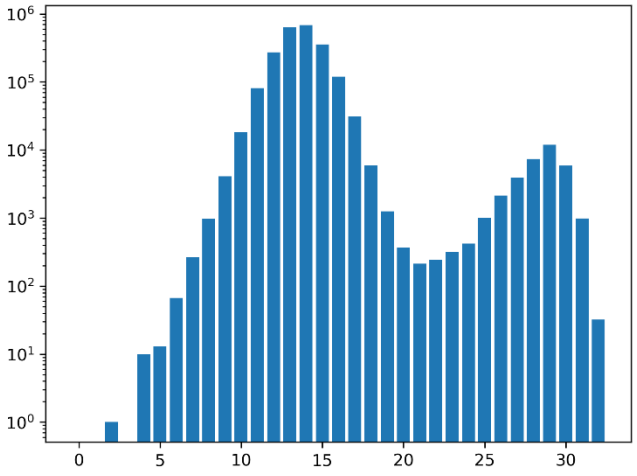

Fig. 6. Histograma de profundidad del árbol.

Del histograma en la figura Fig. 6 se puede observar como la mayor ocurrencia de profundidades se encuentra centrada en 14 y la profundidad máxima es 32. El valor promedio de las profundidades es de 13.8597 por lo que son estos datos los que definen principalmente la estructura del árbol.

B. Consultas por Región

Para las consultas por región se consideraron 1000 repeticiones para una región de radio r desde 1 a 25 y centro aleatorio entre los intervalos $[90 + r, 90 -]$ y $[-180 + r, 180 - r]$ para latitud y longitud correspondientemente.

En la figura Fig. 7 se puede apreciar claramente como la consulta tiene un comportamiento exponencial, lo que calza con el análisis asintótico, ya que a medida que desciende por el árbol tiene una cantidad exponencialmente mayor de nodos por revisar.

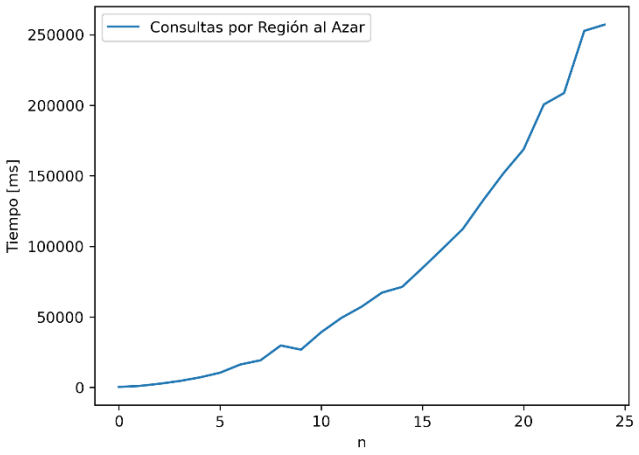

Fig. 7. Búsqueda por región.

Se midió la cantidad de nodos por tipo y totales para cada inserción de puntos que no estuviesen repetidos, de esta forma se asegura una reestructuración del árbol. El total de nodos que forman el PR-QuadTree es de 2,243,467 nodos negros, 1,834,068 nodos grises y 3,883,467 nodos blancos, dando un total de 7,961,002 nodos. En la figura Fig. 8 se puede apreciar que la mayor parte de la estructura está conformada por nodos blancos, por lo que considerarlos como punteros a *NULL* en la implementación significa una gran optimización de memoria. También es posible observar una leve tendencia exponencial en el total de nodos blancos, mientras que la curva de nodos negros siempre es lineal.

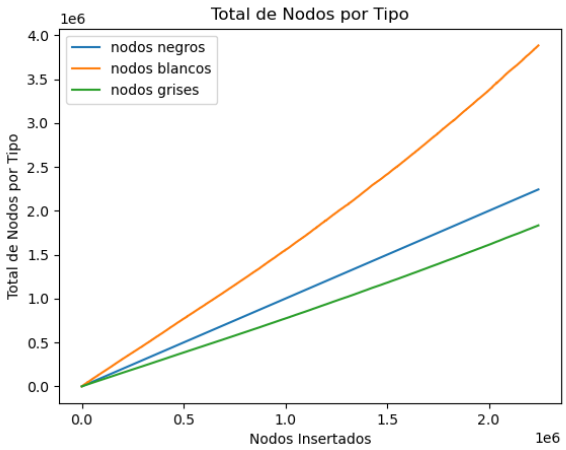

Fig. 8. Ocupancia del árbol.

Realizando un acercamiento a la región correspondiente a los primeros 100 nodos insertados, como se ve en la figura Fig. 9, no es difícil ver el comportamiento de la estructura. Se puede identificar en las primeras inserciones un salto brusco en el total de nodos grises y sobre todo en los blancos. Esto coincide con que los primeros puntos en ser insertados se encuentran muy cercanos, provocando una serie de colisiones, lo que genera un gran número de nodos grises y blancos, además de aumentar la profundidad. Además, se puede ver que mientras el total de nodos negros aumenta de forma lineal con cada inserción, el total de nodos grises solo aumenta o se mantiene constante entre inserciones, no puede disminuir,

mientras que para los nodos blancos se tiene que solo pueden aumentar o disminuir en cantidad, no pueden mantenerse constantes ya que para cada inserción se creará al menos un nodo gris o negro.

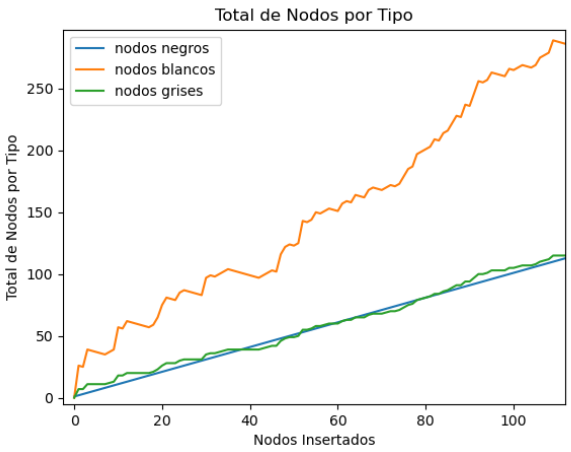

Fig. 9. Detalle de la ocupancia del árbol.

C. *Consultas por Barrido de Regiones*

Las consultas de barrido por regiones se hicieron con una precisión de 20 veces el tamaño del plano.

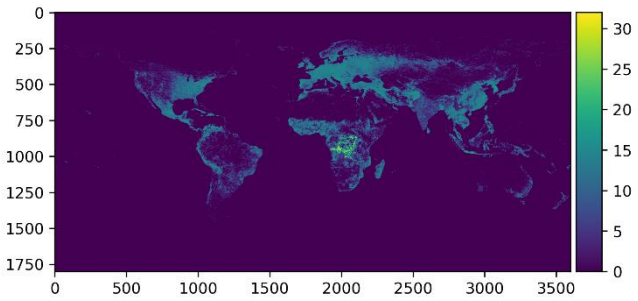

Fig. 10. Histograma 2D de profundidad de nodos negros.

En la figura Fig. 10 se observa el barrido por regiones de las profundidades de los datos de entrada. Este gráfico corresponde a la densidad de ciudades por región medida. Se puede ver claramente la distribución de ciudades a lo largo del planeta y como la mayor concentración está centrada en la República Democrática del Congo. También se puede apreciar que todas las regiones de color lilas son ramas muy bajas del árbol, mientras que las regiones más amarillas corresponden a ramas donde se producen muchas subdivisiones. Por lo que la búsqueda en estas últimas tendrá un costo mucho mayor.

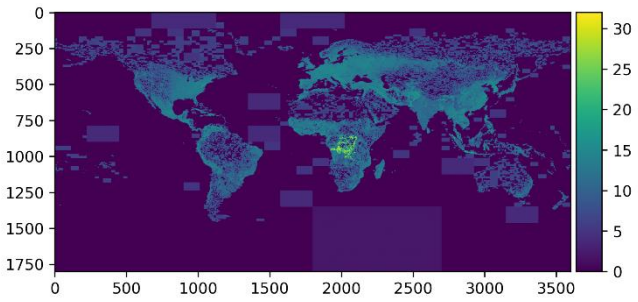

Fig. 11. Histograma de profundidad por cuadrante.

En la figura Fig. 11 se ven los tamaños de los cuadrantes para el barrido por regiones. De esta imagen se puede apreciar la estructura del árbol. En particular como están distribuidos los nodos en los niveles más grandes. Por ejemplo, se puede ver claramente el cuadrante del único nodo de profundidad 2 al sur de África. Por lo que el árbol tiene la mayor cantidad de nodos distribuidos en los otros cuadrantes.

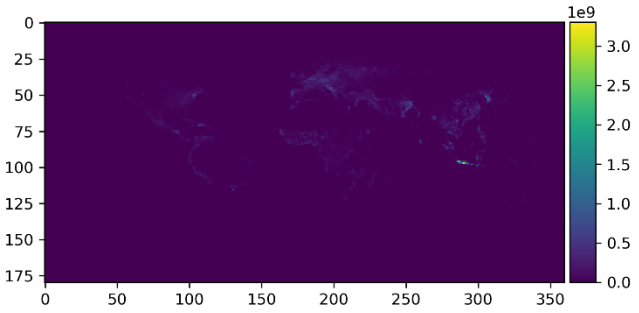

Fig. 12. Histograma 2D de población.

La figura Fig. 12 debería mostrar una imagen similar a la de la figura Fig. 10, sin embargo como los datos de cantidad poblacional no son reales, se puede observar que su distribución es normal y las regiones más amarillas en el plano corresponden directamente con las ciudades que tienen mayor población.

V. Conclusiones

Este trabajo involucró una gran cantidad de esfuerzo para verificar que los resultados fueran correctos, por lo que se entiende que la implementación de un PR-QuadTree no es trivial.

De los resultados de consultas por punto se observa claramente el comportamiento logarítmico de la estructura para cada tipo de consulta. Como para el conjunto de datos de entrada se tiene un máximo de 32 niveles, el acceso a los datos es extremadamente rápido. Tan así que no fue posible obtener una curva de representación que involucrara solo datos individuales, fue necesario hacer una ponderación por ventanas de datos. Por ejemplo, se intento forzar a la estructura a tiempos más altos definiendo las dimensiones del plano hasta 200 millones por 200 millones, para que todo el conjunto de datos ingresados estuviera cercano entre sí, por lo que esto generaría una mayor cantidad de subdivisiones, forzando una mayor profundidad en el árbol. Sin embargo, la altura máxima solo incremento hasta 51 niveles, lo que al momento de realizar búsqueda es despreciable con respecto a los 32 niveles originales, ya que solo se realiza un descenso lineal hacia el punto buscado. Esto demuestra que la búsqueda en este tipo de estructuras es extremadamente rápida, considerando el costo en memoria que lo acompaña.

El histograma de profundidades muestra como su distribución define en cierta forma la estructura del árbol. Mientras más bajas sean las profundidades mejor distribuidos están los datos en el plano representado. Mientras más altas sean las profundidades, se tendrán zonas de mayor densidad de puntos, lo que puede verse reflejado en altos tiempos de consultas por región. Esto es afín a las complejidades de los algoritmos, ya que, para las consultas por puntos la complejidad logarítmica de estos respecto a los niveles de profundidad se ve muy poco afectada por un aumento en la densidad de puntos. Sin embargo, la búsqueda por región se

ve afectada de gran manera debido a su complejidad exponencial.

El costo en memoria de la estructura se puede deducir de la figura Fig. 8, ya que teniendo el total de nodos de cada tipo se calcula el peso utilizado en nodos grises y negros. Esto puede servir para estimar la utilidad de implementar nodos más complejos. Si el nodo tiene demasiados miembros, repercutirá en el uso de memoria de gran manera ya que la complejidad espacial es exponencial. Por ejemplo, en este trabajo se desarrollaron dos implementaciones, una que consideraba nodos blancos como punteros a *NULL* y otra que creaba objetos nodo para los blancos y que guardaba varios parámetros como: sus propias dimensiones y coordenadas de la región que representaban y su profundidad en el árbol. Esto hace que la segunda implementación tuviera un nodo considerablemente más pesado que la primera. En este caso una implementación tenía un peso de 72 bytes y la otra de 96 que considera nodos blancos como objetos creados. Finalmente, esto se tradujo a que la primera implementación utiliza 280 megabytes solamente en nodos grises y negros, mientras que la segunda utiliza 672 megabytes en nodos blancos, grises y negros. De está forma se puede decidir como que limitaciones se puede tener al momento de implementar una estructura como esta.

De los histogramas en las figuras Fig. 10 se puede obtener mucha información útil para analizar los datos. Por ejemplo, en este caso se ve claramente como en la República Democrática del Congo hay una gran densidad de ciudades, por lo que cualquier consulta por región en esta zona tendrá un costo considerablemente mayor que en el resto del plano. De la figura Fig. 11 se puede visualizar que zonas no están siendo aprovechadas por la estructura, por lo que para otro tipo de aplicaciones puede ser de interés saber como se están distribuyendo los nodos por cuadrantes y tratar de optimizar para este caso. Además, con las profundidades se puede obtener cual es el tiempo mínimo promedio para encontrar un dato en una zona dada.

VI. Trabajo Futuro

La implementación de la estructura PR-QuadTree podría mejorarse realizando algunas optimizaciones. Por ejemplo, se podría redefinir la estructura de los nodos para permitirles almacenar más de un solo punto, esto resultaría en menos colisiones de datos y menos subdivisiones, por lo que se tendrían estructuras menos profundas, mejorando los costos de tiempo para búsqueda, inserción y eliminación. Además, la implementación actual considera en cada nodo un puntero a su nodo padre, lo que permite acceder a nodos superiores de manera rápida al momento de colapsar después de eliminar un nodo. De ser necesario, podría redefinirse la función *get_father*, que actualmente retorna el puntero a nodo padre, por una función que calcule el puntero padre por descenso y lo retorne. Esto permitiría reducir el uso de memoria a cambio de un pequeño costo en tiempo. Otra forma en que quizás se podría mejorar la eficiencia de la estructura para casos en que se tuvieran regiones de puntos muy densos sería realizar una transformación espacial a los puntos antes de ser procesados por el PR-QuadTree. Una transformación de dispersión podría aplicarse a un punto antes de ser buscado, insertado o removido, de esta forma se podría controlar la distribución de los puntos en el árbol y así la profundidad máxima de este. Por último, cuando se tienen zonas de puntos muy densas, esto se ve reflejado en un gran número de nodos grises, es decir una estructura profunda, podrían existir situaciones en las que fuese muy necesario evitar descender repetidamente por estas largas ramas, para estos casos se podrían incluir una serie de punteros en la estructura del PR-QuadTree que actuasen como atajos a distintos niveles de profundidad que resultasen convenientes.

VII. Anexos

Los códigos de la tarea pueden encontrarse en: https://github.com/chromosome/proyecto-dsa

Referencias

[1] Samet, Hanan, The Design and Analysis of Spatial Data Structures. Addison-Wesley Longman Publishing Co., Inc., 1990.