**Estructuras**

Cuando se tienen datos de puntos multidimensionales, estos se pueden representar de distintas maneras. La decisión sobre qué tipo de representación utilizar se verá fuertemente afectada por el tipo de operaciones que se ejecutarán sobre estos datos. Para el problema propuesto se debe tener especial cuidado con el uso de memoria, los tiempos de acceso a datos y la capacidad de tener datos dinámicos, los que puedan ser insertados o removidos de la estructura sin perjudicar al resto de los datos.

Los registros de datos espaciales suelen incluir la localización de los datos como uno de sus atributos. Para el caso de datos que se encuentran en un espacio bidimensional, es necesario hacer uso de estructuras de datos que permitan accesos eficientes a los datos para inserción y búsqueda. Un tipo de estructura que permite esto, es el de los Quadtree, que corresponde a un árbol de nodos que subdividen el espacio.

**QuadTree**

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.

**Point-Region Quadtree**

Un PR-Quadtree corresponde a un tipo de Quadtree donde cada nodo puede ser un nodo padre, el cual posee exactamente 4 nodos hijo, o es un nodo hoja. Esta estructura representa una colección de puntos en un espacio bidimensional los que se encuentran relacionados a datos. Así, 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 nodos hoja separados. 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 puede llamar NW, NE, SW, SE.

A diferencia de estructuras como los árboles binarios, donde la estructura 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 remuevan nuevos puntos en función de la localización de los puntos.

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, y 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 especie de Tabla Hash, donde se puede ocupar una coordenada bidimensional como llave al dato.

A picture containing bird

Description automatically generated

A close up of a device

Description automatically generated

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 de nodo, los nodos blancos, los nodos negros y los nodos 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 son nodos que contienen más de un punto 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 en un principio. Así, es posible representar el PR-Quadtree de la siguiente manera mediante estructuras:

**PR-Quadtree**:

Nodo\* root

**Data**:

dato

pos x

pos y

**Nodo**:

Data\* dato

Nodo\* cuadrante\_1

Nodo\* cuadrante\_2

Nodo\* cuadrante\_3

Nodo\* cuadrante\_4

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 al buscado 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 dato del nodo negro es igual al 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.

**Search(pos):**

node = root

While(node is grey):

If: pos in node->first->quad: node = node->first

If: pos in node->second->quad: node = node->second

If: pos in node->third->quad: node = node->third

If: pos in node->fourth->quad: node = node->fourth

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 subcuadrante 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 un nodo 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 de un 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 descender creando nodos grises moviendo los dos puntos a nodos hijos se repite hasta que ambos puntos puedan estar en cuadrantes diferentes.

**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(cuadrants)

set pos and data in correct cuadrant

set old\_pos and old\_data in correct cuadrant

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 por lo menos 2 nodos negros al momento de eliminar el nodo, y estos casos si cumplen con la invariante. Los casos anteriores se ejecutan de forma repetida hasta cumplir con la invariante y a este proceso se le llama Compactación.

**Remove(pos):**

node = search(pos)

father = null

if node doesn’t exist:

end

node->color = white

if node == root:

node->data = null

end

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->childs

father->color = white

node = father

else if ( b == 1 && w == 3 ):

father->color = black

father->data = node->data

delete father->childs

node = father

else

break // si g>=1 o b>=2

end

La **Búsqueda por Región**

Para la búsqueda por región se puede utilizar una búsqueda por profundidad o una búsqueda por anchura, para poder llegar a nodos específicos dentro del árbol. Para una implementación recursiva, en el momento en que se debiese realizar la llamada recursiva de búsqueda en cada nodo hijo, se puede comprobar cuál de los 4 subcuadrantes colisiona en términos de área con la región de búsqueda deseada, de esta forma pueden filtrarse las ramas de nodos en los que se realice la búsqueda, reduciendo el costo en tiempo. Una vez que se llega a los nodos negros es necesario comprobar que el punto del nodo se encuentra contenido por la región de búsqueda deseada, de ser así, se puede aplicar una función específica que procese los datos de cada nodo que cumplan con alguna condición deseada, así se generalizan todos los problemas de procesamiento en regiones. Una de las formas que se implementó fue la de búsqueda por profundidad de forma recursiva.

**Search\_by\_Region( node, region, func ):**

If ( node == NULL or node->color == white ):

Return( 0 )

If ( node->color == black ):

C = position of data in node

If( region contains C ):

Return( func(node->data) )

Else:

Return( neutral\_accum )

Else if ( node->color == grey ):

Int accum = 0

If ( region collides with node->first->cuadrant ):

Accum += Search\_by\_Region ( node->first, region, func )

If ( region collides with node->first->cuadrant ):

Accum += Search\_by\_Region ( node->first, region, func )

If ( region collides with node->first->cuadrant ):

Accum += Search\_by\_Region ( node->first, region, func )

If ( region collides with node->first->cuadrant ):

Accum += Search\_by\_Region ( node->first, region, func )

Return( counter )

A picture containing clock

Description automatically generated

Para la función **collide**, 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 ( C1x,y - C2x,y ) <= r1x,y + r2x,y 🡪 Si Colisionan.

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.

2 \* dx/r \* i - dx + dx/r, 2 \* dy/r \* j - dy + dy/r

Tiempo promedio de inserción de datos = 0.001379421449 = 13.79 microsegundos

**Tiempos de Insertar y Remover Promedio:**

While(para cada dato):

While(repeticiones):

If(existe?):

Timer on

Insertar(dato)

Timer off

Timer on

Remover(dato)

Timer off

**Consideraciones para la Implementación**

* Nombramiento de los cuadrantes
* Dimensiones del cuadrante inicial y centro
* Datos almacenados
* Variables guardadas por Nodo
* Nodos blancos como NULL
* Uso de Doubles para posiciones y Unsigned long long para total de poblaciones
* Cuadrantes no se guardan en el nodo, los cuadrantes se entregan como un marco que viaja con el nodo por las ramas y va cambiando de tamaño con cada nivel, así se reduce el uso de memoria.
* xMin <= x < xMax y

**Resultados**

* Total de nodos blancos, negros, grises en estructura
* Profundidad máxima de la estructura con todos los datos cargados
* Profundidad de la estructura a medida que se insertan nodos

A close up of a sign

Description automatically generated

Ilustración Ciudades por región con subdivisiones de 180x360

A picture containing computer, sign

Description automatically generated

Ilustración Población por región con subdivisiones de 180x360

A picture containing computer, sign

Description automatically generated

A picture containing computer, sign

Description automatically generatedIlustración Profundidades por región

Ilustración Profundidades por región considerando cuadrantes completos

Los puntos amarillos en la gráfica anterior corresponden a la República Democrática del Congo sin considerar a la República del Congo.

A close up of a map

Description automatically generated

Ilustración República Democrática del Congo

**Conclusión**

* Las PR-Quadtree se pueden utilizar como hash-tables de llave bidimensional.
* Se puede mejorar el rendimiento almacenando mas de un punto por cuadrante
* Se podría hacer algo análogo a las funciones Hash, donde antes de aplicar un insert se aplicase una función a los puntos de forma que estos tengan una transformación en el espacio. Si se utilizara una transformación que distribuyese mejor los puntos en el espacio (como para casos en que existan zonas muy densas) se podría obtener un árbol de nodos mejor distribuidos, es decir ramas menos profundas, lo que impactaría directamente en la velocidad de acceso a los datos.
* Al crear la estructura existen algunos nodos hoja con bajas profundidades (2 :O??) esto quiere decir que hay algunos cuadrantes de GRAN tamaño que poseen 1 solo punto, lo mas probable es que sean islas. De este dato se puede inferir/concluir que entre mas arriba este un nodo en el árbol (es decir entre menor sea su profundidad), más alejado estará este punto de los otros. De esta forma la estructura PR-Quadtree puede servir para deducir el aislamiento entre puntos. AHORA, esto no siempre es así, ya que podríamos estar en nodos de baja profundidad pero un nodo hermano puede tener un punto que este muy cercano, por ejemplo, si dos puntos están al lado del mismo borde en sus cuadrantes, entonces no están aislados! De todas maneras debe existir una forma de decidir el nivel de aislamiento, quizás se puede calcular la existencia de al menos un nodo entre ambos nodos, y eso aseguraría que se encuentran alejados.
* El puntero a nodo padre podría eliminarse reemplazando get\_father() por un proceso que calcule el father.

- los tiempos de insercion son tan rapidos ( menores a 50 uSeg) que las pruebas de velocidad tienden a mostrar

ruido provocado por "quien sabe qué estará haciendo el compu xD". Debido a esto hicimos la prueba de aumentar

el area del cuadrante inicial de 360x360 a 200MillonesX200Millones XD pero esto solo aumento la profundidad

maxima de los nodos en el quadtree de 32 a 51, por lo que sigue siendo una profundidad muy pequeña para realizar

pruebas :p

por obtener:!!!!

* Obtener grafica de hisrograma de profundidades por inserción para mostrar como crece el árbol en profundidad a medida que se insertan elementos.
* Histogramas y profundidades
* El quadtree ideal es como una pirámide de pascal con todos los nodos en las hojas que es equivalente a tener todos los puntos separados de forma equidistante en el espacio. Usando la recurrencia de la pirámide de pascal se puede obtener el total de nodos grises (ya que no habrían nodos blancos en un quadtree ideal) que serian necesarios para guardar los n nodos grises (con lo que se puede concluir el uso de memoria mínimo para guardar n datos y la eficiencia en memoria al comparar nodos grises vs nodos negros. :OOO!
* A(n) = 4\*A(n-1) + 1
* [4:03 AM]
* an = 5an-1 - 4an-2
* [4:04 AM]
* x^2 - 5x +4
* [4:04 AM]
* (x-4)(x-1)
* [4:08 AM]
* an = 4/3 \* 4^n - 1/3
* **NEW**
* [4:14 AM]
* 5592405

<https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/PRquadtree.html>

<http://courses.cs.vt.edu/~cs3114/Summer15/Notes/T04_PRQuadTrees.pdf>

<https://es.wikipedia.org/wiki/Quadtree>

<http://www.cs.umd.edu/~hjs/pubs/AngSSD89.pdf>