Tarea 1
CC4102 - Diseño y Análisis de Algoritmos

Diego Gajardo, Francisco Hafon, Rodrigo Alonso

21 de octubre de 2012

Índice

1. Introducción **3**

2. Diseño Experimental **4**

3. Resultados y Análisis **6**

4. Conclusiones **7**

1. Introducción

Un R-Tree es una estructura similar a un B+ Tree, donde las hojas contienen punteros a datos reales, y donde los nodos internos corresponden a elementos agrupadores, en este caso, el menor rectángulo tal que todo rectángulo en su nodo hijo esté encerrado por el primero, i.e., su minimum bounding rectangle (MBR). En este caso la estructura está orientada a emplearse bajo el modelo de memoria secundaria; de esta manera, se busca minimizar la cantidad de accesos a disco, dado que cada nodo del árbol se almacena en una página de disco. Este tipo de estructuras se utiliza principalmente para catalogar y posteriormente acceder a información geoespacial de forma rápida, por ejemplo, para servicios de mapas o de búsqueda contextual (Google Maps, Foursquare).

Dado un valor b fijo, cada nodo del árbol contiene entre b y $2b$ rectángulos, donde $2b$ es el máximo número de rectángulos que caben, junto con otra información relevante de un nodo, en una página de disco (bloque). La raíz es una excepción, pues puede tener entre 0 y $2b$ rectángulos.

Un R-Tree suele proveer las siguientes operaciones:

- Insertar un rectángulo, que apunta a datos reales, en el árbol.
- Buscar todos los rectángulos del árbol que intersectan a un rectángulo dado.
- Borrar un rectángulo en las hojas del árbol.

El algoritmo de inserción consiste en descender desde la raíz del árbol hasta llegar a una hoja. En cada nodo interno, se sigue la ruta indicada por el MBR que requiera menor incremento de área si se le inserta el nuevo rectángulo, continuando en su hijo. Si hay dos o más MBRs cuyo incremento es el mismo, se escoge el MBR de menor área. Al llegar a una hoja, se actualizan los MBRs de sus ancestros, y luego se intenta insertar el rectángulo nuevo; si ocurre overflow del nodo, se realiza split (recursivamente si es necesario, hasta llegar a la raíz), agregando una nueva raíz si se hace split de ésta. Para dividir un nodo, existen múltiples métodos o variantes; se revisarán dos de ellas en este trabajo.

En tanto, el algoritmo de borrado consiste en, dada una hoja del árbol y un rectángulo en ella, eliminar el rectángulo, actualizar los MBRs de los ancestros de la hoja y posteriormente, resolver underflow en caso de ocurrir.

Se pretende corroborar con datos experimentales lo predicho bajo el modelo de memoria secundaria respecto de la cantidad de accesos a disco, y por ende de la complejidad de los algoritmos mencionados. En particular, se busca corroborar la siguiente hipótesis:

$$find(C) \in O(n^{\alpha})$$
$$insert(C), delete(C) \in O(\log_b n)$$

2. Diseño Experimental

Cada nodo *en disco* presenta la siguiente estructura:

n	$path_{padre}$	$rect_1$	$path_{hijo_1}$	$\cdots$	$rect_n$	$path_{hijo_n}$
2	8	16	8		16	8

Cuadro 1: Estructura de un nodo en disco (tamaños en bytes)

Si n es el número de pares de rectángulos/hijos en un nodo, este último utiliza $2 + 8 + n \cdot (16 + 8) = 10 + 24n$ bytes, y para $n = 2b$ debe ser menor que el tamaño de bloque B, por lo tanto:

$$b \leq (B - 10)/48$$

$$\Rightarrow b_{MAX} = \lfloor (B - 10)/48 \rfloor$$

Cada nodo *en memoria* consta de su nombre de archivo, la cantidad de rectángulos que contiene, el nombre de archivo de su padre, el arreglo de los rectángulos que contiene, el arreglo de los nombres de archivo de sus hijos, y un puntero auxiliar a otro nodo (el cual se emplea en la implementación como caché para reducir el número de accesos a disco).

Cada rectángulo en memoria se representa únicamente por las coordenadas de dos vértices opuestos $(x1, y1)$ y $(x2, y2)$, que lo definen completamente. Se impone en la implementación que $x1 \leq x2$ e $y1 \leq y2$. La estructura que representa a un R-Tree en memoria, en tanto, es global a toda la ejecución y está descrita por el nombre de archivo de su raíz y el valor de su altura.

Para verificar la hipótesis se tienen dos variantes de los algoritmos de inserción y borrado. En el caso de inserción, las variantes de split a considerar son las siguientes:

- Escoger los dos rectángulos que formen el mayor MBR, removiéndolos del nodo original y separándolos en grupos distintos. Luego ir tomando uno a uno los rectángulos, escogiendo cada vez el que tenga mayor diferencia entre los incrementos del MBR de cada grupo si se les insertase tal rectángulo respectivamente, y agregarlo al grupo cuyo MBR deba crecer menos.
- Tomar el par de rectángulos cuyos centros estén más alejados entre sí, y asignarlos a grupos distintos. Luego, se procede de igual forma que en la variante anterior.

Para manejar el underflow en el borrado, se considerarán los siguientes métodos:

- Eliminar el nodo con underflow, y propagar recursivamente hacia la raíz. Luego de encontrar un nodo que no quede con underflow, se insertan los rectángulos huérfanos de los nodos eliminados (insertando los rectángulos reales en las hojas y los MBRs en niveles más altos) y se ajusta el MBR del nodo.
- Consultar a un nodo hermano si puede donar un rectángulo sin quedar con underflow, y en caso de ser posible, tomar el rectángulo que minimice la suma de las dos nuevas áreas de los MBRs de ambos nodos. De lo contrario, se fusionan ambos nodos y se actualiza el MBR.

Se estudiará el comportamiento de ambas variantes para verificar la hipótesis para n rectángulos, con n entre 2^9 y 2^19, experimentando con secuencias de operaciones del tipo $(Insert^n)(Find^n)$. Además, se utilizarán los resultados para escoger la mejor variante, a emplear posteriormente en pruebas con datos geoespaciales reales.

3. Resultados y Análisis

Se ha elegido hacer split por área (variante #1), ya que presenta un mejor rendimiento en número de lecturas a disco y tiempo por secuencia de operaciones (ver datos adjuntos).

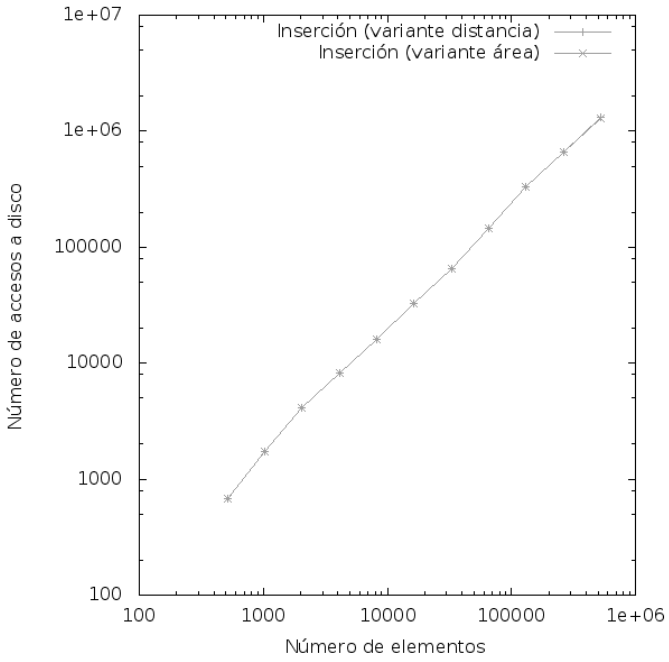

Figura 1: Accesos a disco por secuencia de operaciones $i^k m$ en función del número de nodos

Se obtuvo un comportamiento $O(n \log n)$ en el número de accesos a disco por secuencia de operaciones, en función del número de elementos n. Por lo tanto, el algoritmo es de orden logarítmico por cada operación.

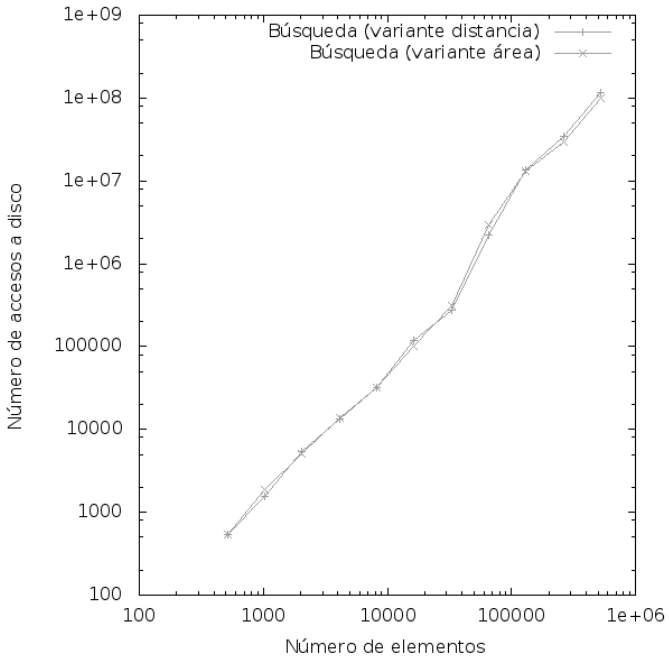

Figura 2: Accesos a disco por secuencia de operaciones $f^k m$ en función del número de nodos

4. Conclusiones

Se verifica el comportamiento $O(n^\alpha)$ de la operación de búsqueda y $O(\log_b n)$ de la operación de inserción, lo cual corrobora parcialmente la hipótesis planteada. No se logró constatar $O(\log_b n)$ para la operación de borrado, al no haber sido implementada correctamente. No se logró operar con los datos geoespaciales reales dentro del plazo estipulado.