Tarea 1 - R*-Trees

CC4102 - Diseño y Análisis de Algoritmos
Profesor: Pablo Barceló Auxiliar: Jorge Bahamonde

Fecha de Entrega: 24 de Abril

1 Introducción

Un R-tree[1] es un árbol, parecido al B-tree, que permite manipular *rectángulos* en memoria secundaria. Se permite *insertar* rectángulos y *buscar* rectángulos. La operación de búsqueda es como sigue: Dado un rectángulo C como entrada, debemos retornar todos los rectángulos en el R-tree que intersectan C.

El objetivo de esta tarea es implementar y evaluar en la práctica una variante del R*-tree conocida como el R*-tree. En particular, nos interesa analizar el impacto de variantes de inserción sobre la operación de búsqueda. Para la evaluación consideraremos datos generados al azar. Se espera que se implementen los algoritmos y se entregue un informe que indique claramente los siguientes puntos:

1. Las *hipótesis* escogidas antes de realizar los experimentos.
2. El *diseño experimental*, incluyendo los detalles de la implementación de los algoritmos, la generación de las instancias y las medidas de rendimiento utilizadas.
3. La *presentación de los resultados* en forma de una descripción textual, tablas y/o gráficos.
4. El *análisis e interpretación* de los resultados.

2 Las Estructuras

Cada nodo de un R*-tree almacena al menos t rectángulos, excepto la raíz que almacena al menos 2 rectángulos. Todos los nodos almacenan a lo más $2t$ rectángulos. Cada rectángulo está asociado a un subárbol (salvo cuando el nodo es hoja). Los rectángulos reales se encuentran almacenados en las hojas del árbol. Cada rectángulo en un nodo interno es un MBR (*Minimum Bounding Rectangle*) que es el rectángulo más pequeño que contiene a todos los rectángulos en el subárbol correspondiente.

Se pide implementar las siguientes operaciones:

1. *buscar*(C): Recorremos el árbol para buscar todos los rectángulos en los nodos hojas que intersectan a C. Si en algún nodo interno, un MBR no interesecta a C, podemos descartar el subárbol correspondiente.i
2. *insertar*(C): Recorremos el árbol desde la raíz hasta alguna hoja, e insertamos C en esa hoja.

   Dependiendo de en qué posición estemos en el árbol, buscaremos minimizar el aumento de área o el aumento del *overlap*, definido de la siguiente forma:

$$\text{overlap}(E_k) = \sum_{i=1, i\neq k}^{p} \text{area}(E_i \cap E_k), 1 \leq k \leq p$$

[1] No confundir con `/r/trees`.

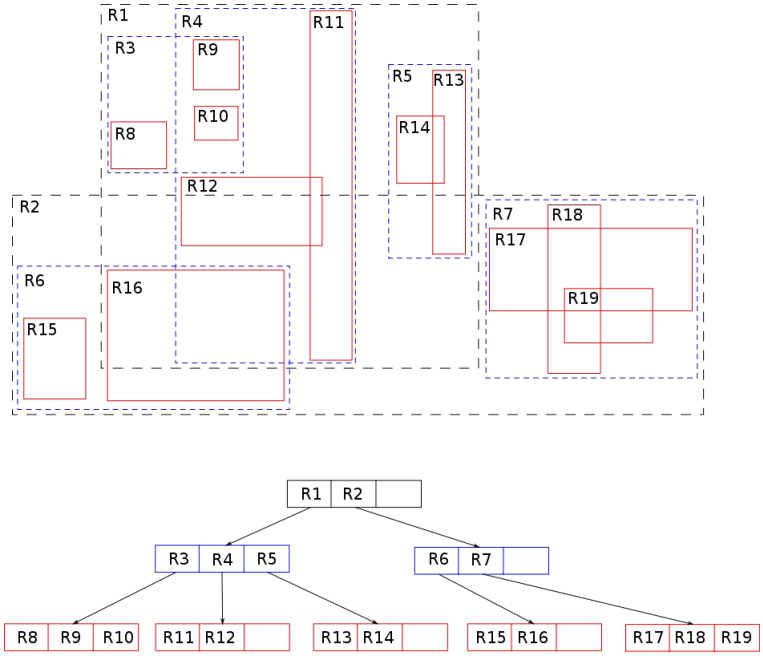

Figure 1: Ejemplo de R*-tree

siendo $E_1 ... E_p$ los MBRs correspondientes los hijos de un nodo dado.

El recorrido para la inserción se realiza de la siguiente forma: si el nodo actual no es una hoja (en cuyo caso el recorrido termina) se chequea si los hijos de éste son hojas. Si es así, se escoge el MBR F que requiera el menor incremento de *overlap* para incluir C y se baja por éste. En caso de empates, se escoge el MBR que requiera el menor incremento de área (en caso de un nuevo empate, se escoge el MBR de menor área).

En caso de que los hijos del nodo actual no sean hojas, se escoge el MBR que requiera el menor incremento de área (en caso de empate, se escoge el MBR de menor área) para continuar.

Una vez insertado C se actualizan los MBRs desde la hoja hacia la raíz. En caso de haber *overflow*, existen dos algoritmos a utilizar. Por un lado, puede hacerse un *split* del nodo.

Split

Se define el *margen* (para seguir la nomenclatura de la literatura) de un rectángulo como su perímetro. Por otro lado, es necesario definir lo que será una *distribución* del conjunto de rectángulos. Suponiendo que se tienen $2t+1$ rectángulos ordenados en una lista, llamaremos distribución a dividir la lista en dos conjuntos no vacíos: el de los $m-1+k$ primeros y el de los restantes, dado un parámetro m. Así, hay $2t-2m+2$ distribuciones posibles.

En el primer paso del split, se escoge la dimensión de corte. Se ordenan, para cada dimensión, los rectángulos por sus valores mínimos a lo largo de ésta. Luego, para cada distribución (dado un m fijo), se calcula la suma del margen del MBR de cada uno de los dos grupos generados. Se calcula la suma total S de estos márgenes para todas las distribuciones posibles dado m. Se repite el proceso ordenando según los valores máximos. Finalmente, se escoge la dimensión que tenga el mínimo valor de S.

En un segundo paso, se escoge el índice de corte a lo largo de la dimensión elegida. Para esto, se elige la distribución que tenga la mínima intersección entre los MBRs que engloban a cada uno de los dos grupos de la distribución. En caso de empate, se escoge la distribución que genere los MBRs de menor área.

Reinsert

Por otro lado, se tiene el algoritmo de *reinserción* para el caso de overflow. En primer lugar, se ordenan los $2t + 1$ rectángulos según su distancia al centro del MBR correspondiente a N (en orden decreciente). A continuación, se toman los primeros p rectángulos y se remueven de N. Luego se reinsertan estos rectángulos en el árbol, comenzando por aquel con la menor distancia calculada anteriormente. Nótese que deben ser insertados a la misma altura de la que son extraídos.

Finalmente, el algoritmo para tratar los casos de overflow es como sigue:

---

function OnOverflow(...)
  if no estamos en la raíz y es la primera llamada de OnOverflow en este nivel del árbol para esta inserción **then**
    Reinsert
  else
    Split
  end if
end function

Borrado

Finalmente, el algoritmo de reinserción requiere implementar el borrado de rectángulos del árbol:

Se elimina el nodo (guardando los rectángulos huérfanos) y se propagan los cambios hacia arriba (es decir, se elimina el MBR del padre y si se produce underflow, repite); en caso contrario, se ajusta el MBR de forma de contener a los rectángulos restantes. Finalmente, los rectángulos huérfanos son reinsertados en el árbol. Tenga cuidado que al guardar los huérfanos también puede guardar MBRs: en ese caso, debe insertar sólo los rectángulos reales en las hojas, mientras que los MBRs deben ir quedando en niveles más altos del árbol, de forma que las hojas de esos subárboles queden al mismo nivel que las hojas del árbol principal.

Este es el algoritmo usual de borrado para un R*-Tree.

3 Implementación

Implemente dos versiones del algoritmo. Por un lado, la antes descrita, que usa Split y Reinsert. Además de esta versión, implemente una en que no se utiliza Reinsert ante un overflow (es decir, sólo usa Split).

4 Experimentos

Escoja el parámetro t de acuerdo a las características de su máquina de manera que un nodo caiga siempre en una página de disco. Documente las características de su máquina, el sistema operativo, lenguaje y compilador utilizados, RAM, y características del disco duro. Utilice $m = 40\%$ para el split y $p = 30\%$ para la reinserción.

En los experimentos se pide comparar la construcción del R*-tree con cada variante de insertar, y evaluar el desempeño de la operación buscar.

Genere conjuntos de n rectángulos, con $n \in \{2^9, 2^{12}, 2^{15}, \ldots, 2^{21}\}$. Usted deberá generar los rectángulos al azar de manera que las coordenadas sean reales uniformemente distribuidos en el rango $[0, \ldots, 500000]$. El área de cada rectángulo debe estar uniformemente distribuido en el rango [1,. . . ,100].

Para cada conjunto de tamaño n, construya el R*-tree con las dos variantes. Documente el tiempo utilizado y la cantidad de accesos a disco. Una vez construido el R*-tree, genere $n/10$ rectángulos al azar para hacer consultas de búsqueda. Documente en cada caso el tiempo y cantidad de accesos a disco promedio de búsqueda.

5 Entrega de la Tarea

- La tarea puede realizarse en grupos de a lo más 2 personas.
- No se permiten atrasos.
- Para la implementación puede utilizar `C`, `C++`, `Java` o `Python`. Para el informe se recomienda utilizar LaTeX.
- Escriba un informe claro y conciso. Las ponderaciones del informe y la implementación en su nota final son las mismas.
- La entrega será a través de U-Cursos y deberá incluir el informe junto con el código fuente de la implementación (y todas las indicaciones necesarias para su ejecución). El día hábil siguiente a la entrega, deberá dejar un informe impreso donde Sandra.

6 Links

- El paper original de los R*-Trees: `http://dbs.mathematik.uni-marburg.de/publications/myPapers/1990/BKSS90.pdf`
- El paper original de los R-Trees: `http://www-db.deis.unibo.it/courses/SI-LS/papers/Gut84.pdf`