Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Depto. de Ciencias de la Computación
CC4102 - Diseño y Análisis de Algoritmos

Tarea 1

Integrantes : Rodrigo Delgado
Belisario Panay
Gabriel Sanhueza
Profesor : Gonzalo Navarro
Ayudante : Sebastián Ferrada
Auxiliar : Jorge Bahamondes

Índice

1. Introducción

1.1. Problema a resolver

Los R-trees son un tipo de árbol que se maneja en memoria secundaria, el cual contiene rectángulos como información. El problema a resolver consiste en implementar un R-Tree, una herramienta de búsquedas de rectángulos y 2 heurísticas de inserción de rectángulos en el R-Tree.
En particular, se busca evaluar el impacto entre distintas versiones de *inserción* que manejan una versión específica de manejo de *overflow* (para este caso, *Linear Split* y *Greene Split*.

1.2. Hipótesis

Especificaciones de la máquina utilizada (Anakena)

- Procesador: Intel ®Xeon ®CPU E5620 @ 2.40GHz
- Arquitectura: x86_64
- Número de CPUs: 12
- Memoria RAM: 12288 KB
- Tamaño de página de disco: M = 512 bytes ($\ulcorner 40\,\%M \urcorner = 205$)
- Sistema Operativo: Linux version 4.4.6-gentoo
- Lenguaje usado: C
- Compilador: gcc version 4.9.3

Usaremos siempre la misma semilla de aleatoriedad, para poder tener experimentos "aleatorios" repetibles.
Limitamos la cantidad máxima de rectángulos en un nodo con respecto al tamaño de página del disco.
Así, si *BLOCK_SIZE* = 4096 y tenemos un *struct rectangle* llamado *Rectangle*:

- M = *BLOCK_SIZE / sizeof(struct rectangle)* = 4096/32 = 128
- m = 40 % de M. = $40\,\% * 128 = 51$

Por último, la idea es nunca tener más de dos archivos abiertos en un instante dado.

2. Diseño Experimental

En nuestra implementación hicimos 2 *structs* para manejar los rectángulos en el R-Tree.

2.1. Structs

2.1.1. Rectangle

Esta estructura posee información sobre:

- Coordenada X.
- Coordenada Y.
- Ancho del rectángulo.
- Alto del rectángulo.
- Identificador (nombre) del rectángulo.
- Identificador del hijo de este rectángulo.

2.1.2. Node

Esta estructura posee información sobre:

- Arreglo dinámico de rectángulos.
- Número de rectángulos escritos en el arreglo.
- Nombre del nodo actual (para uso como nombre de archivo en disco).

2.2. Constantes

- **BLOCK_SIZE:** Tamaño del bloque en disco.
- **count**: Variable global para diferenciar nodos al escribirlos a disco.
- **M**: BLOCK_SIZE / sizeof(Rectangle) — Máximo número de rectángulos en un nodo.
- **m**: 40 % de M — Mínimo número de rectángulos en un nodo.

2.3. Funciones

- *Rectangle* createRectangle(int x, int y, int w, int h, int id)*: Crea un rectángulo con coordenadas y nombre.
- *Node* createNode()*: Crea un nodo con un arreglo de rectángulos como información interna.
- *writeToDisk(Node *data)*: Escribe los datos de un nodo a disco.
- *Node* loadFromDisk(char *filename)*: Carga un archivo del disco.
- *int intersect (Rectangle *r1, Rectangle *r2)*: Retorna un "booleano"que dice si los dos rectángulos se intersectan.
- *int MBR(Rectangle *r1, Rectangle *r2)*: Calcula la nueva Area si se agrega r2 a r1.
- *void mergeRectangle(Rectangle *r1, Rectangle *r2)*: Actualiza las coordenadas de r1 al agregarle r2. (Solo las actualiza, no añade r2 a r1).

- *int partitionX(Node *header,int inicio,int final)*: Funcion auxiliar para quicksort, eje X.
- *int partitionY(Node *header,int inicio,int final)*: Funcion auxiliar para quicksort, eje Y.
- *void quicksort(Node *header,int inicio,int final,int d)*: Quicksort para rectángulos.
- *Rectangle **makeRandom(Node pNode)*: Desordena el orden de los rectángulos de un nodo para aleatorizar el split.
- *void printRectangle(Rectangle *r)*: Imprime información de un rectángulo.
- *Rectangle **calculateXRectangles(Node *pNode)*: Calcula los rectangulos con mayor bajo y menor alto en un arreglo para el eje X e Y.
- *int *calculateBounds(Node *pNode)*: Calcula el rectángulo más grande de todos los que están en el nodo.
- *int randomNum(int max)*: Retorna un número aleatorio acotado.
- *Rectangle ** bateriaRectangulos(int n)*: Crea n rectángulos distintos para experimentación.
- *Rectangle **copy(Rectangle **pRectangle, int n)*: Copia un arreglo de n elementos.
- *Node *search(char *nodeName, Rectangle *rect)*: Busca en el nodo todos los rectángulos que intersectan a *rect.
- *void insertToRootLinear(char *nodeName, Rectangle *r)*: Inserción Linear Split con control de Overflow en Root.
- *void insertToRootGreene(char* nodeName, Rectangle *r)*: Inserción Greene Split con control de Overflow en Root.
- *void insertLinear(char *nodeName , Rectangle *r)*: Inserción Linear Split con control de Overflow en el resto de los nodos.
- *void insertGreene(char *nodeName, Rectangle *r)*: Inserción Greene Split con control de Overflow en el resto de los nodos.
- *Rectangle ** linearSplit(Node *header)*: Control de overflow usando Linear Split.
- *Rectangle ** greeneSplit(Node *header)*: Control de overflow usando Greene Split.

3. Presentación de los Resultados

3.1. Tiempo de Construcción del R-Tree

3.1.1. Linear Split

Hacer un gráfico?

3.1.2. Greene Split

Hacer otro gráfico?

3.2. Espacio ocupado y porcentaje de llenado de páginas de disco

Aún más gráficos?

3.3. Desempeño de operación *Buscar*

4. Análisis e Interpretación

Aquí hay que hacer el chamullo correspondiente (?)

5. Conclusiones

TODO