CI4251 - Programación Funcional Avanzada
Tarea 4

Ernesto Hernández-Novich
86-17791
<emhn@usb.ve>

Junio 14, 2012

Hilbert R-Tree

El R-Tree de Hilbert [1], es una estructura de datos diseñada para almacenar y consultar datos geométricos bidimensionales (rectas, polígonos, curvas, etc.)

Esta estructura de datos fue presentada por primera vez en [2], trabajo de notable influencia para mejorar las técnicas de almacenamiento de datos no escalares. En particular, PostgreSQL emplea R-Trees para construir índices eficientes para información vectorial, geográfica y para búsqueda de texto.

Su implantación debe estar en el módulo `RTree.hs`, siguiendo de cerca los detalles de [2], bajo las siguientes condiciones:

- Su implantación de R-Tree *solamente* va a operar con rectángulos, de coordenadas X e Y entre 0 y 65536. Para eso, basta representar los cuatro vértices con un tipo de datos similar a

```
data Rectangle = R { ul, ll, lr, ur :: (Int,Int) }
```

- Debe diseñar el tipo de datos para el R-Tree.

```
data RTree = ...
```

  El tipo de datos debe ser completamente puro, con las operaciones

  - `insert` – agrega un nuevo rectángulo a la estructura. Insertar un rectángulo duplicado es causa de error.

```
insert :: RTree -> Rectangle -> Either e RTree
```

  - `delete` – elimina un rectángulo presente en la estructura. Eliminar un rectángulo inexistente es causa de error.

```
delete :: RTree -> Rectangle -> Either e RTree
```

  - `search` – permite consultar la estructura para determinar si el rectángulo suministrado como parámetro se solapa con uno o más rectángulos en la estructura. El resultado de la función es la lista de rectángulos solapados.

```
search :: RTree -> Rectangle -> Maybe [Rectangle]
```

Uso de la aplicación

La aplicación de manipulación de R-Trees será empleada desde la línea de comandos. Su ejecución tiene tres fases:

1. **Inicio** – la aplicación recibirá como argumento de línea de comando el nombre de un archivo de texto con la colección inicial de rectangulos a cargar. Los archivos contendrán una línea por cada rectángulo, similares a

   ```
   2830,4212,2830,4189,3255,4189,3255,4212
   ```

   que representan las coordenadas (x_0, y_0), (x_0, y_1), (x_1, y_1) y (x_1, y_0) respectivamente. Debe verificar que en efecto se trate de un rectángulo antes de aceptarlo.

   Puede descargar un archivo de muestra con suficientes rectángulos del mismo lugar del cual descargó este archivo – si le causa curiosidad, los rectángulos salieron de [3] [1]

   El usuario podría omitir el archivo o suministrar uno vacío, lo cual no es causa de error, pero debe indicarse la acción efectivamente completada, e.g.

   ```
   $ rtree /tmp/foo.txt
   Leídos 42 rectángulos desde el archivo /tmp/foo.txt
   ...
   ```

   y en caso de omitir el argumento

   ```
   $ rtree
   No se indicó base de datos inicial.
   ```

   Al terminar esta fase, el programa debe haber leído el archivo y convertido sus contenidos a la estructura `RTree` – note que la estructura podría estar vacía. Esa estructura *inicial* debe permanecer **inmutable** durante el resto de la operación de la aplicación.

[1] ¡No haga click en este enlace a menos que quiera verse succionado fatalmente por una espiral de procrastinación!

2. **Leer-Evaluar-Responder** – la aplicación presentará un *prompt* para que el usuario indique las operaciones a efectuar sobre la base de datos cargada. Las operaciones se indican con los comandos:

   - `insert r` – agrega el rectángulo `r` a la base de datos.
   - `delete r` – elimina el rectángulo `r` de la base de datos.
   - `search r n` – si el rectángulo `r` se solapa con uno o más rectángulos en la base de datos, indicará la cantidad *total* de triángulos con los que solapa y mostrará hasta `n` de esos rectángulos. En caso contrario, indicará que no se encontraron solapamientos.
   - `kthxbye` – termina la ejecución.

   Tan pronto como el usuario solicite una modificación a la estructura original, el programa debe construir una copia *mutable* para soportar todos los `insert` y `delete` que el usuario quiere aplicar. Así mismo, a partir de la primera modificación, cada vez que se ejecute un `search`, es necesario realizar la consulta sobre *ambas* estructuras y reportar los resultados convenientemente.

   Los errores derivados de operaciones `insert` y `delete` deben ser manejados inmediatamente y nunca puede hacer abortar al programa.

3. **Resumen** – la aplicación terminará la ejecución mostrando

   - Cantidad de operaciones exitosas, discriminando inserciones, eliminaciones y búsquedas. En el caso de las búsquedas, debe discriminar aquellas contra la estructura original y aquellas contra la estructura modificada, solamente si en efecto hubo alguna modificación.
   - Cantidad de operaciones fallidas, discriminando inserciones duplicadas, eliminación de inexistentes y búsquedas sin solapamientos.

Su solución *debe* explotar los transformadores de Monads para que toda la Fase 2 sea completada en un Monad único que combine las funcionalidades necesarias para mantener la base de datos original, la base de datos modificada, el registro de operaciones, el manejo de errores y la gestión de IO con el usuario.

Pruebas

Su implantación de `RTree` debe venir acompañada de una instancia `Arbitrary` que permita generar casos de prueba para un arnés QuickCheck de manera que haya una buena mezcla de casos y se garantice la terminación del generador.

Así mismo, debe ofrecer suficientes propiedades QuickCheck que permitan comprobar la correctitud de su implantación. Ud. debe comprobar:

- El tipo `RTree` y sus operaciones – incluso la generación de errores.
- La construcción de `Rectangle` a partir de los datos leídos desde el archivo o desde la línea de comandos.

Entrega

Ud. deberá entregar los archivos `RTree.hs` y `Main.hs` correspondientes a su implantación. Los archivos deben documentarse usando Haddock para las funciones públicas de los módulos y con comentarios privados para las funciones internas.

Trabajen en equipos de **dos** personas, y completen la entrega antes del 2012-06-14 23:59 VET. Note que los equipos conformados para esta entrega serán los mismos que para el proyecto final.

Referencias

[1] Hilbert R-Tree en WikiPedia

[2] Kamel & Faloutsos
Hilbert R-tree: An improved R-tree using Fractals

[3] Visual Transistor-Level Simulation of the 6502 CPU