DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico - N^o 2

Algoritmos y Estructura de Datos III — 1er cuatrimestre 2007

Integrante	LU	Correo electrónico
García, Ana Daniela	530/05	danita719@yahoo.com.ar
Joaquin Miguez	120/04	jmiguez@dc.uba.ar
Fernando Bugni	611/05	fernando.bugni@gmail.com
Engler, Christian Alejandro	314/05	caeycae@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

1. Introducción

1.1. Enunciado

UBA - FCEyN - Departamento de Computación
ALGORITMOS Y ESTRUCTURAS DE DATOS III
Trabajo Práctico N^o 1
Primera entrega: 7-5-2007, hasta las 19:30 horas
Segunda entrega: 28-5-2007, hasta las 19:30 horas
Ver información general sobre los Trabajos Prácticos en la página de la materia en Internet.

Ver información general sobre los Trabajos Prácticos en la página de la materia en Internet. El Booble art es un programa que muestra el mapa de Buenos Aires y sus atracciones turísticas. A medida que vamos recorriendo el mapa en la pantalla, podemos hacer un click en un punto y nos muestra varias fotos que corresponden a lugares que están cerca del punto que marcamos.

Nuestro objetivo es realizar un algoritmo que, dado un punto en el plano (la pantalla), decida cuáles son las fotos a mostrar. Para esto, vamos a considerar los siguientes criterios:

- Cada foto tiene una posición en el plano dada por sus coordenadas.
- La pantalla tiene un tamaño fijo de $s * t$
- Al clickear en un punto del plano, las fotos que deben mostrarse son TODAS aquellas fotos que contienen a ese punto (es decir, el punto está dentro del área dado por las coordenadas de la foto).
- Las fotos pueden tener diferentes tamaños, intersecarse, estar contenidas una dentro de otra, etc.
- Por supuesto, al listar las fotos a mostrar, pueden haber fotos que se intersecan, etc. Esto no nos preocupa, sólo queremos conocer la lista de fotos a mostrarse en otra pantalla, donde otro programa (que se hará en Algoritmos 4) acomodará estas fotos

Vamos a realizar un preprocesamiento del conjunto de fotos para luego, cada vez que se realiza un click, podamos decidir qué fotos deben mostrarse en forma eficiente.

1. (a) Dada un conjunto de fotos, construir un *arboldeintervalos* usando **árboles red black** (será explicado en clase) con las operación *Insertar*.

   (b) Indicar el orden de las operación *insertar* No es obligatorio que figure en el informe la justificación del orden de *insertar* pero se preguntará sobre esto durante el coloquio.

   (c) Calcular la complejidad TEMPORAL Y ESPACIAL de construir el árbol de intervalos.

   (d) Diseñar un algoritmo que, dado un punto en el plano, decida el conjunto de fotos a mostrar. Para esto, implementar la operación *buscarInterseccion*.

   (e) Calcular la complejidad TEMPORAL Y ESPACIAL de realizar una consulta (llamamos "consulta" a clickear sobre un punto del plano).

   (f) Analizar el algoritmo midiendo el tiempo de ejecución tanto para el armado del árbol como para consultas. Es decir, para diferentes conjuntos de fotos (instancias), realizar varias consultas.

2. Ahora, pensemos la siguiente variante. Cada tanto, el programa Booble art se actualiza y adiciona nuevas fotos a su mapa. Explicar cómo cambia el algoritmo anterior, sus funciones, etc. Calcular la complejidad de la función *insertar* para este caso.

Entrada `Tp2Ej1.in` Este archivo contiene varias instancias de entrada. La primera línea contiene el número de instancias. Cada instancia es definida de la siguiente manera. La primera línea contiene el número de fotos n, las siguientes n líneas contiene las coordenadas de las fotos: $x0$ $x1$ $y0$ $y1$, donde $x0 \leq x1$ y $y0 \leq y1$.

`Tp2Ej1d.in` Este archivo contiene consultas para hacer sobre todas las instancias del archivo anterior. La primera línea contiene la cantidad de consultas (k) y las siguientes k líneas contienen las coordenadas del punto del plano de la forma x y.

NOTA: NO ES OBLIGATORIO QUE LA ENTRADA SE INGRESE POR LINEA DE COMANDOS, PERO EL QUE QUIERE PUEDE HACERLO DE AMBAS MANERAS.

Salida `Tp2Ej4.out` Por cada instancia, cada línea contiene una foto que se debe mostrar identificada por sus cuatro coordenadas. La salida de cada instancias está separada por un 0.

Bibliografía:

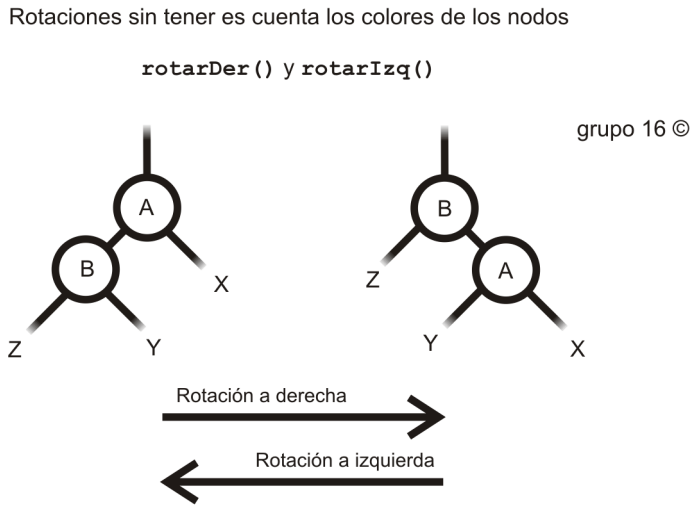
Fotocopiadora: "Introduction to Algorithms", Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Capítulo 13.

Sugerencia: Para buscar información sobre interval-trees pueden consultar en
`http://www.dgp.toronto.edu/people/JamesStewart/378notes/22intervals/`

2. Ejercicio 1

2.1. *Árbol Rojo y Negro*

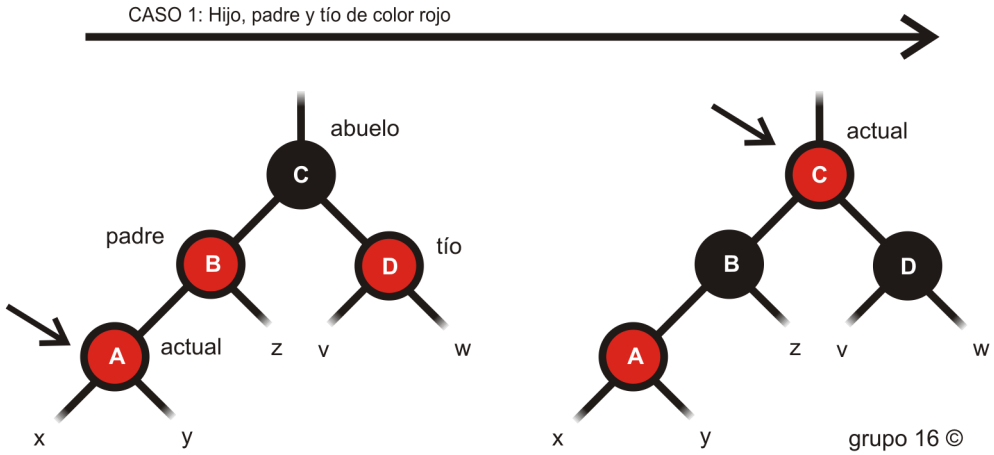
2.1.1. ROTACIONES



(figura 1.1)

Rotaciones simples para mantener el balanceo. (No son para hacerlo un AVL)

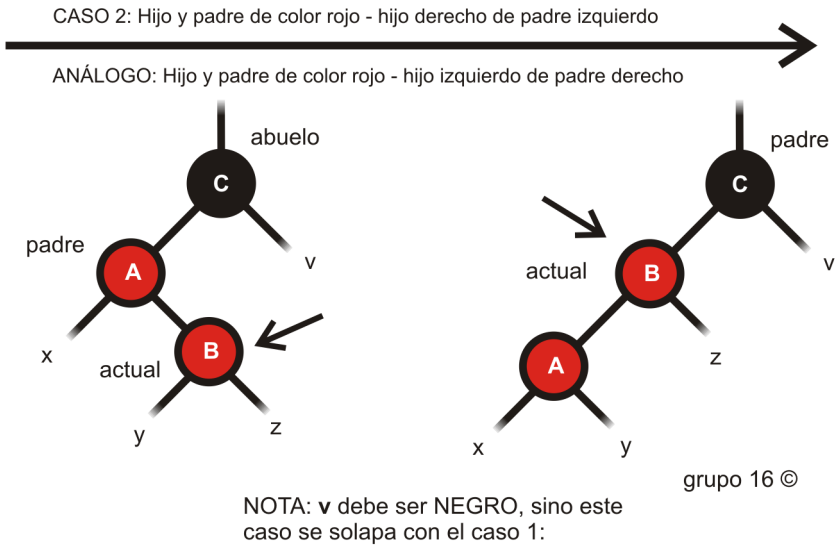
2.1.2. CASO 1



(figura 1.2)

Si A (Actual), B (Padre) y D (tio) son Rojos, estamos en CASO 1.
Pintamos al padre y tio de Negro, y pintamos a C (Abuelo) de Rojo. Como el arbol era un *Árbol **Rojo** y **Negro*** ese abuelo era negro.
Al hacer los cambios, alturas negras no se modifican. Por lo que hasta el abuelo tenemos todo arreglado.

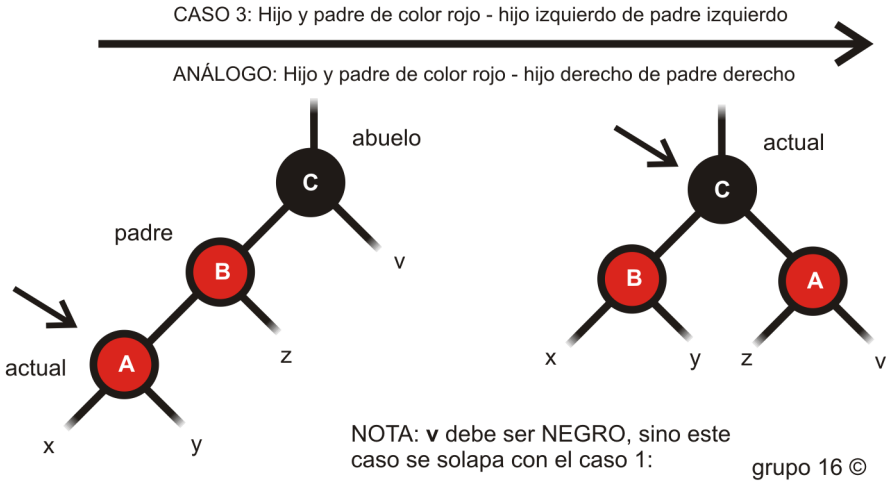
2.1.3. CASO 2



(figura 1.3)

Si B (Actual) y A (padre) son rojos y el tio es negro (o es nil - tambien considerado como regro) y ademas B es hijo derecho de un padre que es hijo izquierdo (o B es hijo izquierdo de un padre que es hijo derecho), estamos en CASO 2
Rotamos el padre a izquierda (o derecha segun el caso), y hemos reducido el CASO 2 o un CASO 3.

2.1.4. CASO 3



(figura 1.4)

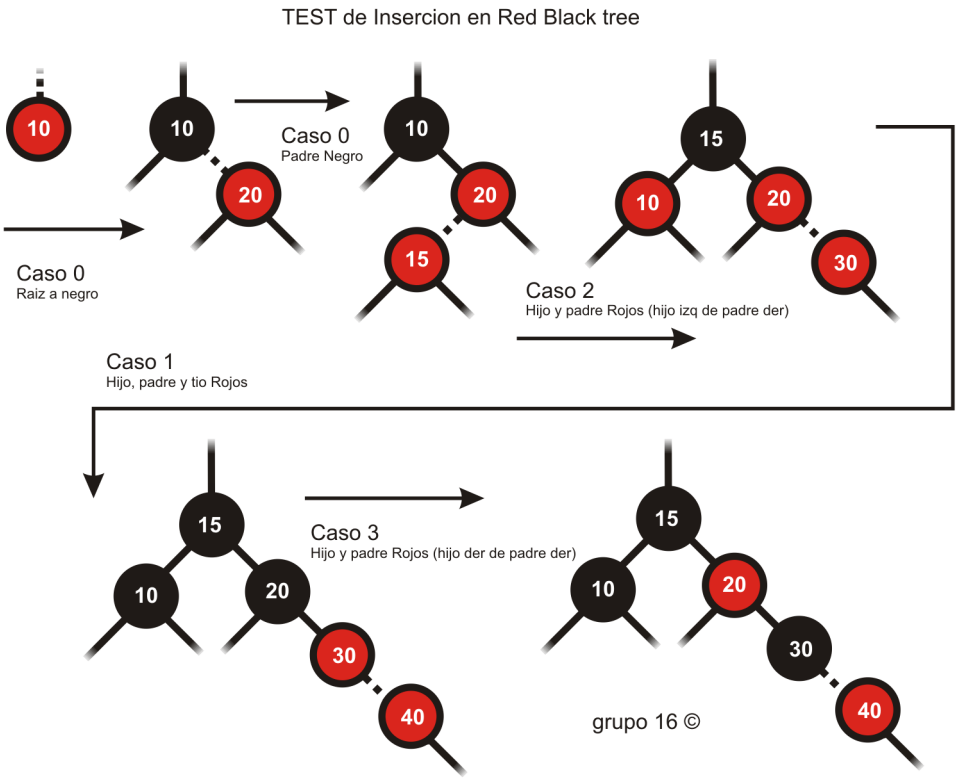
Si A (Actual) y B (padre) son rojos y el tio es negro (o es nil - tambien considerado como regro) y ademas B es hijo derecho de un padre que es hijo derecho (o B es hijo izquierdo de un padre que es hijo izquierdo), estamos en CASO 3
Pintamos a padre negro, al abuelo rojo y y rotamos a derecha (o Izquerda segun el caso) al abuelo
Las alturas negras no se modifican y la posición del abuelo quedaría negra, con ambos hijos rojo.

2.1.5. TEST de Inserciones

Hemos realizado un par de test para "verificar" el correcto fuincionemiento de las inserciones en el *Árbol* ***Rojo*** *y* ***Negro***
Las inserciones fueron las siguientes $[10, 20, 15, 30, 40]$.
De esta manera probamos los casos mas interesantes:

de *ninguno*	a $[10]$	(CASO 0: Insertar raiz)
de $[10]$	a $[10, 20]$	(CASO 0: Insertar con padres negro)
de $[10, 20]$	a $[10, 20, 15]$	(CASO 2)
de $[10, 20, 15]$	a $[10, 20, 15, 30]$	(CASO 1)
de $[10, 20, 15, 30]$	a $[10, 20, 15, 30, 40]$	(CASO 3)

Nótese que en algunas de estas inserciones,
los llamados recursivos, llaman tambien a otros casos de "reacomodacion".
Para correr el test y observar la salida del programa se debe llamar al `main` con parametro `test`



(figura 1.5)

2.2. *Árbol de Intervalos*

3. Ejercicio 2

3.1. insertar en *Árbol Rojo y Negro*

3.2. insertar en *Árbol de Intervalos*

Índice