Auxiliar 9 - Árboles auto-balanceantes

Profesores: Iván Sipirán
Patricio Poblete
Nelson Baloian

Auxiliares: Alonso Almendras, Valentina Aravena
Gabriel Norambuena, Sebastián Acuña
Rodrigo Llull, Lucas Oyarzún

P1. Árbol Rojo-Negro

Un árbol Rojo-Negro es un tipo de árbol de búsqueda binaria auto-balanceante. Esto quiere decir que intenta a toda costa mantener mínima su altura y evitar 'ramas' muy largas. Cumple las siguientes propiedades:

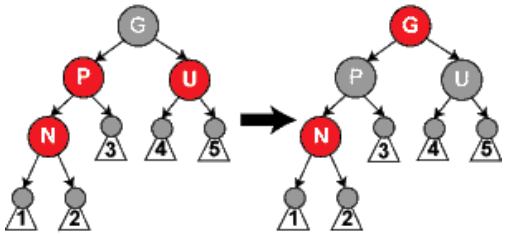
(a) La raíz es negra.

(b) Todo nodo rojo debe tener dos nodos hijos negros.

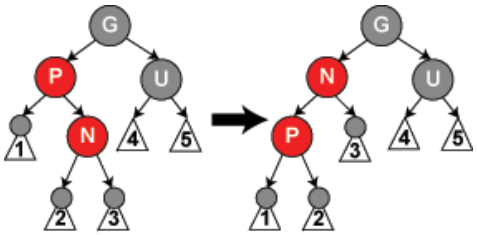
(c) Cada camino desde un nodo dado a sus hojas descendientes contiene el mismo número de nodos negros.

(d) Cada nodo se inserta de la misma forma que en un árbol binario de búsqueda y es de color rojo.

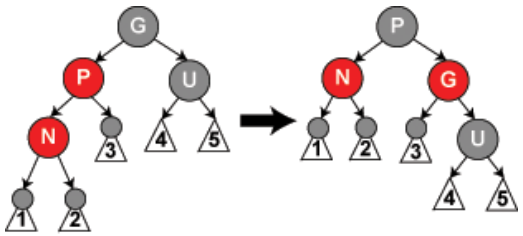
- **Caso 1:** *El nodo insertado es la raíz.*
  La raíz se pinta de negro.
- **Caso 2:** *El padre del nodo insertado es negro.*
  Se deja igual.
- **Caso 3:** *El padre del nodo y el tío son rojos.*
  Se cambia de color al padre y tío, y se revisa al abuelo como si se hubiese insertado.



- **Caso 4:** *El padre es rojo pero el tío es negro. El nodo insertado está hacia adentro.*
  Se hace una rotación simple sobre el padre para poder pasar al caso 5.



- **Caso 5:** *El padre es rojo pero el tío es negro. El nodo insertado está hacia afuera.*
  Se hace una rotación simple sobre el abuelo, y se hacen cambios de color.



Dibuje paso a paso el árbol Rojo-Negro resultante luego de insertar cada uno los siguientes elementos: $\{2, 1, 4, 5, 9, 3, 6, 7\}$, indique además todos los estados intermedios.

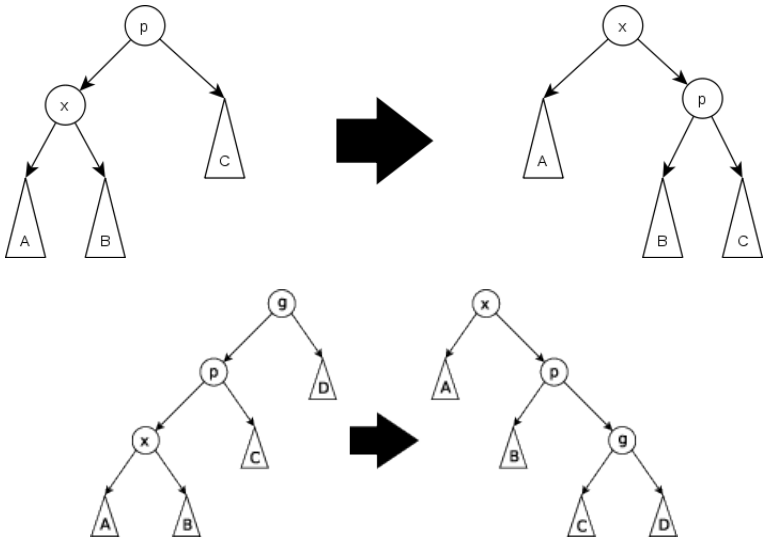
P2. ***Splay trees***

Un *splay tree* (o árbol biselado) es otro tipo de árbol de búsqueda binaria auto-balanceante, cuya propiedad más interesante es que los últimos nodos buscados e insertados están más cerca de la raíz que los nodos más antiguos. En particular, el último nodo insertado/accedido x va a ser la nueva raíz del árbol. Esto se logra mediante una operación llamada *splaying* sobre x.

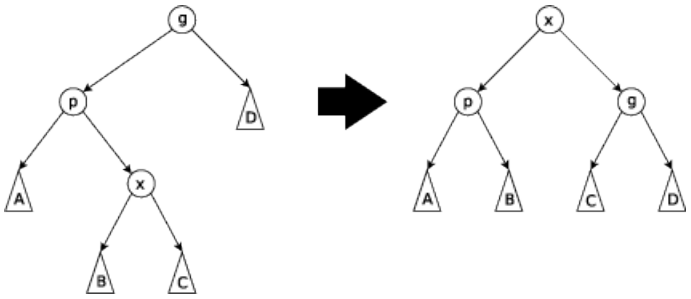
Esta estrategia garantiza que cualquier secuencia de m operaciones en un árbol que llega a tener n elementos, partiendo de un árbol vacío, toma tiempo $O(m \log n)$. Es importante notar que no garantiza que alguna secuencia en particular no va a tomar $O(n)$, sino que el costo acumulado dividido por el número de operaciones da un promedio de $O(\log n)$ por operación. Se dice que una estructura de este tipo es eficiente en el sentido *amortizado*.

Hay tres casos para ejecutar operación de *splaying*:

- **Caso 1:** ***zig*** *El padre de x es la raíz*
  Se hace una rotación simple sobre el padre (al igual que en árboles AVL), dejando a x como la nueva raíz.
- **Caso 2:** ***zig-zig*** *Tanto x como su padre son hijos del mismo lado (izquierdo o derecho) de su abuelo*
  El *zig-zig* se convierte en *zag-zag*, resultando en un árbol con x en la raíz, el padre de x como su hijo, y el abuelo de x como su nieto.



- **Caso 3:** ***zig-zag*** *El nodo x es hijo del lado derecho de su padre, y el padre es hijo izquierdo del abuelo (están en zigzag). El caso zag-zig es análogo*
  Se hace una rotación simple entre x y su padre. Luego, se rota x con su abuelo (ahora el padre) como en el caso 1 *(en la imagen, el árbol (p A B) cumple el rol del árbol A resultante en el diagrama del caso 1)*. Esto es igual a la rotación doble de los árboles AVL.



Si x no se encuentra en el árbol, se realiza la operación *splay(x')*, con x' el último nodo visitado.

(a) Inserte en un *Splay Tree* vacío la siguiente secuencia: 1, 2, 3, 4, 5, 6, 7, 8.

(b) Acceda al nodo con valor 1, realizando el *splaying* correspondiente. Comente acerca de la complejidad de búsqueda en este caso en particular.

(c) Practique accediendo a los nodos con valor: 4, 5. Termine insertando el 9.