OBSERVACIONES DEL LA PRÁCTICA

Tomas Alvarado - 202421564

Pablo de Leon - 202422545

Juan Pablo Camacho - 202110977

Preguntas de análisis

1. ¿Qué diferencia existe entre las alturas de los dos árboles (BST y RBT)?

Un árbol binario de búsqueda (BST) puro no garantiza ninguna cota sobre su altura: depende enteramente del orden de inserción. Con datos aleatorios la altura promedio crece en Θ(log n), pero si las claves llegan ya ordenadas el BST degenera en una lista y su altura alcanza n. En cambio, un Red–Black Tree (RBT) emplea reglas de coloreo y rotaciones que aseguran que, tras cada operación, su altura nunca excede $2 \cdot \log_2(n+1)$. De este modo, mientras un BST puede llegar a desbalancearse gravemente, un RBT permanece siempre cercano al óptimo logarítmico en el peor caso.

2. ¿Percibe alguna diferencia entre la ejecución de los dos árboles (RBT y BST)? ¿Por qué pasa esto?

En la práctica, cuando las inserciones llegan en un orden adverso (por ejemplo, orden creciente), el BST experimenta cada vez más comparaciones y accesos a punteros, lo que ralentiza dramáticamente sus operaciones de búsqueda, inserción y eliminación. Un RBT, por el contrario, realiza un pequeño conjunto de rotaciones y recoloreos (en el orden de 1–3 cambios de puntero por inserción o borrado) para restaurar su balance. Este coste extra constante hace que cada operación en un RBT sea ligeramente más lenta que la operación promedio de un BST balanceado, pero muy superior cuando el BST se descompensa: las latencias se mantienen estables y predecibles.

3. ¿Existe alguna diferencia de complejidad entre los dos árboles (RBT y BST)? Justifique su respuesta.

En un BST aleatorio alcanzamos Θ(log n) en el caso promedio, pero en el peor caso pagamos Θ(n). Todas las operaciones fundamentales (búsqueda, inserción, eliminación) pueden degenerar linealmente. Un RBT añade sobrecarga de coloreo y rotaciones, pero **garantiza** Θ(log n) en el peor, medio y mejor caso. Por tanto, mientras un BST sólo ofrece eficiencia probabilística, el RBT brinda un rendimiento O(log n) determinista, lo cual es fundamental en aplicaciones donde el patrón de datos no puede considerarse aleatorio.

4. ¿Existe alguna manera de cargar los datos en un árbol RBT de tal forma que su funcionamiento mejore? Si es así, mencione cuál.

Aunque el RBT mantiene altura logarítmica con cualquier orden de inserción, podemos reducir el número total de rotaciones si pre procesamos los datos. Una técnica habitual es

barajar (shuffle) la secuencia de claves antes de insertarlas, garantizando diversidad y evitando clustering local de rotaciones. Aún mejor, si disponemos de las claves ya ordenadas, podemos construir de forma “bulk” un BST perfectamente balanceado en $\Theta(n)$ (por ejemplo, dividiendo recursivamente el array ordenado) y luego asignar colores negros y rojos con un pase lineal para obtener un RBT válido. De este modo eliminamos casi por completo el coste de reequilibrio inicial y arranquemos con un árbol óptimo.