B-Tree

• If there is just one item in the node, then the B-Tree is organised as a binary search tree: all items in the left sub-tree must be less than the item in the node, and all items in the right sub-tree must be greater.

• If there are two elements in the node, then:
  • all items in the left sub-tree must be less than the smallest item in the node
  • all items in the middle sub-tree must be between the two items in the node
  • all elements in the right sub-tree must be greater than the largest item in the node

• Also, every non-leaf node must have at least two successors and all leaf nodes must be at the same level.
Red-Black Tree

• A red-black tree is another variation of a binary search tree that is slightly more efficient (and complicated) than B-trees and AVL trees.

• A red-black tree is a binary tree where each node is coloured either red or black such that the colouring satisfies:
  • the root property — the root is black
  • the external property — every external node is black
  • the internal property — the children of a red node are black
  • the depth property — every external node has the same number of black ancestors.
Red-Black tree
Red-Black tree: Height

• A subtree rooted at node \( x \) has at least \( 2^{bh(x)} - 1 \) internal nodes
  • \( bh(x) \) = the number of black nodes (not counting \( x \) if it is black) from \( x \) to any leaf in the subtree (called the black-height).

• **Proof by Induction** (on the height of \( x \))

• **Basis Step:**
  • If the height of \( x \) is 0, then \( x \) must be a leaf (NIL), and the subtree rooted at \( x \) indeed contains at least \( 2^{bh(x)} - 1 = 2^0 - 1 = 0 \) internal nodes.

• **Inductive Step:**
  • Consider a node \( x \) that has positive height
  • \( x \) is an internal node with two children
  • So, each child has a black-height of either \( bh(x) \) or \( bh(x) - 1 \), depending on whether its color is red or black, respectively
Red-Black tree: Height (cont.)

• The height of a child of \( x \) is less than the height of \( x \) itself;
• Apply the inductive hypothesis to conclude that each child has at least \( 2^{bh(x)} - 1 \) internal nodes.
• Thus, the subtree rooted at \( x \) contains at least \( (2^{bh(x)} - 1) + (2^{bh(x)} - 1) + 1 = 2^{bh(x)} - 1 \) internal nodes, which proves the claim. (proof)

• Further, let \( h \) be the height of the tree.
• According to property 3 (internal), at least half the nodes on any simple path from the root to a leaf, not including the root, must be black.
• Consequently, the black-height of the root must be at least \( h/2 \)
Red-Black tree: Height (cont.)

• $n \geq 2^{h/2} - 1$
• So $h \leq 2 \log(n + 1)$
Red-Black tree: Insert

• Use the BST insert algorithm to add K to the tree
• Colour the node containing K red (why?)
• Restore red-black tree properties (if necessary)
  • Recoloring
  • Restructuring (rotation)
• Colour the root node black.
Red-Black tree: Insert (cont.)

- Insert($x$)  

1) If $y$ is black; we are done.

2) If $y$ is red  
   - (we need to look at the grandparent $w$ and uncle $u$)

2a) If $u$ is red  
   - Just recolor
   
   - Climb up and recolor the root (if necessary)
Red-Black tree: Insert (cont.)

• 2b) If $u$ is black or NIL

• Rotations + Recoloring
  • root of the restructured subtree is colored black and its children are colored red.
Red-Black tree: Insert (cont.)
RB-tree: An example

• Insert (4)

• Insert (7)

• Insert (12)
  • Rotate left
RB-tree: An example (cont.)

- Insert (15)
  - Recolor

- Insert (3)
- Insert (5)
RB-tree: An example (cont.)

• Insert(14)
  • Needs 2 rotations

• Insert(18)
  • Recolor
RB-tree: An example (cont.)

- Insert (16)
RB-tree: An example (cont.)

- Insert (17)
  - Recolor

- New problem
- Rotation
Red-Black tree: Delete

1) Perform standard BST delete
   • Delete a node which is either leaf or has only one child.
   • Let $v$ be the node deleted and $u$ be the child that replaces $v$.
   • Restore the RB properties

2) If either $u$ or $v$ is red
   • we mark the replaced child as black

Ref: http://www.geeksforgeeks.org/red-black-tree-set-3-delete-2/
Red-Black tree: Delete (cont.)

3) If Both u and v are Black.
   • 3.1) Color u as **double** black
   • 3.2) Do following while the current node u is double black or it is not root. Let sibling of node be **s**.

3.2a) If sibling s is black and at least one of sibling’s children is **red**, perform rotation(s). Let the red child of s be **r**.
Red-Black tree: Delete (cont.)

- **3.2a(i)** Left Left Case (s is left child of its parent and r is left child of s or both children of s are red)
- **3.2a(ii)** Left Right Case (s is left child of its parent and r is right child).
- **3.2a(iii)** Right Right Case (s is right child of its parent and r is right child of s or both children of s are red)
Red-Black tree: Delete (cont.)

• **3.2a(iv)** Right Left Case (s is right child of its parent and r is left child of s)
Red-Black tree: Delete (cont.)

• 3.2b): If sibling is black and its both children are black, perform recoloring, and recur for the parent if parent is black.

![Diagram of Red-Black Tree Operations](image-url)
Red-Black tree: Delete (cont.)

- **3.2(c): If sibling is red**, perform a rotation to move old sibling up, recolor the old sibling and parent. The new sibling is always black.

- **3.2c(i) Left Case** (s is left child of its parent). This is mirror of right right case shown in below diagram. We right rotate the parent p.

- **3.2c(ii) Right Case** (s is right child of its parent). We left rotate the parent p.
Red-Black tree: Delete (cont.)

• 3.3 If u is root, make it single black and return (Black height of complete tree reduces by 1).