Dictionaries

Sets and Multisets; Opers: (Ins., Del., Mem.)

- Sequential sorted or unsorted lists.
- Linked sorted or unsorted lists.
- **Tries** and **Hash Tables**.
- Binary Search Trees.

Priority Queues

Multisets; Opers: (Ins., Del-Max. (or Del-Min.))

- Sequential sorted or unsorted lists.
- Linked sorted or unsorted lists.
- Tries and Hash Tables.
- Binary Search Trees.
- **Heaps**

Min-Max Priority Queues

Matching Heaps, Min-Max Heaps and Deaps are for the case of Multisets and the operations are Ins., Del-Max. and Del-Min.

Generalized Dictionaries

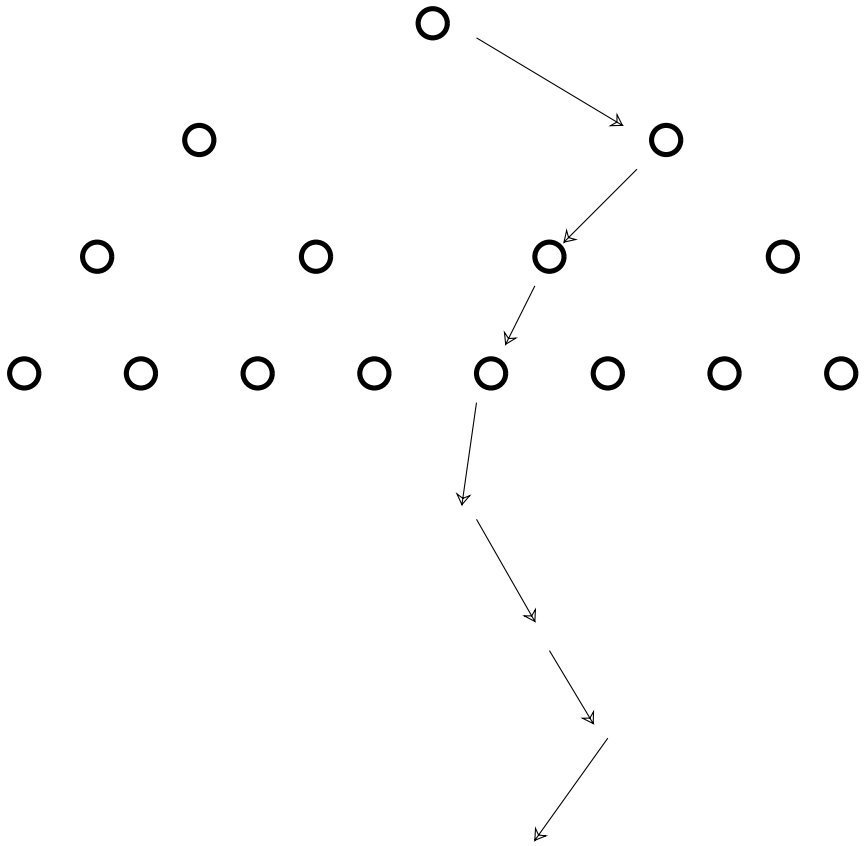
Set and Multisets; Opers: Insert, Delete, Del-Min, Del-Max, Concatenate, Split, Find kth Smallest element.

- Sequential sorted or unsorted lists, Linked sorted or unsorted lists, Tries and Hash Tables, Binary Search Trees.
- **Red-Black Trees**

Binary Search Trees

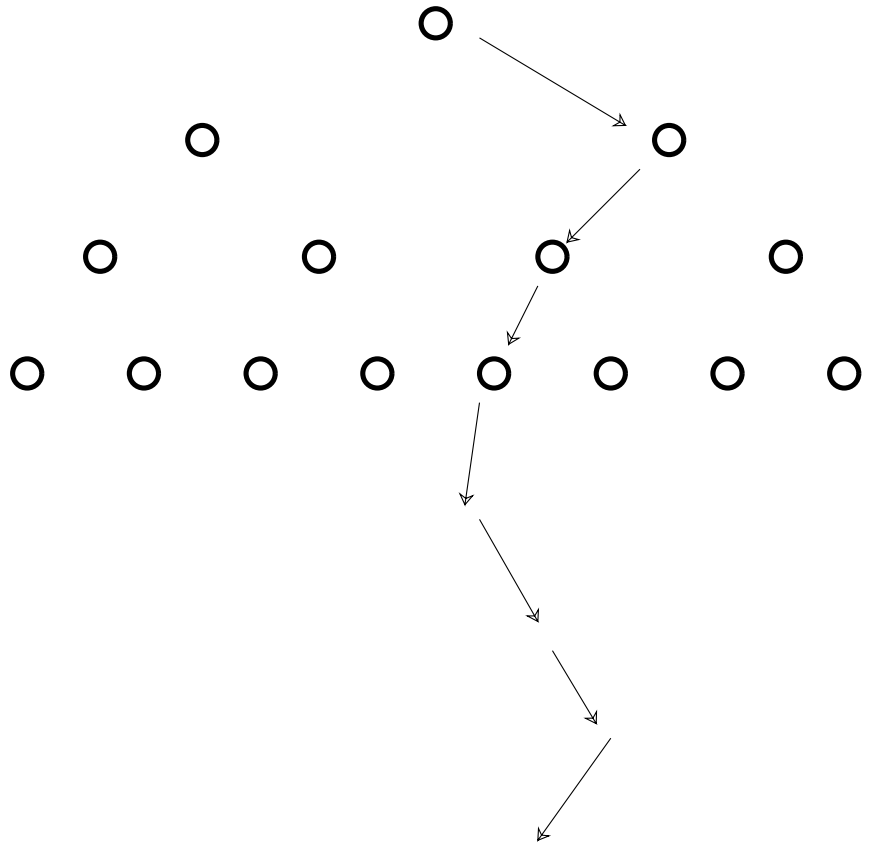
- Work for Sets and Multisets (we use sets here).
- For each node x, the values in the left subtree of the tree rooted at x are less than the value stored at x, and the values in the right subtree of the tree rooted at x are greater than the value at stored at x.
- Insert, Delete, Delete-Min, Delete-Max, Concatenate, and split take time $O(h)$, where h is the height of the tree. But h may be as large as n, the number of nodes in the tree.

Membership in BSTree

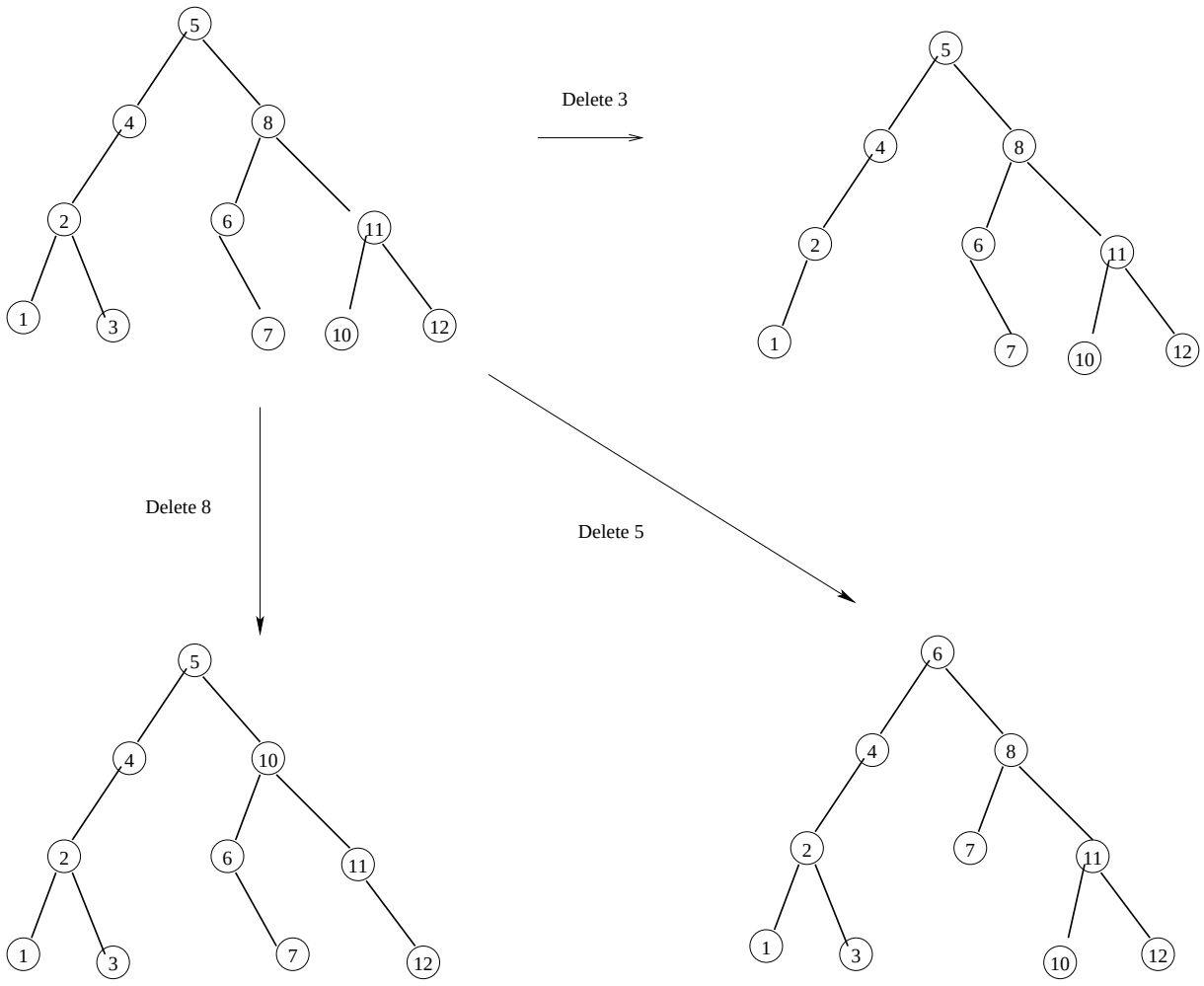


Search for the element until you find it or you reach a null pointer. Time complexity is $O(h)$, where h is the height of the tree.

Insert in BSTree



Delete in BSTree



- Let x be the element to be deleted.
- Search for x in the BST.
- If x is not in the tree (you end at a null pointer), then just return
- Else, let $node(x)$ be the node where x is

located.

- If $node(x)$ does not have a right child, then make the parent of $node(x)$ (or root $node(x)$ is the root) point to left child of $node(x)$ instead of pointing to $node(x)$. Return to the storage pool $node(x)$.
- Else ($node(x)$ has a right child). Let y be the node with smallest value larger than x. Make the parent of y point to the right child of y instead of pointing to y. Assign to $node(x)$ the value stored at node y. Return to the storage pool node y.

Delete takes time $O(h)$,
where h is the height of the tree.

Red-Black Trees

Red-Black Trees: BSTs with $O(\log n)$ height.

Definition: Every Red-Black Tree is a binary search tree with the following properties.

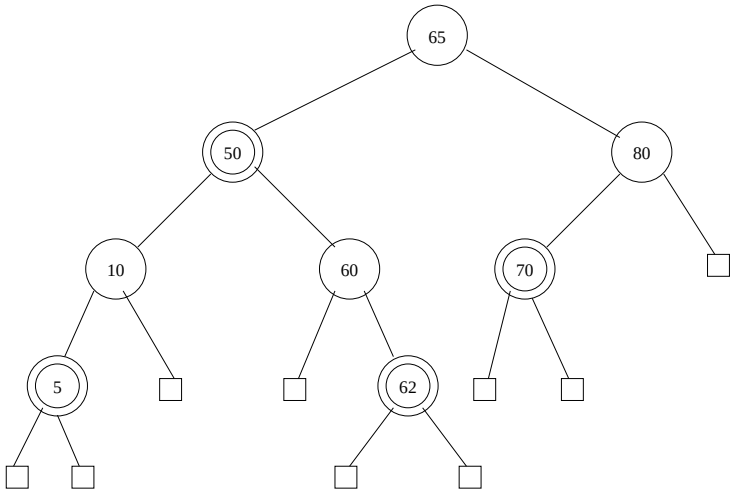
(1) Every node is either colored red or black

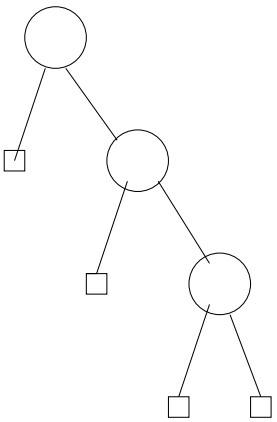
(2) The root is colored black

(3) If a node is colored red then its children must be colored black.

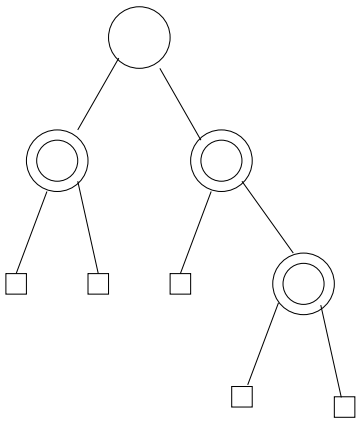
(4) For every node x, every path from node x to a NULL pointer must visit the same number of black nodes.

Note that the lines and boxes represent null pointers. You may think about these nodes as external nodes and the other (regular) nodes as internal nodes. Al external nodes are black nodes.





Not RBT



Not RBT

Height of a Red-Black Tree

The **rank** of a node in a red-black tree is the number of black nodes on any path from the node to any NULL pointer in its subtree.

Lemma 1: Let P and Q be two paths from a node x to a NULL pointer in a red-black tree. Then $length(P) \leq 2length(Q)$, where length is the number of nodes in the path from the node to the given NULL pointer.

Proof: Follows from the fact that P and Q visit the same number of black nodes (4), between every pair of black nodes there is at most one red node (3) and the root node is a black node.

The leftmost path in the above figure has length 4, and the length of the rightmost path is 2.

Lemma 2: Let h be the height of a red-black tree, let n be the number of internal nodes in the tree, and let r be the rank of the root. Then

(a) $h \leq 2r$.

(b) $n \geq 2^r - 1$.

(c) $h \leq 2log_2(n+1)$.

Proof: Item (a) follows from the fact that all paths from the root to a NULL pointer have at most $2r$ nodes.

Inductively one can show that the number of internal nodes with rank $1 \leq j \leq r$ is at least 2^{r-j}. Therefore, $n \geq \sum_{j=1}^{r} 2^{j-1} = 2^r - 1$, and (b) holds.

Since (b) holds we know that $n \geq 2^r - 1$. Substituting (a) we know $n \geq 2^{h/2} - 1$. Or equivalently $n + 1 \geq 2^{h/2}$. Taking logs on both sides and passing the two to the other side we get (c).

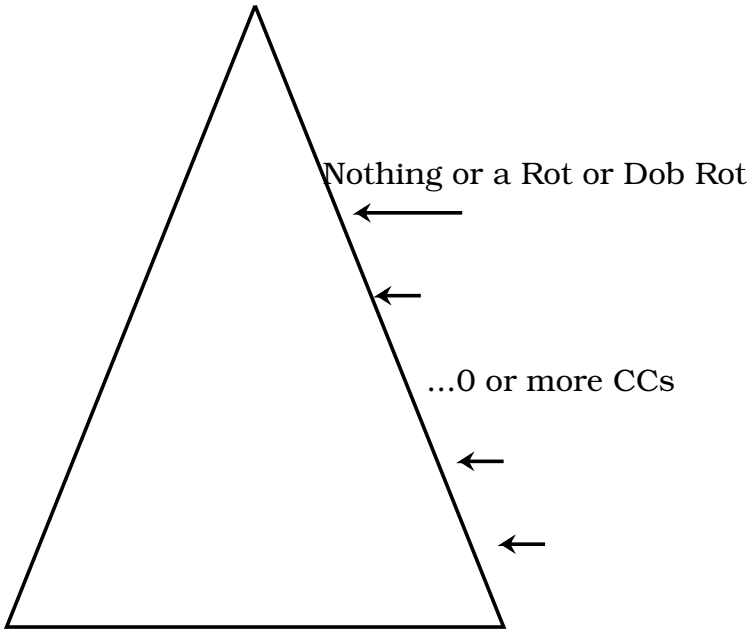
Membership

Procedure Membership is the same as for BSTrees which takes $O(h)$. Since $h = O(log\ n)$, it then follows that membership takes $O(log\ n)$.

Insert

To show that Insert can be performed in $O(log\ n)$ time, we only need to show that rebalancing takes $O(h)$ time.

General Idea



Deletion Transformation

Transform the problem of Deletion to the problem of deletion of a node with two Null Pointers.