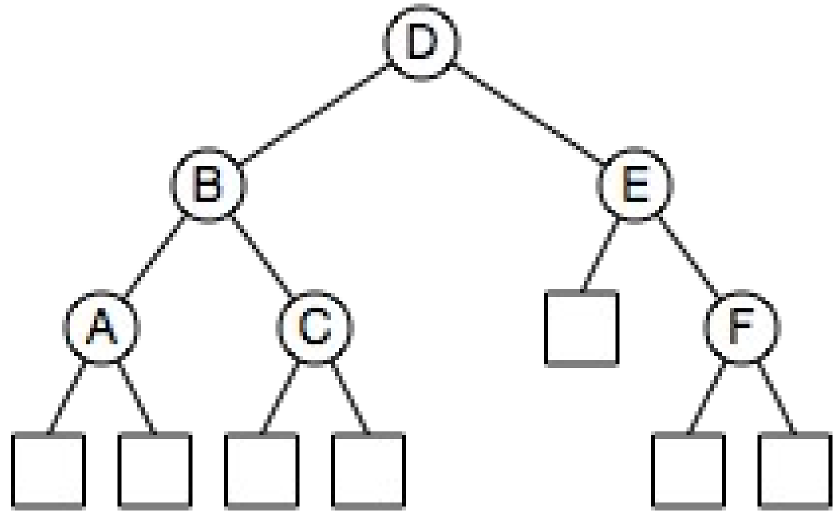
Binary SearchTrees

Binary Tree

- Hierarchical structure of linked nodes—parents and children.
- Binary tree: two children at most.
- Multiple levels: grandchildren, grandparents, descendants, ancestors.
- Top node is called the root.

Example Search Tree



- Trees are made up of smaller trees — subtrees. Node B is the root of a subtree; so is A.
- Internal nodes (circles) contain keys and values; keys are identifiers that are searched for.
- External nodes mark the bottom of the tree; they have no children and no content.
- Left subtree’s keys are less than the root’s key, right subtree’s keys are greater.

Operations

- Insert: add a node
- Find: locate a node with a specific key
- Erase: remove a node

- Find is used by:
    - Insert, to determine the position of the new node
    - Erase, to determine the position of the node to remove

Find

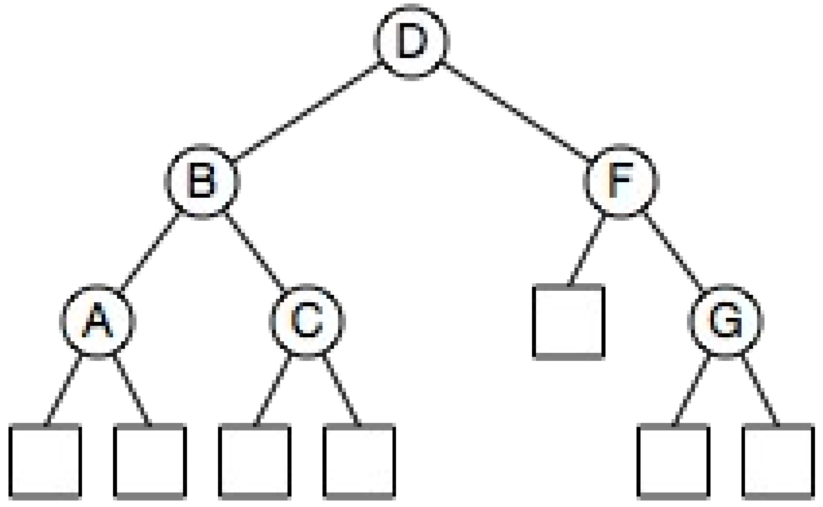
- Navigate left and right down the tree
- At each node, compare node key to search key
  - If they match, search terminates successfully
  - If search key is less, go down to left subtree
  - If search key is greater, go down to right subtree
- If no match has been found when bottom of tree is reached (no left or right subtree to descend to), search terminates unsuccessfully.

Insertion

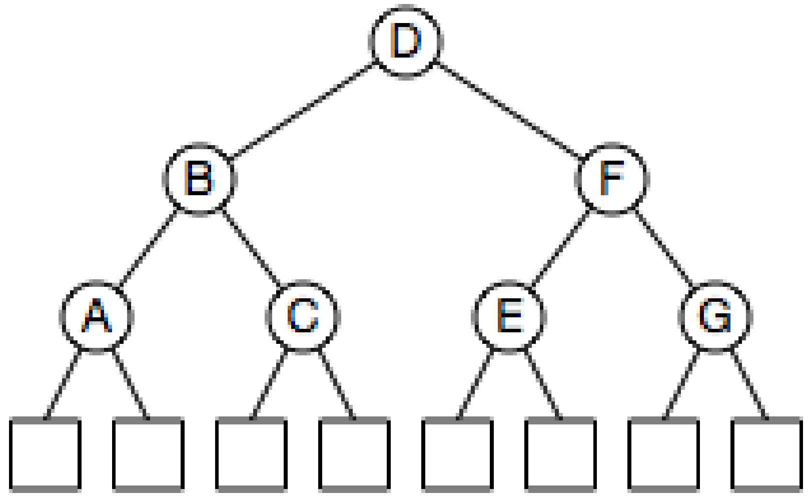
- Two possibilities:
    - Node with key already exists in tree
    - No node with key exists in tree
- A Find for a key that doesn’t exist terminates at the external node at the position where the internal node would have been, and where the new node should be.
- A Find for a key that already exists causes links to be followed down to the right until an external node is reached—the end of the chain of duplicate keys.

Insert: key not already in tree

Before node E is inserted

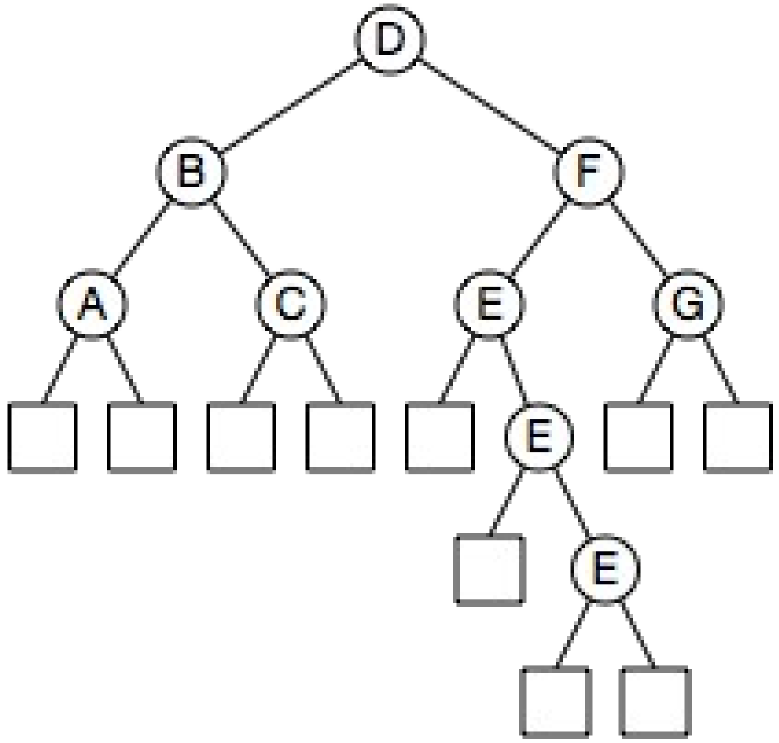


After node E is inserted

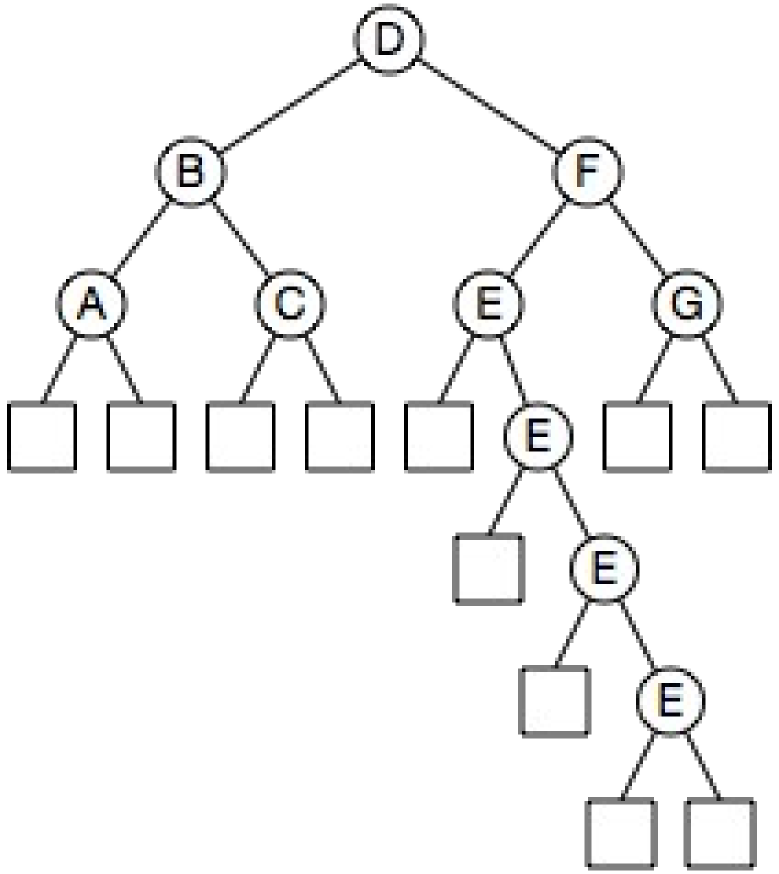


Insert: duplicate keys already in tree

Before another node E is inserted



After one more node E is inserted

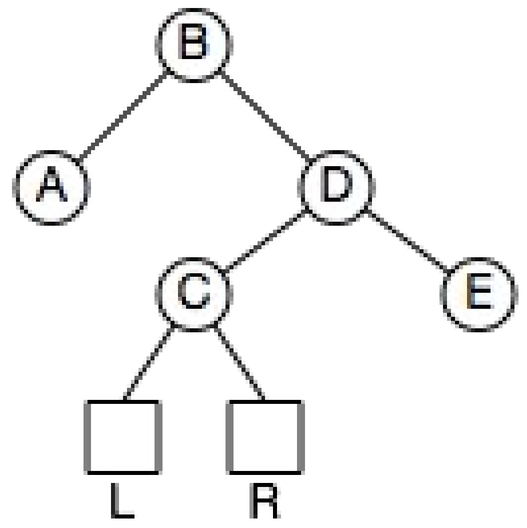


Erasure

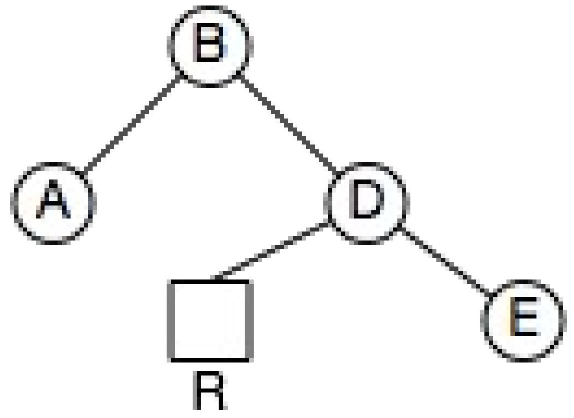
- The most complicated operation.
- Removing a node leaves a hole that must be filled.
- Filling the hole involves promoting another node, internal or external.
- Three different situations, depending on number of node’s external children: two, one, or none.

Two External Children

Erase node C

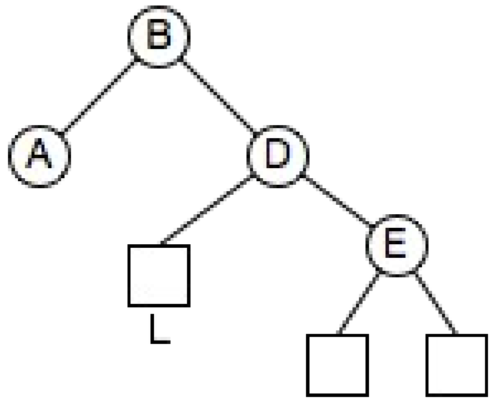


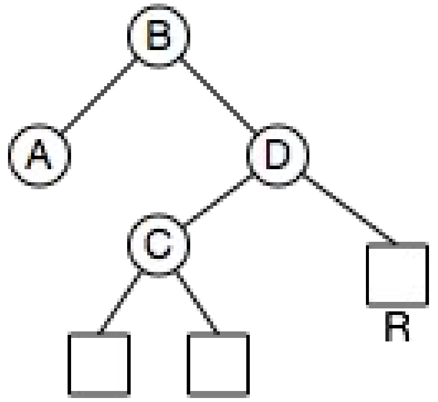
- Promote node C’s right child into node C’s position
- Delete node C and its left child, resulting in:



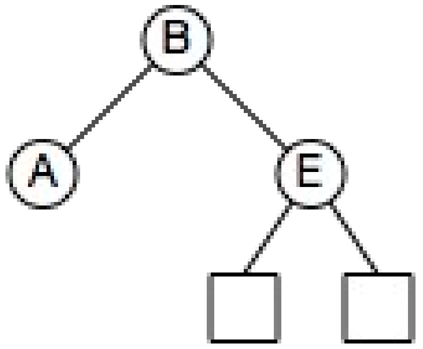
One External Child

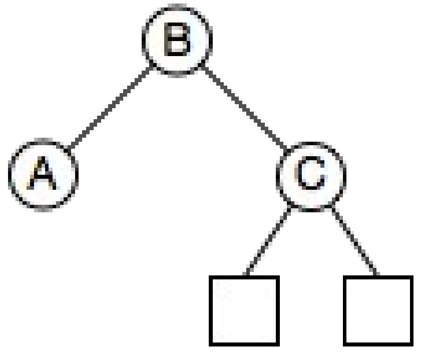
- Two complementary variations





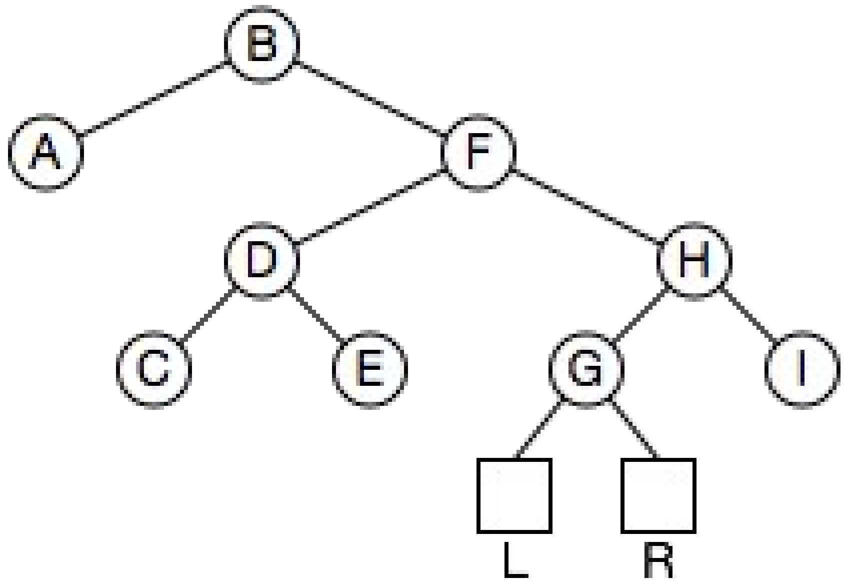
- Erase node D by promoting its internal child





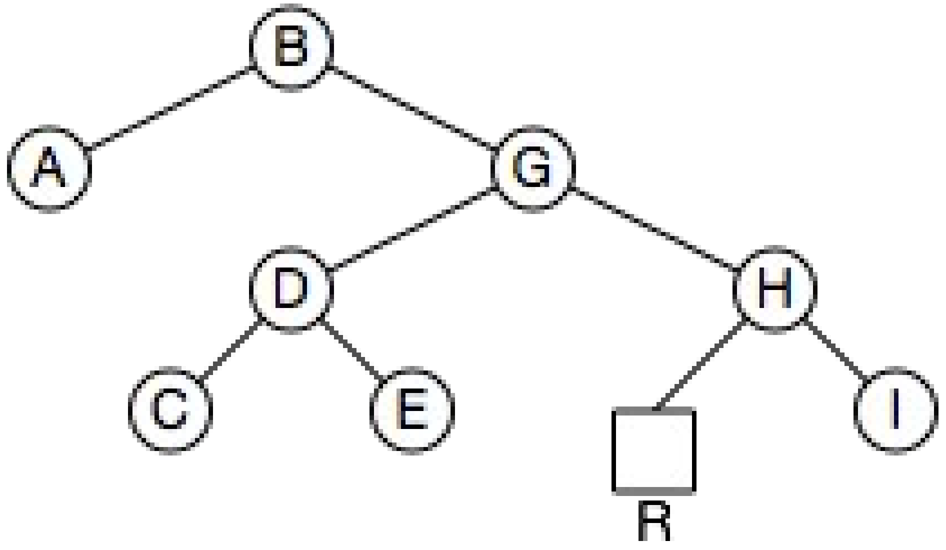
No External Children

- Most complicated case; node F has two internal children.



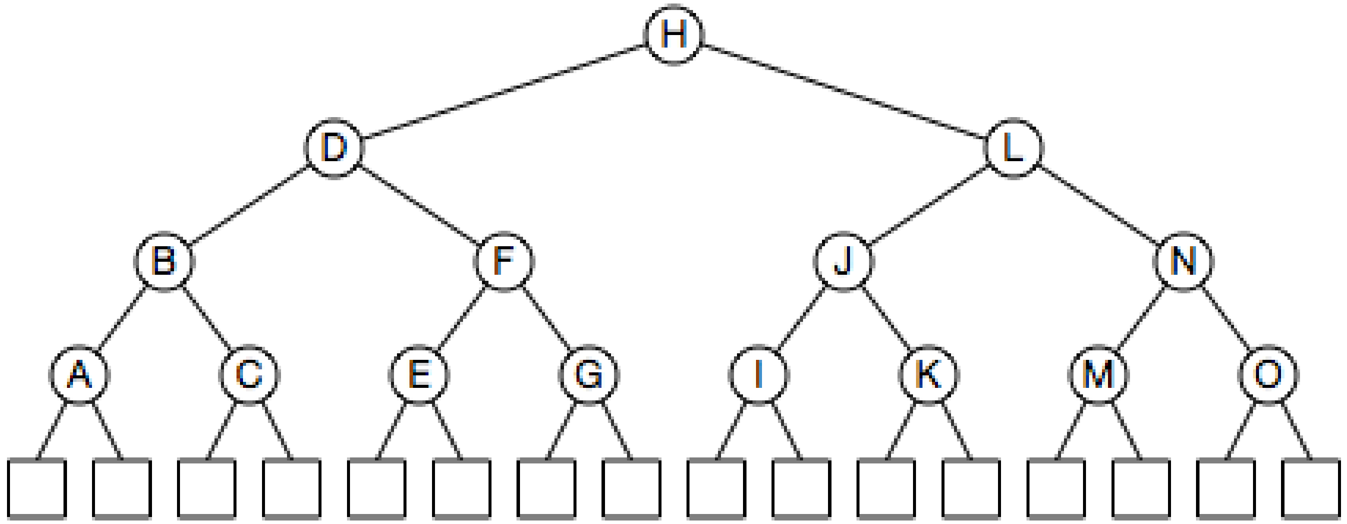
- There's no external child, no single internal child.
- It should be replaced by its logical successor, node G.
- Find F's successor by going right, then left, left,…

- Node G won’t be linked into node F’s position; its key and value will be copied.
- Old node G’s right child will be promoted to fill G’s position.
- Old node G and its left child will be deleted.



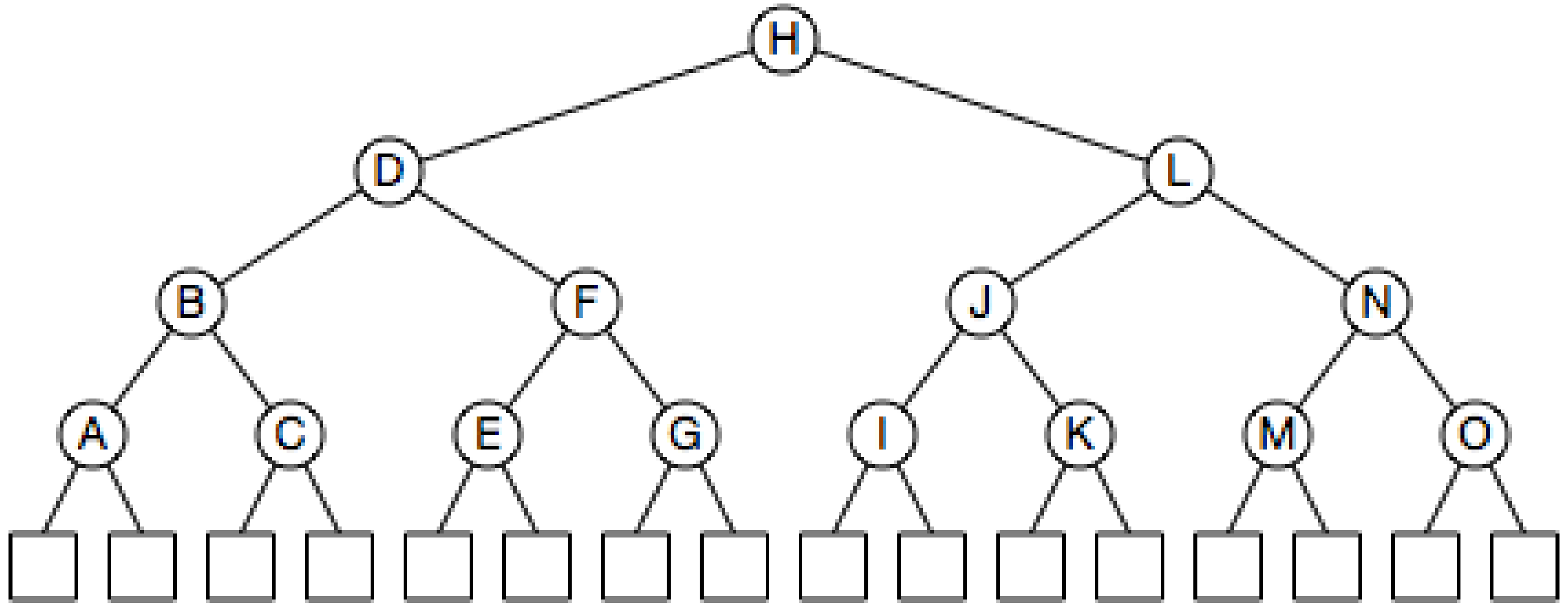
Iterator

- Performs inorder traversal, visits nodes in key order.
- Navigates down, left, right, up, down again



Finding a Node’s Successor

- If there is a right subtree, it’s at the lower left corner.
- If there is no right subtree, it’s at an ancestor—the ancestor of the left subtree that the node is in.
- Node B’s successor is node C; node C’s successor is node D.

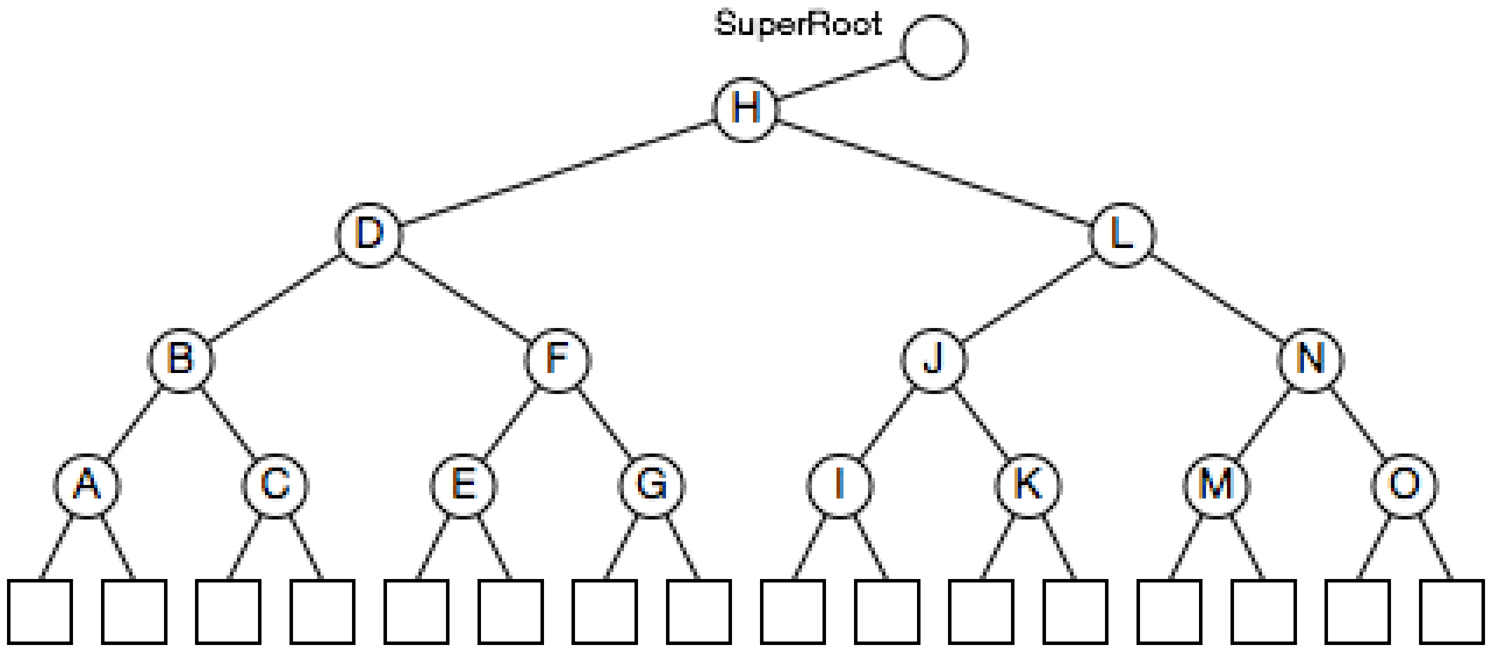


Related Functions

- Begin(). Returns an iterator to the first node.
- End(). Returns an iterator to the end of the tree, which is actually one beyond the last node.
- “Half-open Range” viewpoint—reaching the end of the range means advancing beyond the last element.
- Loop: for(i = tree.Begin(); i != tree.End(); ++i)
- The typical array loop—for (i = 0; i < size; ++i)—uses the same viewpoint:
    - If size is 10, i goes from 0 to 9; loop stops when i is 10, one position beyond the end of the array.

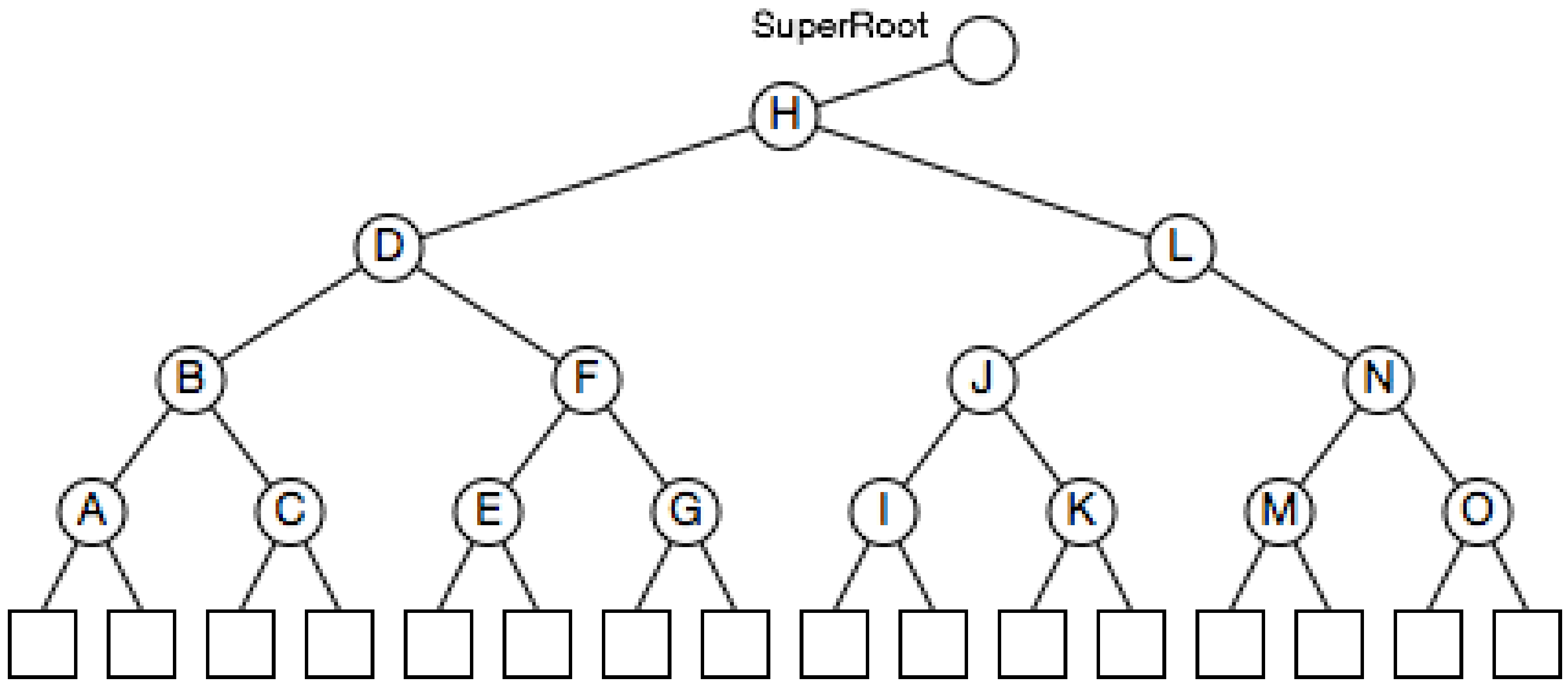
Root and SuperRoot

- The tree needs a node that’s logically beyond its last node.
- The root will get a parent—a SuperRoot.



Reaching the End of the Tree

- Node O’s successor is the ancestor of the left subtree that node O is in.
- That ancestor is the SuperRoot.



C++ Implementation of a Binary Search Tree

See implementation examples