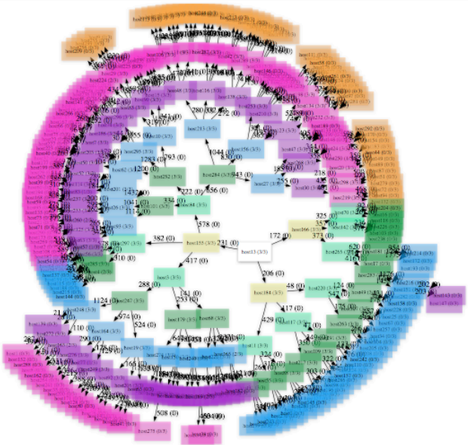
MIAMI UNIVERSITY

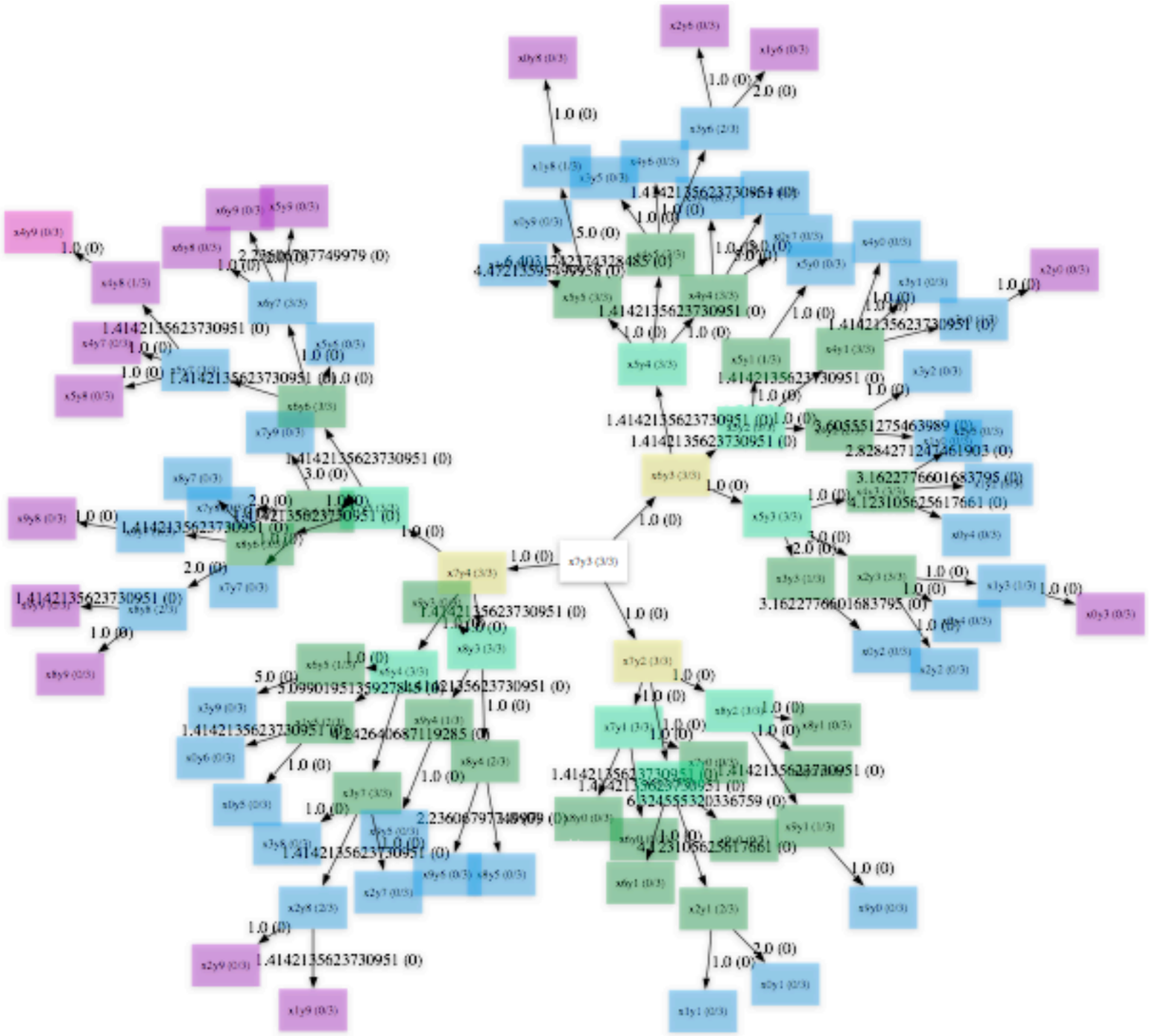
Data Abstraction & Data Structures
Trees

Miami University
CSA 274 - Spring 2007
Mike Helmick



School of Engineering & Applied Science

Trees



MIAMI UNIVERSITY

Previously...

Previously...

- We have discusses linear data structures

Previously...

- We have discusses linear data structures
- Every element has a predecessor and a successor

Previously...

- We have discusses linear data structures
- Every element has a predecessor and a successor
- Traversing the list is a O(n) operation

Previously...

- We have discusses linear data structures
- Every element has a predecessor and a successor
- Traversing the list is a O(n) operation
- The order is well known / well defined

Trees

Trees

- Still one predecessor

Trees

- Still one predecessor
- but

Trees

- Still one predecessor
- but
    - multiple successors

Trees

Trees

- Trees in nature

Trees

- Trees in nature
    - roots

Trees

- Trees in nature
  - roots
  - trunks

Trees

- Trees in nature
  - roots
  - trunks
  - branches

Trees

- Trees in nature
  - roots
  - trunks
  - branches
  - leaves

Trees

- Trees in nature
  - roots
  - trunks
  - branches
  - leaves

Trees

Trees

- In Computer Science

Trees

- In Computer Science
- Have a hierarchical structure

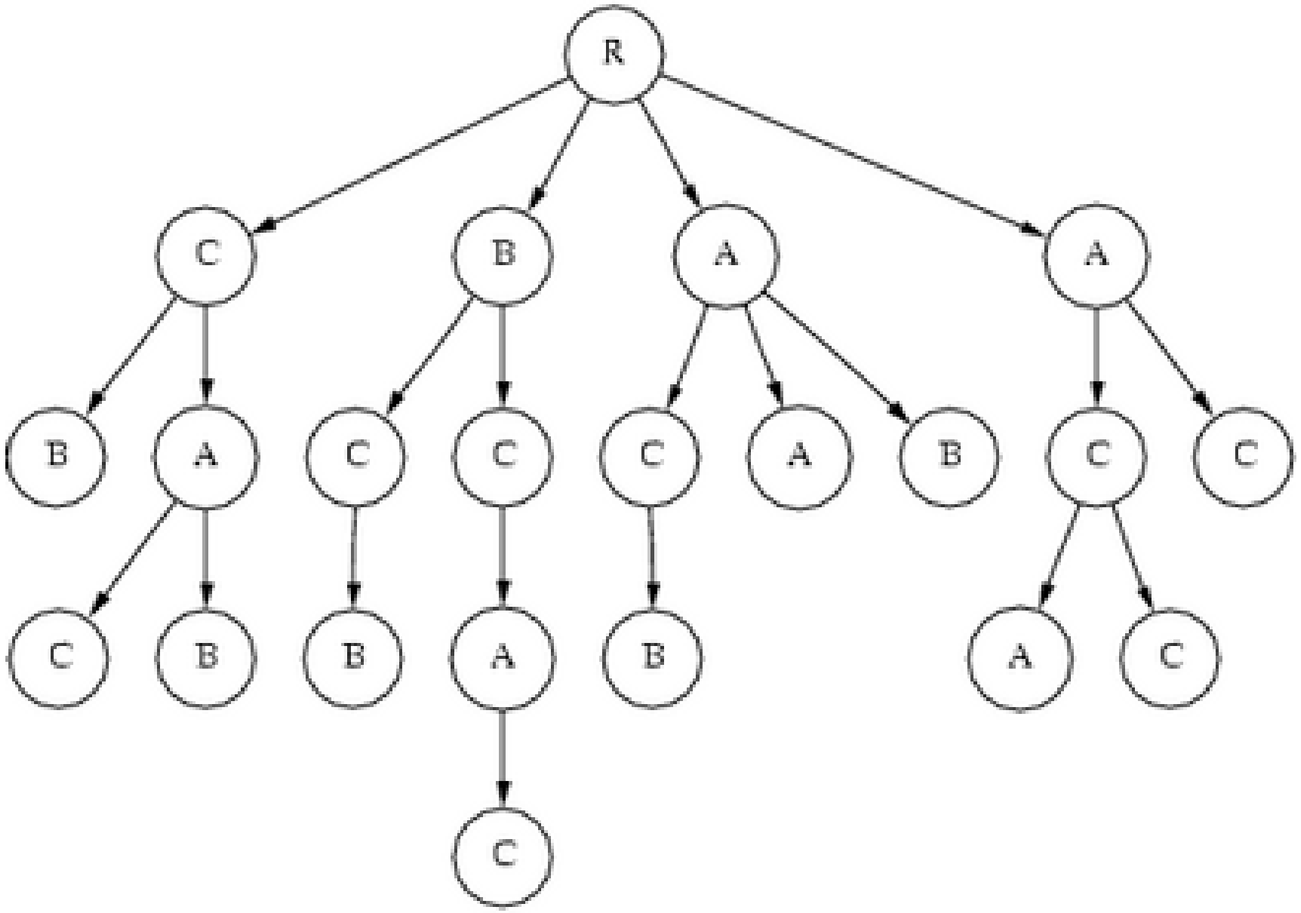
Trees

- In Computer Science
- Have a hierarchical structure
- Drawn with the root at the top

Trees

- In Computer Science
- Have a hierarchical structure
- Drawn with the root at the top
    - and the leaves at the bottom

Example Tree



Recursion

Recursion

- Trees are a naturally recursive structure

Recursion

- Trees are a naturally recursive structure
    - similar to linked list

Recursion

- Trees are a naturally recursive structure
    - similar to linked list
- For this reason - many tree operations are written as recursive functions

Binary Tree

Binary Tree

- The Binary Tree is a special case of the tree structure

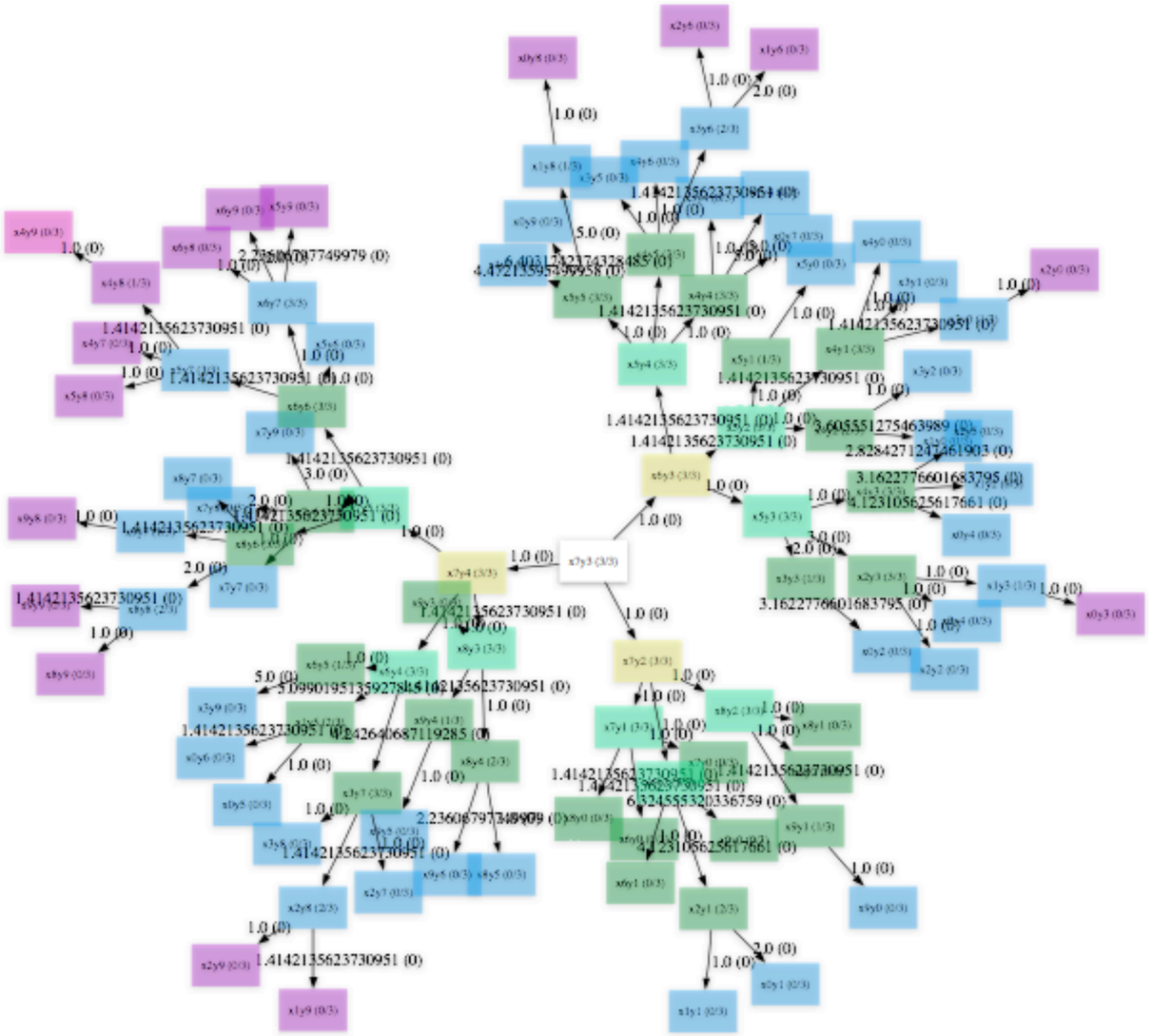
Binary Tree

- The Binary Tree is a special case of the tree structure
    - Every node has at most 2 children

Binary Tree

- The Binary Tree is a special case of the tree structure
    - Every node has at most 2 children
        - 0, 1, or 2 children

Terminology



MIAMI UNIVERSITY

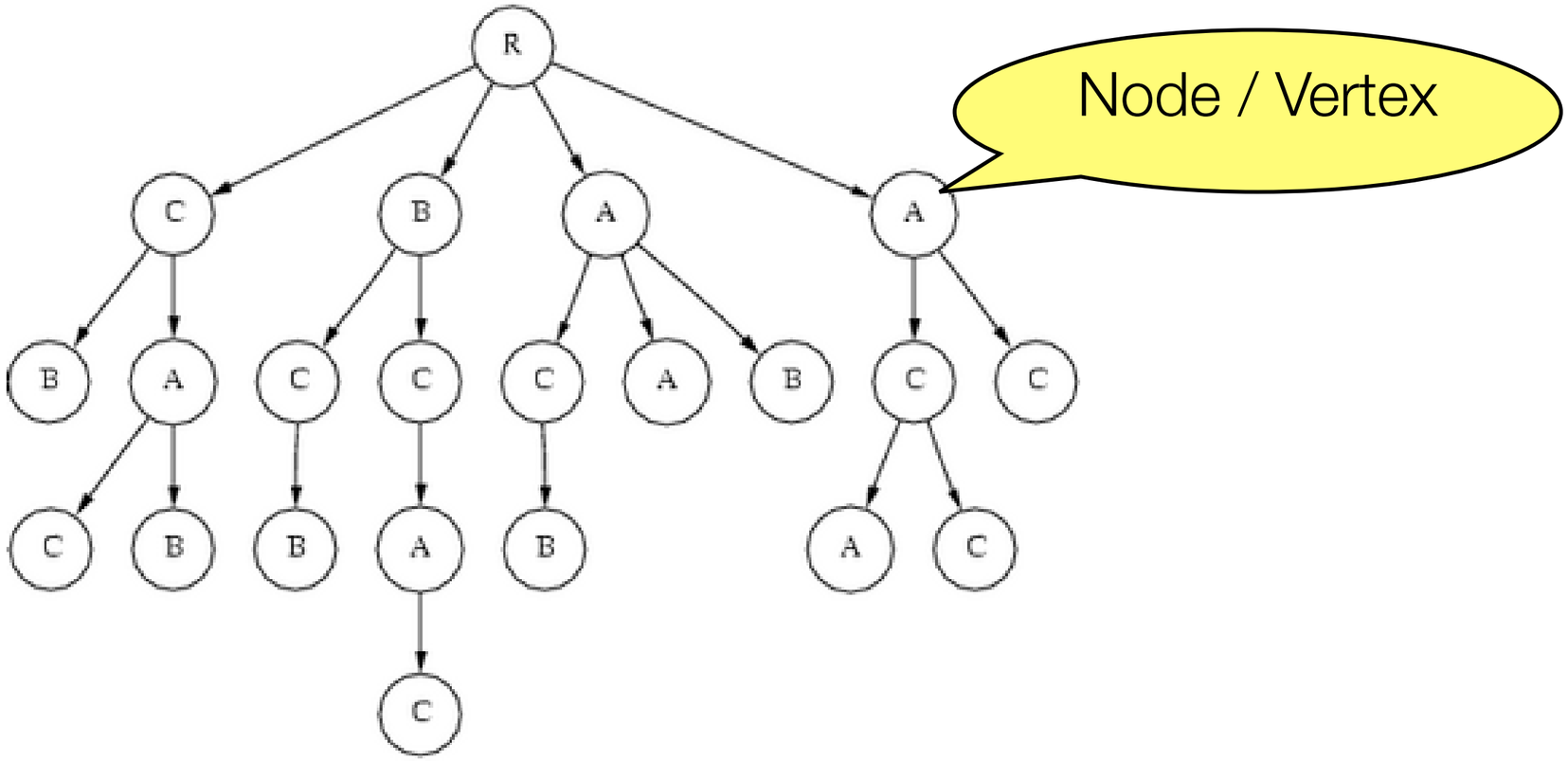
Entries

Entries

- Each entry in the tree is an element or, more commonly, a node

Entries

- Each entry in the tree is an element or, more commonly, a node



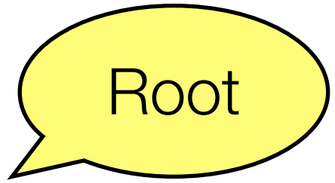
Node / Vertex

MIAMI UNIVERSITY

Root

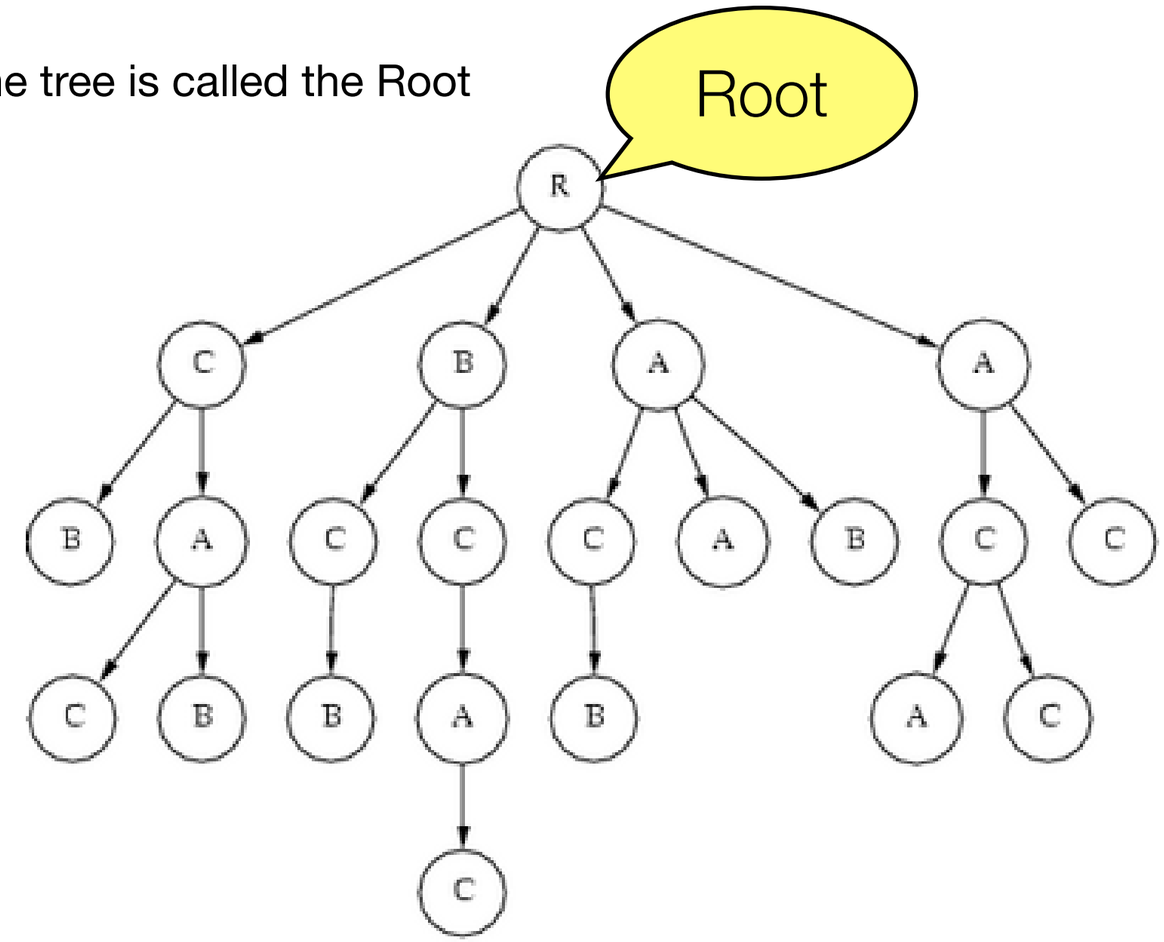
Root

- The node at the top of the tree is called the Root



Root

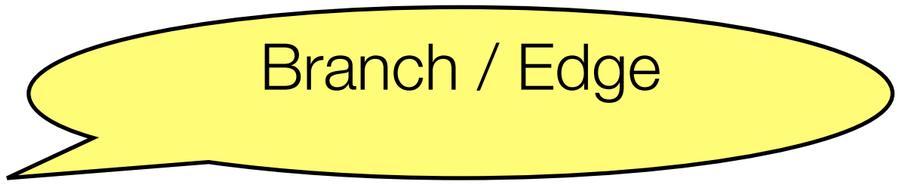
- The node at the top of the tree is called the Root



Branches

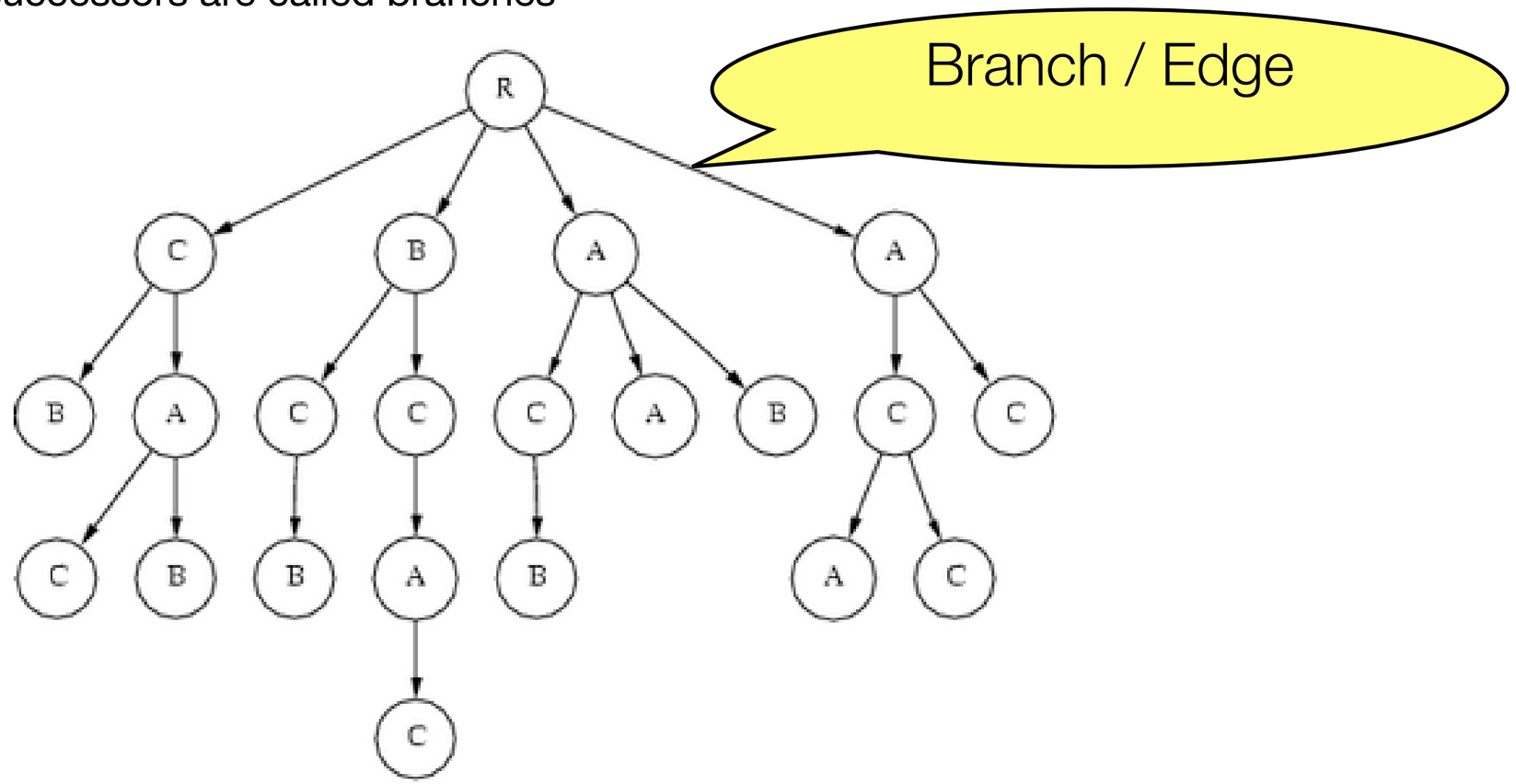
Branches

- Links from a node to its successors are called branches



Branches

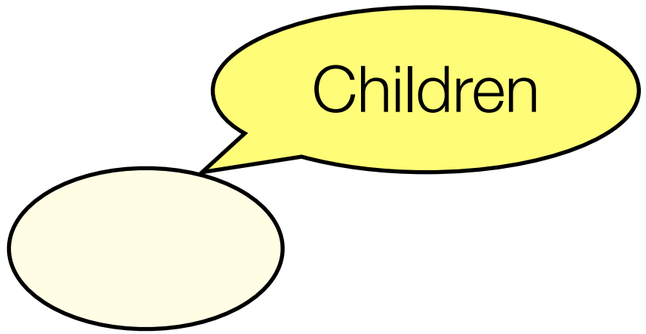
- Links from a node to its successors are called branches



Children

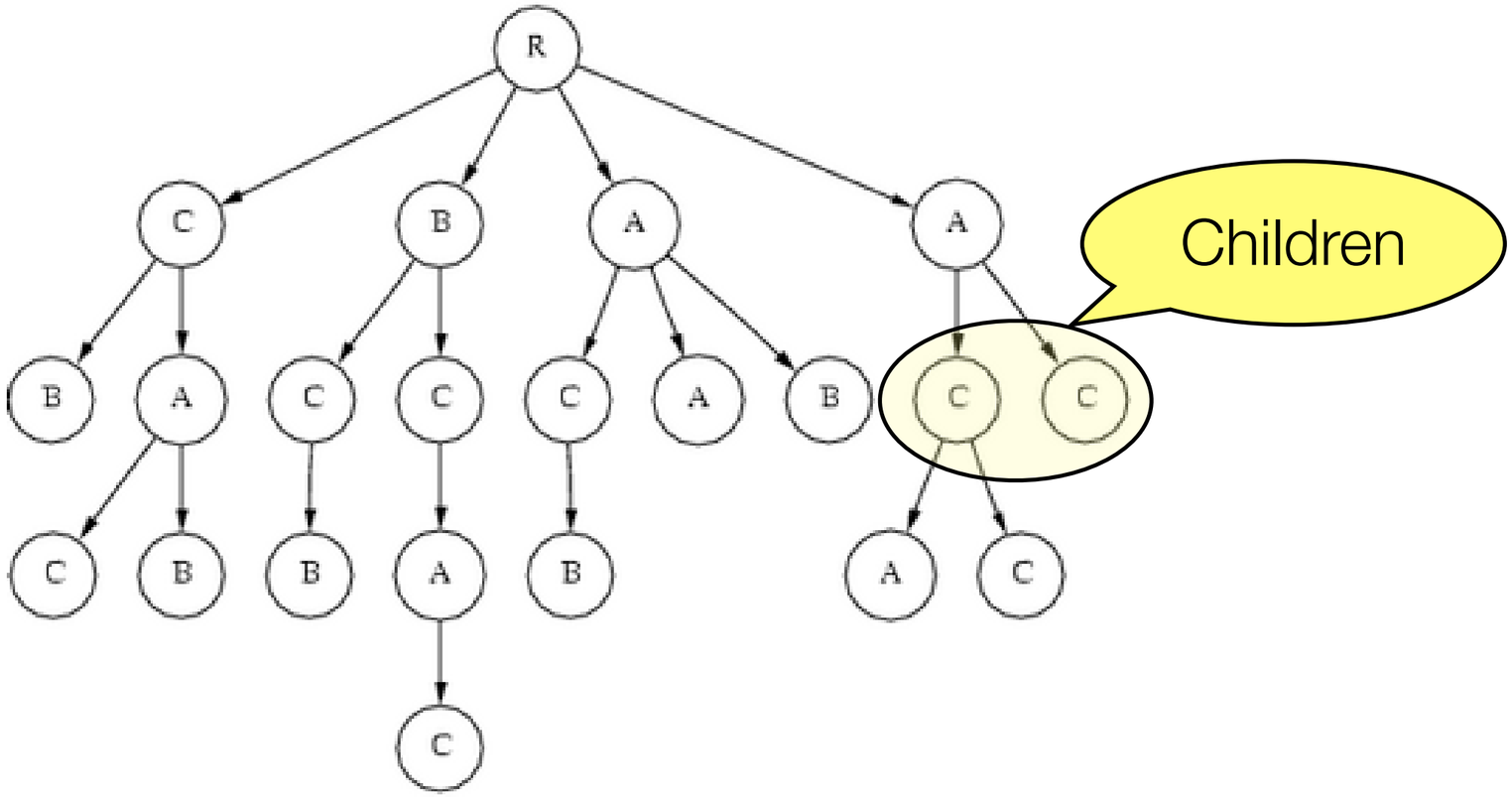
Children

- A node's successors are called it's children (a child node)



Children

- A node's successors are called it's children (a child node)



Children

MIAMI UNIVERSITY

Parents

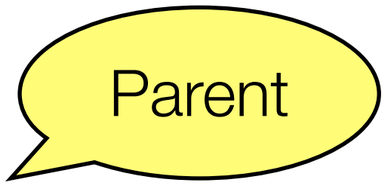
Parents

- A node's predecessor is called it's parent

Parents

- A node's predecessor is called it's parent

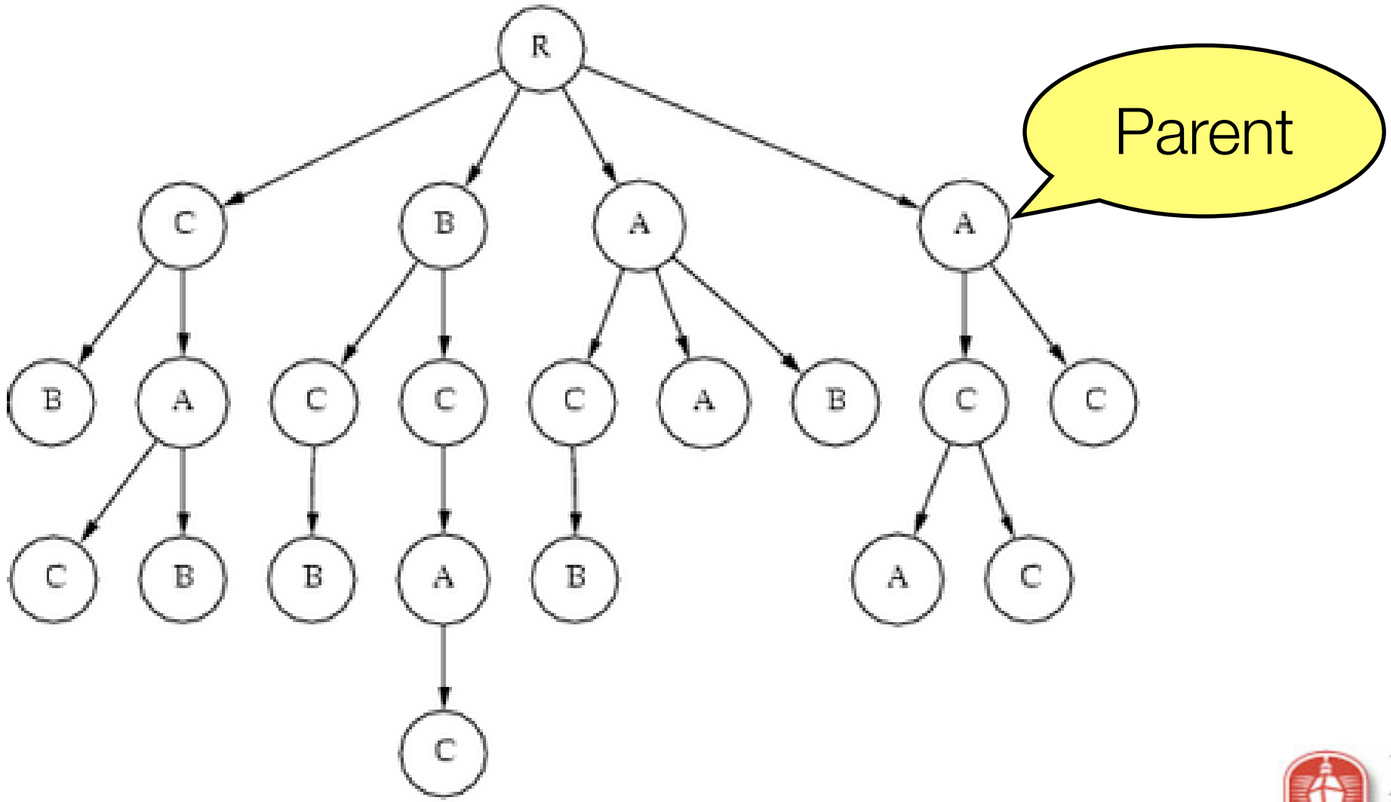
- In a Tree, a node can only have 1 parent, and always has a parent unless it is the root



Parents

- A node's predecessor is called it's parent

- In a Tree, a node can only have 1 parent, and always has a parent unless it is the root



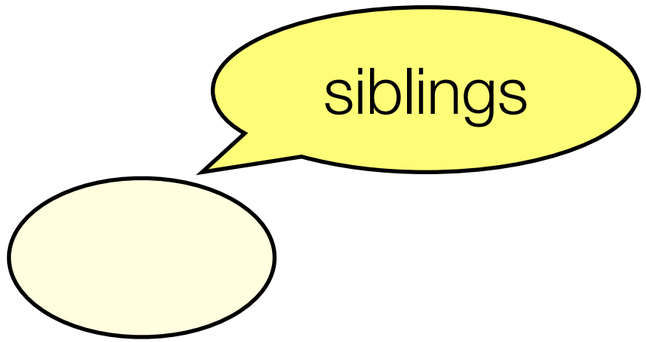
Parent

MIAMI UNIVERSITY

Siblings

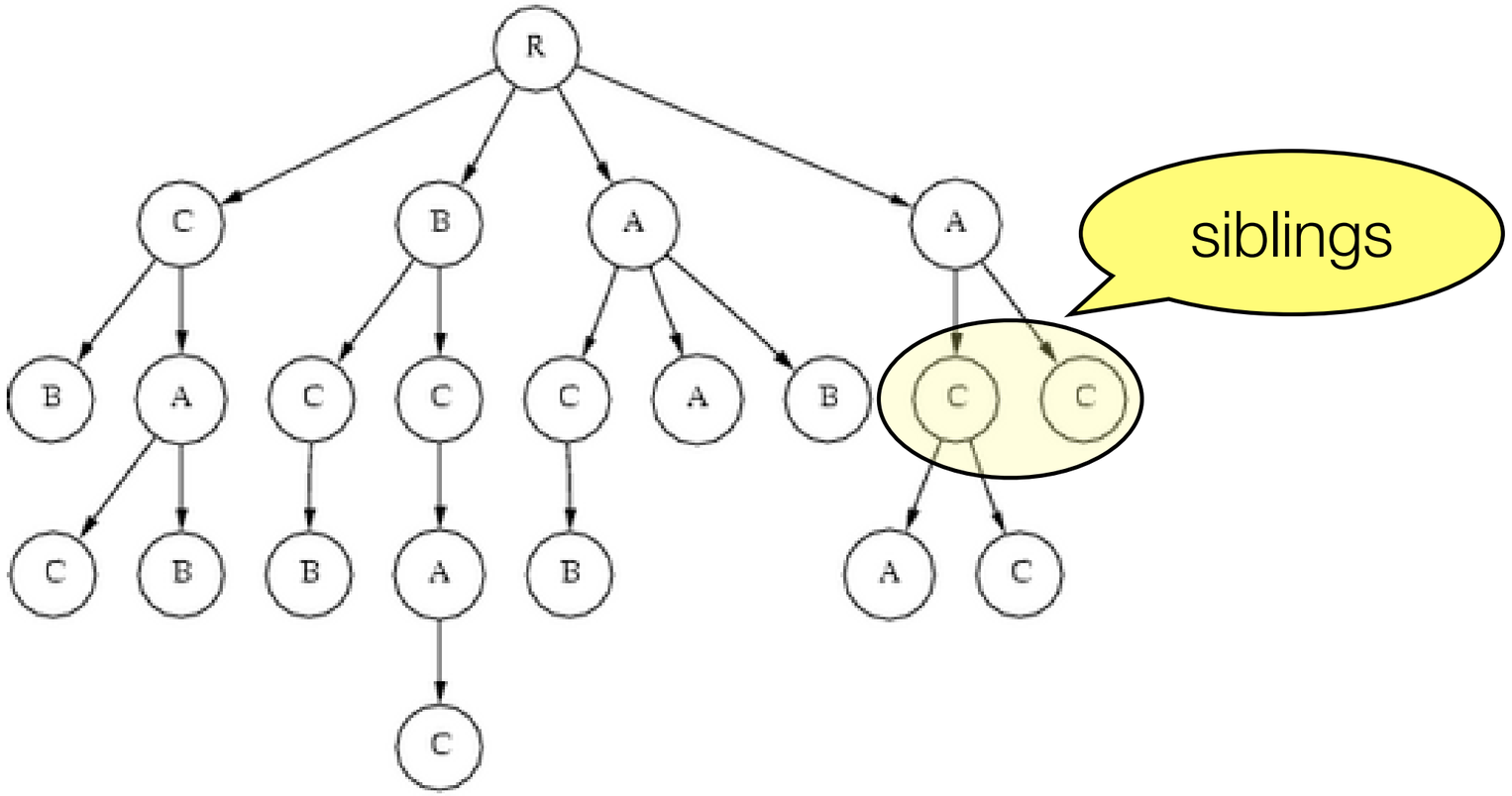
Siblings

- Nodes that have the same parent are siblings



Siblings

- Nodes that have the same parent are siblings

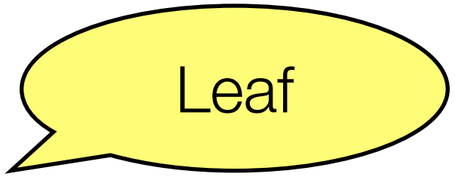


siblings

Leaf

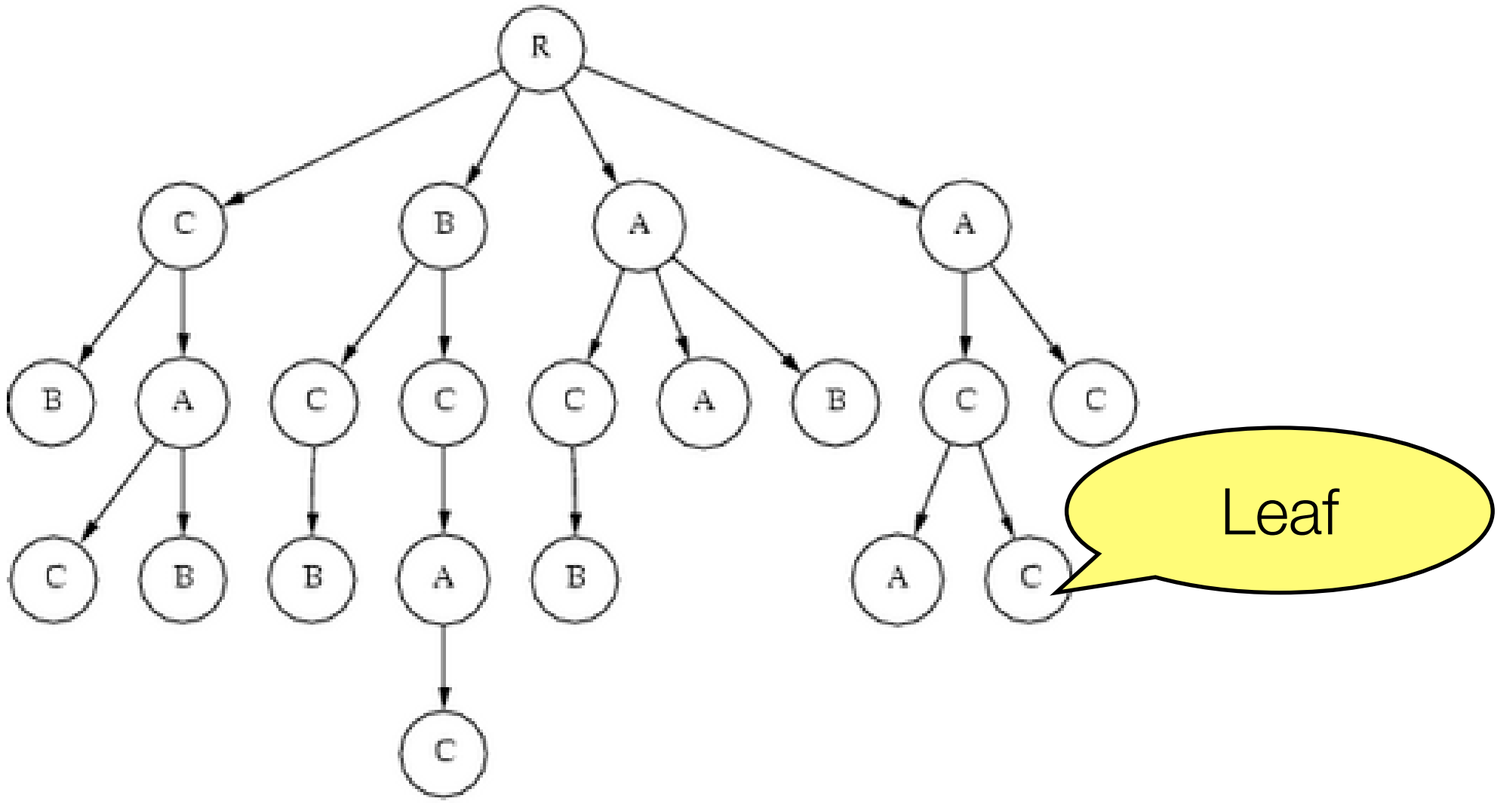
Leaf

- A node with no successors, is called a leaf node, or external node



Leaf

- A node with no successors, is called a leaf node, or external node

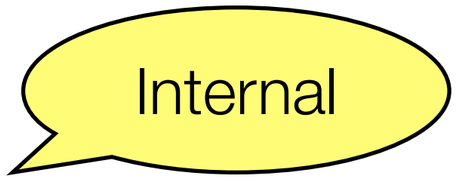


Leaf

Internal Nodes

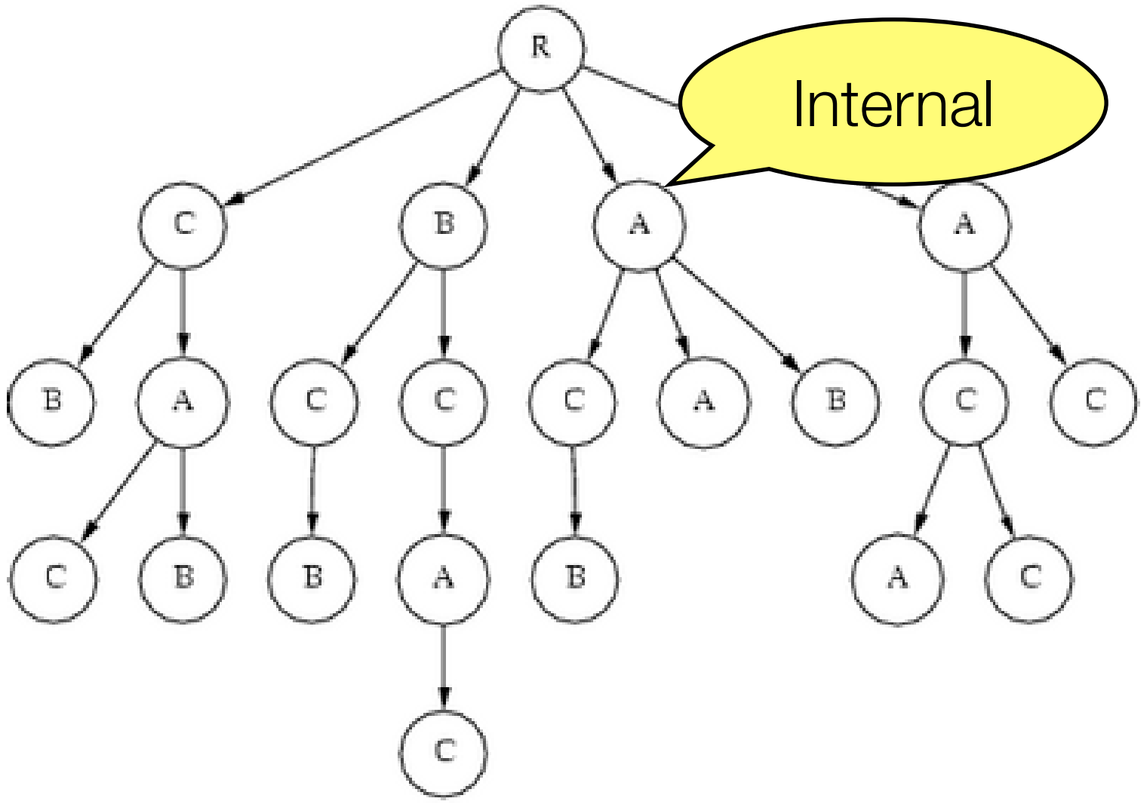
Internal Nodes

- Non-leaf nodes are refereed to as internal nodes



Internal Nodes

- Non-leaf nodes are refereed to as internal nodes



Generations

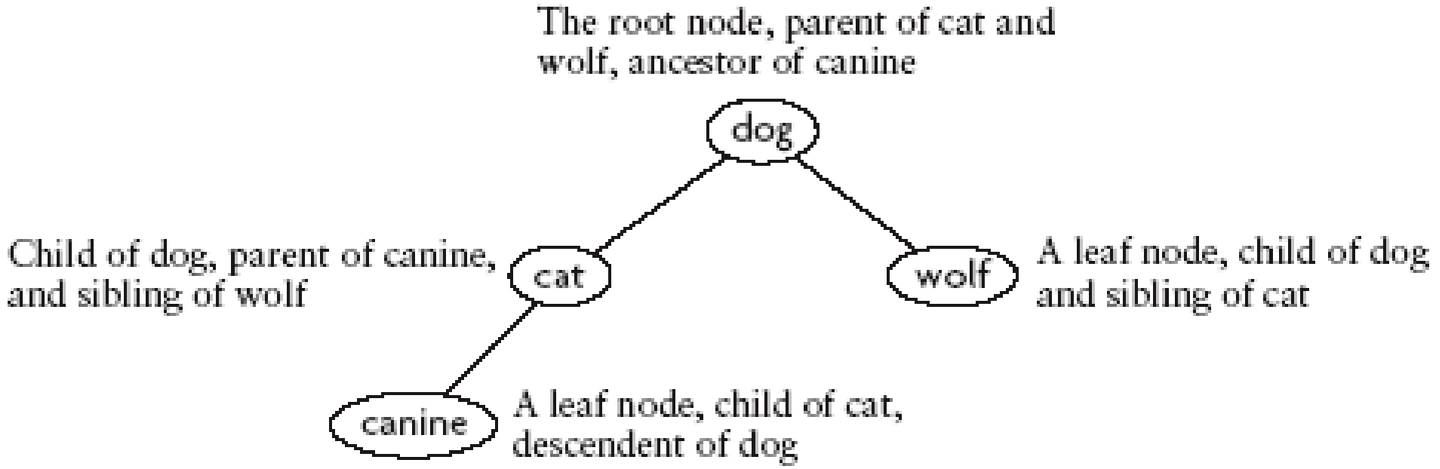
Generations

- Ancestor

Generations

- Ancestor
- Descendant

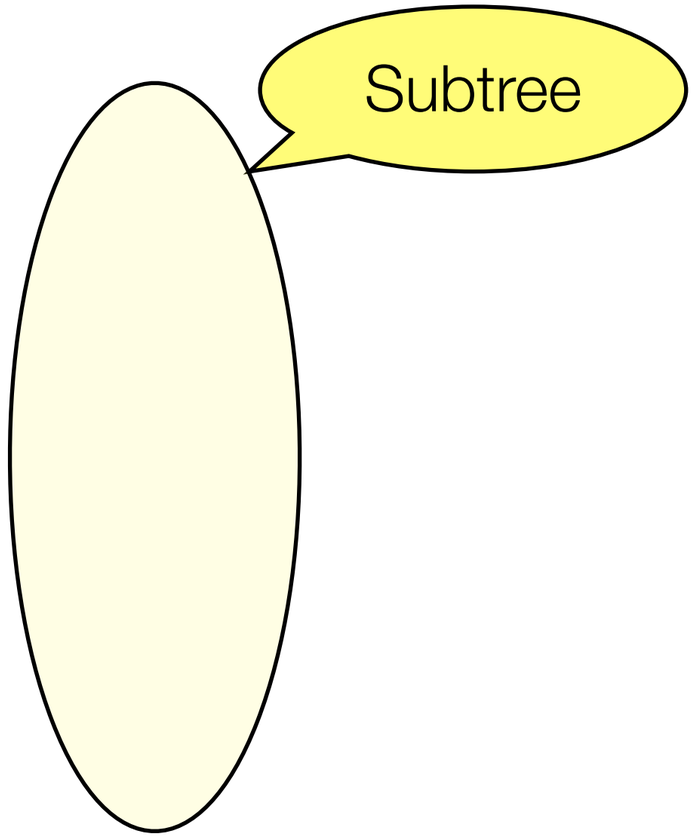
FIGURE 8.2
A Tree of Words



Subtree

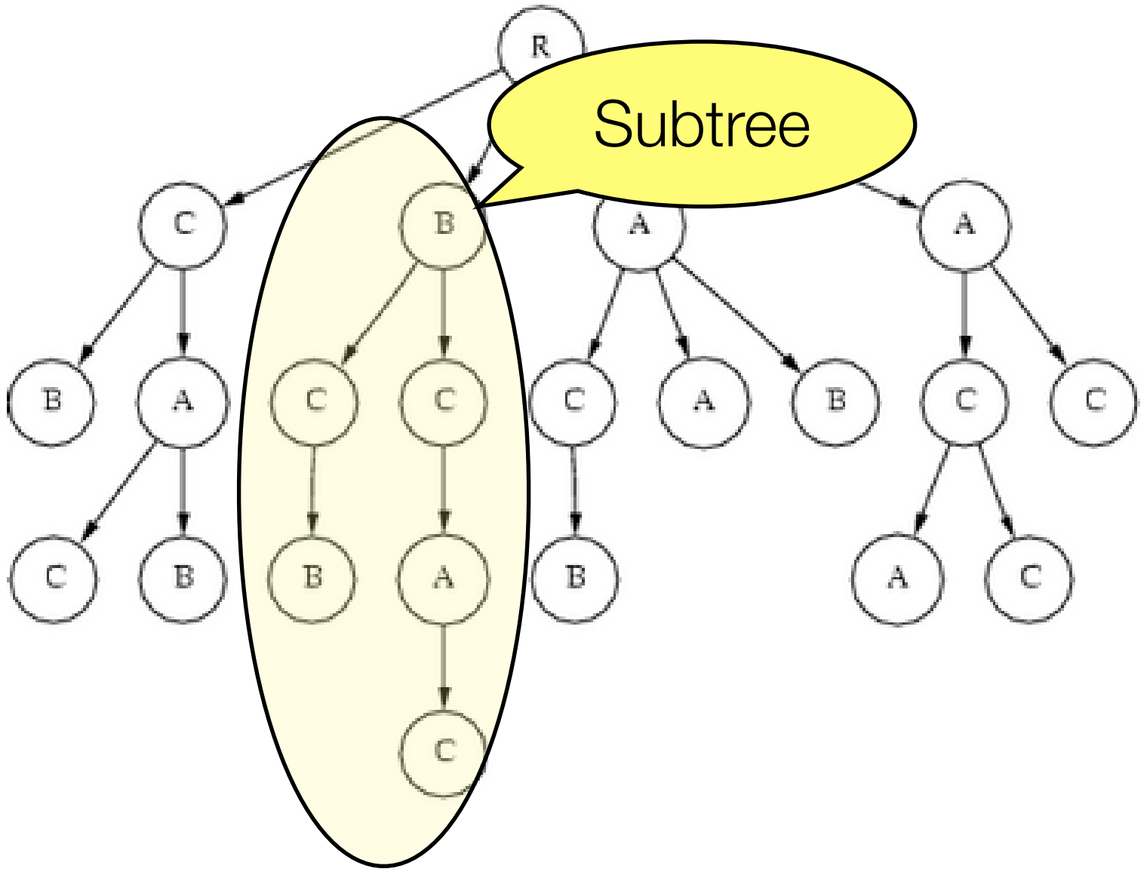
Subtree

- a subtree is a new tree taken with a different root (that is usually an inner node)



Subtree

- a subtree is a new tree taken with a different root (that is usually an inner node)



Level

Level

- Level (or depth) is a measure of the distance of the path from the root to that node

Level

- Level (or depth) is a measure of the distance of the path from the root to that node
- defined recursively

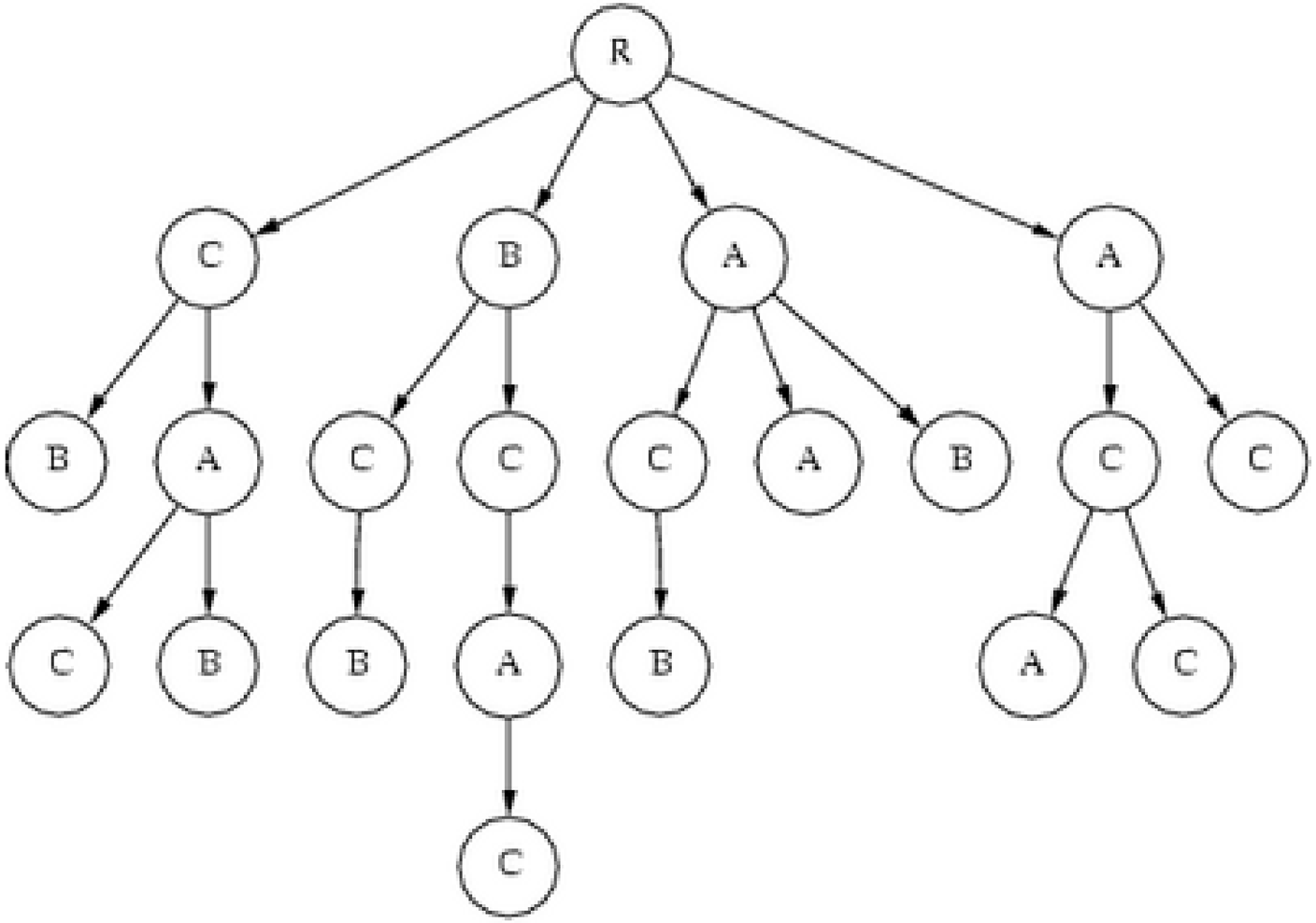
Level

- Level (or depth) is a measure of the distance of the path from the root to that node
- defined recursively
    - if node n is the root, it's level is 1

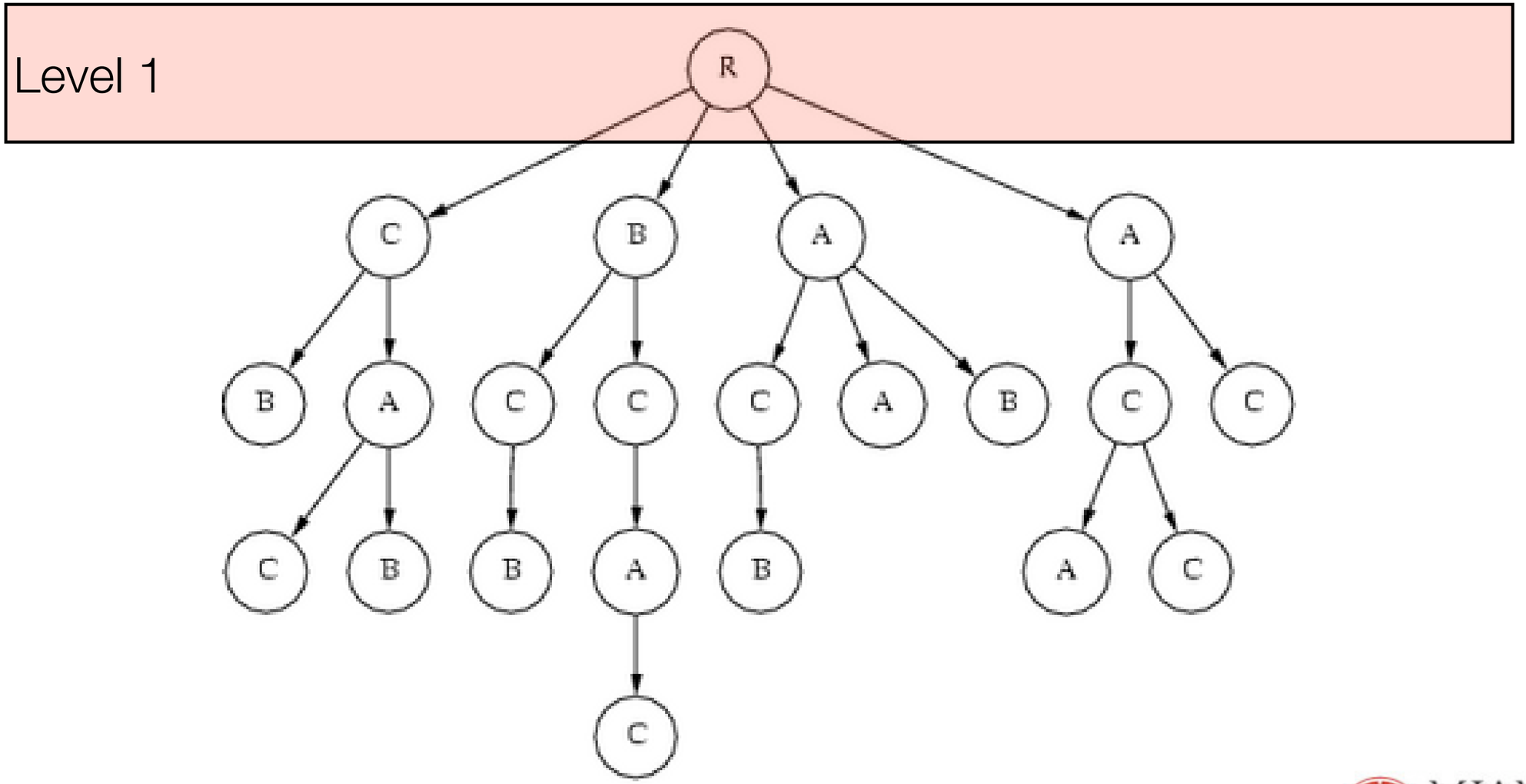
Level

- Level (or depth) is a measure of the distance of the path from the root to that node
- defined recursively
  - if node n is the root, it's level is 1
  - else, it's level is 1 + the level of it's parent

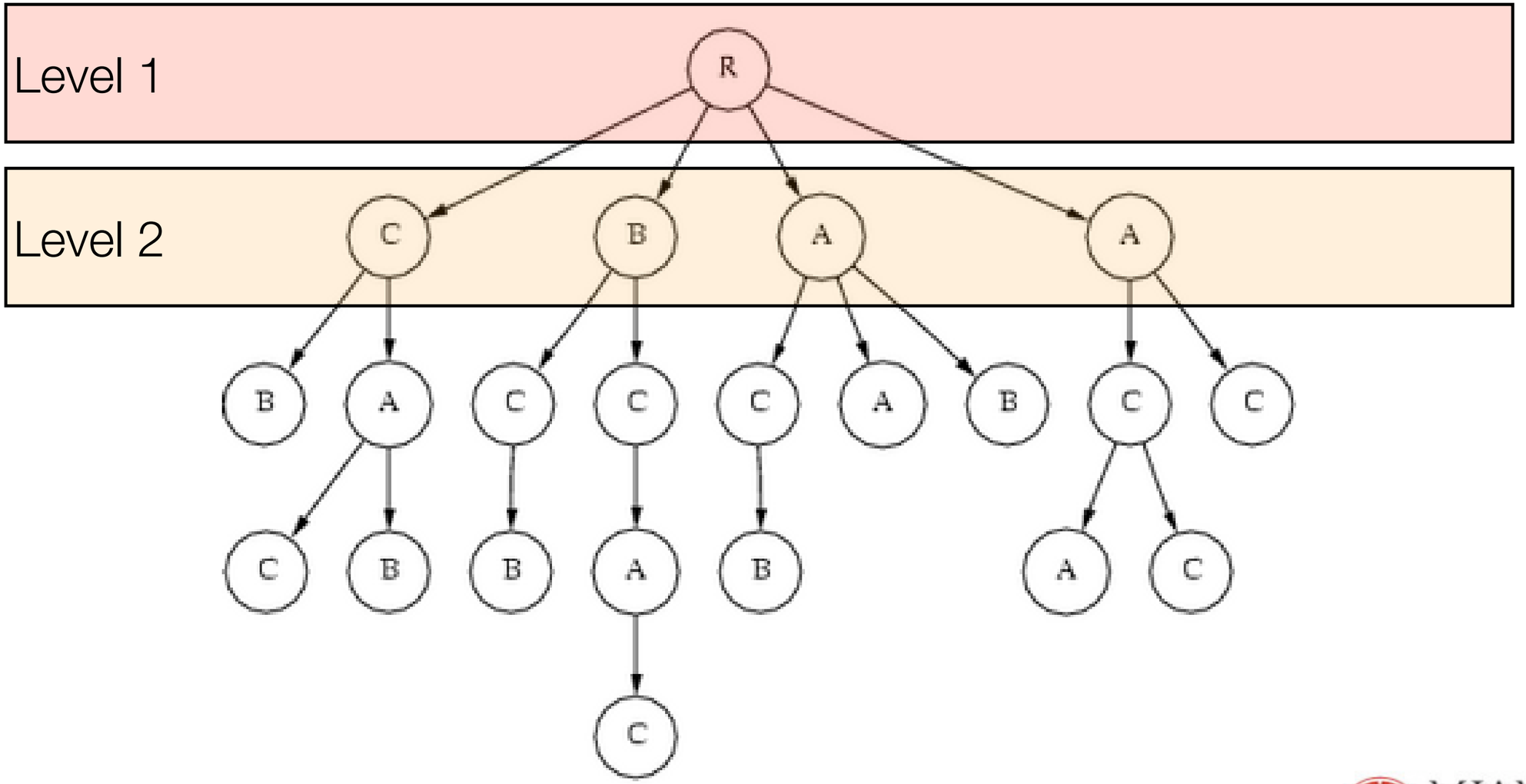
Level



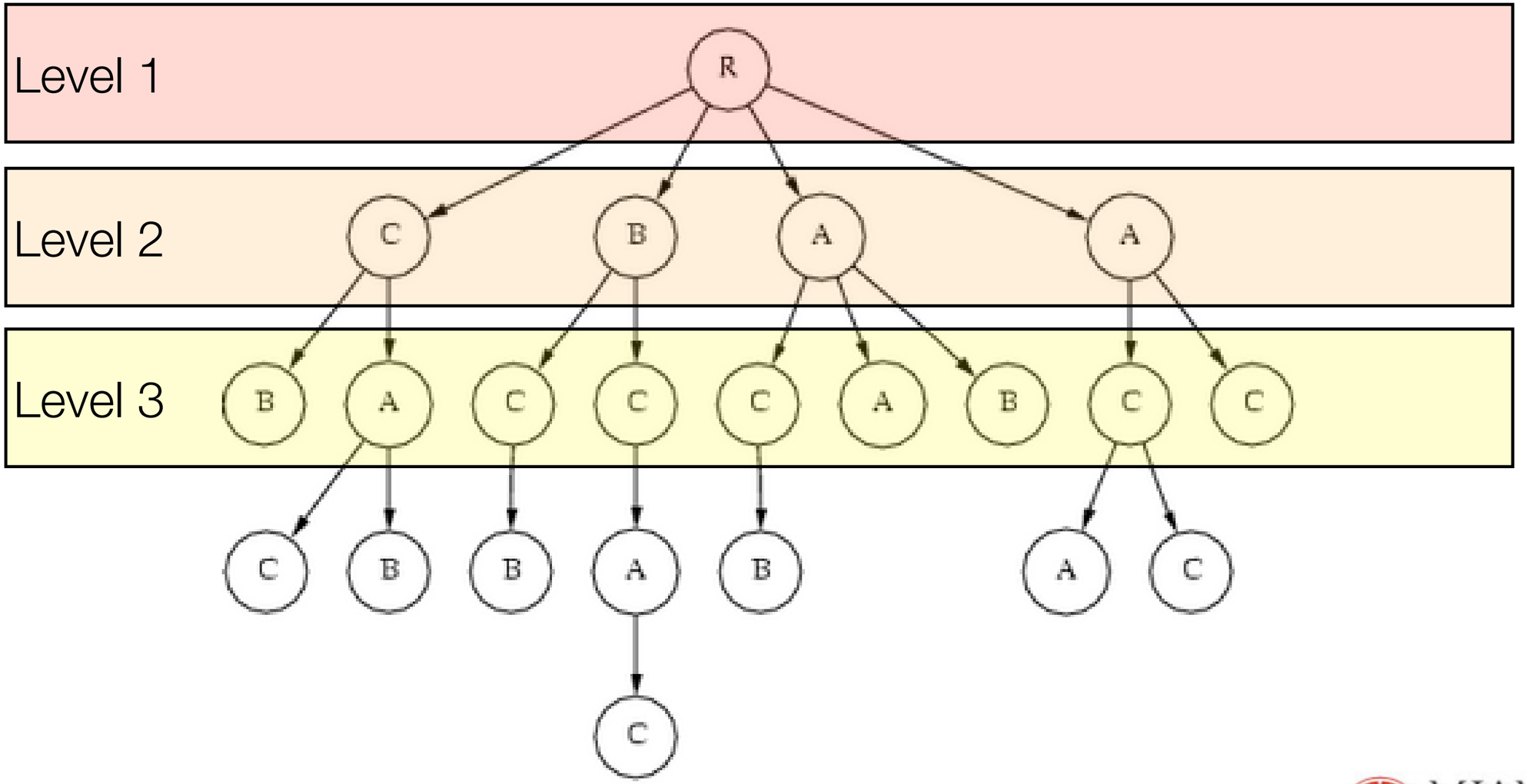
Level



Level

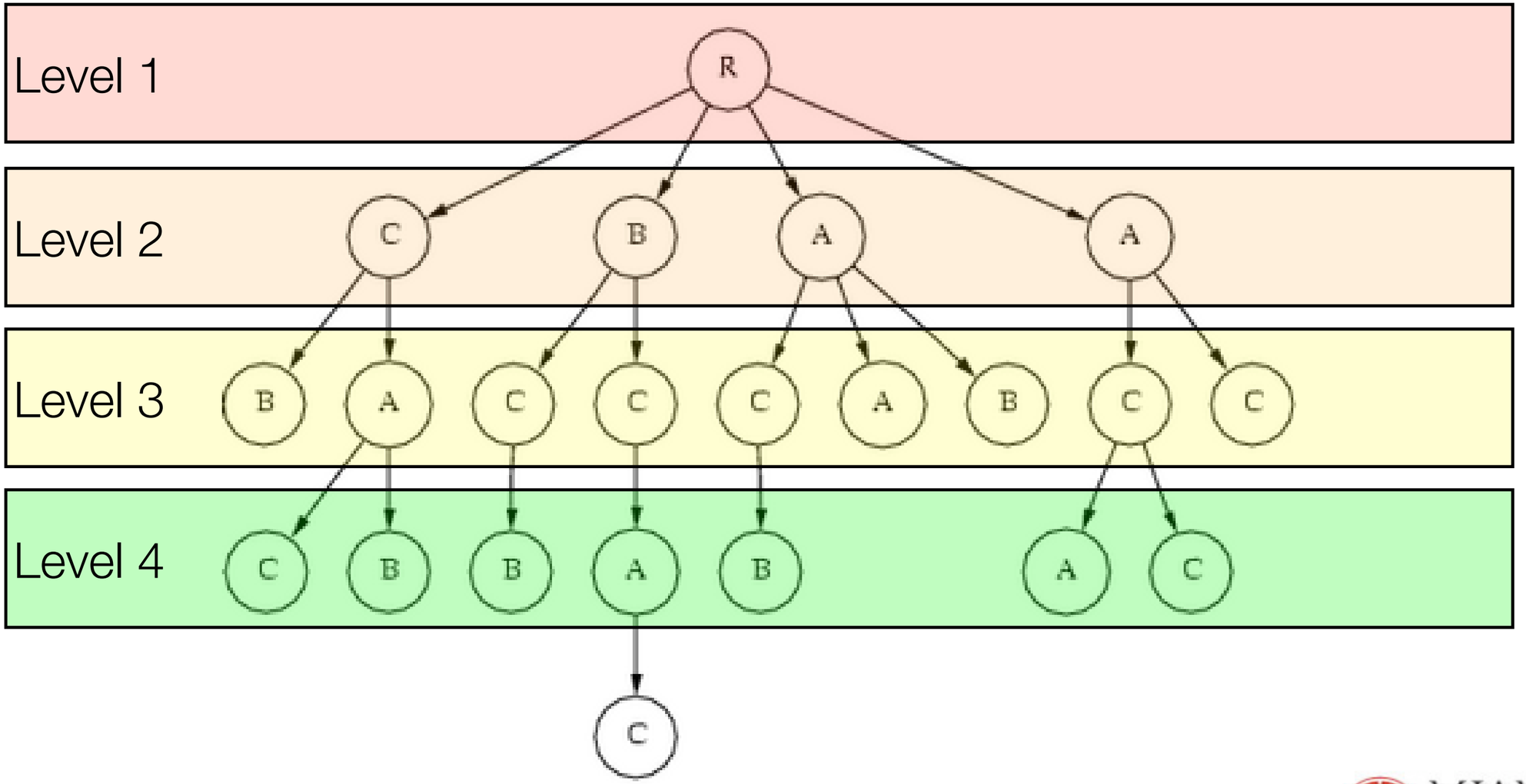


Level

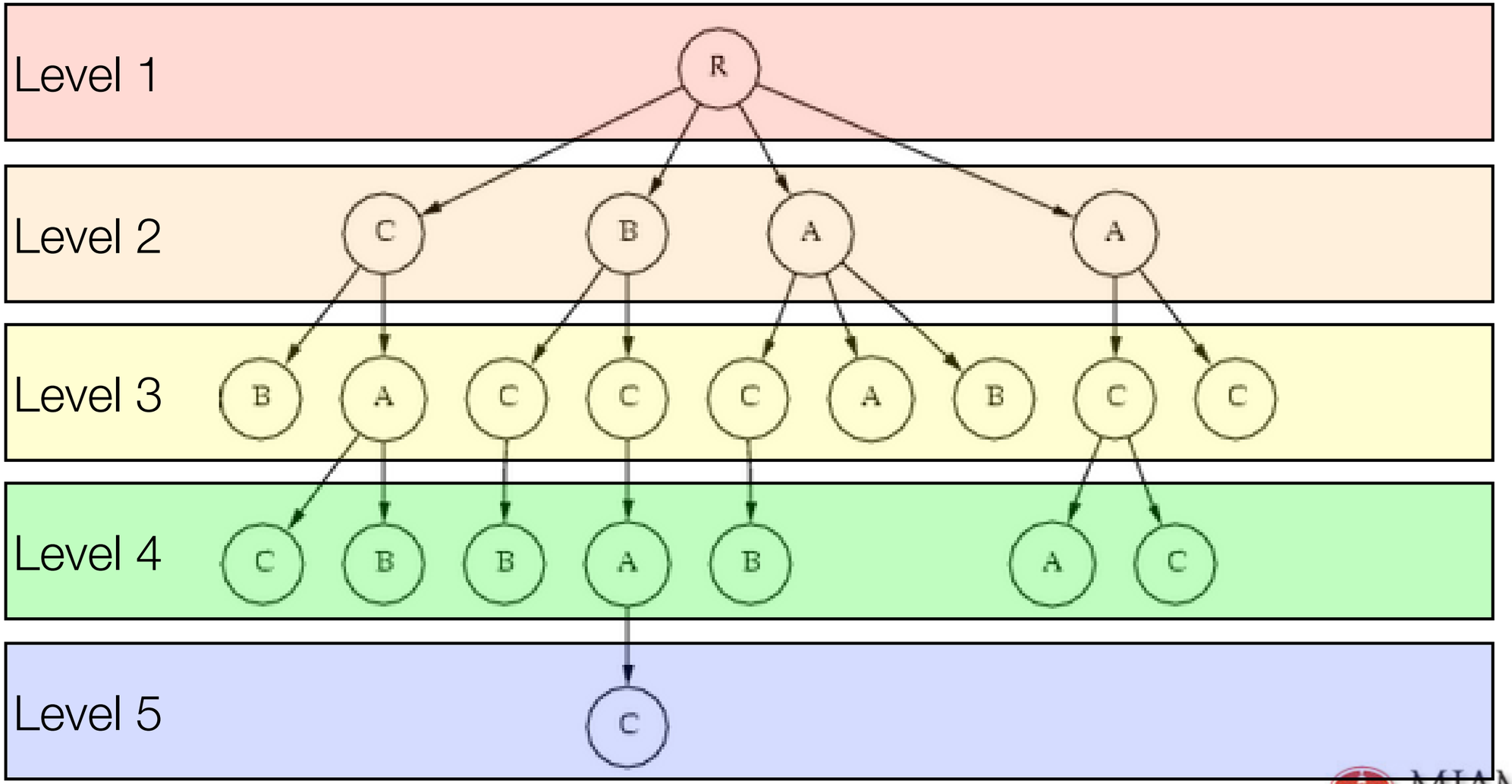


MIAMI UNIVERSITY

Level



Level



Height

Height

- Number of nodes in the longest path from the root to a leaf node

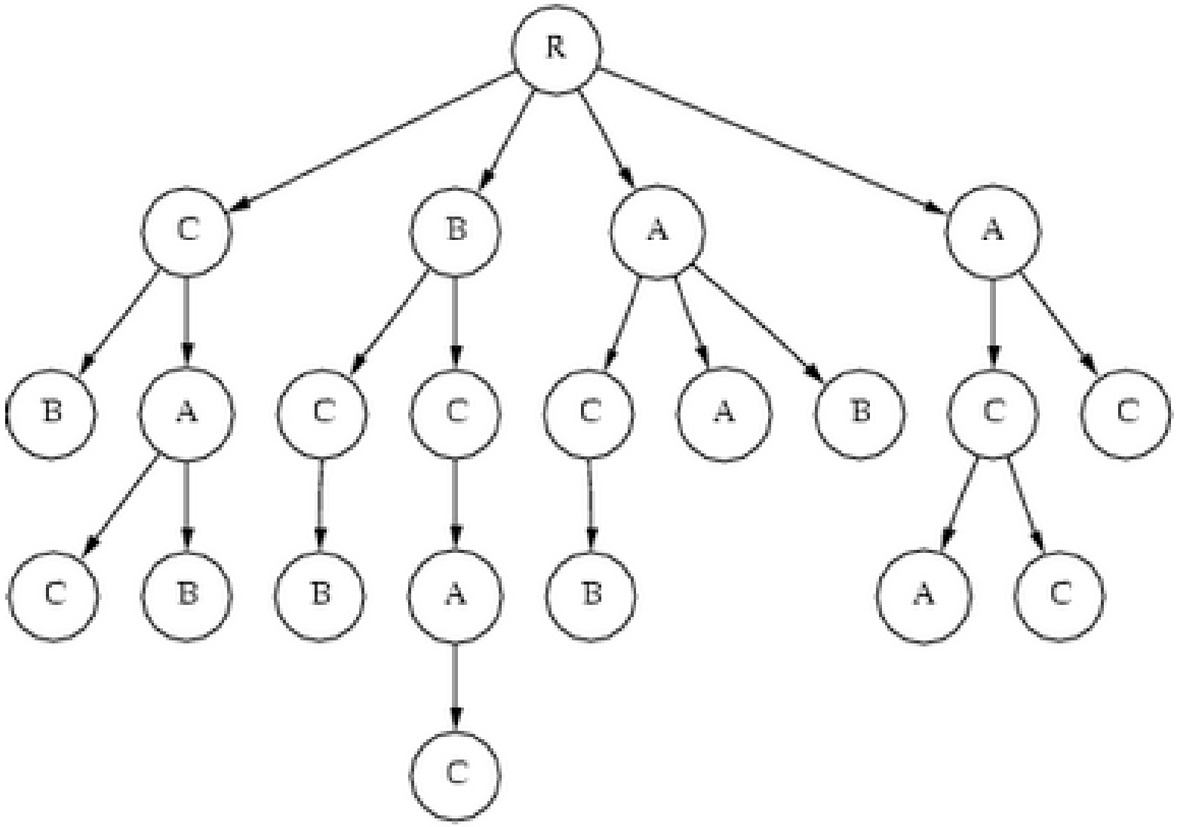
Height

- Number of nodes in the longest path from the root to a leaf node
    - if the tree T is empty, height is 0

Height

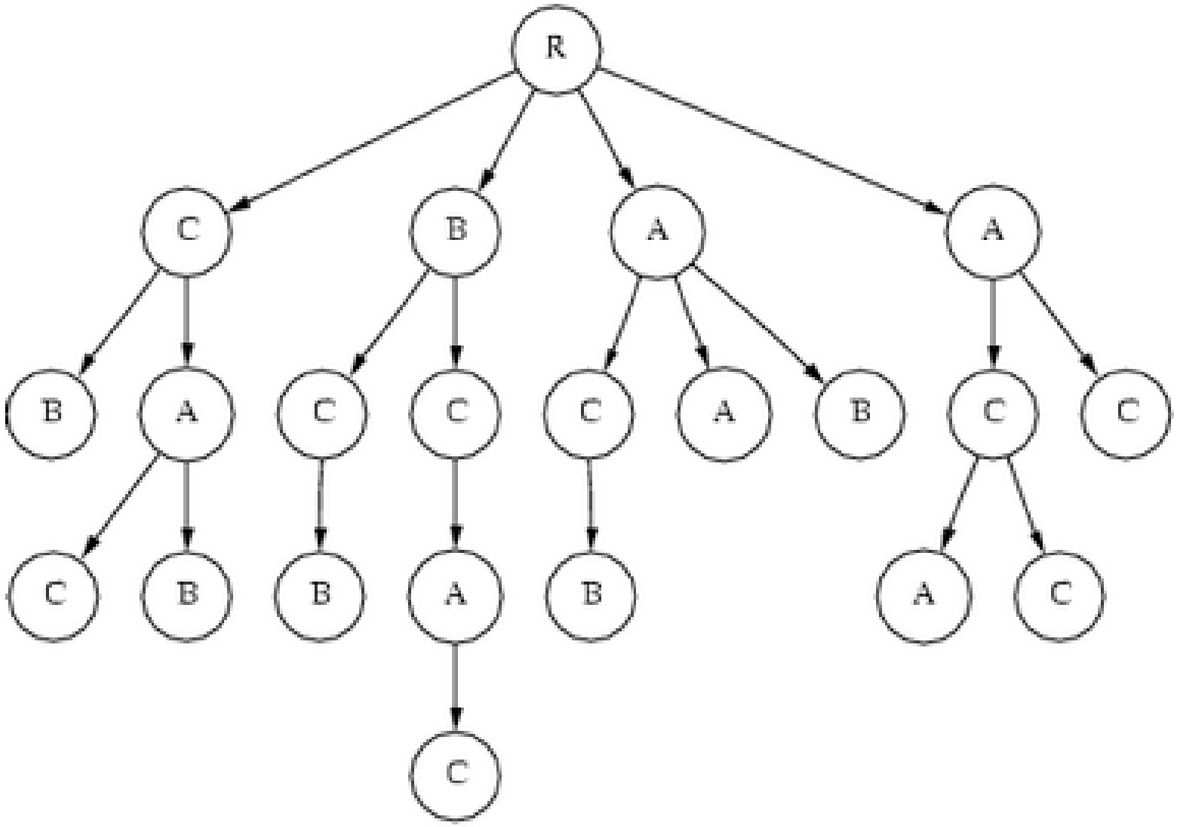
- Number of nodes in the longest path from the root to a leaf node
  - if the tree T is empty, height is 0
  - if the tree T is not empty, the height is the number of nodes in the longest path

Height



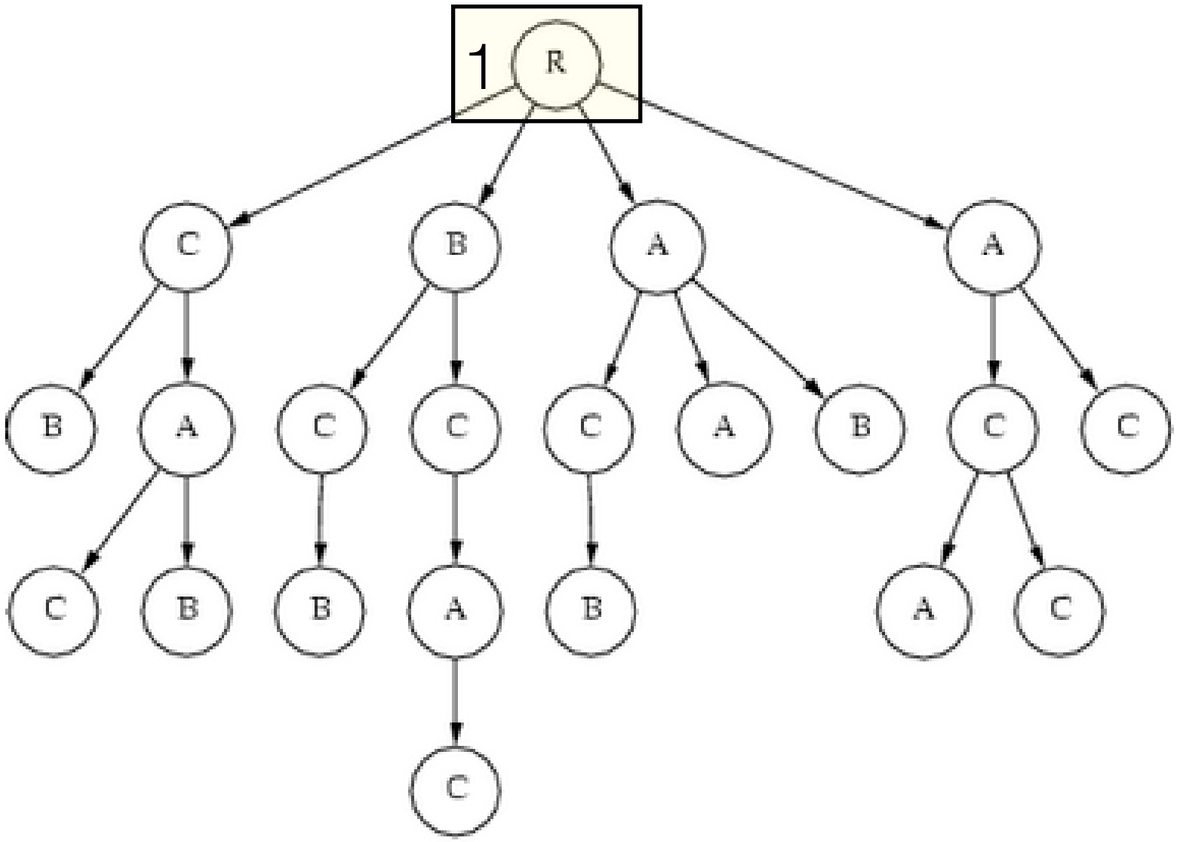
Height

- Each entry in the tree is an element or, more commonly, a node



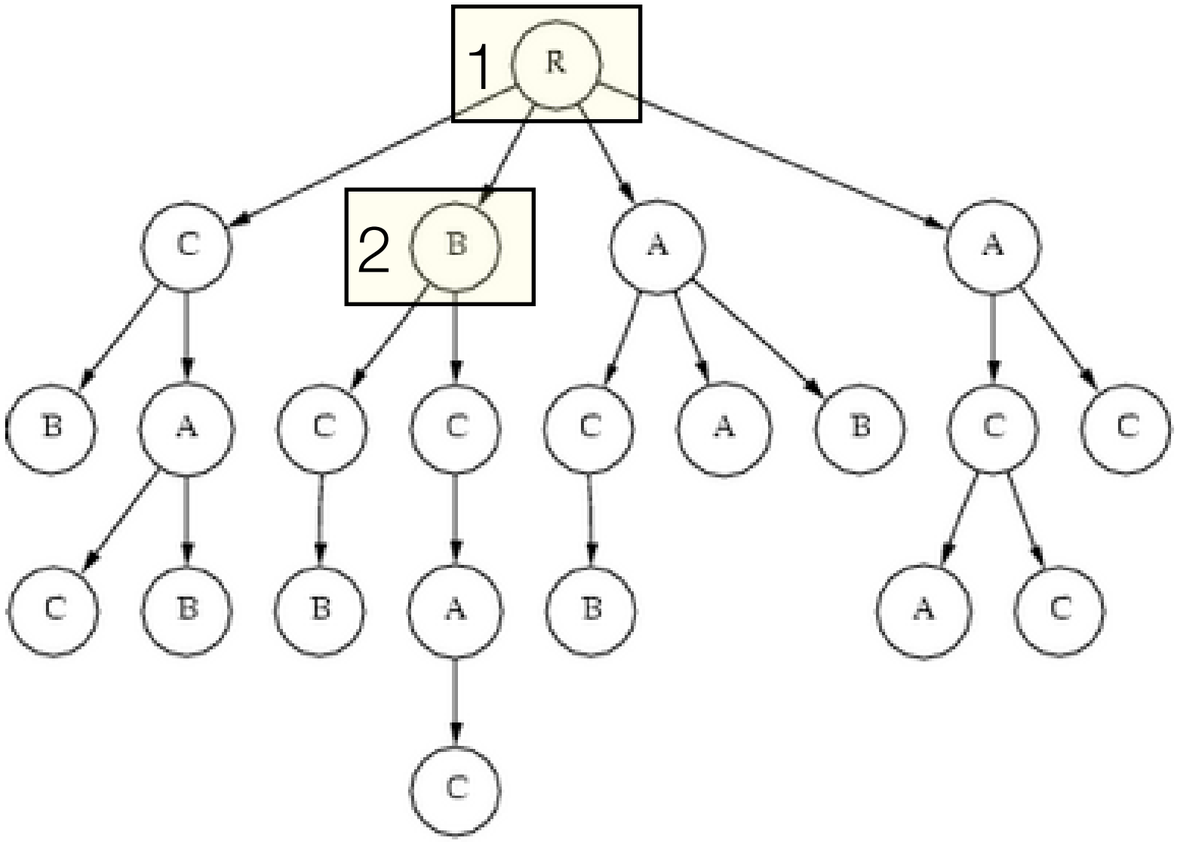
Height

- Each entry in the tree is an element or, more commonly, a node



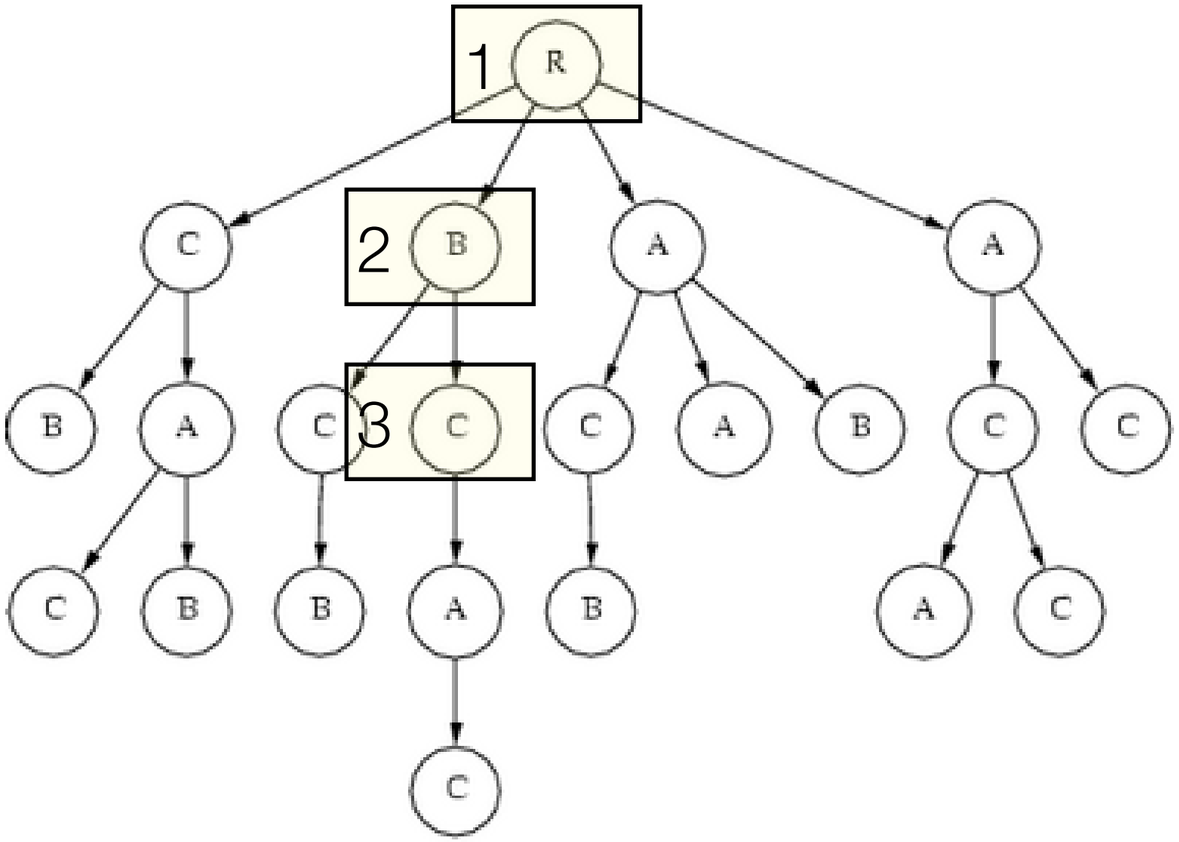
Height

- Each entry in the tree is an element or, more commonly, a node



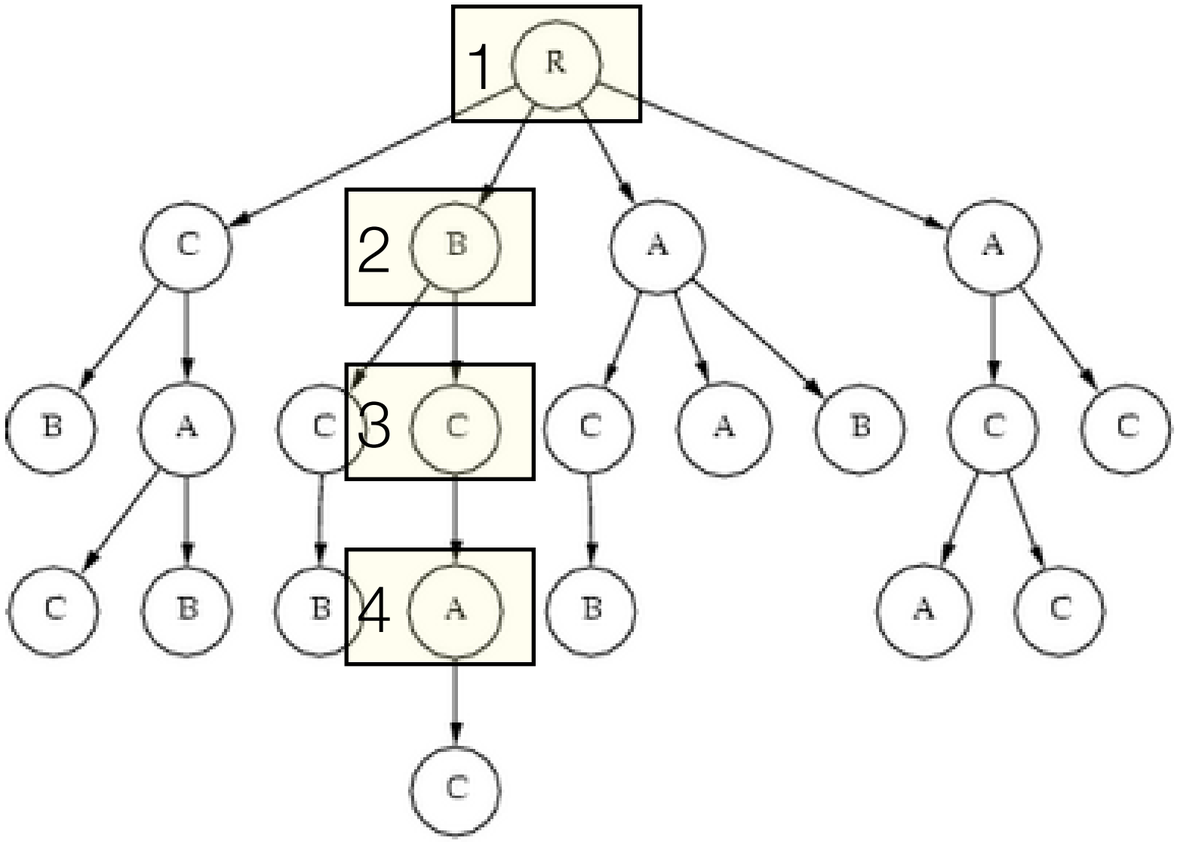
Height

- Each entry in the tree is an element or, more commonly, a node



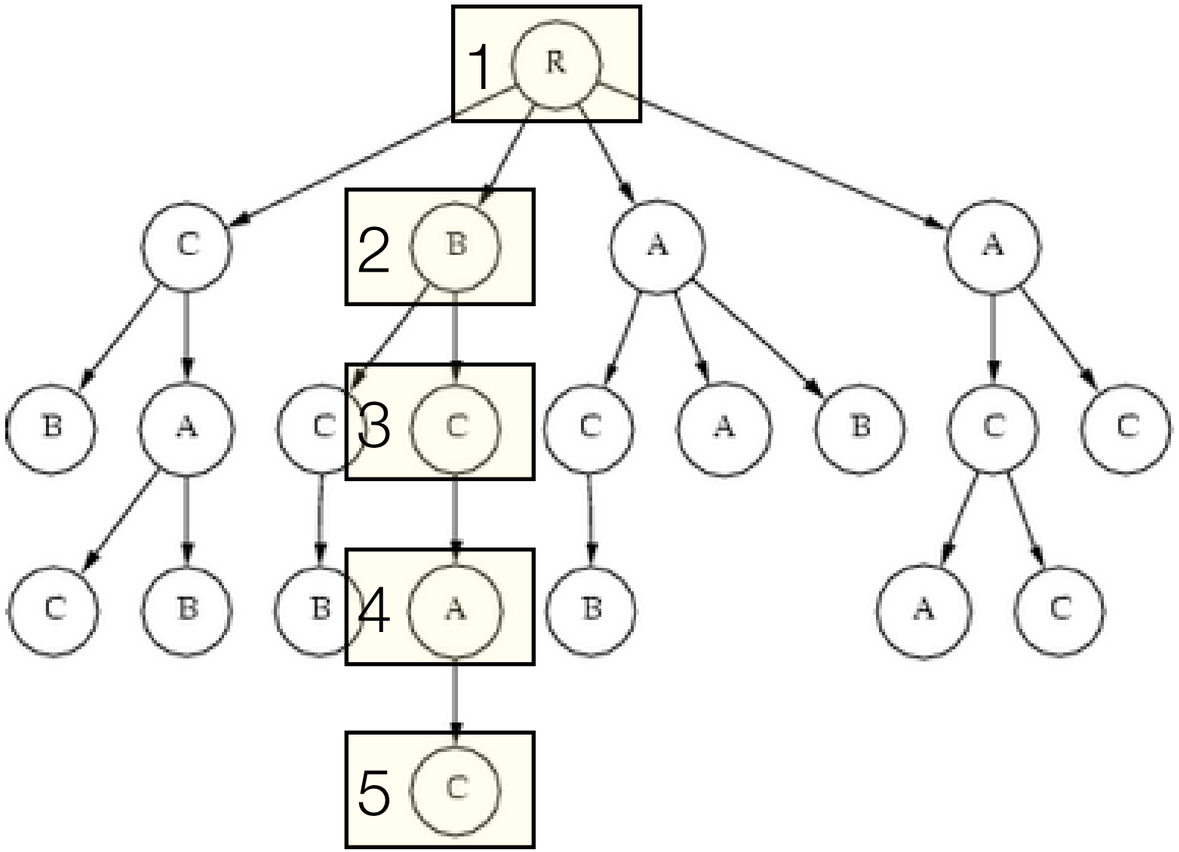
Height

- Each entry in the tree is an element or, more commonly, a node

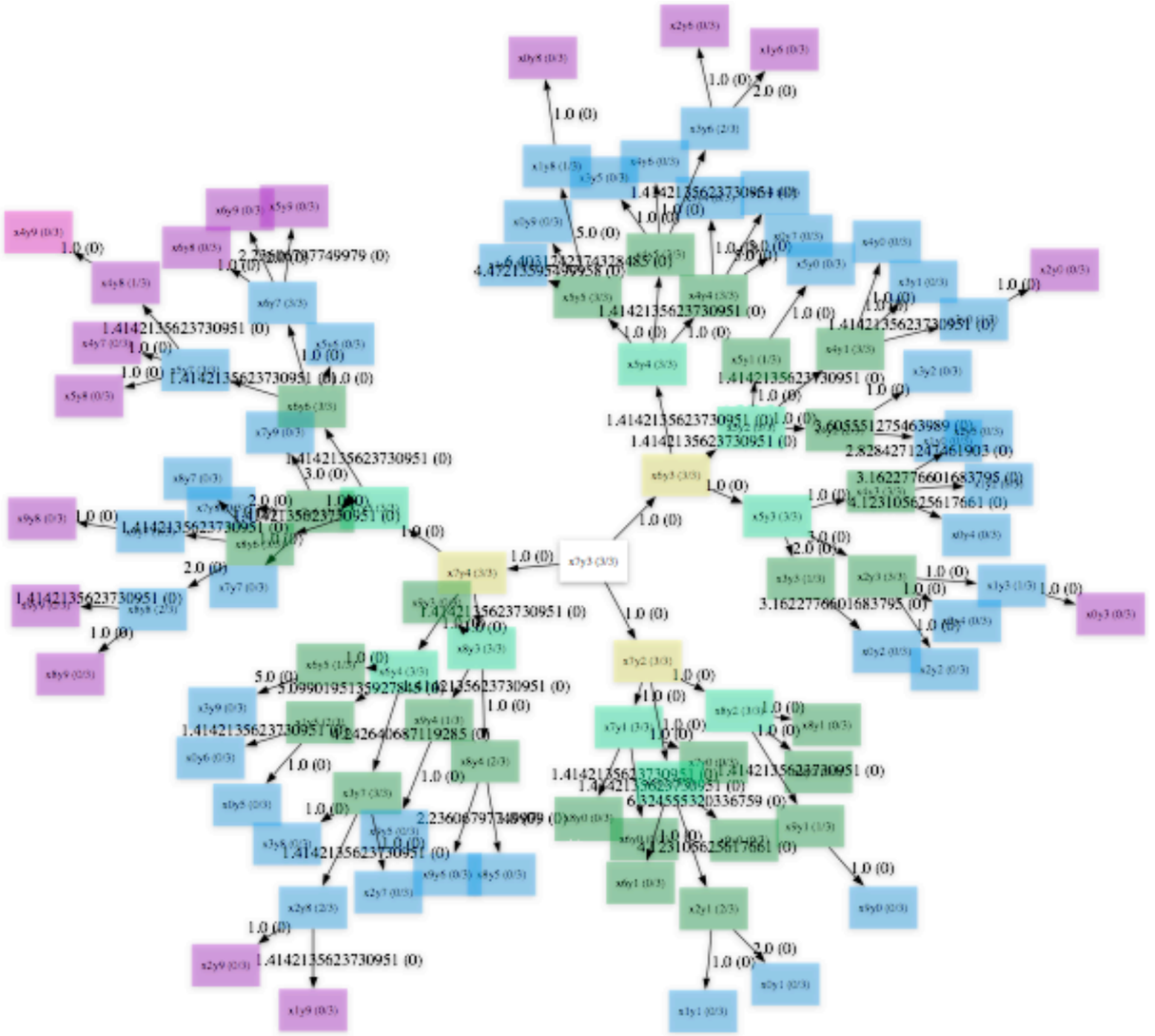


Height

- Each entry in the tree is an element or, more commonly, a node



Binary Tree



Binary Tree

Binary Tree

- A special tree where each node has at most 2 subtrees (or at most 2 children)

Binary Tree

- A special tree where each node has at most 2 subtrees (or at most 2 children)

- Formally, a tree T is a binary tree if:

Binary Tree

- A special tree where each node has at most 2 subtrees (or at most 2 children)
- Formally, a tree T is a binary tree if:
    - T is empty

Binary Tree

- A special tree where each node has at most 2 subtrees (or at most 2 children)
- Formally, a tree T is a binary tree if:
  - T is empty
  - T is not empty, it's root has two subtrees, Tl and Tr, such that Tl and Tr are binary trees.

Example Binary Tree

Example Binary Tree

- Expression Tree

Example Binary Tree

- Expression Tree
- Binary Search Tree

Example Binary Tree

- Expression Tree
- Binary Search Tree
- Huffman Encoding Trees

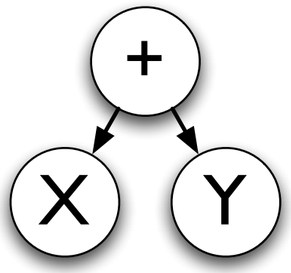
Expression Tree

Expression Tree

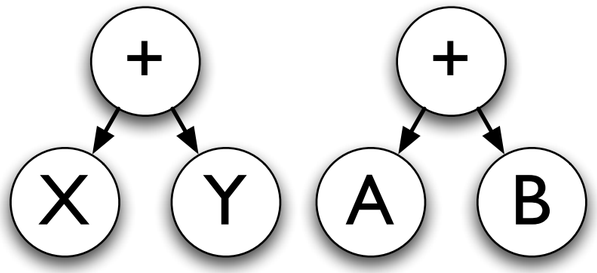
- (x + y) * ((a + b) / c) + (x * (y + z))

Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))

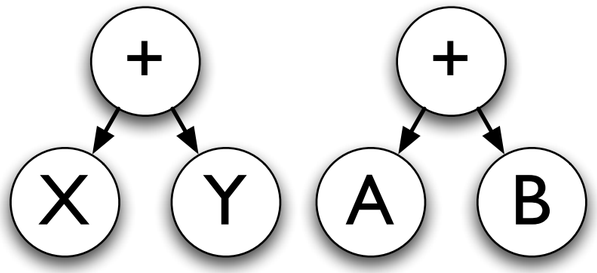


Expression Tree



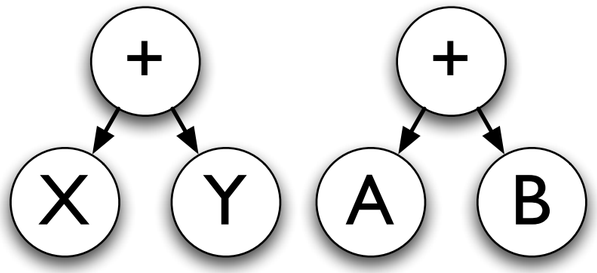
Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))

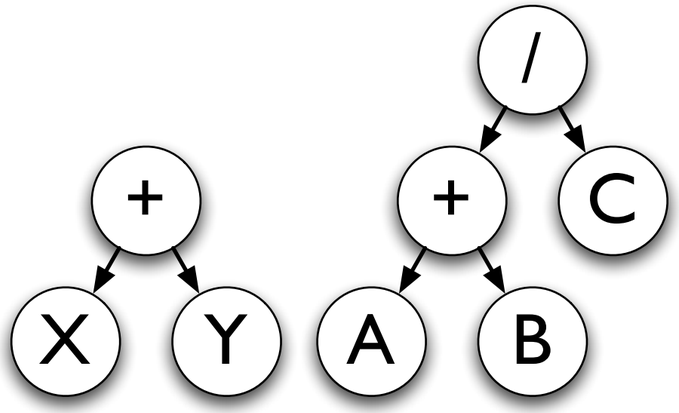


Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))

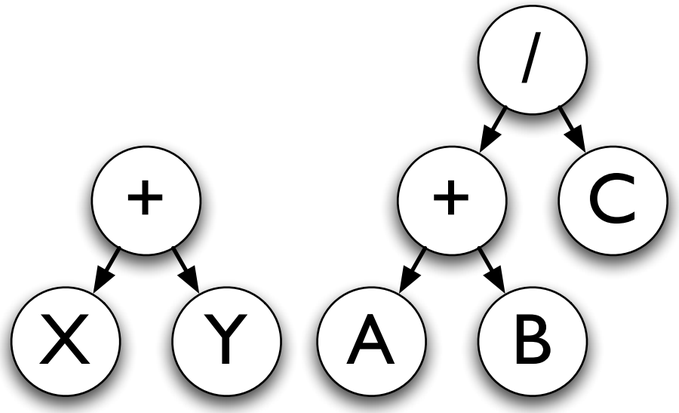


Expression Tree



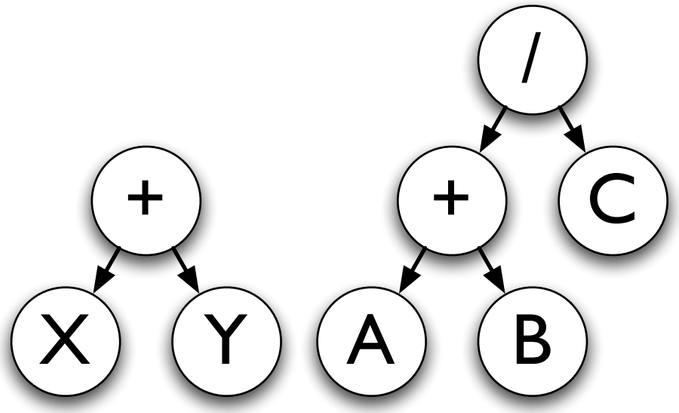
Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))

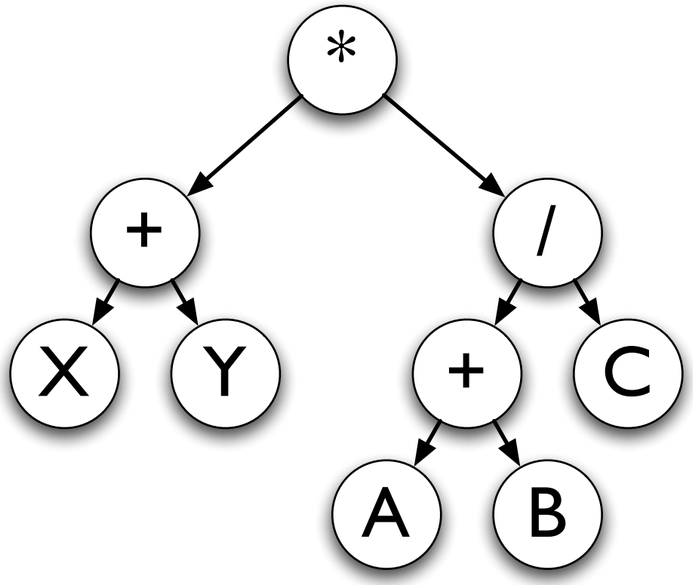


Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))

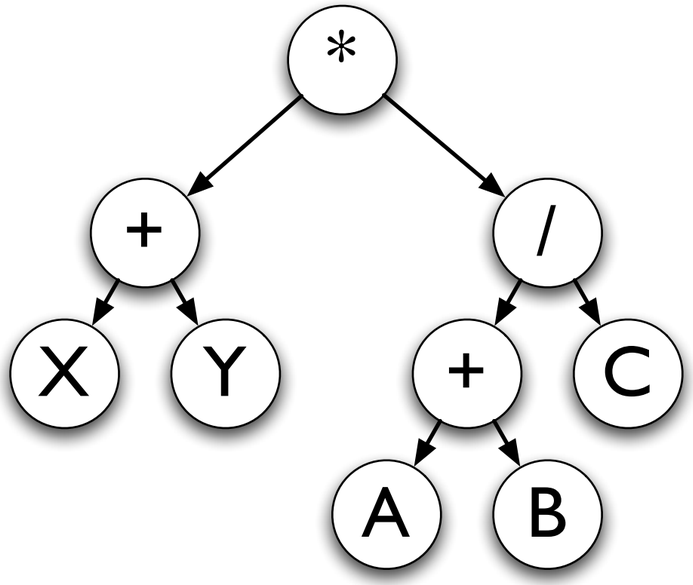


Expression Tree



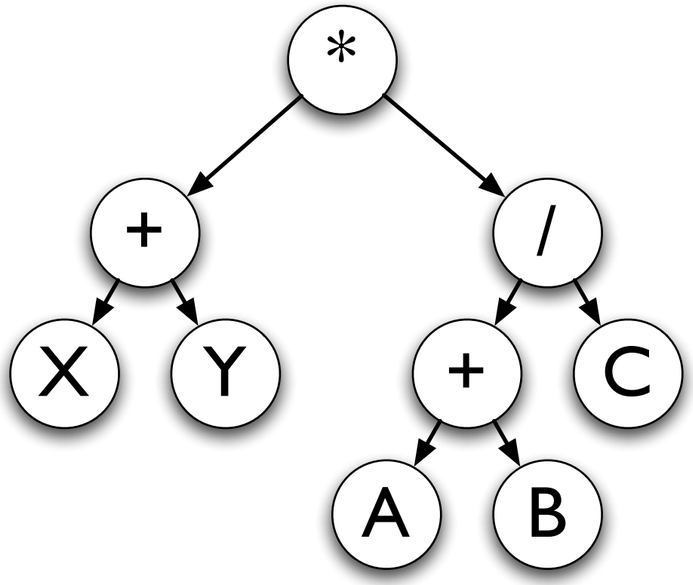
Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))

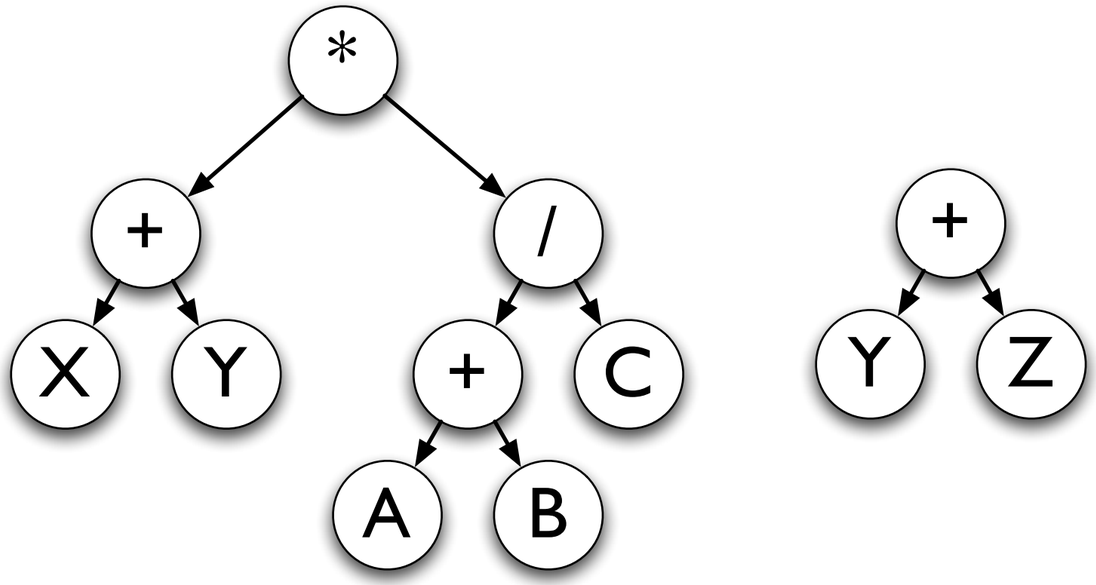


Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))

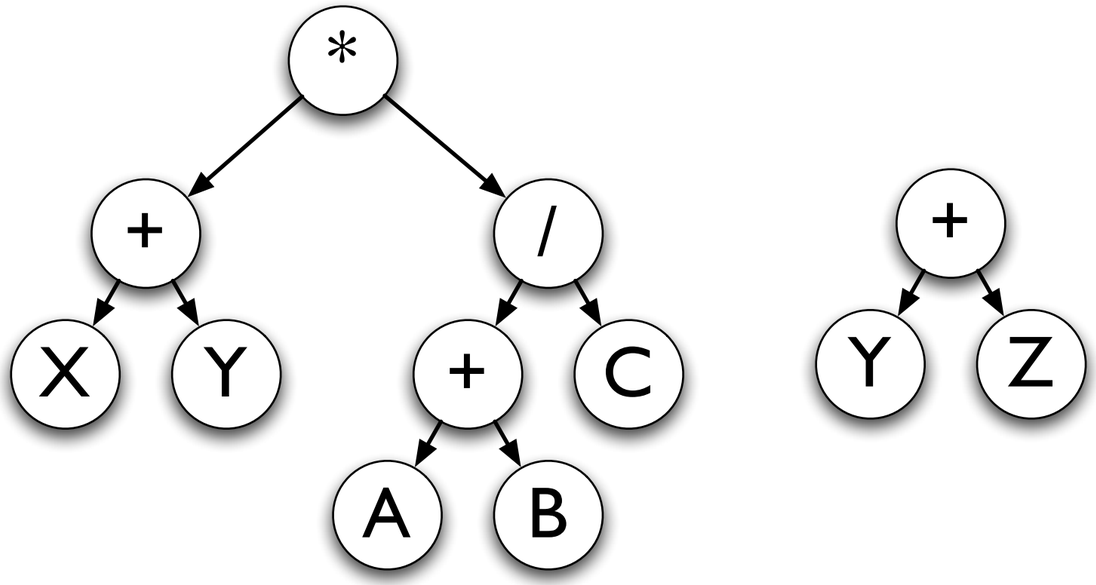


Expression Tree



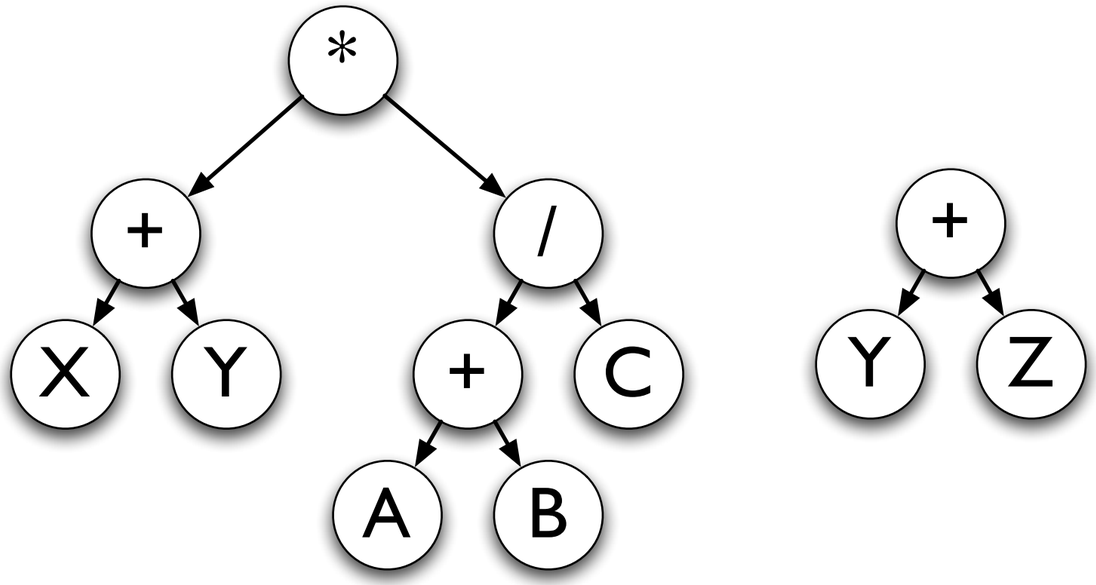
Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))

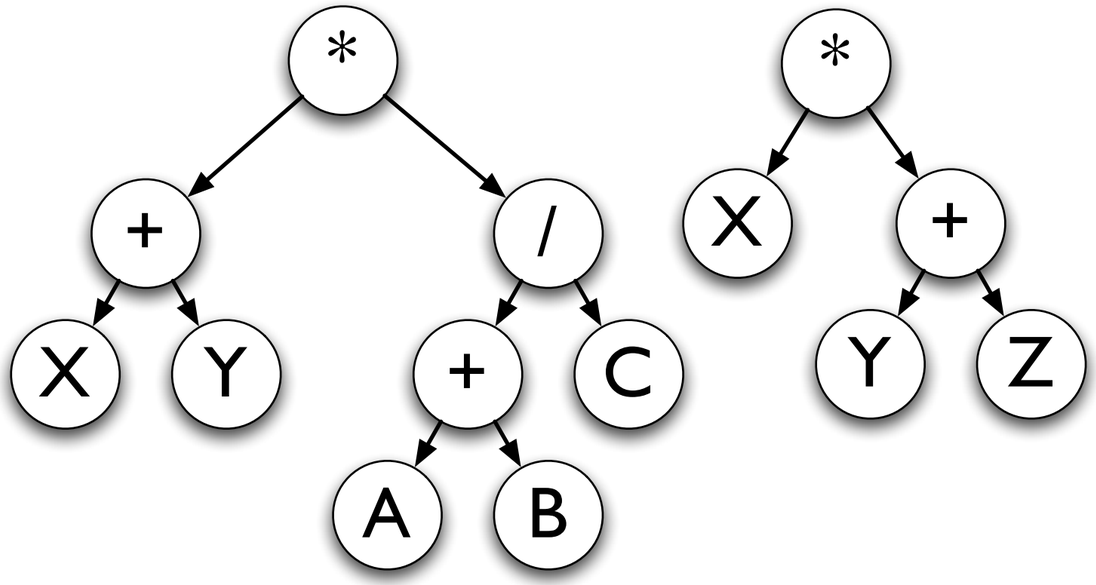


Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))

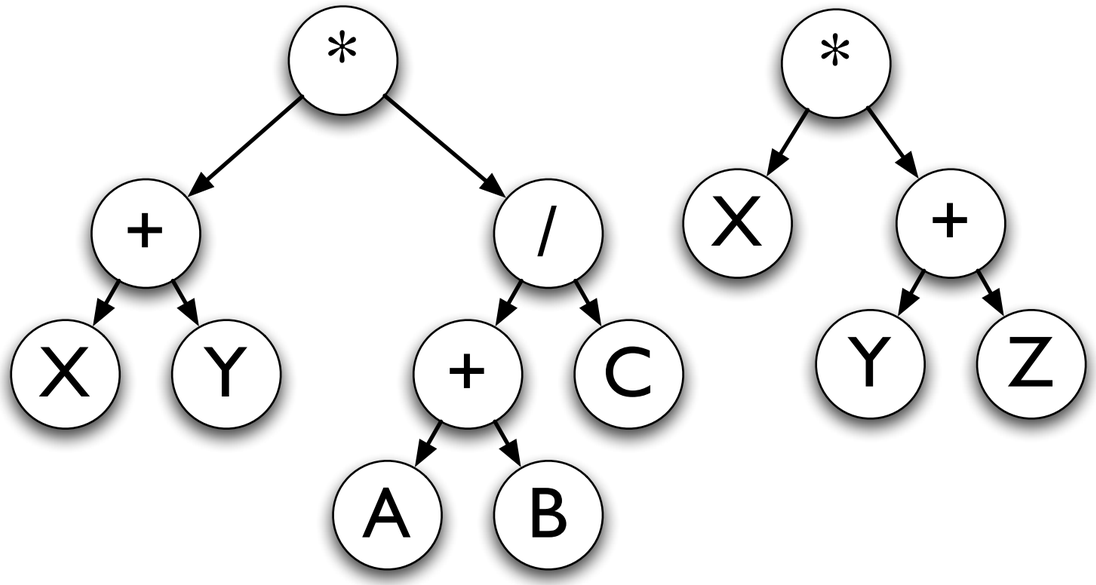


Expression Tree



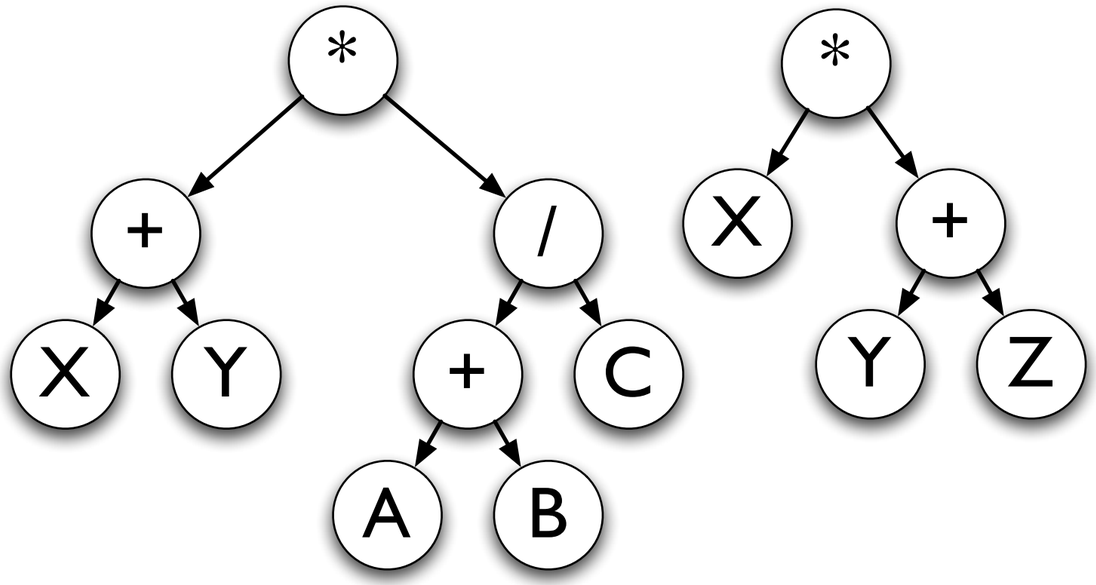
Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))

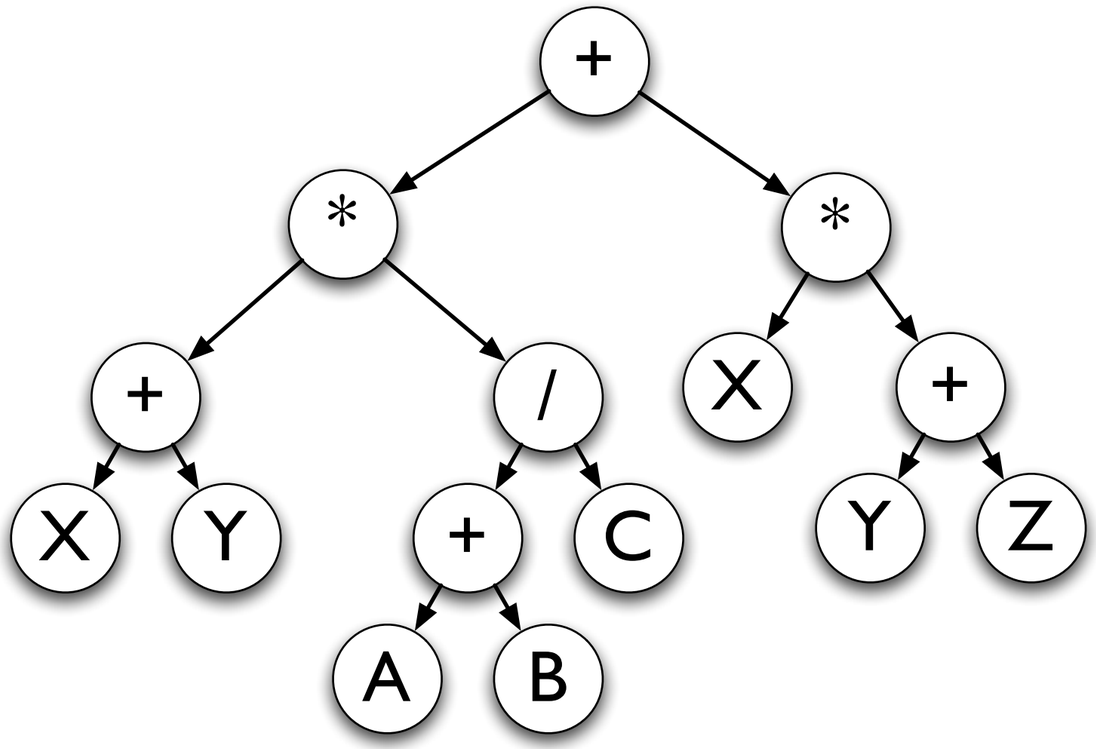


Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))

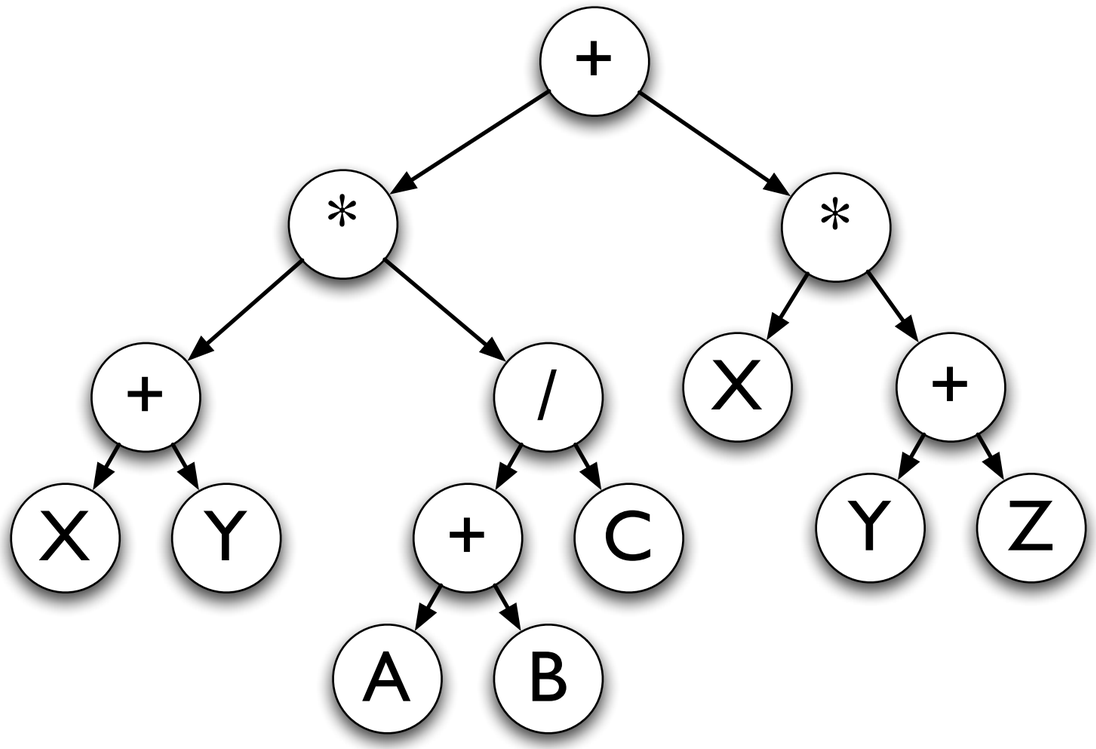


Expression Tree



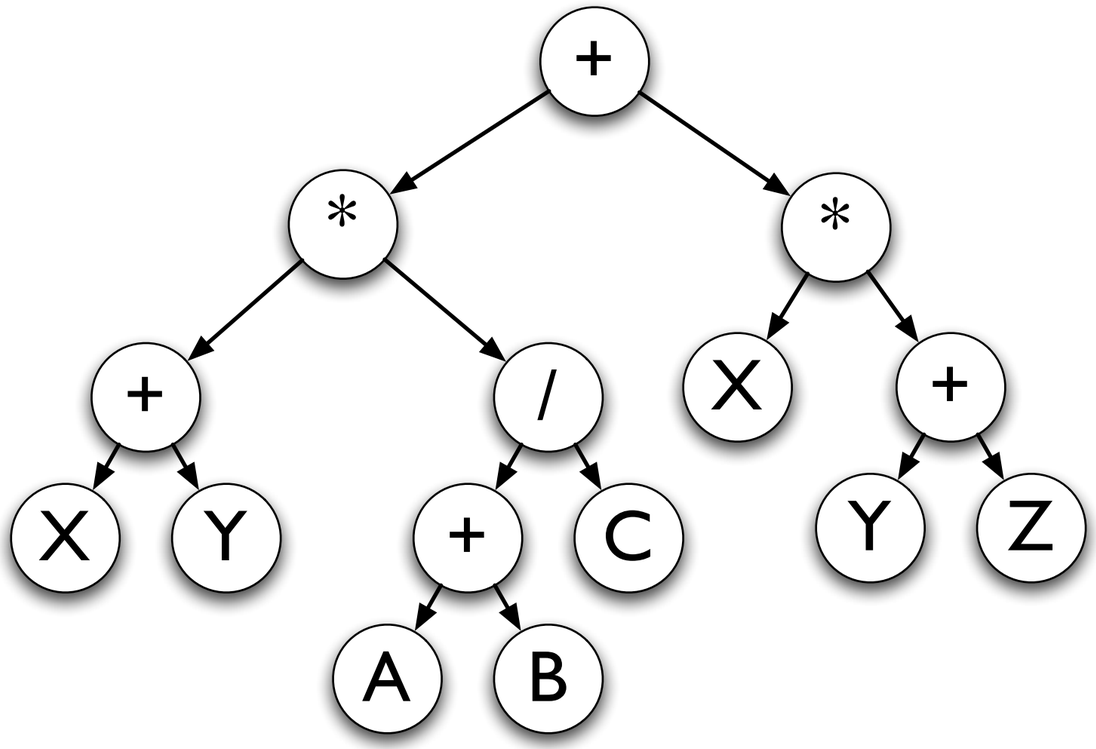
Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))



Expression Tree

- (x + y) * ((a + b) / c) + (x * (y + z))



Binary Tree

Binary Tree

- A special tree where each node has at most 2 subtrees (or at most 2 children)

Binary Tree

- A special tree where each node has at most 2 subtrees (or at most 2 children)

- Formally, a tree T is a binary tree if:

Binary Tree

- A special tree where each node has at most 2 subtrees (or at most 2 children)
- Formally, a tree T is a binary tree if:
  - T is empty

Binary Tree

- A special tree where each node has at most 2 subtrees (or at most 2 children)
- Formally, a tree T is a binary tree if:
  - T is empty
  - T is not empty, it's root has two subtrees, Tl and Tr, such that Tl and Tr are binary trees.

Building

Building

- if the tree is empty

Building

- if the tree is empty
    - insert the node

Building

- if the tree is empty
  - insert the node
- else if the value is the same as the current node

Building

- if the tree is empty
  - insert the node
- else if the value is the same as the current node
  - cannot insert - duplicate value

Building

- if the tree is empty
  - insert the node
- else if the value is the same as the current node
  - cannot insert - duplicate value
- else if the value is less than the current

Building

- if the tree is empty
  - insert the node
- else if the value is the same as the current node
  - cannot insert - duplicate value
- else if the value is less than the current
  - insert in left subtree

Building

- if the tree is empty
  - insert the node
- else if the value is the same as the current node
  - cannot insert - duplicate value
- else if the value is less than the current
  - insert in left subtree
- else

Building

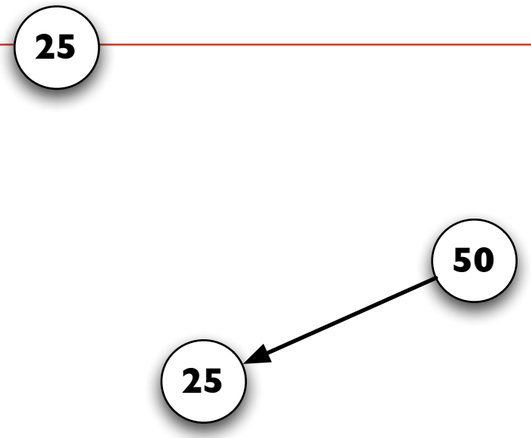
- if the tree is empty
  - insert the node
- else if the value is the same as the current node
  - cannot insert - duplicate value
- else if the value is less than the current
  - insert in left subtree
- else
  - insert in right subtree

BST

BST

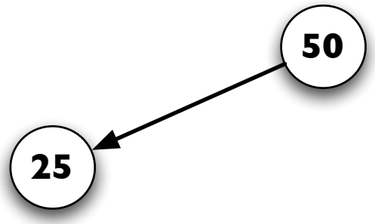
- Insert

BST

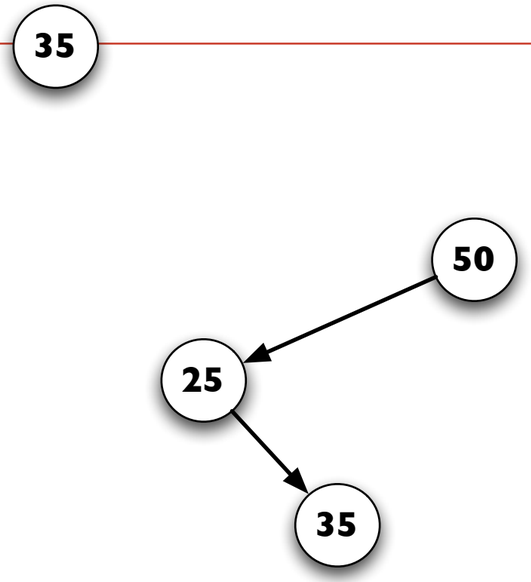


BST

- Insert

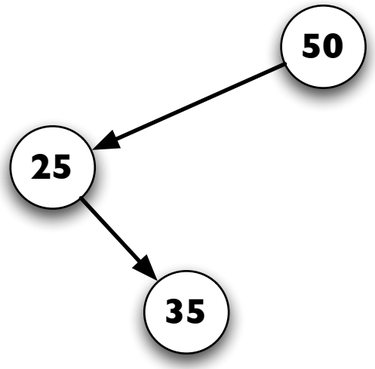


BST

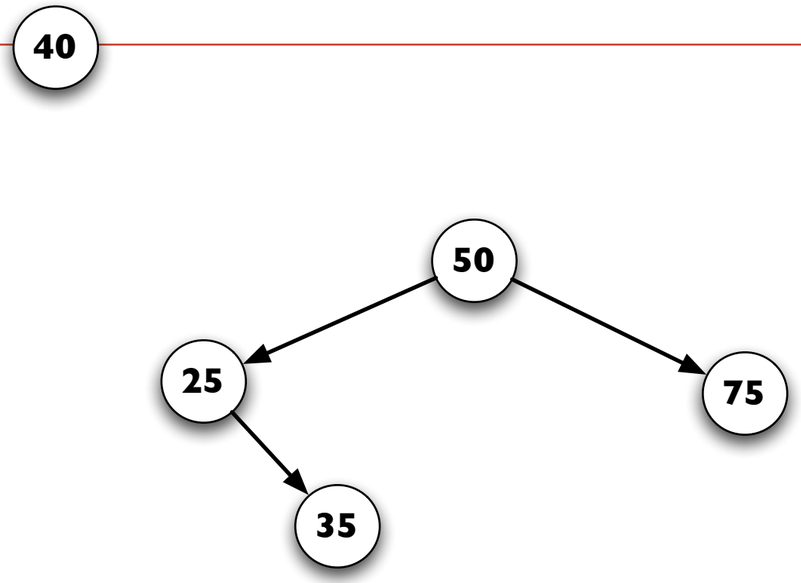


BST

- Insert

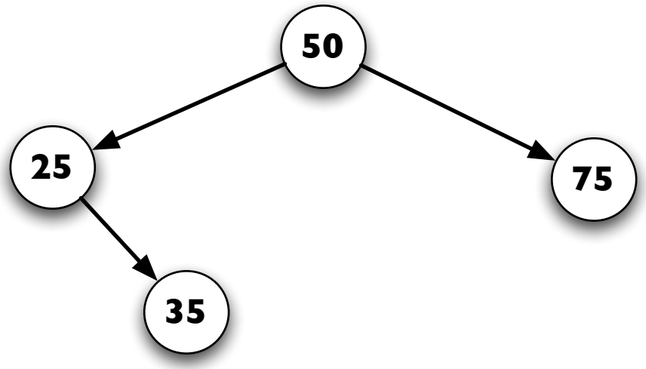


BST



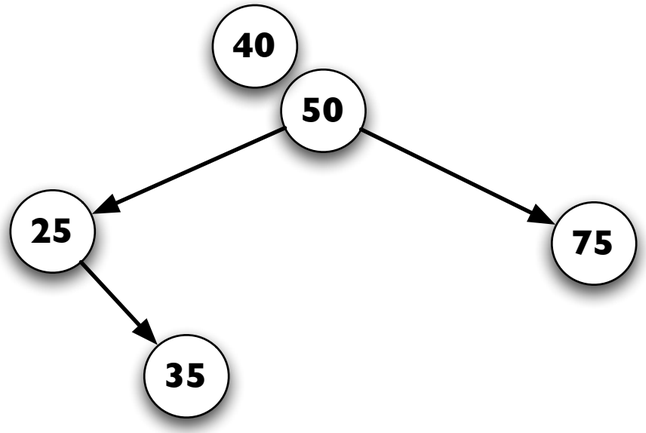
BST

- Insert



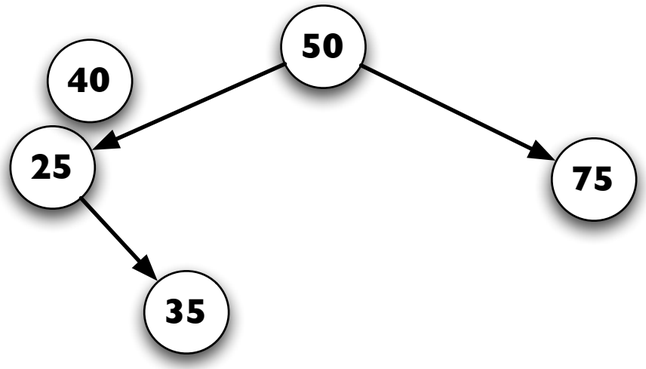
BST

- Insert



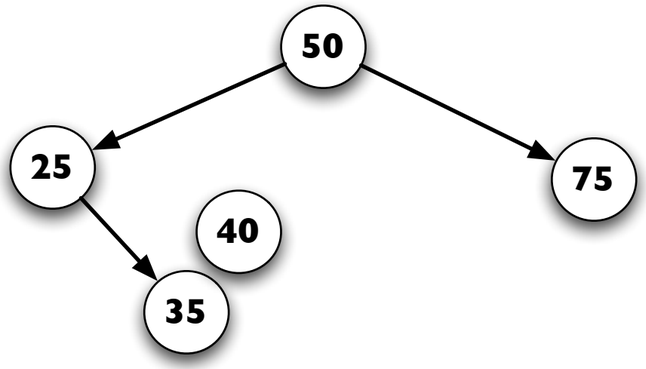
BST

- Insert



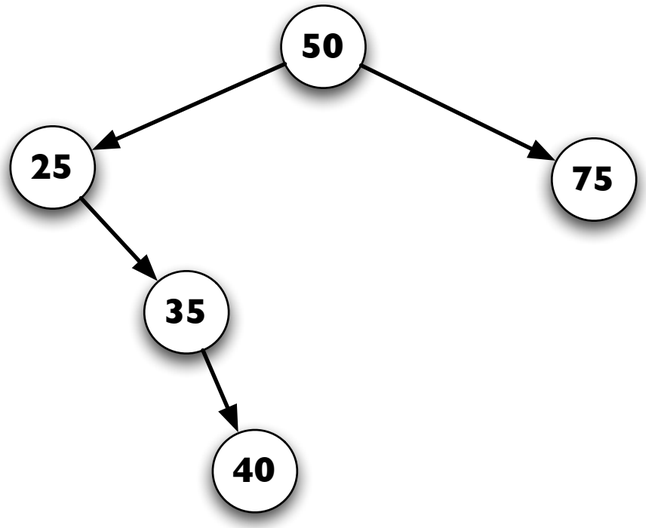
BST

- Insert

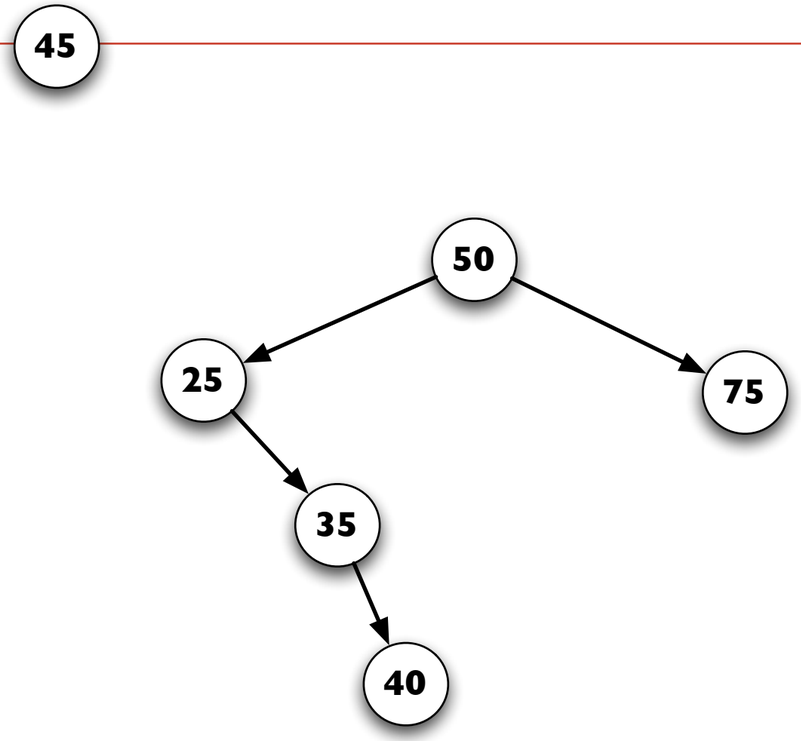


BST

- Insert

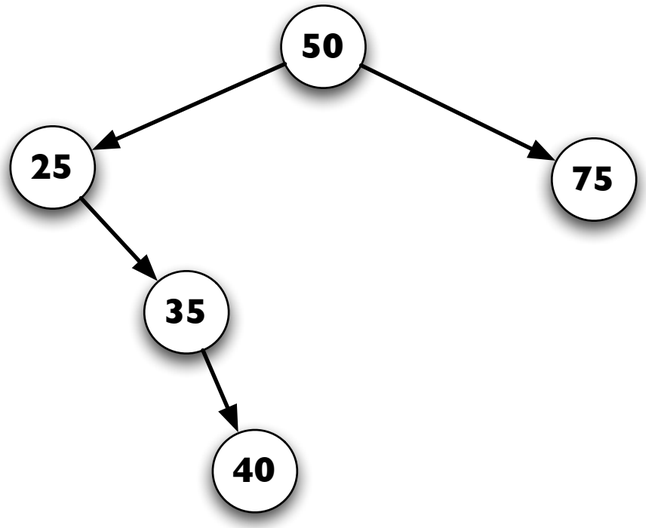


BST



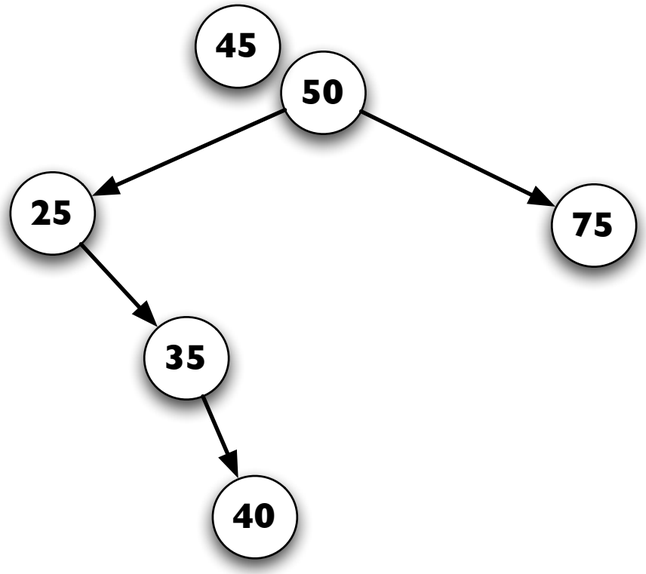
BST

- Insert



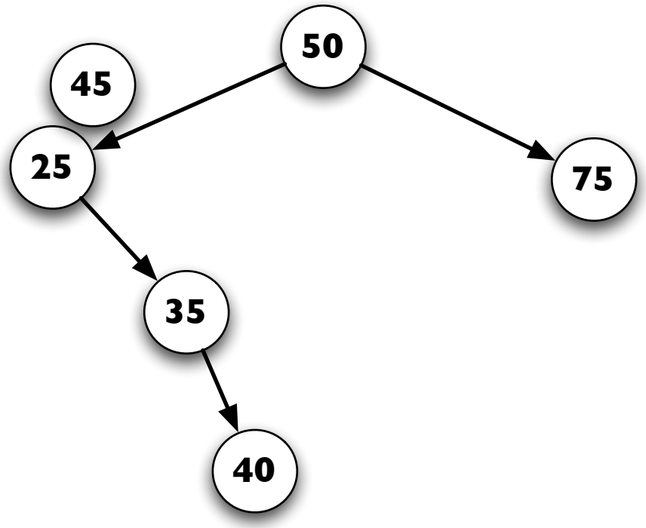
BST

- Insert



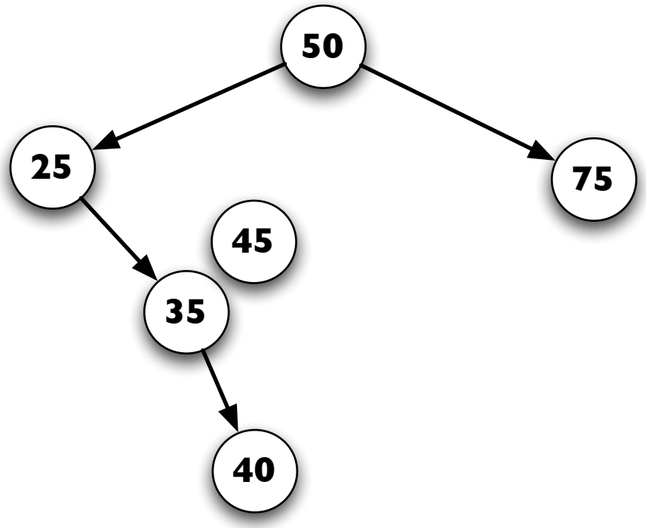
BST

- Insert



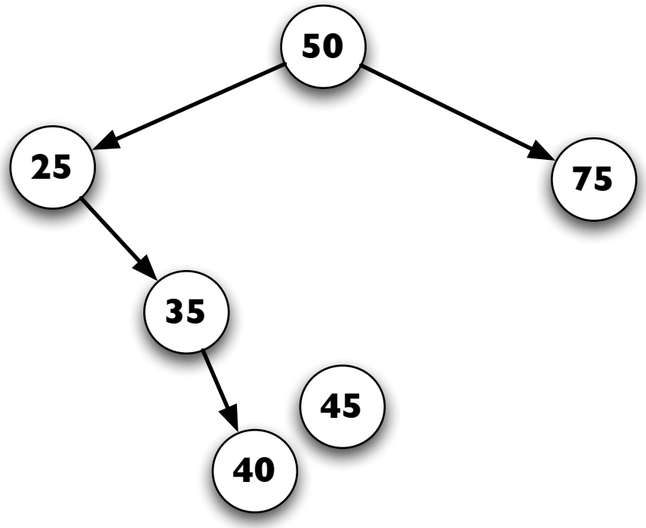
BST

- Insert



BST

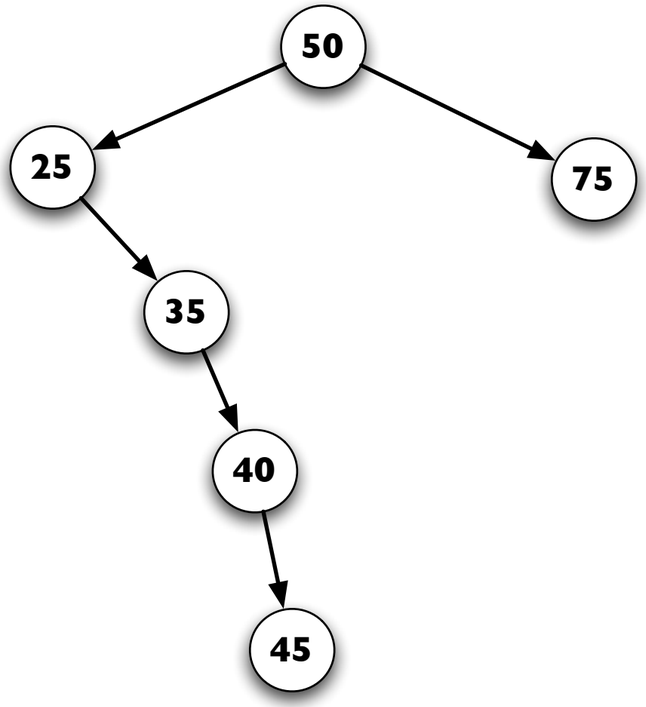
- Insert



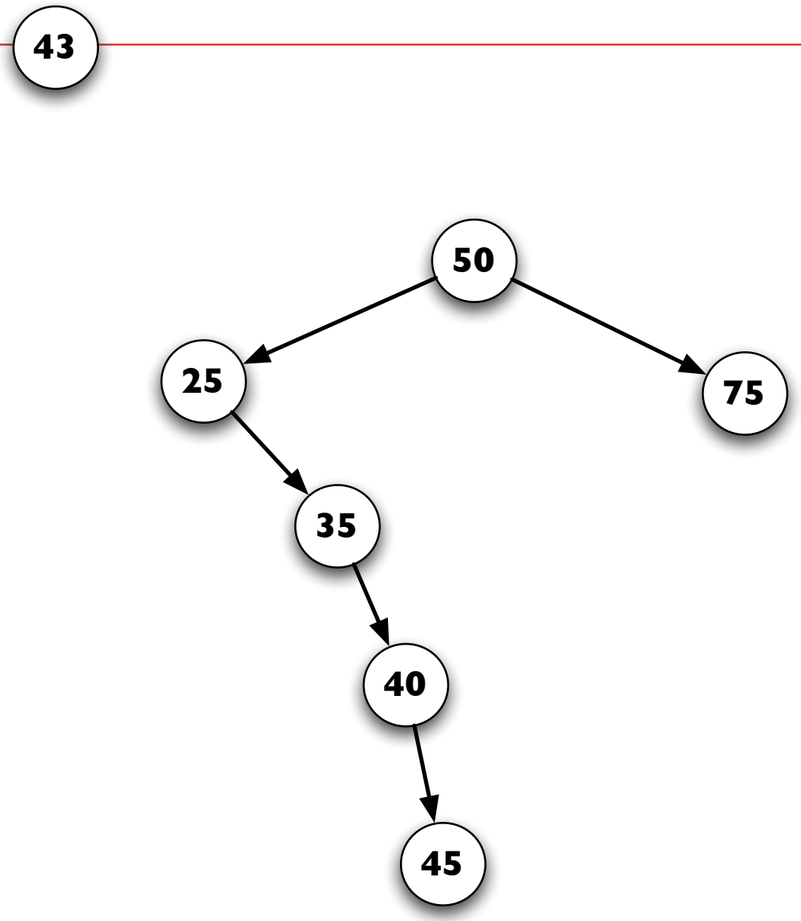
BST

45

- Insert

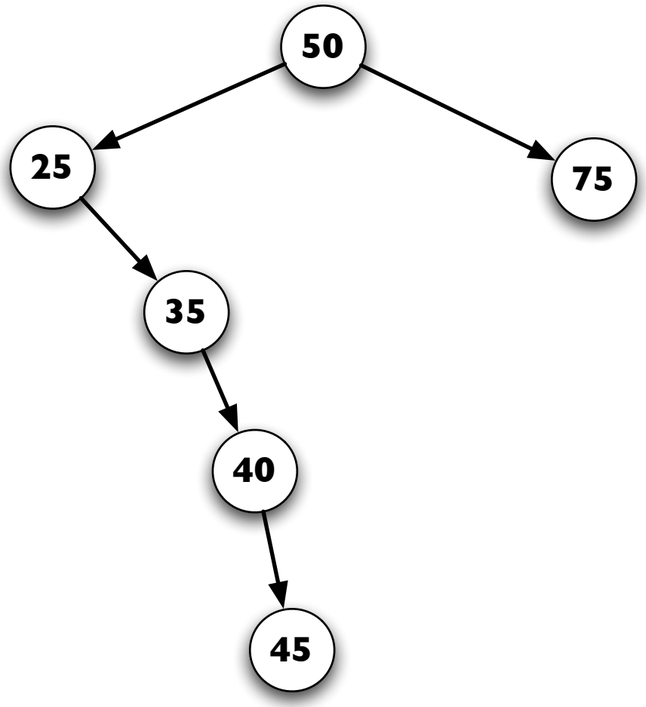


BST



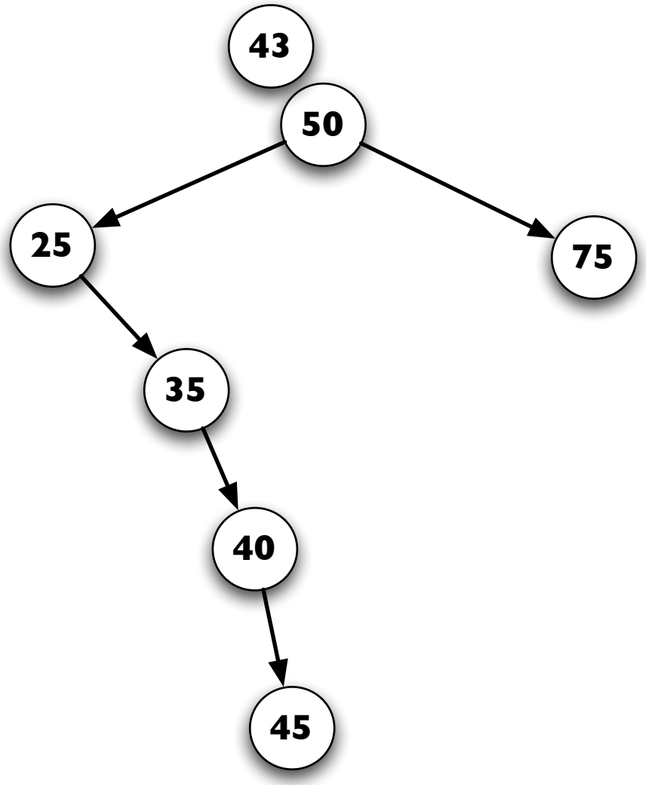
BST

- Insert



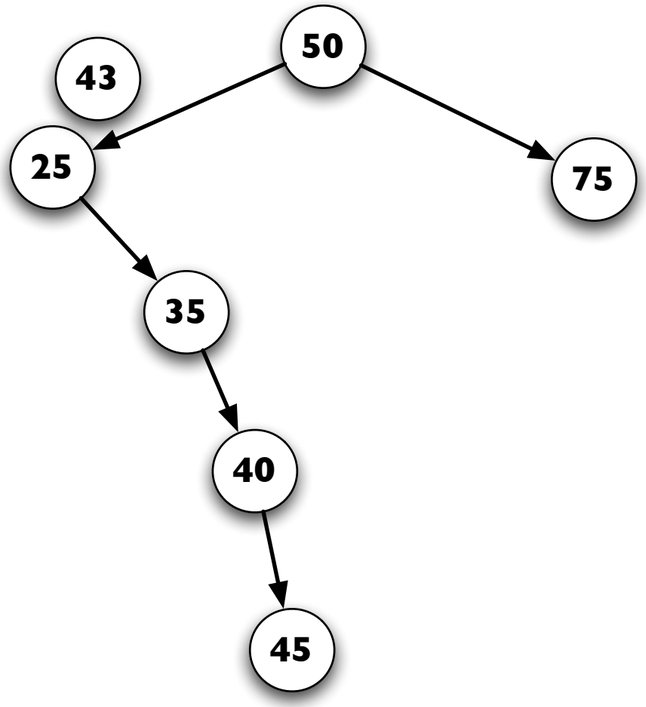
BST

- Insert



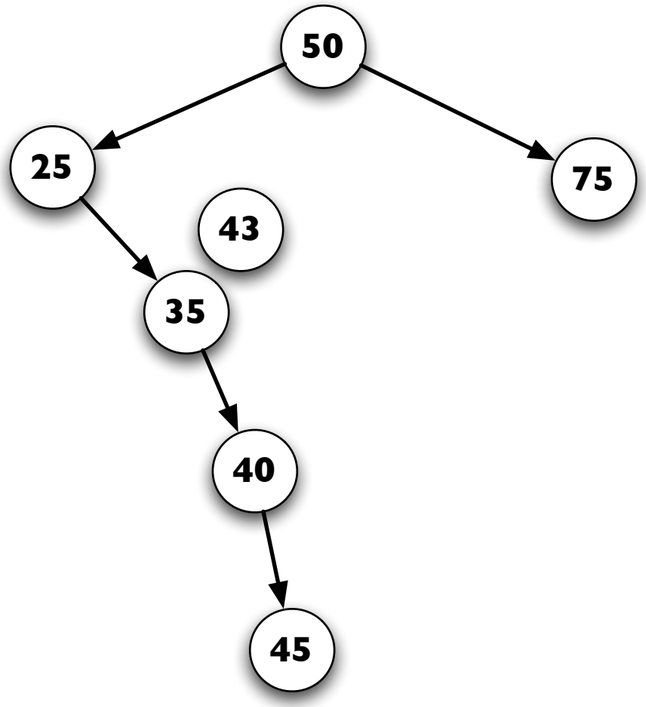
BST

- Insert



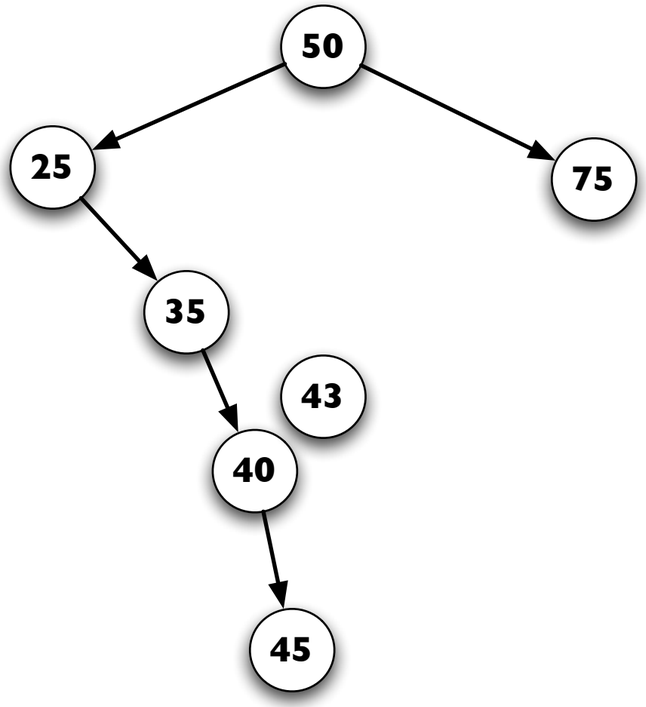
BST

- Insert



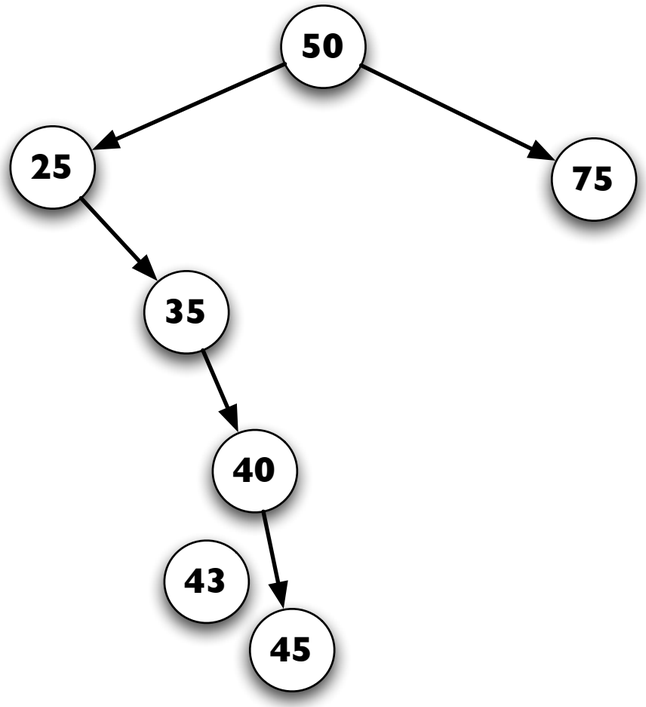
BST

- Insert



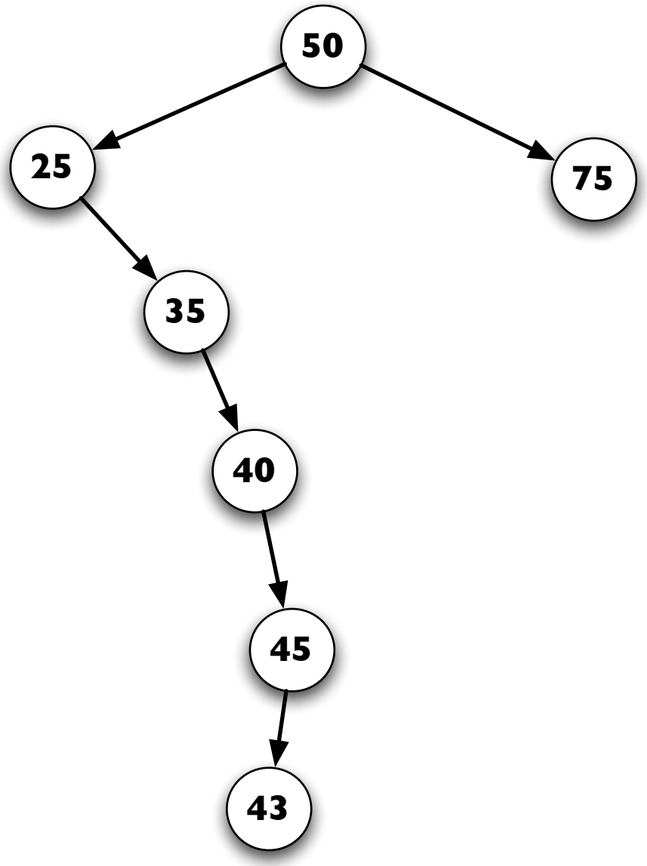
BST

- Insert

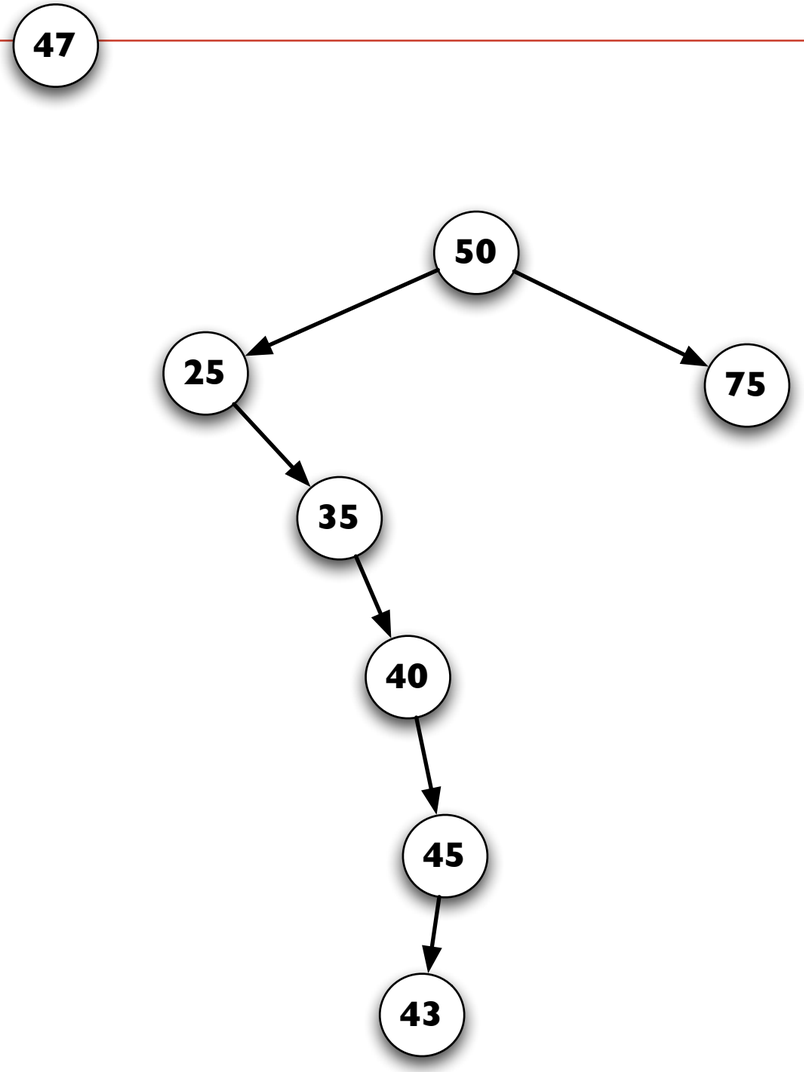


BST

- Insert



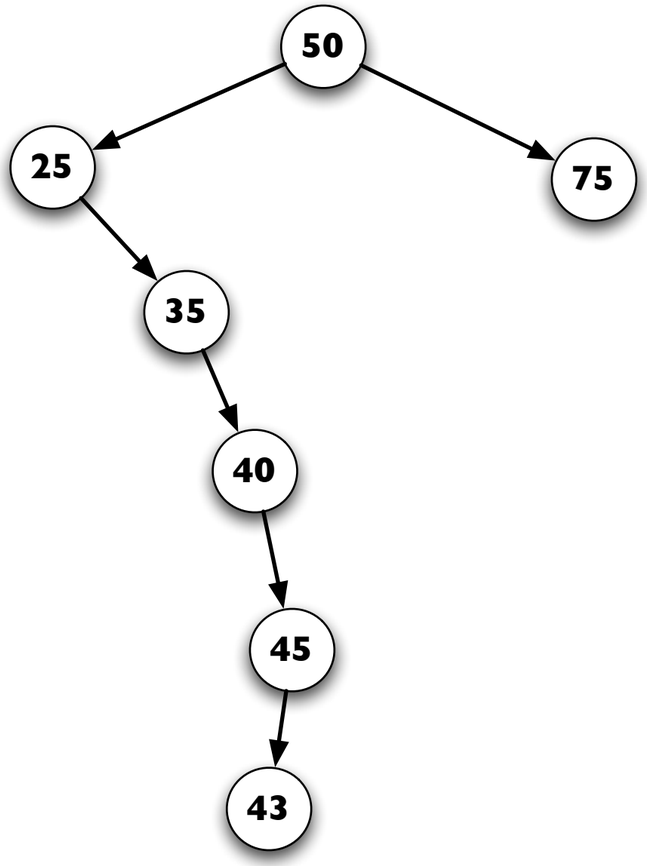
BST



BST

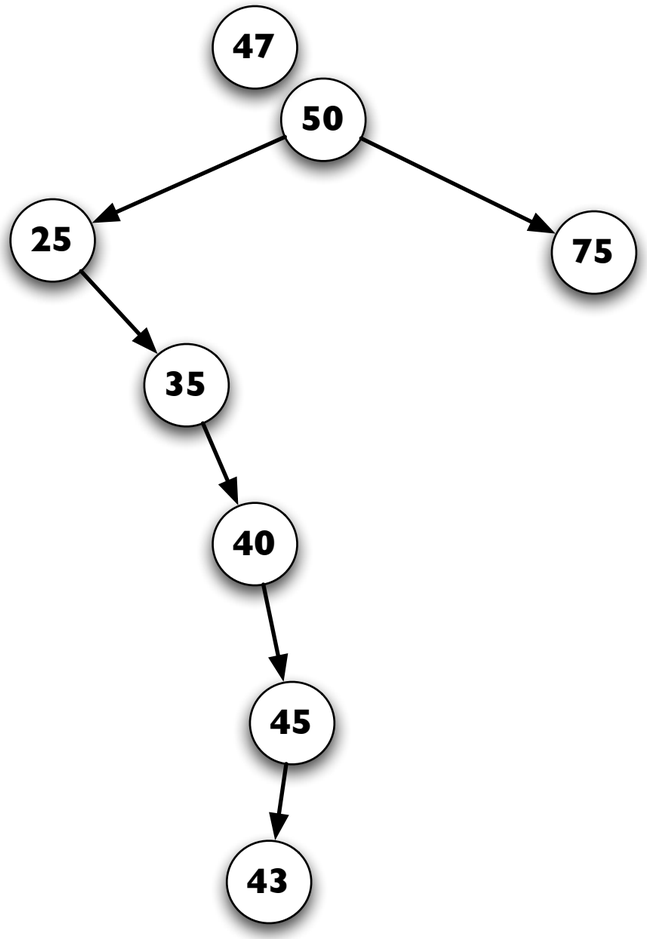
47

- Insert



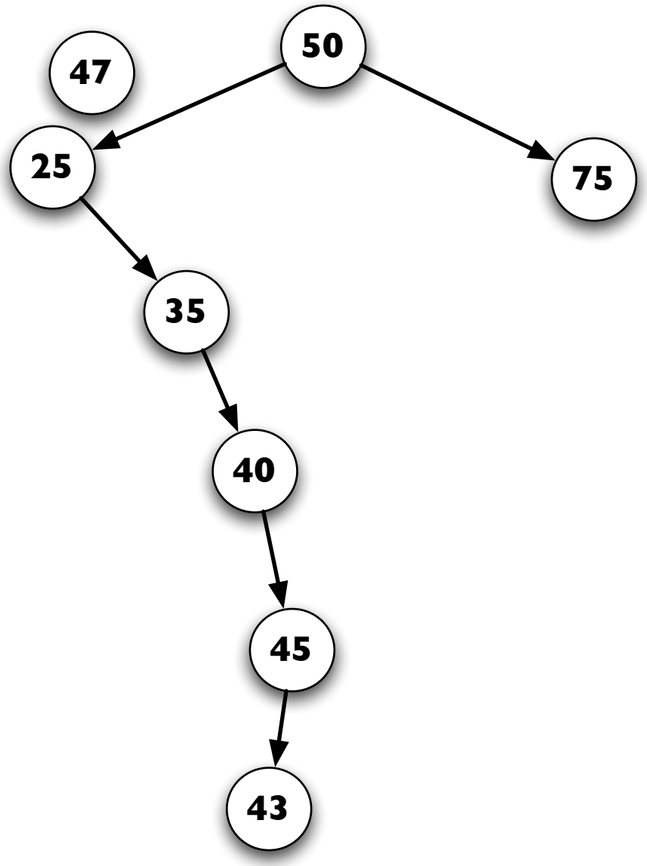
BST

- Insert



BST

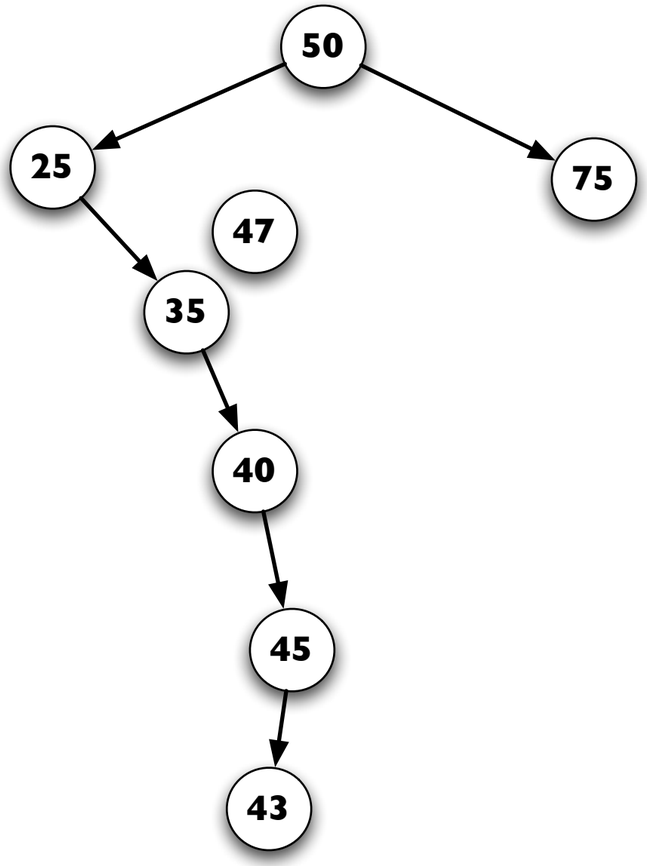
- Insert



BST

47

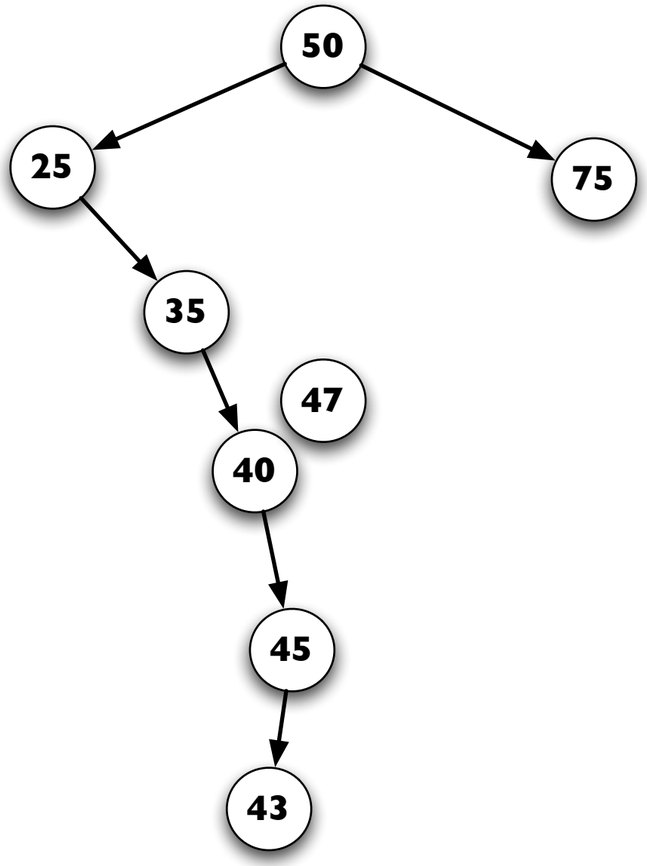
- Insert



BST

47

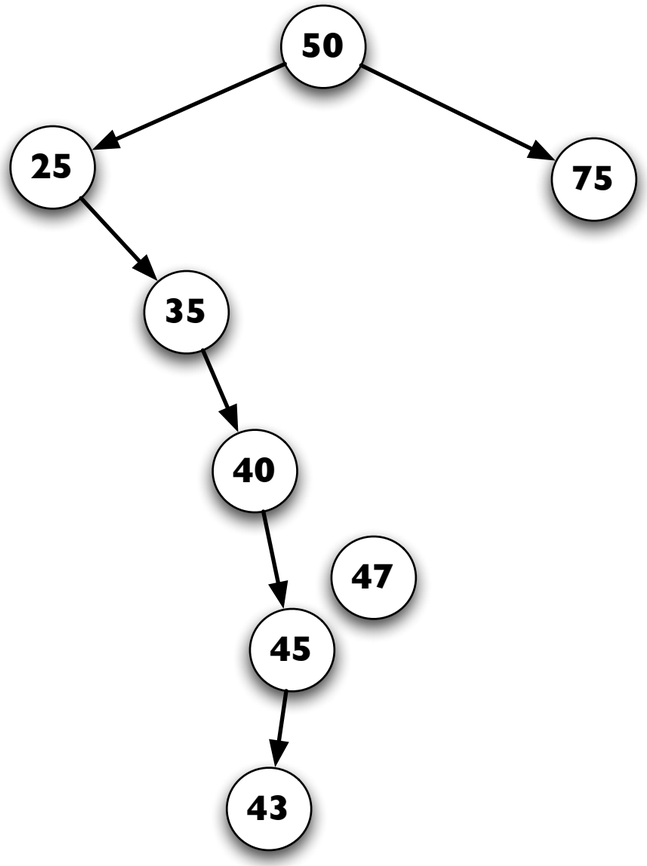
- Insert



BST

47

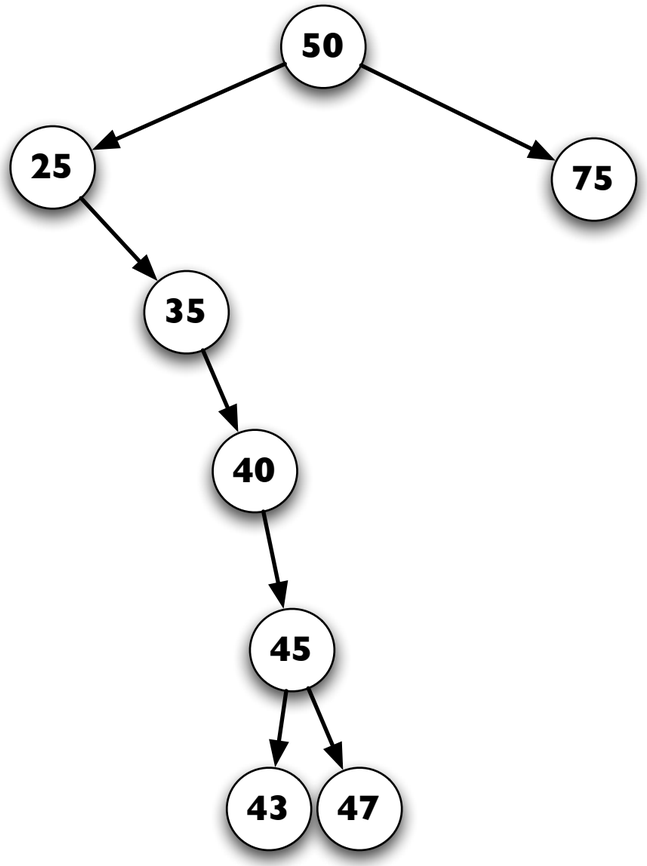
- Insert



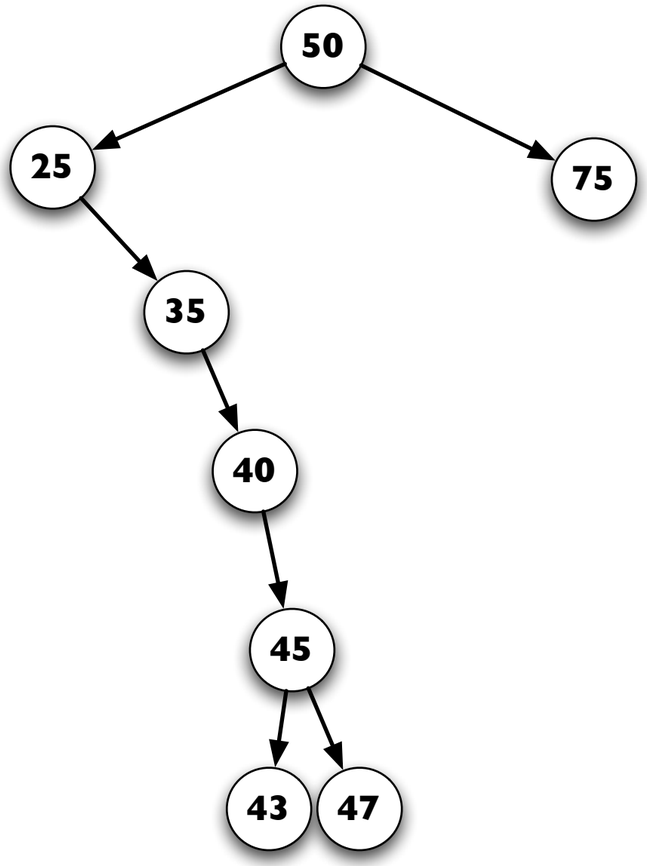
BST

47

- Insert

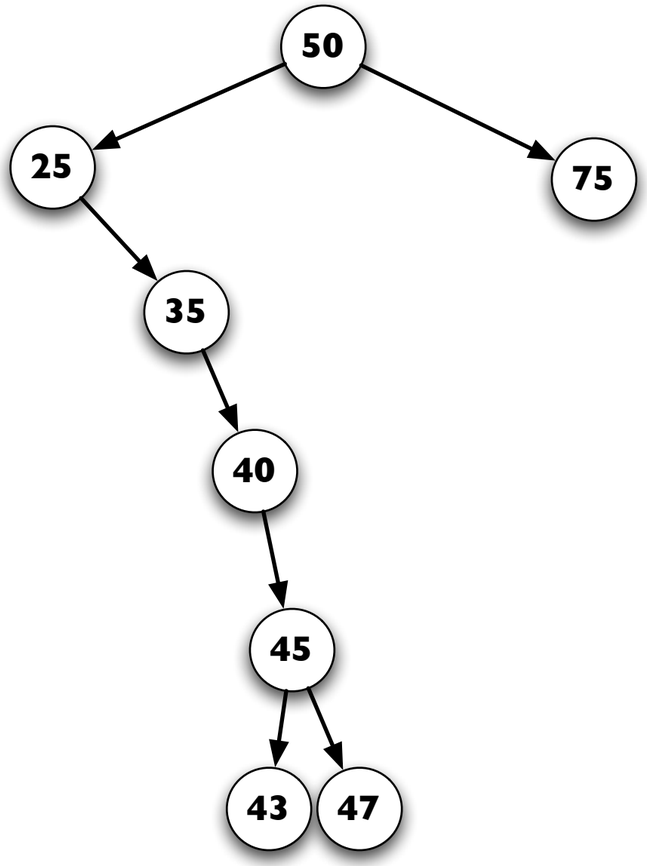


BST



BST

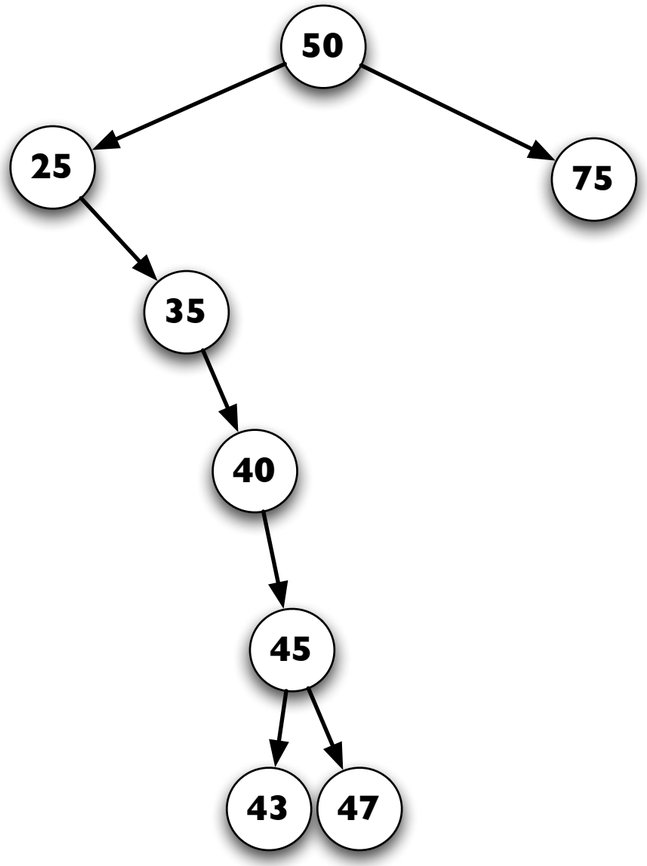
- Insert



BST

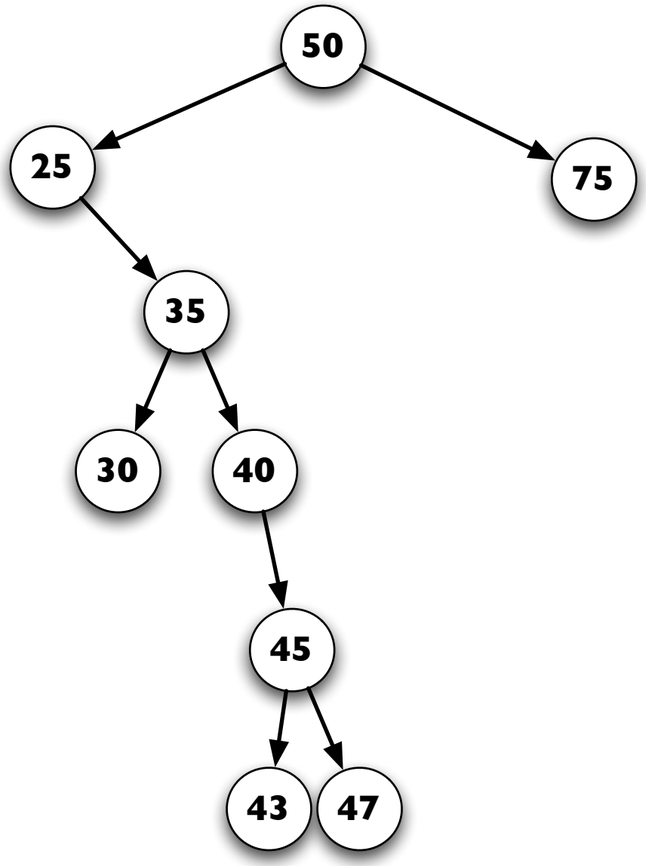
30

- Insert

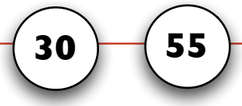


BST

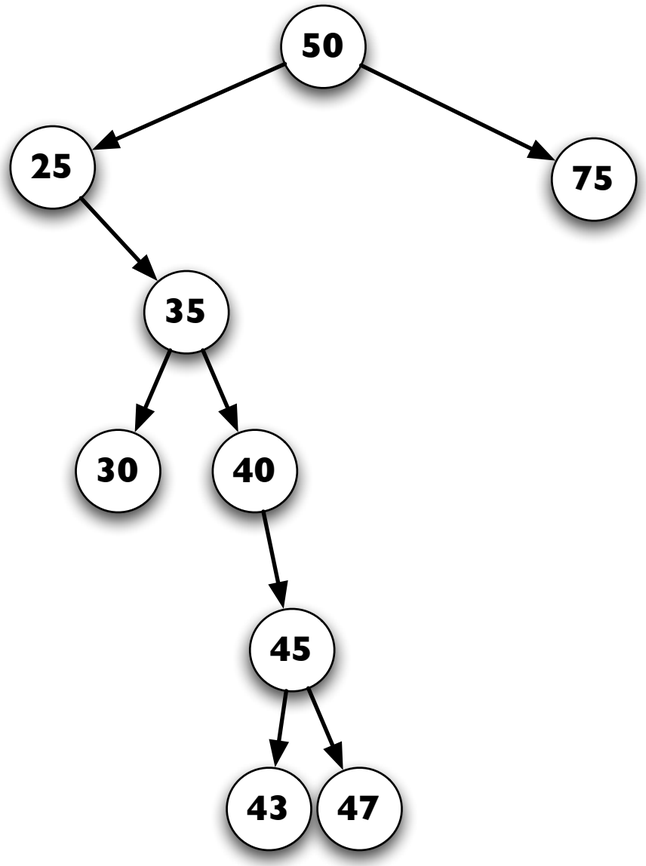
- Insert



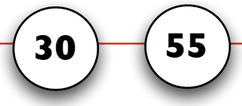
BST



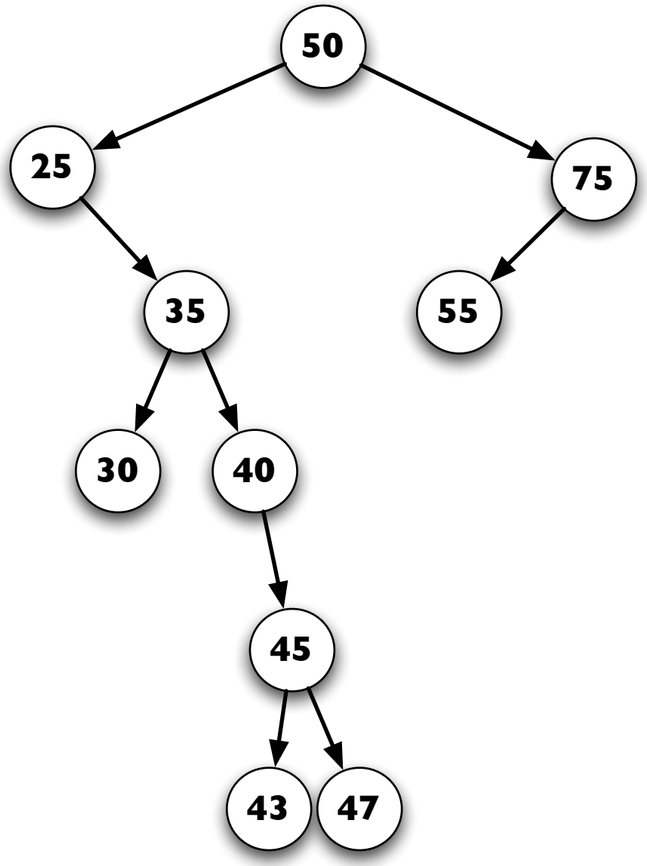
- Insert



BST

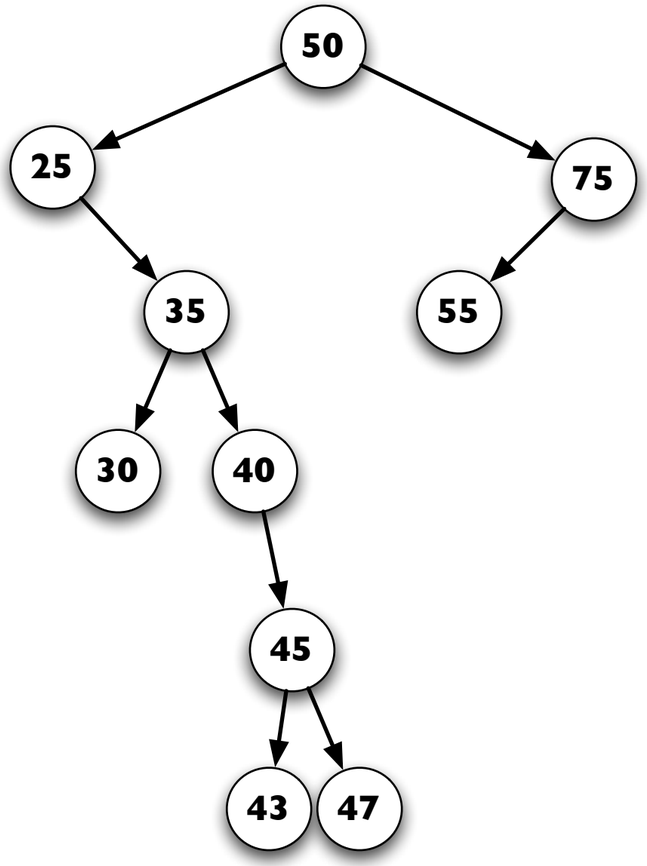


- Insert



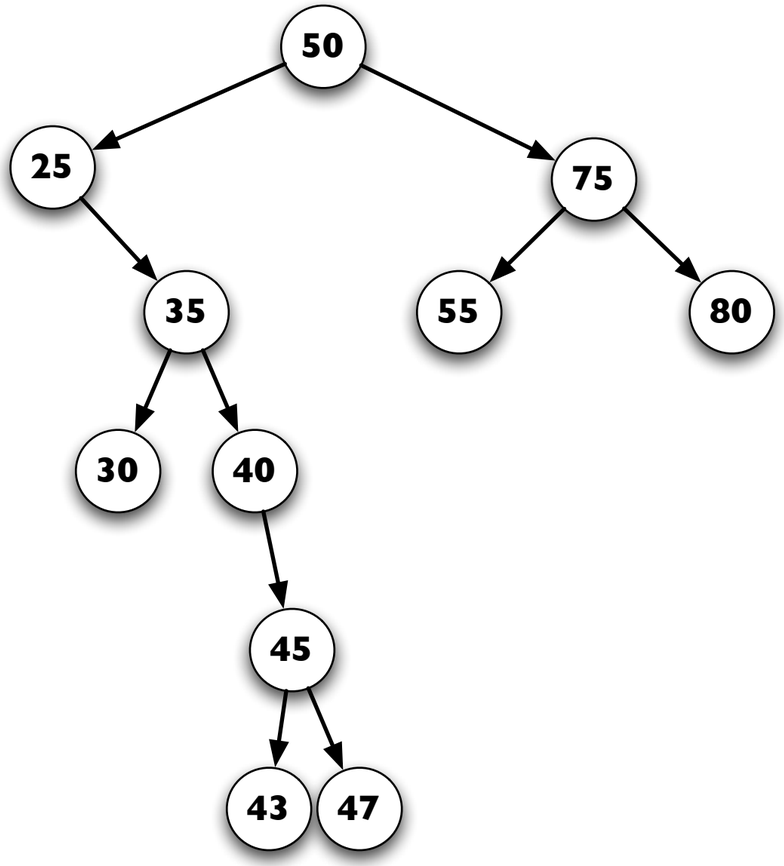
BST

- Insert

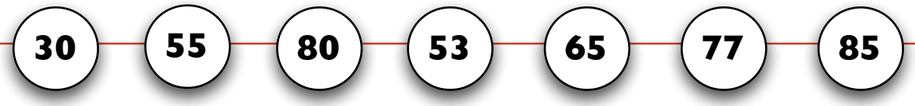


BST

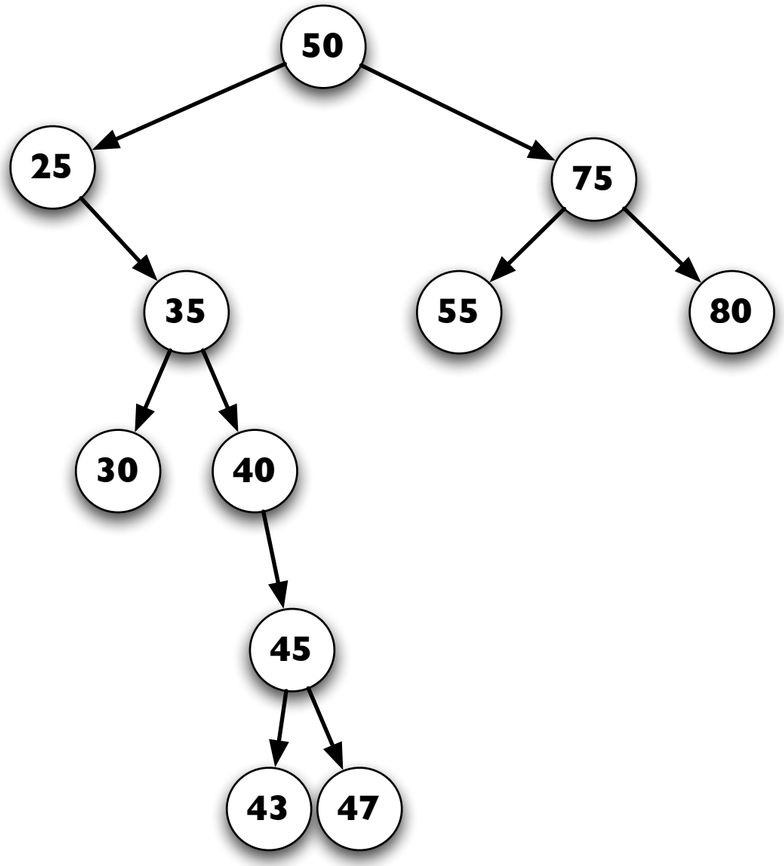
- Insert



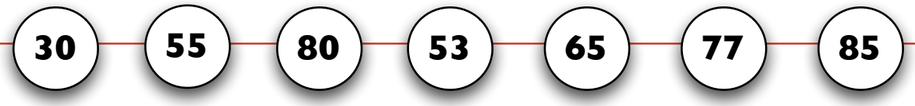
BST



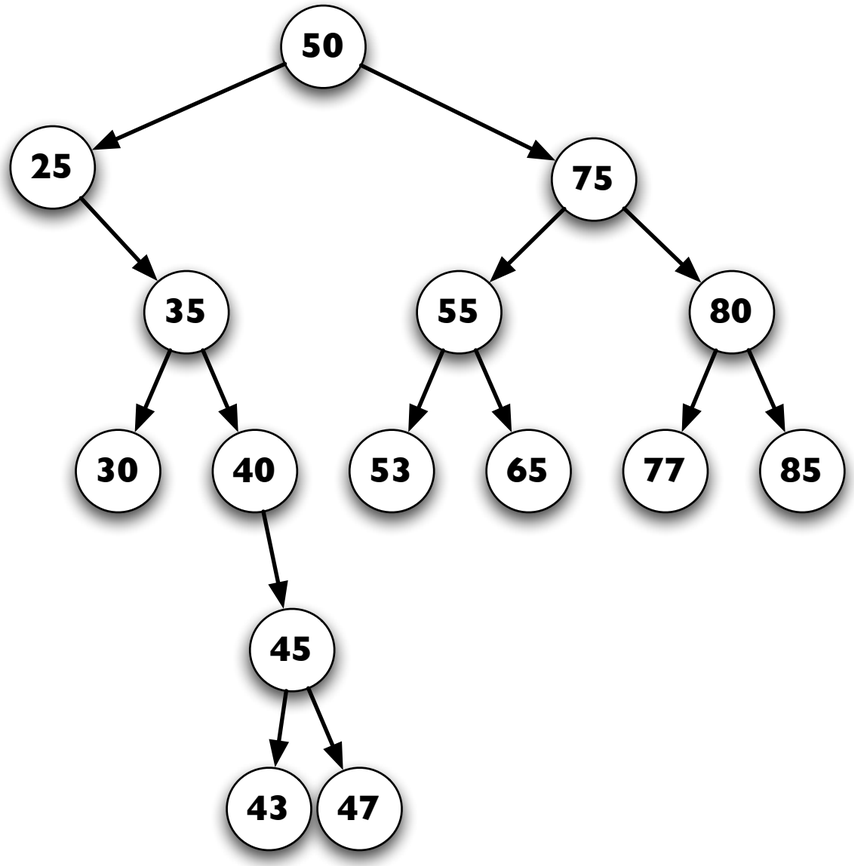
- Insert



BST



- Insert

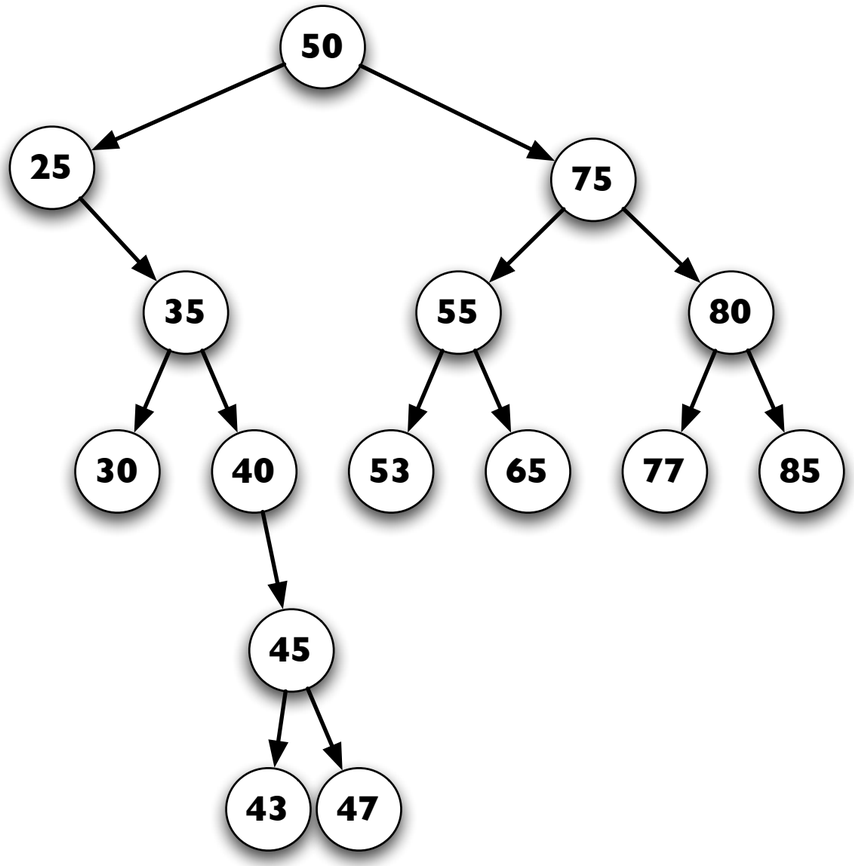


BST

```
if the tree is empty
    return null (no match)
else if the target matches the root node's data
    return the data (a match)
else if the target is less than the root node's data
    return the result of left subtree search
else
    return the result of the right subtree search
```

Searching

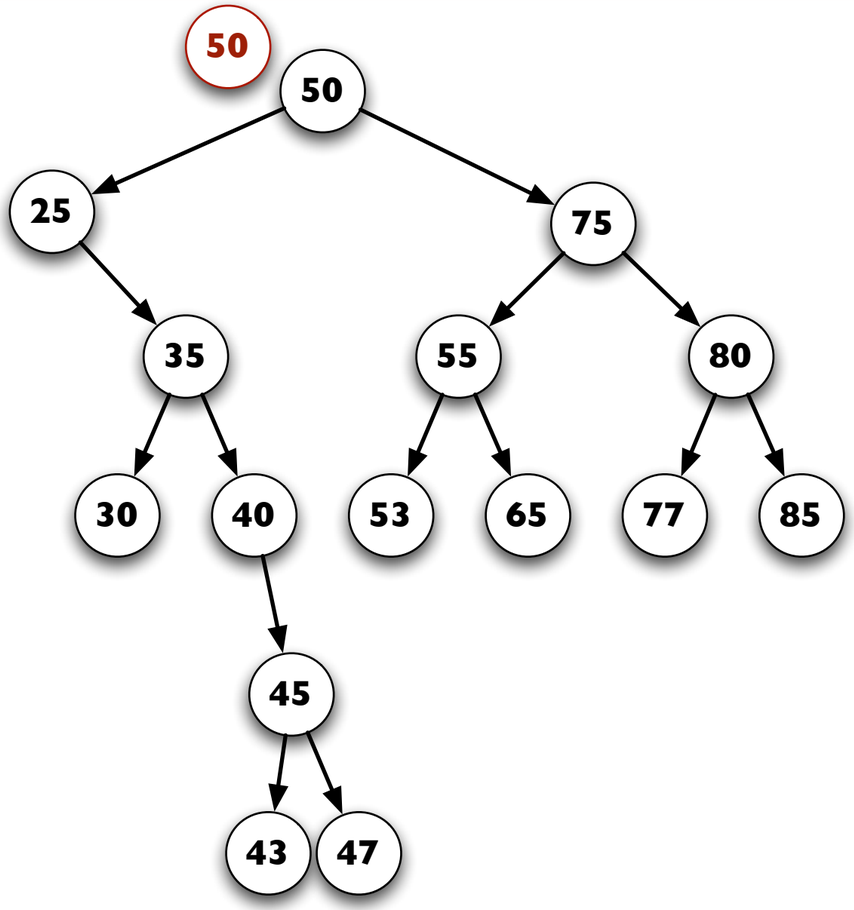
Search For 50



MIAMI UNIVERSITY

Searching

Search For 50

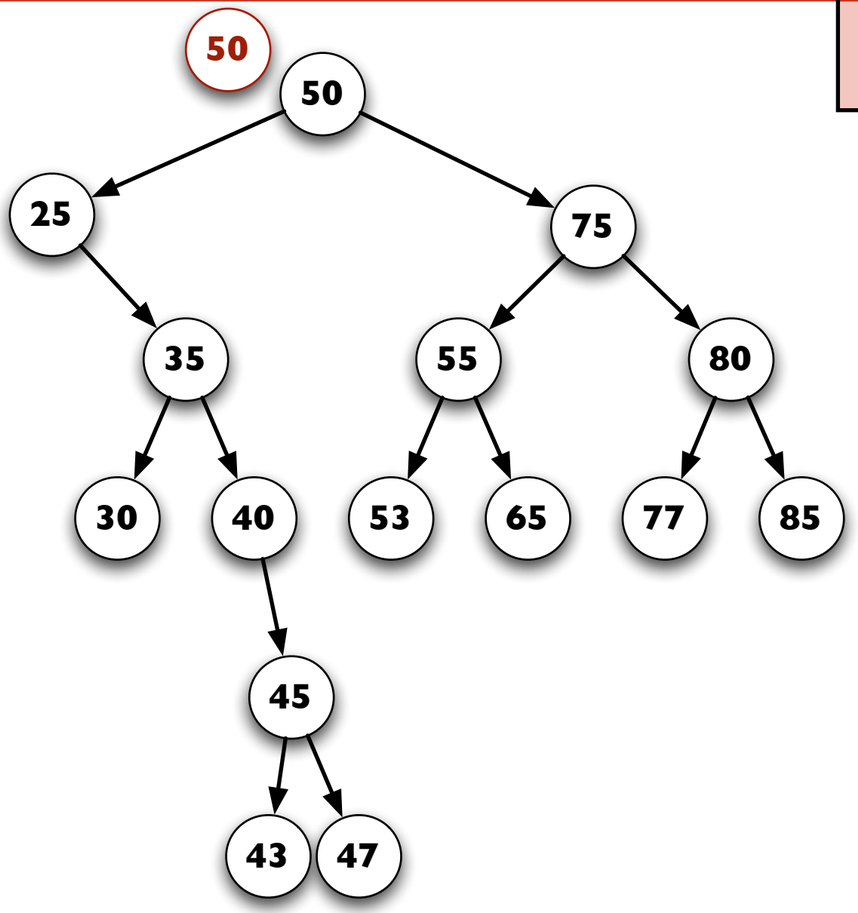


MIAMI UNIVERSITY

Searching

Search For 50

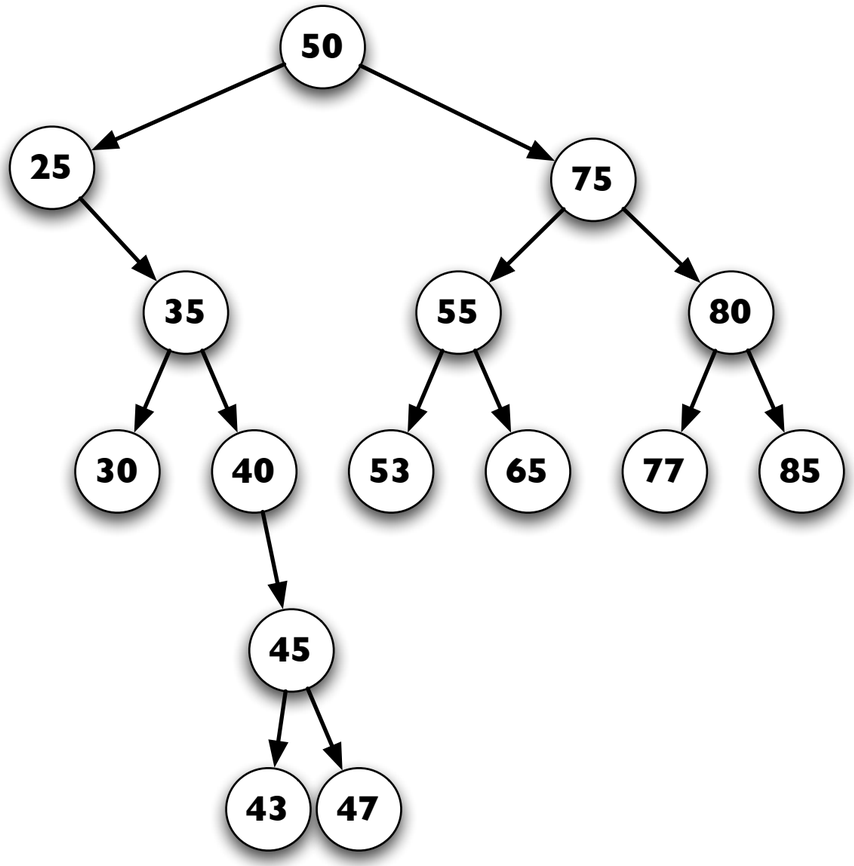
Found!



MIAMI UNIVERSITY

Searching

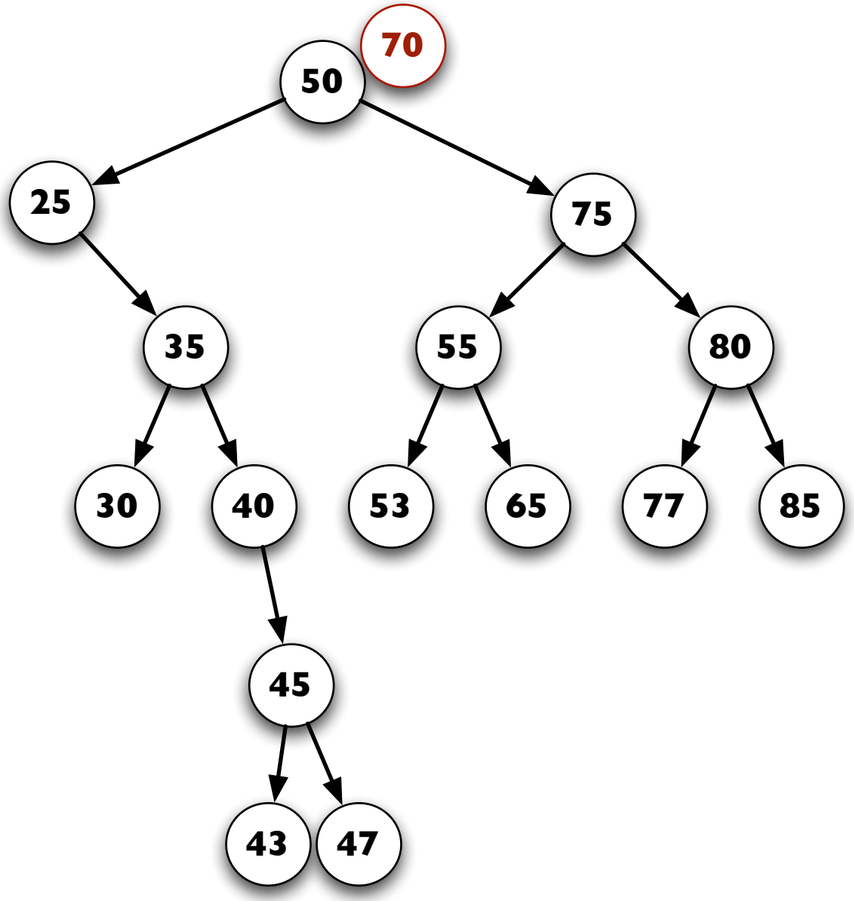
Search For 70



MIAMI UNIVERSITY

Searching

Search For 70

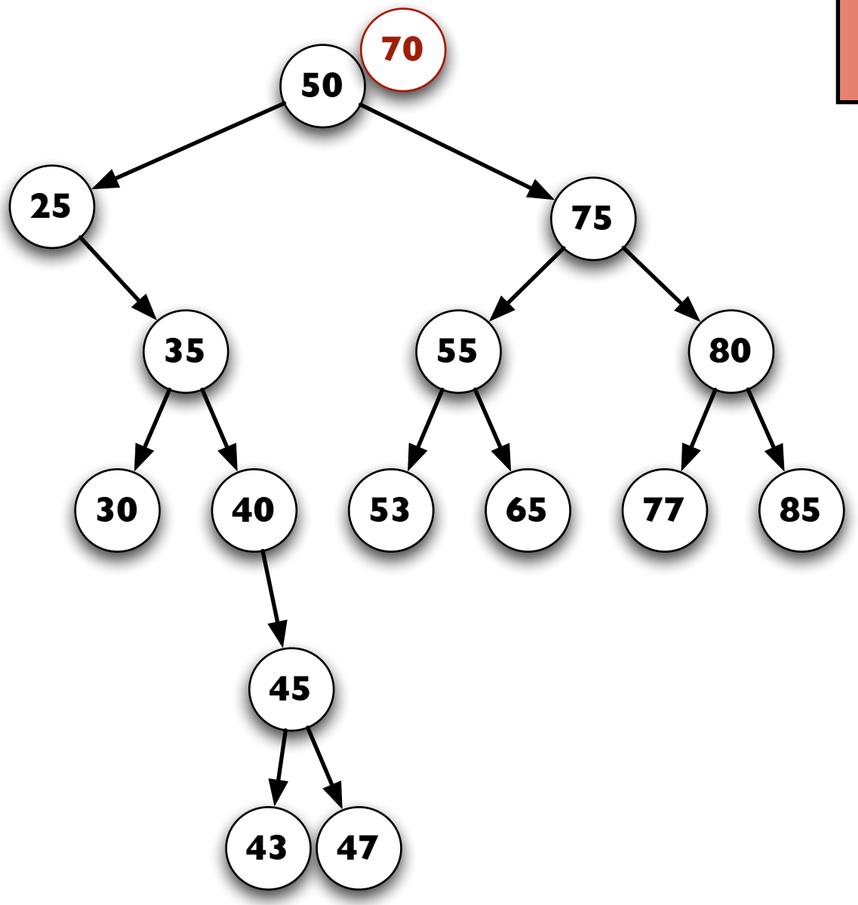


MIAMI UNIVERSITY

Searching

Search For 70

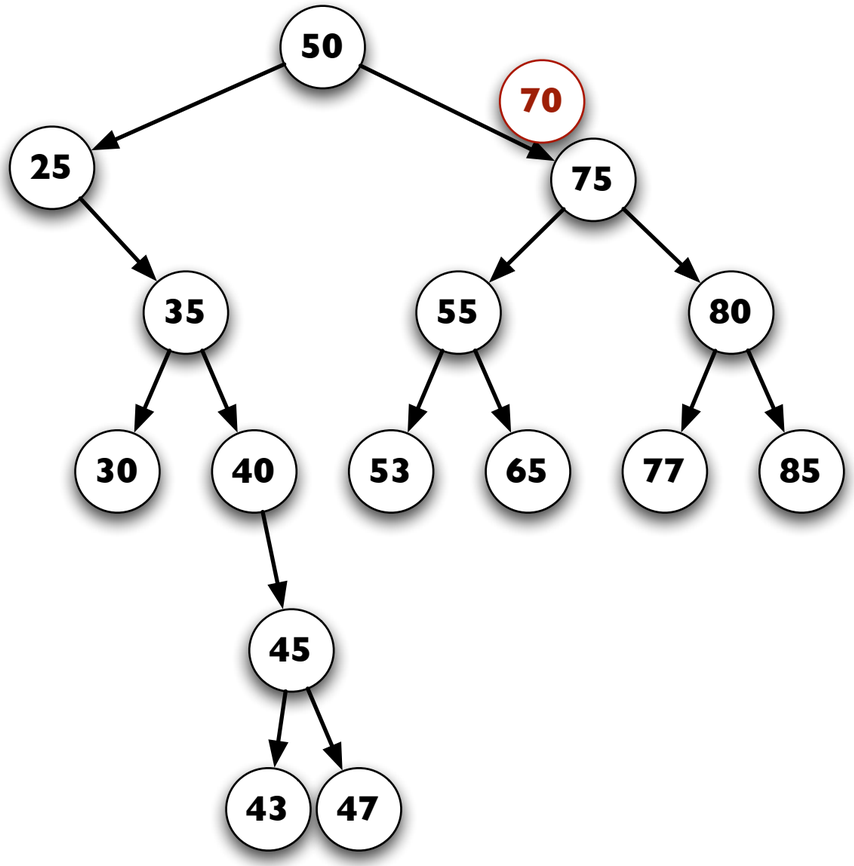
> 50 - TR



MIAMI UNIVERSITY

Searching

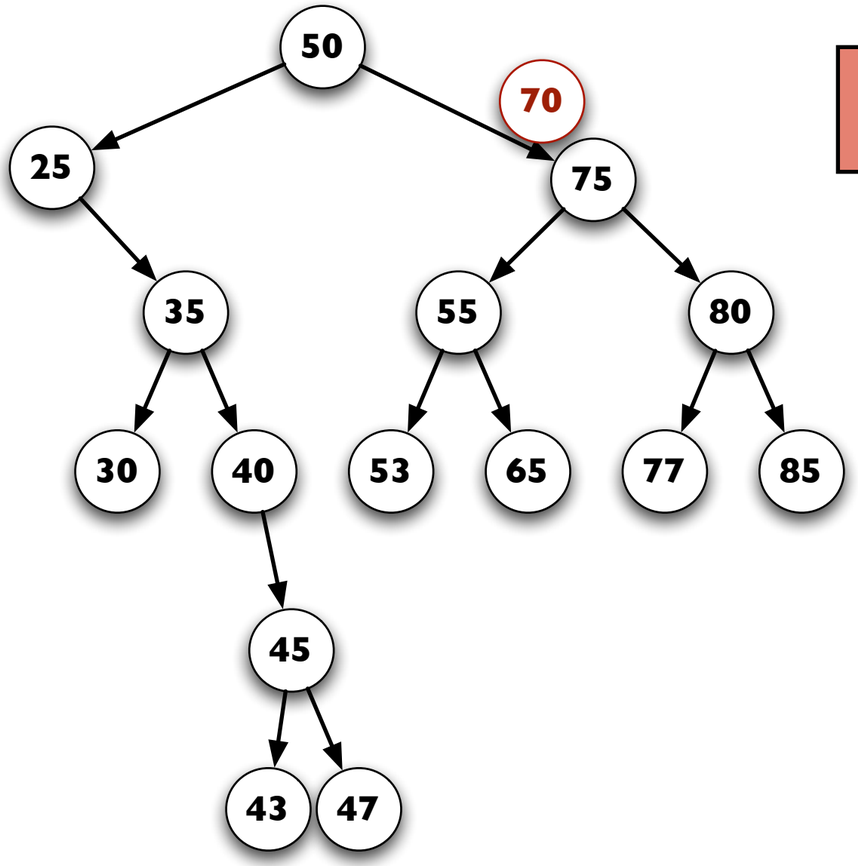
Search For 70



MIAMI UNIVERSITY

Searching

Search For 70

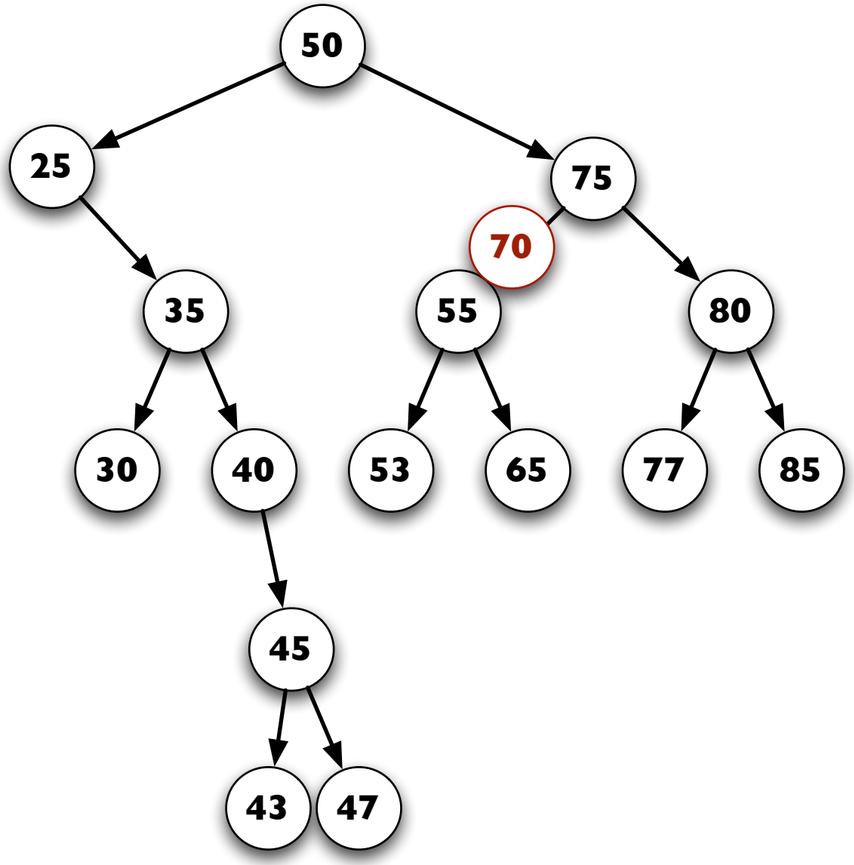


< 75 - TL

MIAMI UNIVERSITY

Searching

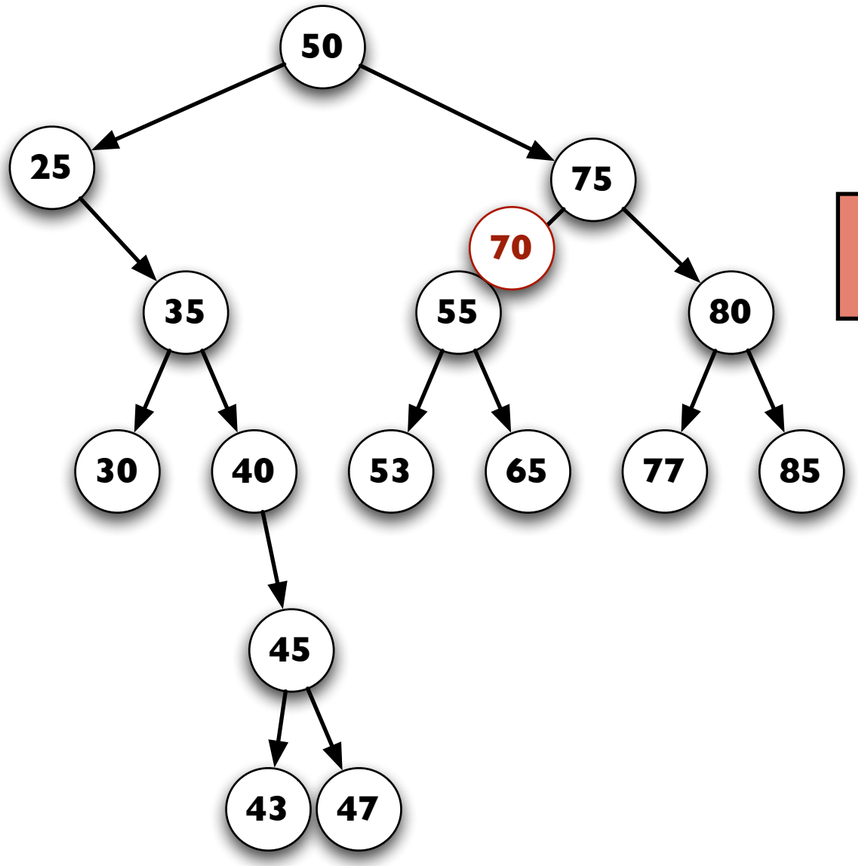
Search For 70



MIAMI UNIVERSITY

Searching

Search For 70

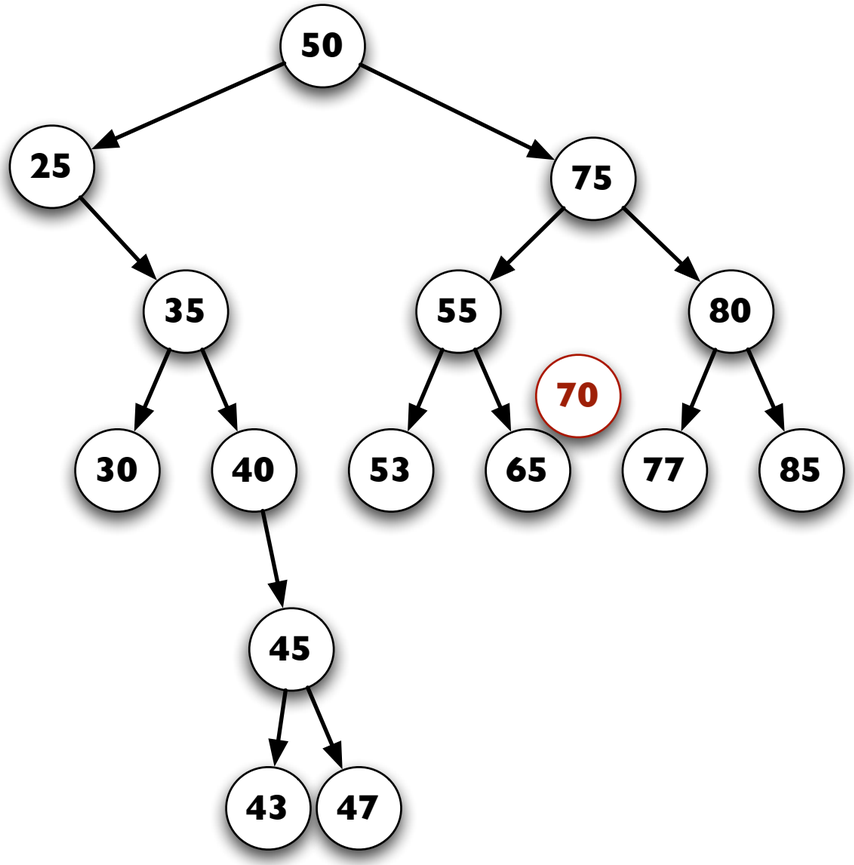


> 55 - TR

MIAMI UNIVERSITY

Searching

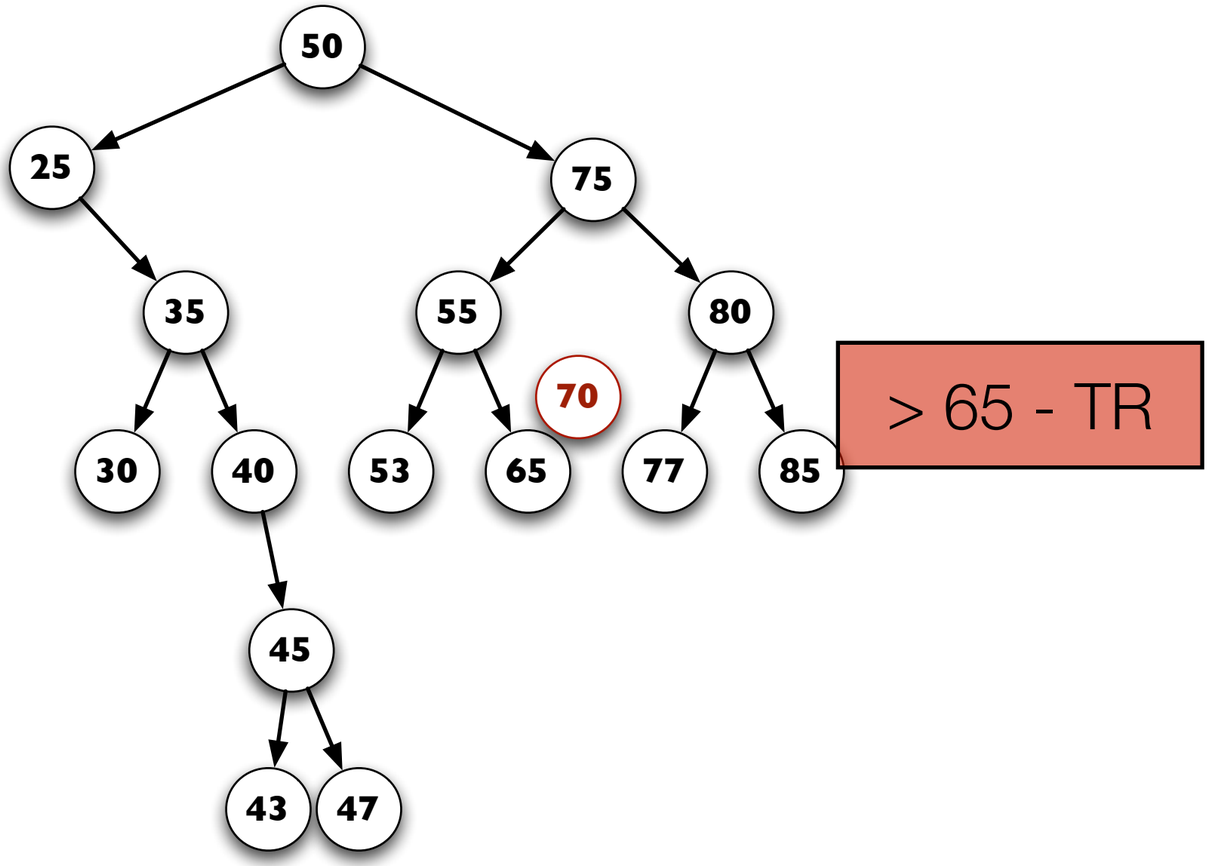
Search For 70



MIAMI UNIVERSITY

Searching

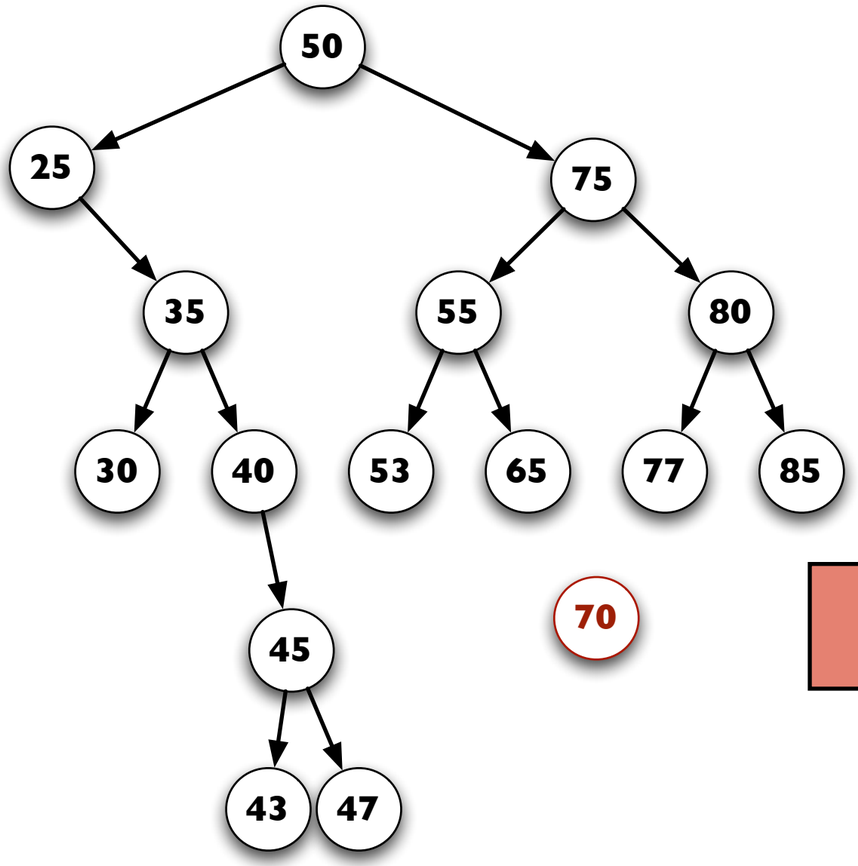
Search For 70



MIAMI UNIVERSITY

Searching

Search For 70



70

Not Found

MIAMI UNIVERSITY

Performance

Performance

- The average performance of operations on the BST is O(lg n)

Performance

- The average performance of operations on the BST is O(lg n)
    - includes searching and inserting

Performance

- The average performance of operations on the BST is O(lg n)
    - includes searching and inserting
- This is not the case with binary search on an array

Performance

- The average performance of operations on the BST is O(lg n)
    - includes searching and inserting
- This is not the case with binary search on an array
    - you would have to reorganize the array O(n) time

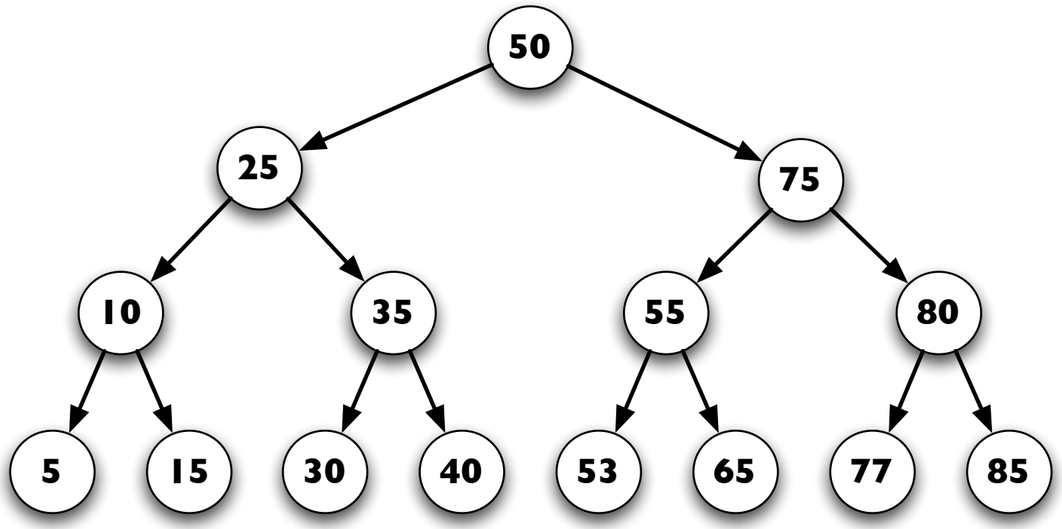
Fullness

Fullness

- A full binary tree is one where all levels of the tree contain their maximum number of nodes

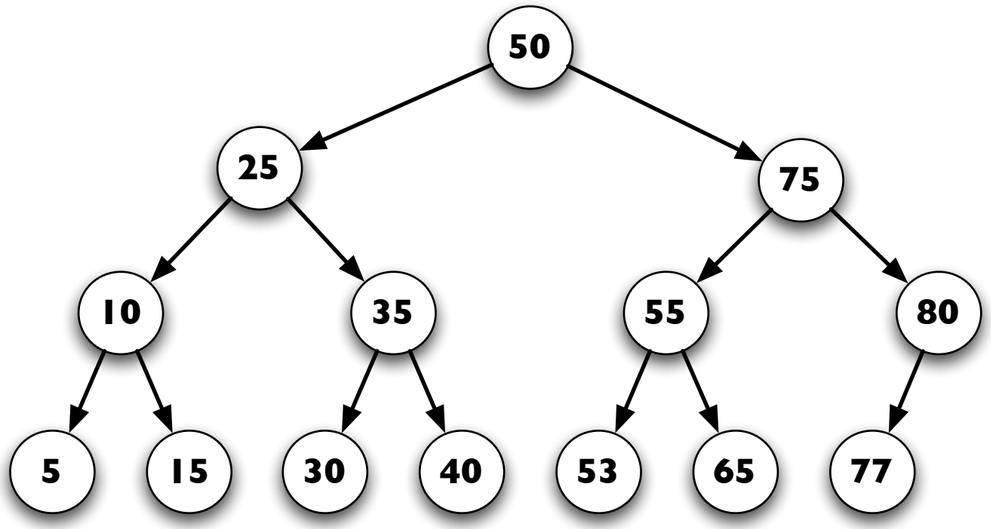
Fullness

- A full binary tree is one where all levels of the tree contain their maximum number of nodes



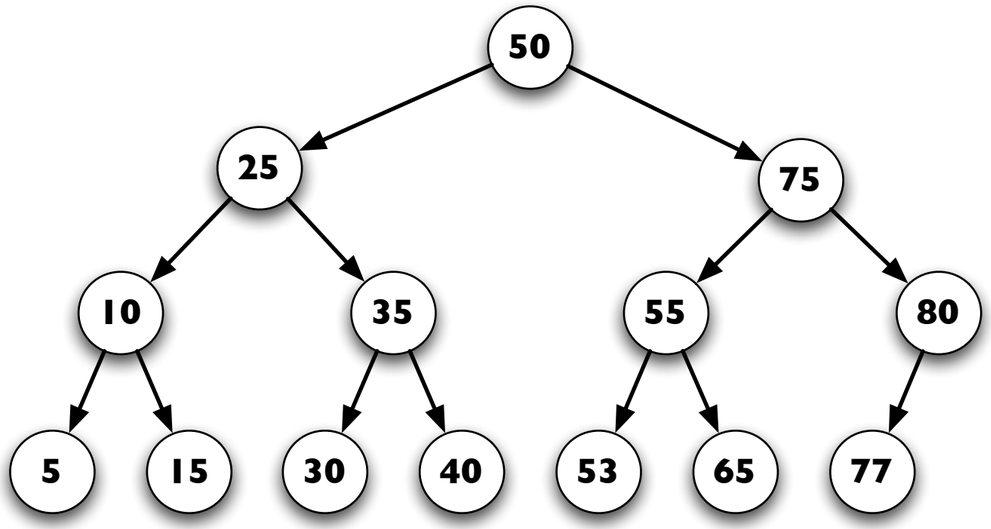
A full binary tree of height 4

Completeness



Completeness

- A complete binary tree is one where the only level that is not full is the last level (i.e. h-1 is not full)



A complete binary tree of height 4

Completion

Completion

- all nodes at level h-2 and above have 2 children

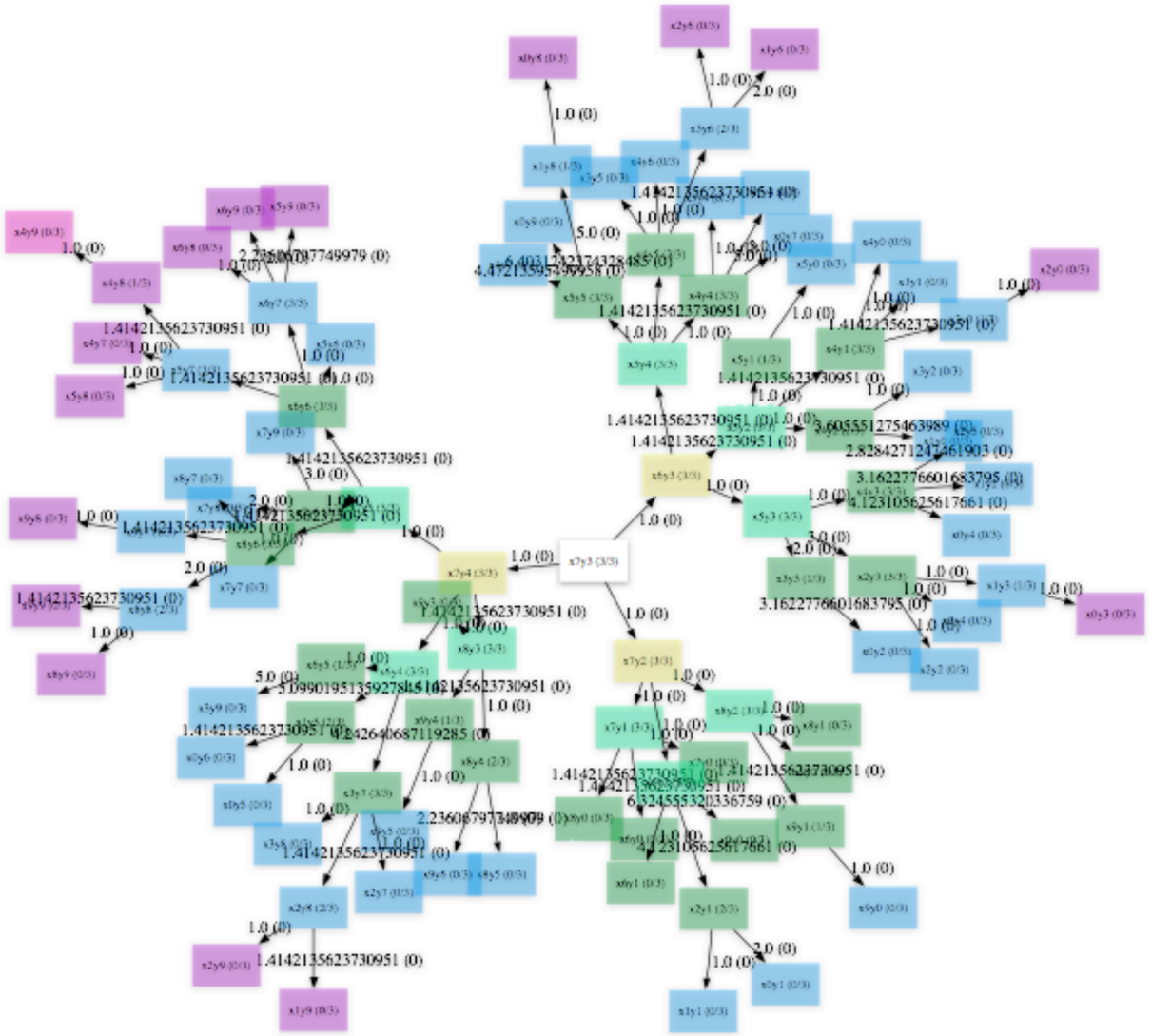
Completion

- all nodes at level h-2 and above have 2 children
- When a node at level h-1 has children, all nodes to the left of it have two children

Completion

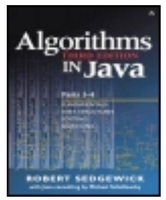
- all nodes at level h-2 and above have 2 children
- When a node at level h-1 has children, all nodes to the left of it have two children
- If a node at level h-1 has one child, it is a left child

Mathematical Properties



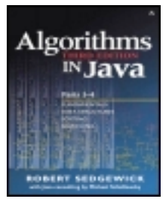
MIAMI UNIVERSITY

Leaf Nodes



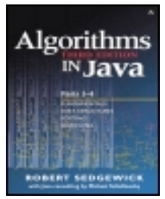
Leaf Nodes

- “A binary tree with N internal nodes has N+1 external nodes.”



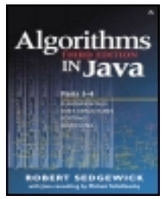
MIAMI UNIVERSITY

Links

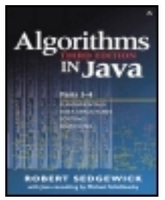


Links

- “A binary tree with N internal nodes has 2N links: N-1 links to internal nodes and N+1 links to external nodes.”

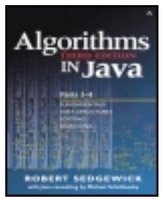


Performance

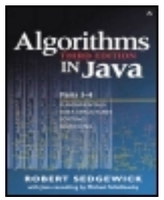


Performance

- “The level of a node in a tree is one higher than the level of its parent (with the root at level 0). The height of a tree is the maximum of the levels of the tree’s nodes. The path length of a tree is the sum of the levels of all the tree’s nodes. The internal path length of a binary tree is the sum of the levels of all the tree’s internal nodes. The external path length of a binary tree is the sum of all the tree’s external nodes.”

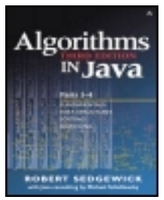


Height

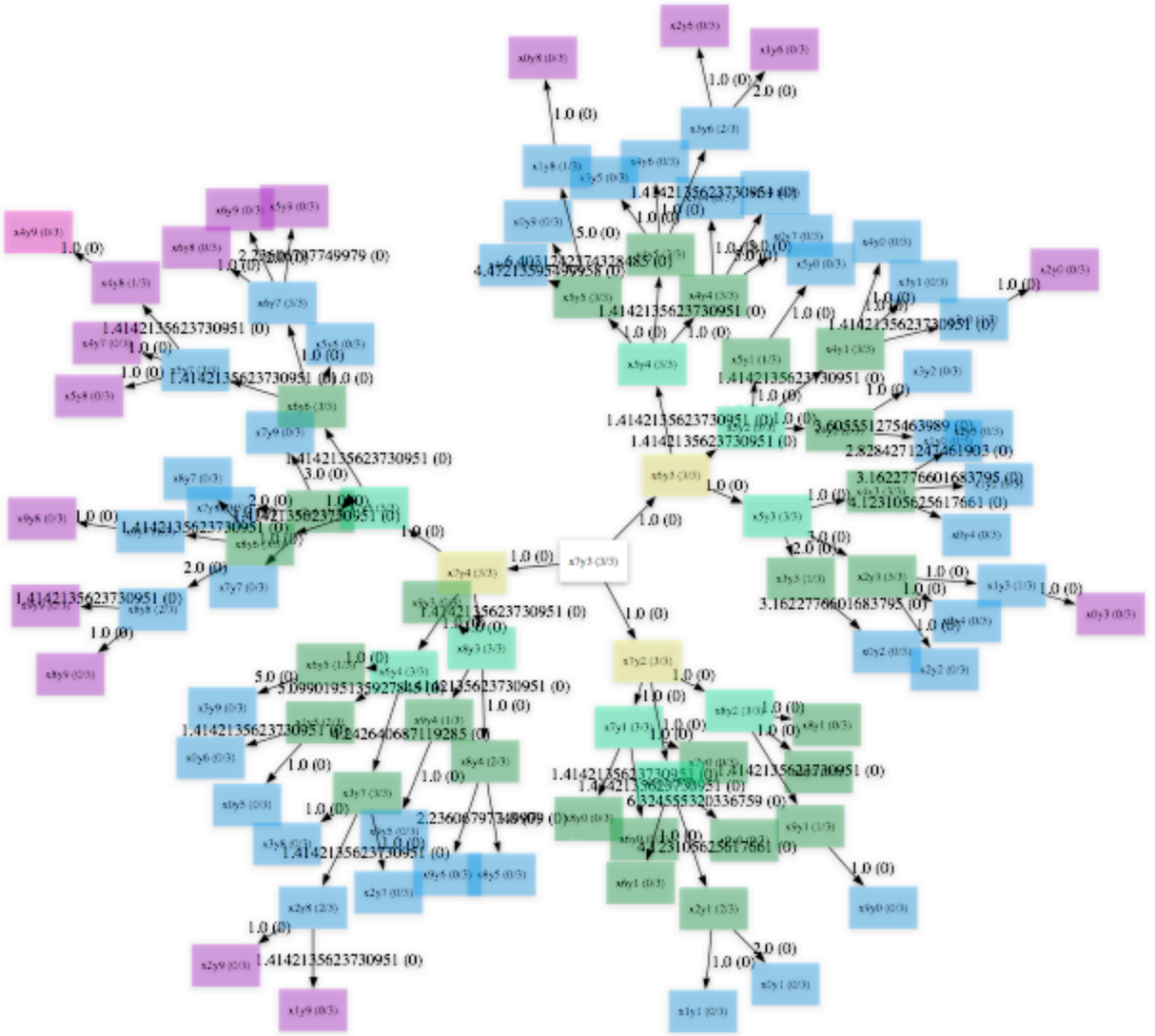


Height

- “The height of a binary tree with N internal nodes is at least lg N and at most N - 1.”



Traversals



MIAMI UNIVERSITY

Traversals

Traversals

- Preorder

Traversals

- Preorder
- Inorder

Traversals

- Preorder
- Inorder
- Postorder

Preorder

Preorder

- Visit root node

Preorder

- Visit root node
- Traverse TL

Preorder

- Visit root node
- Traverse TL
- Traverse TR

Inorder

Inorder

- Traverse TL

Inorder

- Traverse TL
- Visit root node

Inorder

- Traverse TL
- Visit root node
- Traverse TR

Postorder

Postorder

- Traverse TL

Postorder

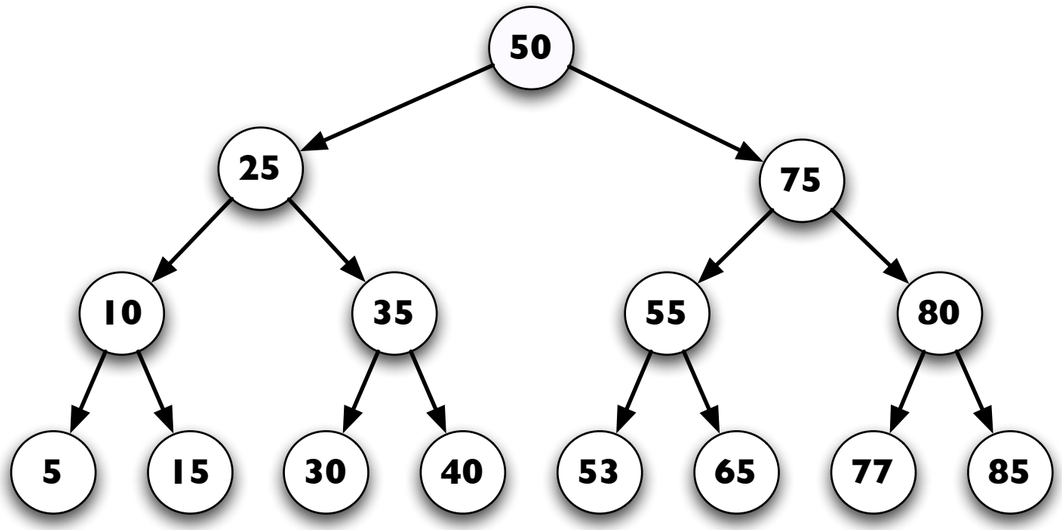
- Traverse TL
- Traverse TR

Postorder

- Traverse TL
- Traverse TR
- Visit root node

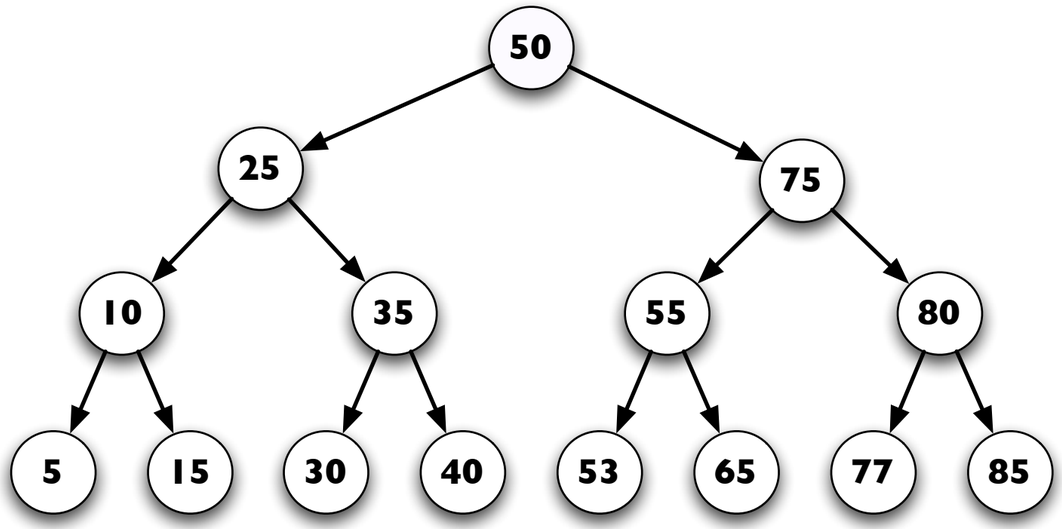
Preorder

Preorder



Preorder

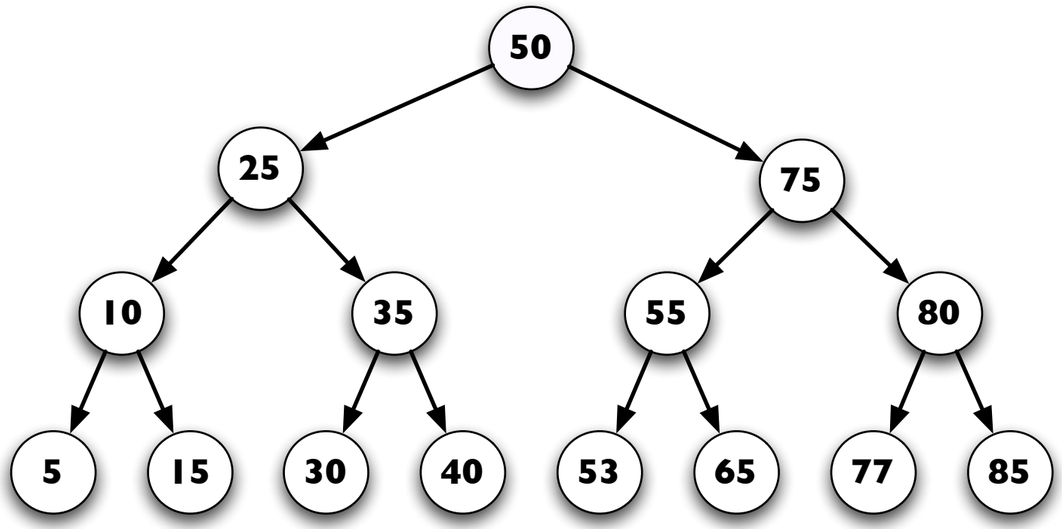
- Visit root node



Preorder

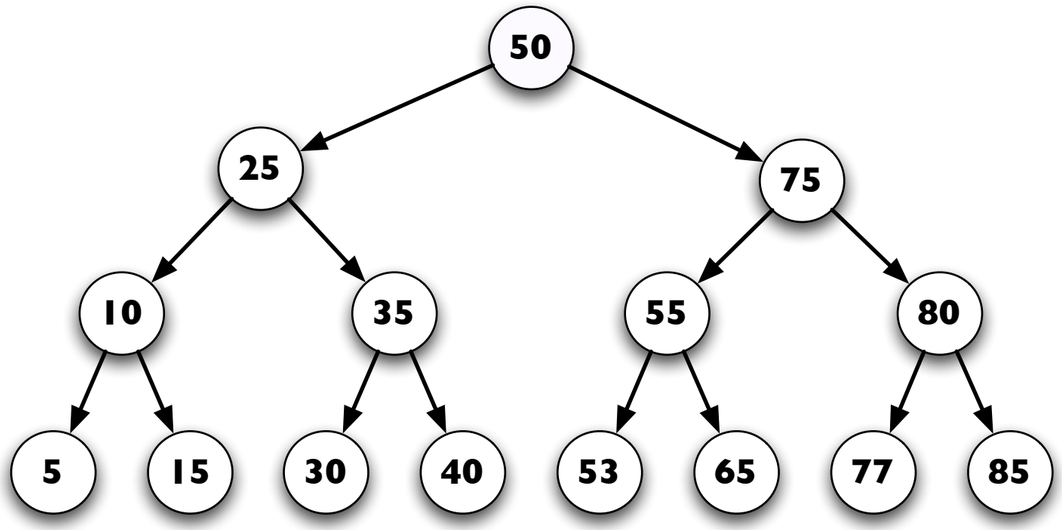
- Visit root node

- Traverse TL

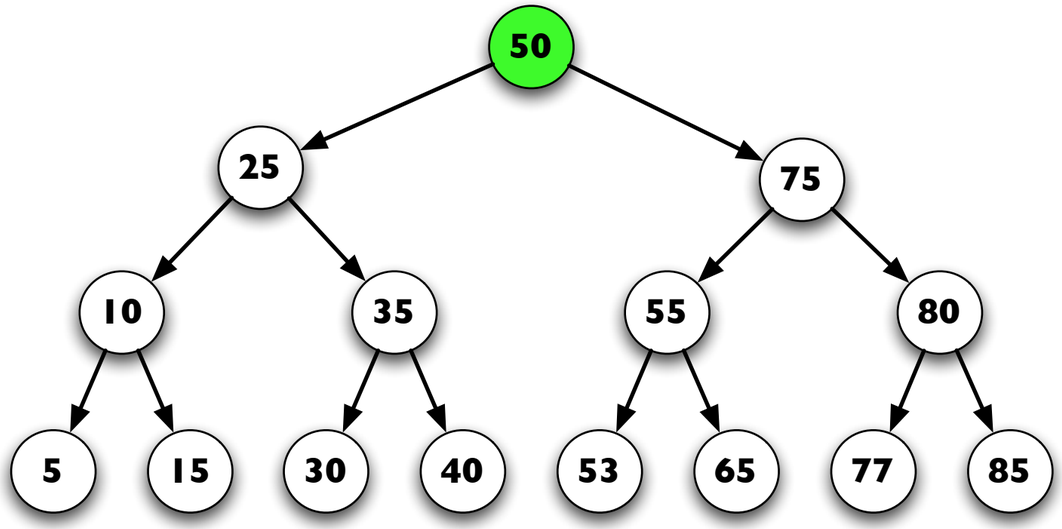


Preorder

- Visit root node
- Traverse TL
- Traverse TR



Preorder



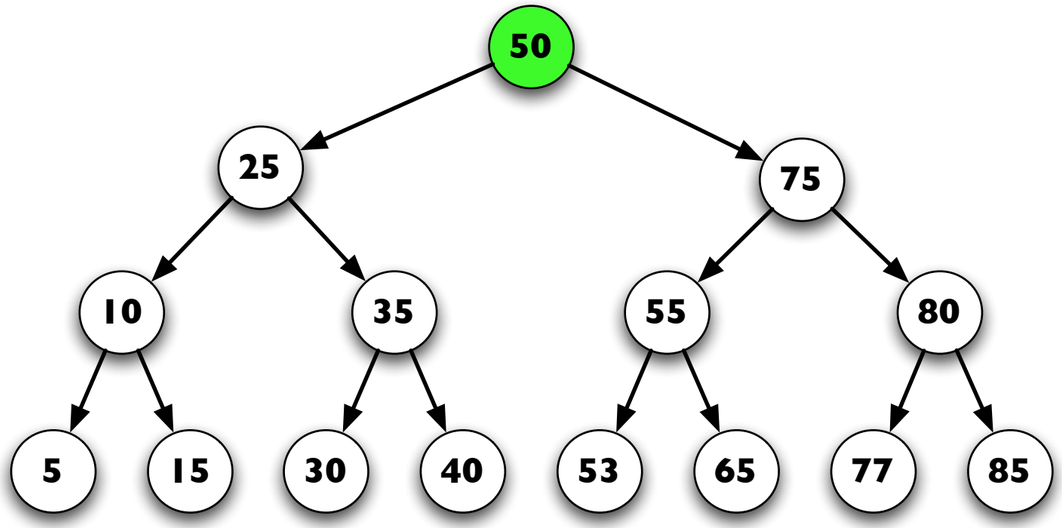
Visit root node

Traverse TL

Traverse TR

Preorder

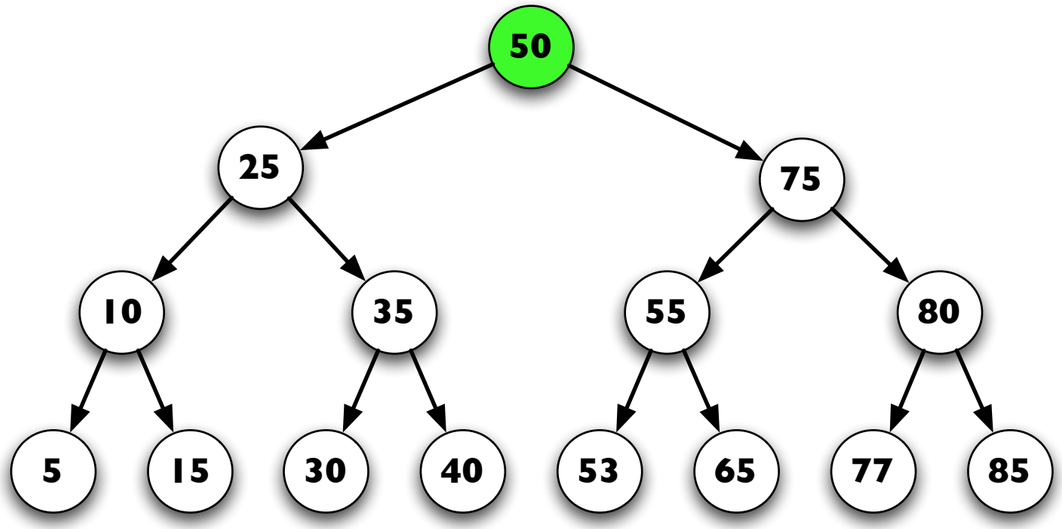
- Visit root node



Visit root node
Traverse TL
Traverse TR

Preorder

- Visit root node
- Traverse TL
- Traverse TR

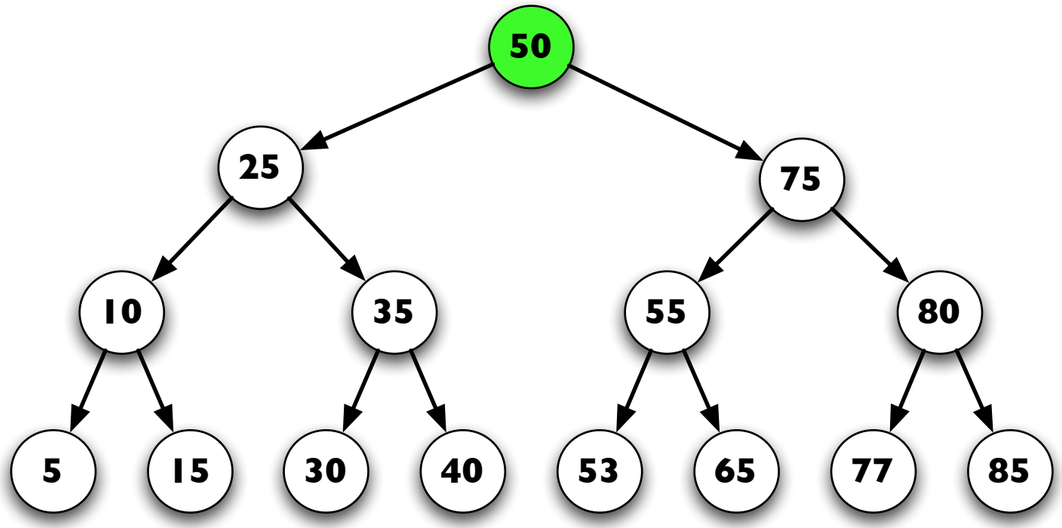


Visit root node
Traverse TL
Traverse TR

Preorder

50

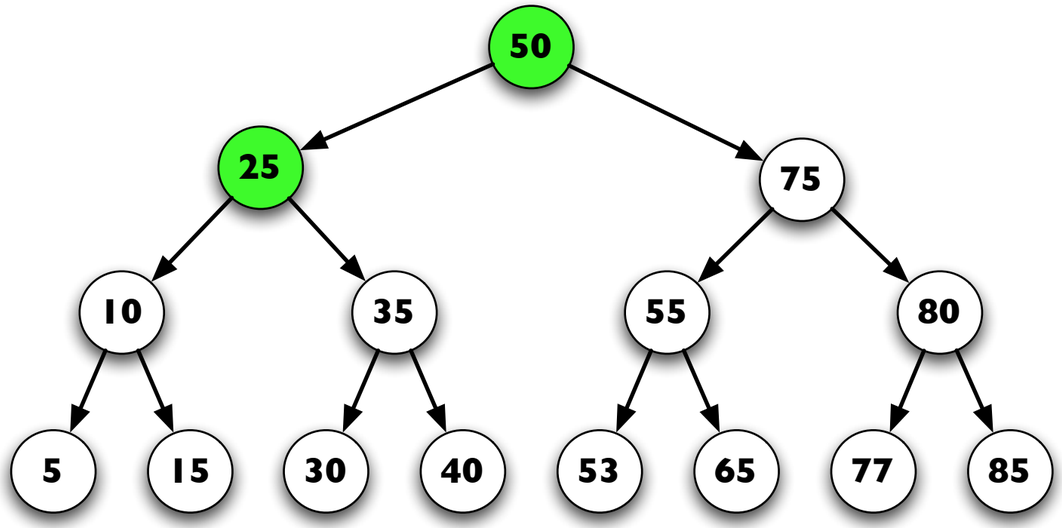
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50



Visit root node

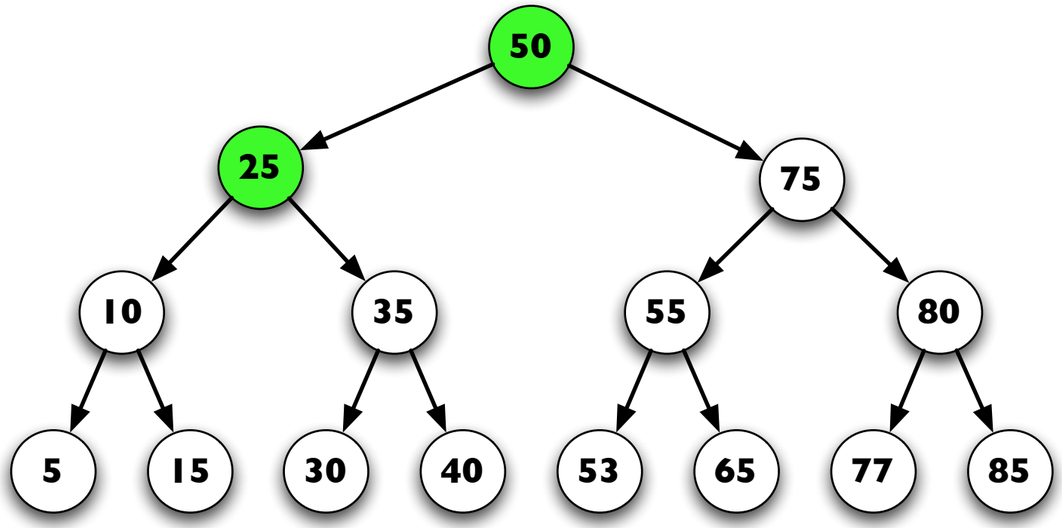
Traverse TL

Traverse TR

Preorder

50

- Visit root node

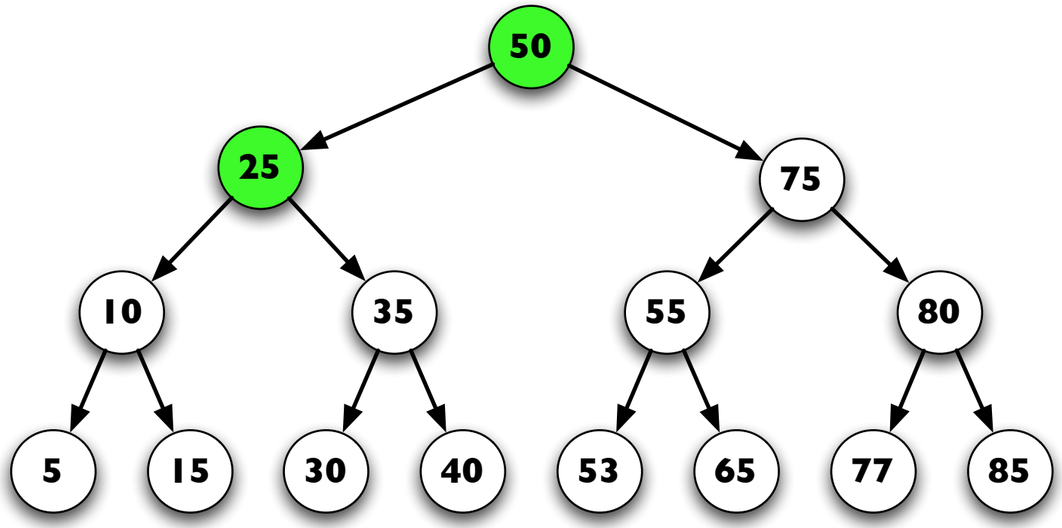


Visit root node
Traverse TL
Traverse TR

Preorder

50

- Visit root node
- Traverse TL
- Traverse TR

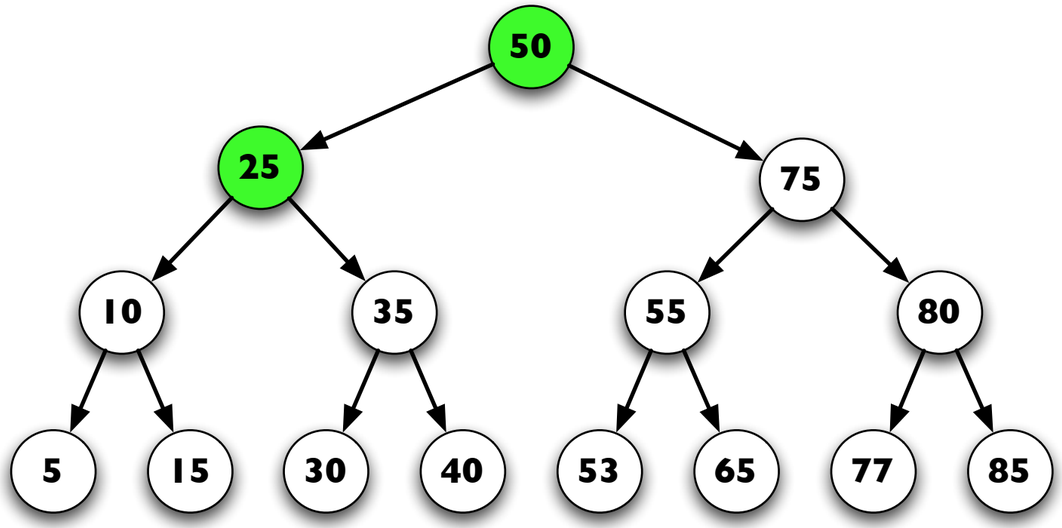


Visit root node
Traverse TL
Traverse TR

Preorder

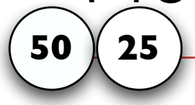
50 25

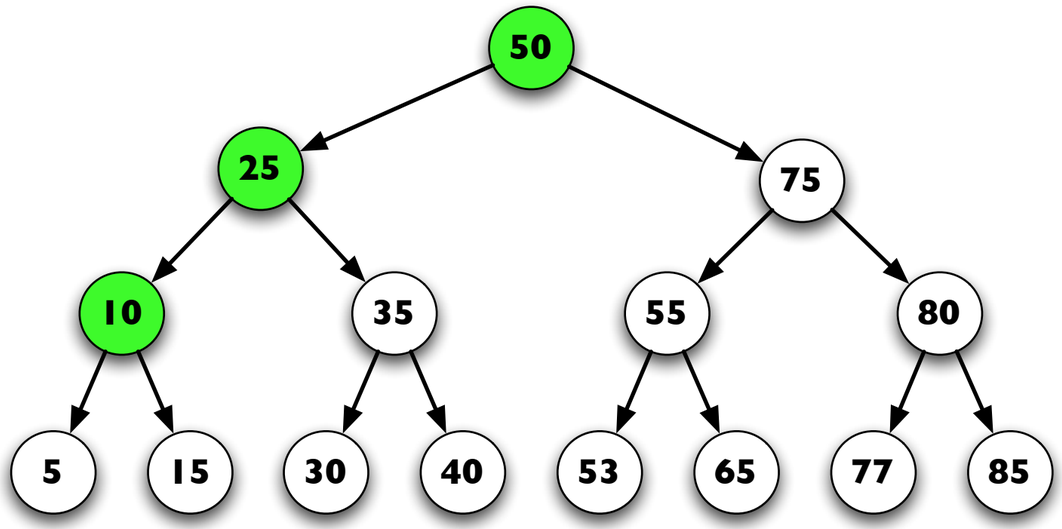
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder





Visit root node

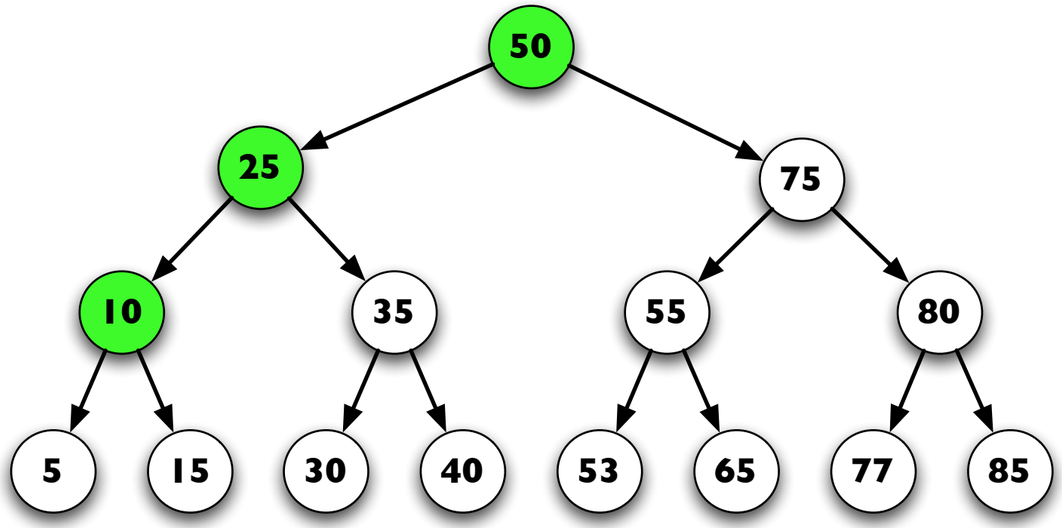
Traverse TL

Traverse TR

Preorder

50 25

- Visit root node

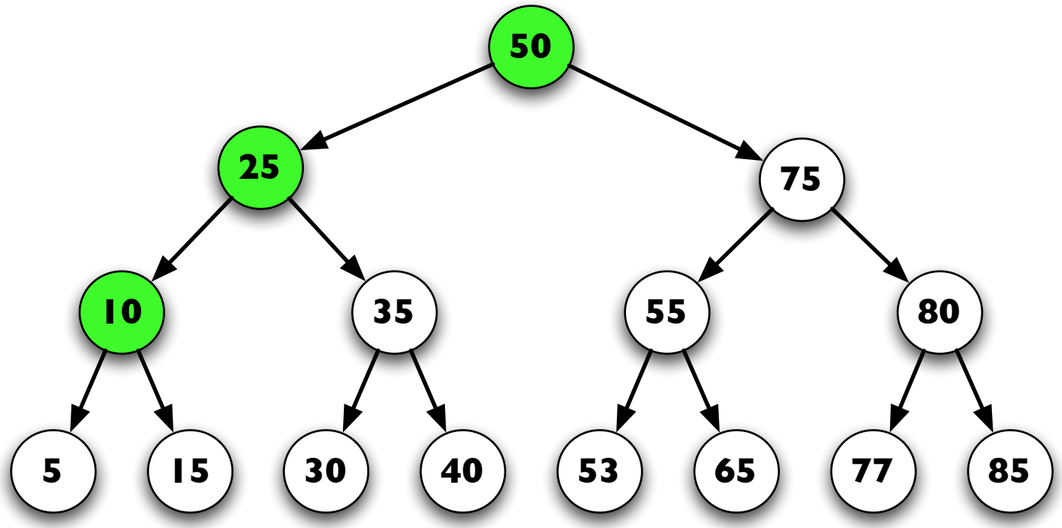


Visit root node
Traverse TL
Traverse TR

Preorder

50 25

- Visit root node
- Traverse TL
- Traverse TR

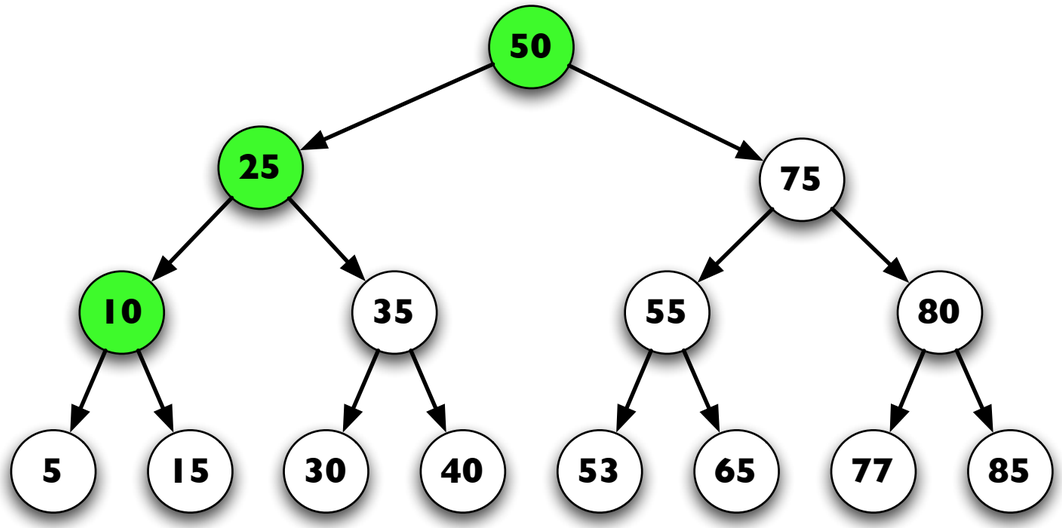


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10

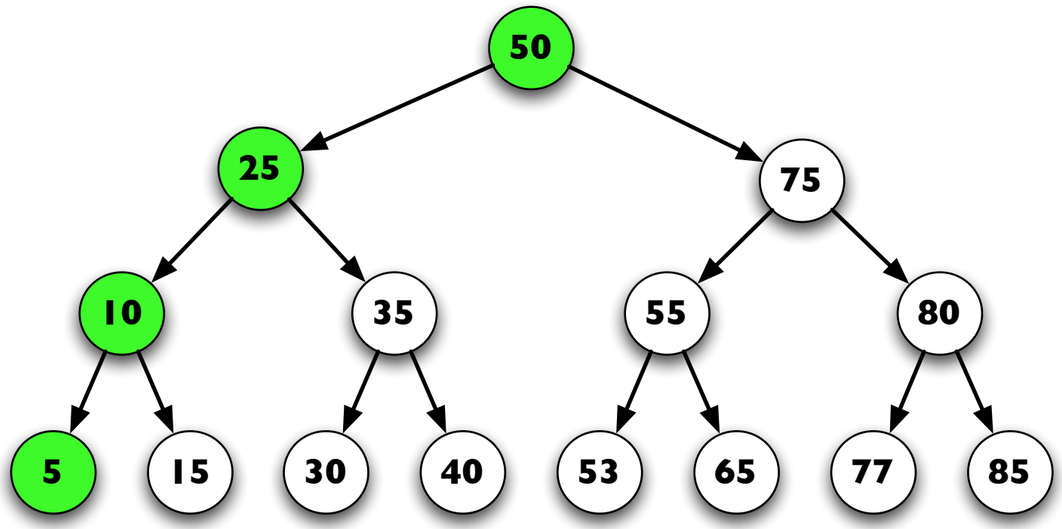
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10



Visit root node

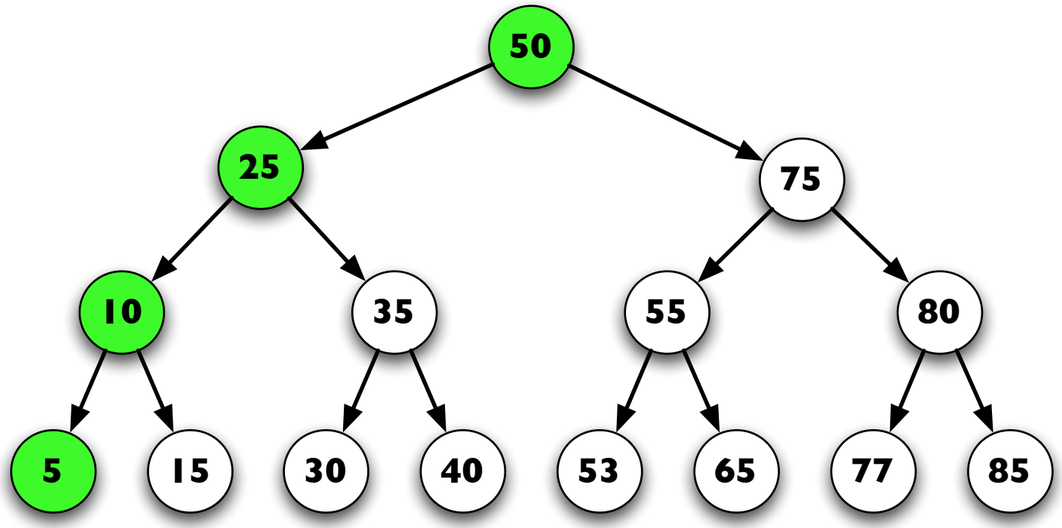
Traverse TL

Traverse TR

Preorder

50 25 10

- Visit root node

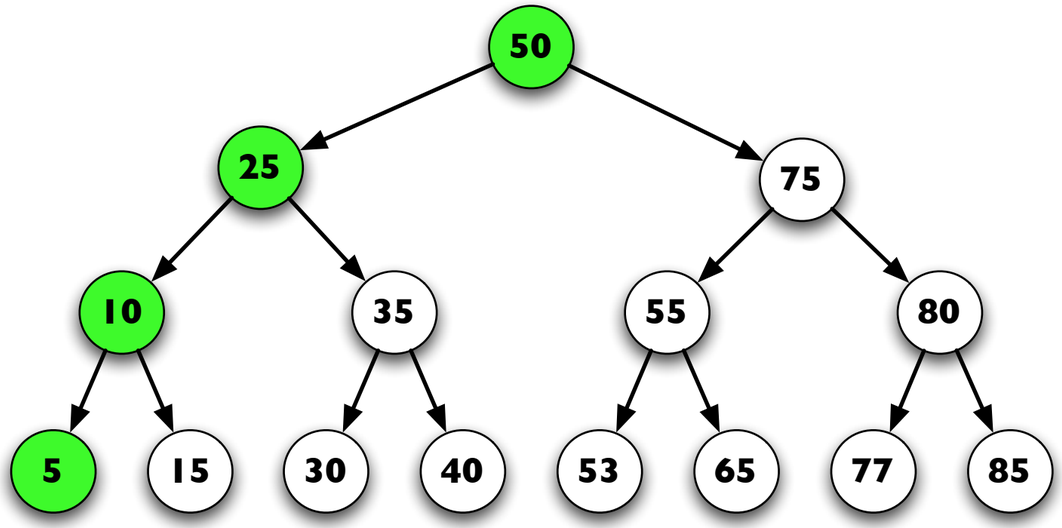


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10

- Visit root node
- Traverse TL
- Traverse TR

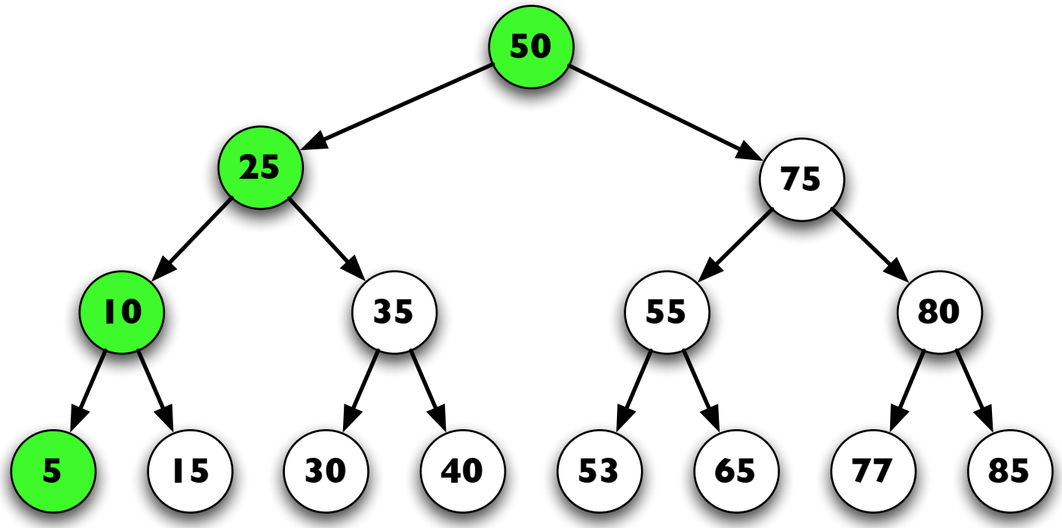


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5

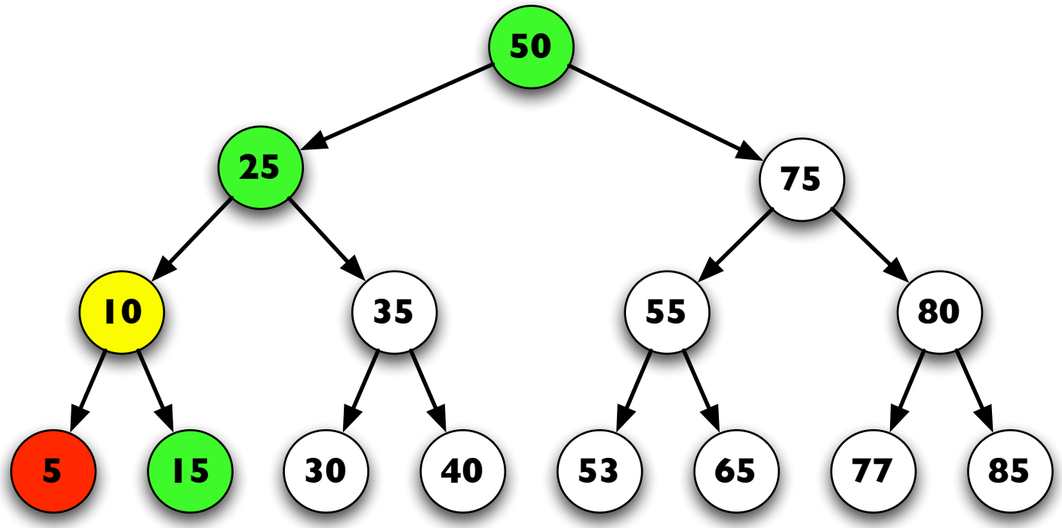
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5



Visit root node

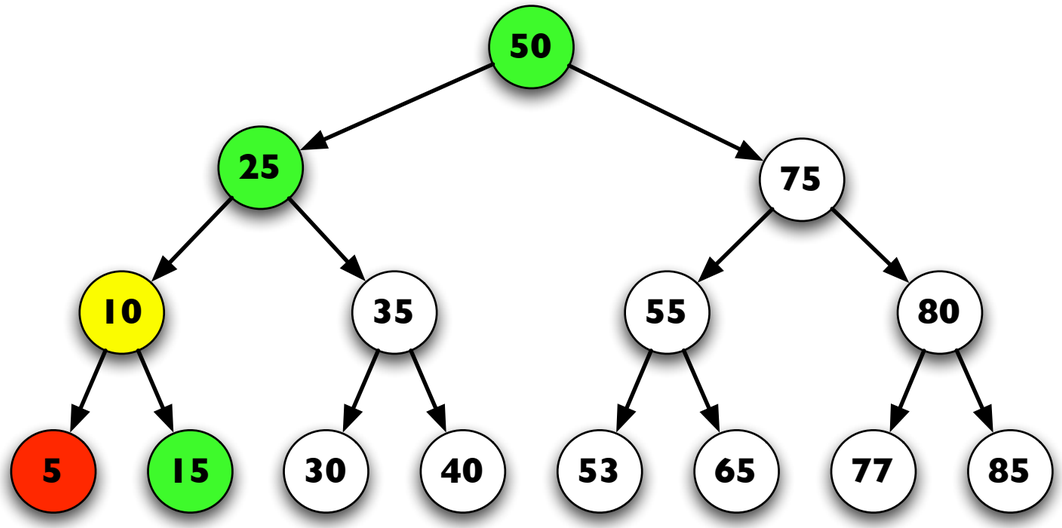
Traverse TL

Traverse TR

Preorder

50 25 10 5

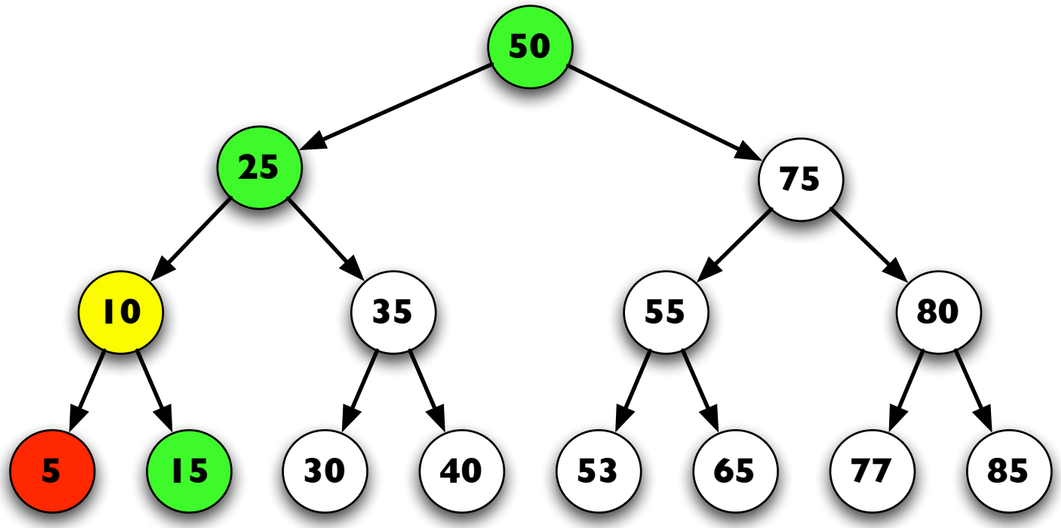
- Visit root node



Visit root node
Traverse TL
Traverse TR

Preorder

- Visit root node
- Traverse TL
- Traverse TR

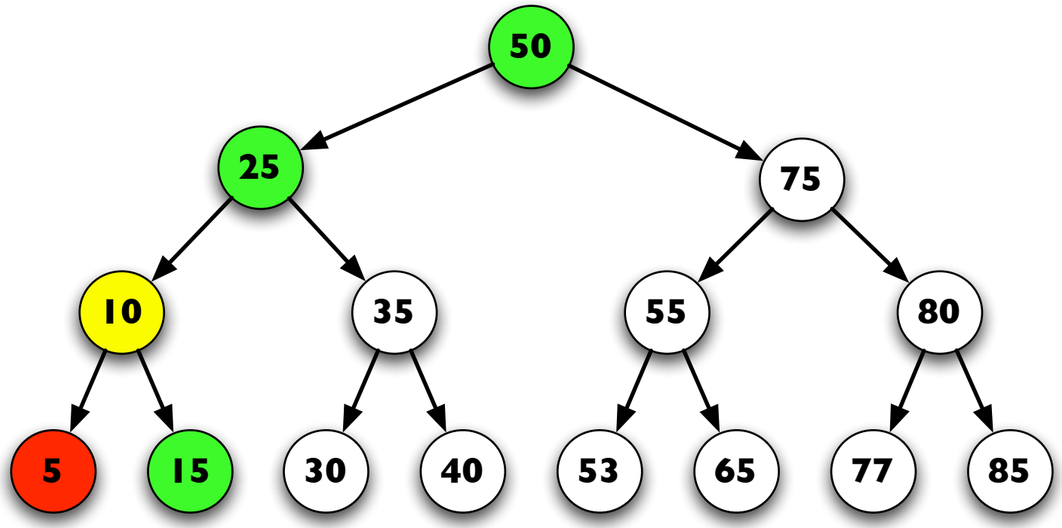


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15

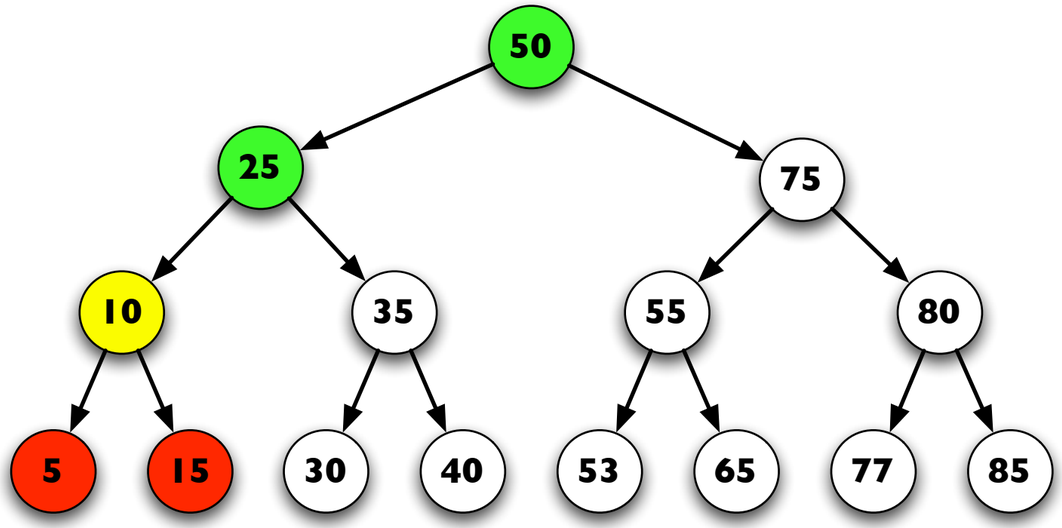
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15



Visit root node

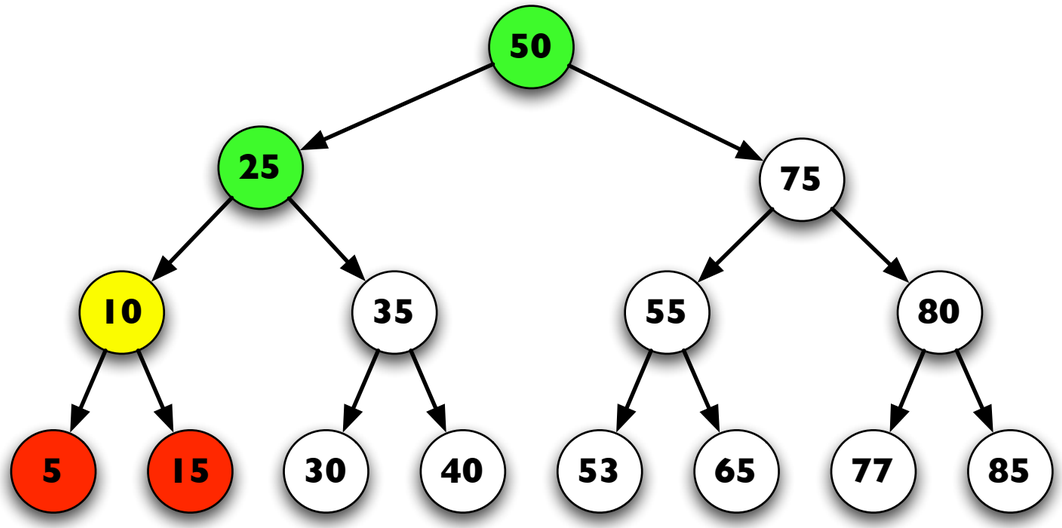
Traverse TL

Traverse TR

Preorder

50 25 10 5 15

- Visit root node



Visit root node

Traverse TL

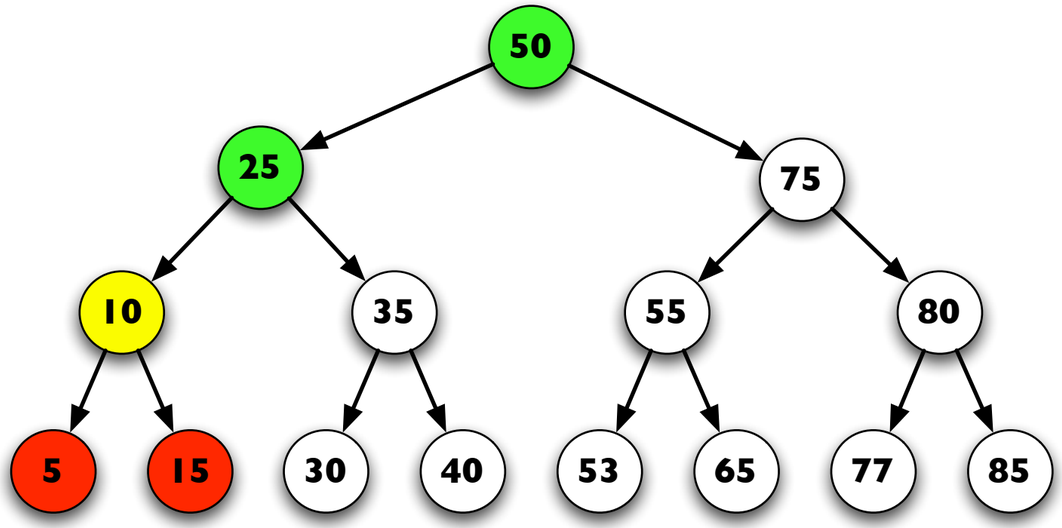
Traverse TR

Preorder

50 25 10 5 15

- Visit root node

- Traverse TL



Visit root node

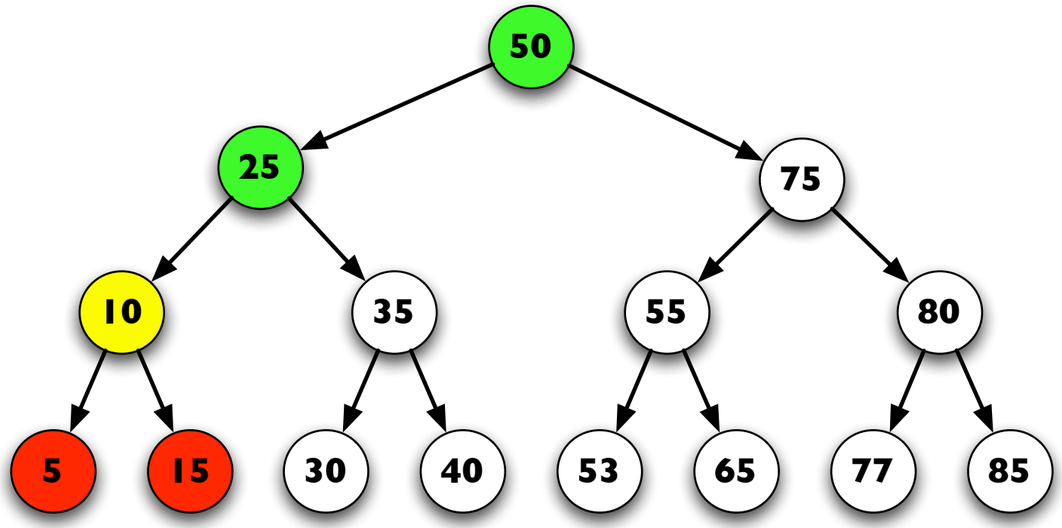
Traverse TL

Traverse TR

Preorder

50 25 10 5 15

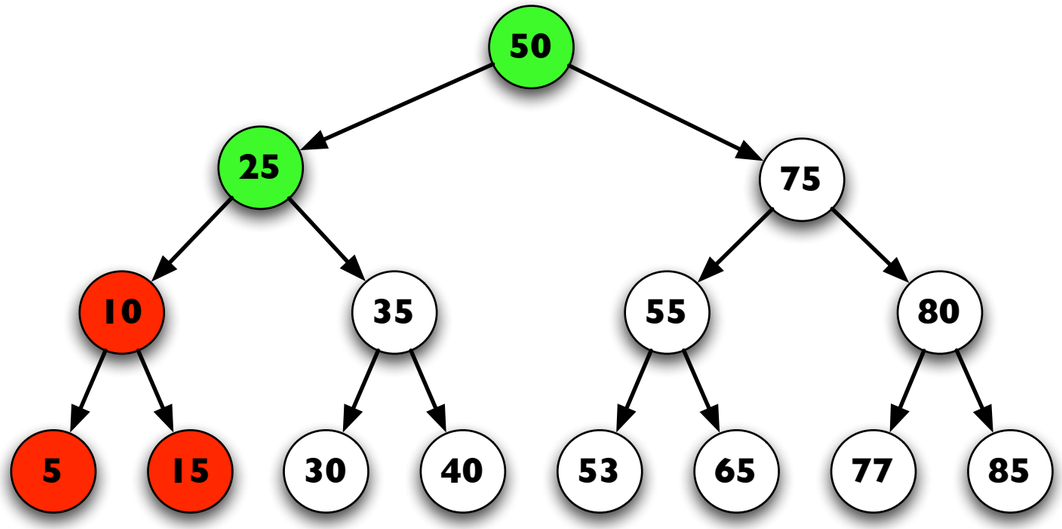
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15



Visit root node

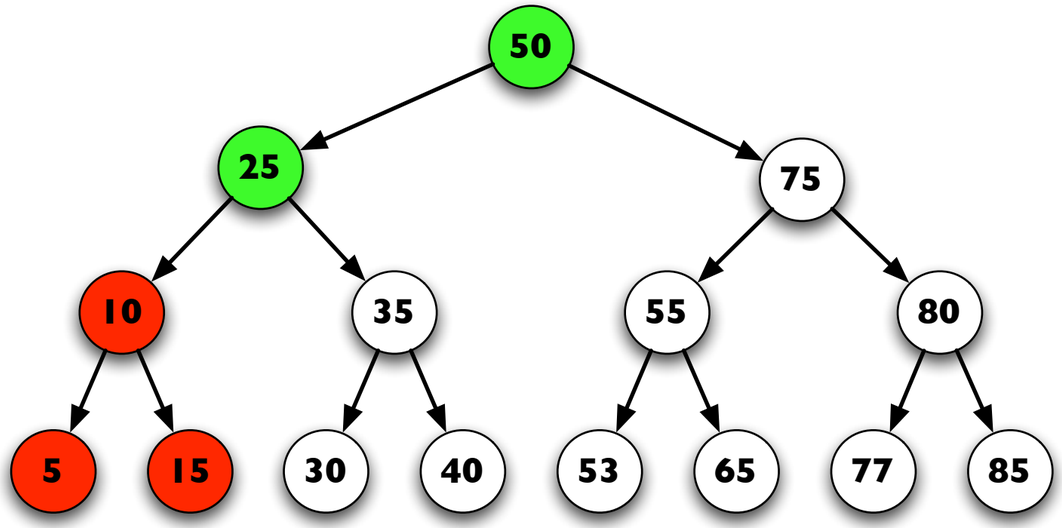
Traverse TL

Traverse TR

Preorder

50 25 10 5 15

- Visit root node



Visit root node

Traverse TL

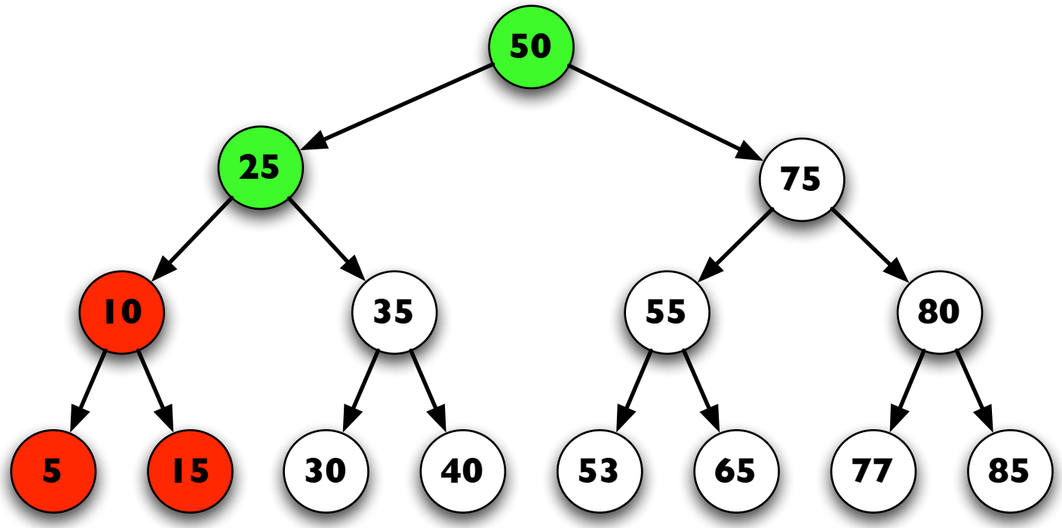
Traverse TR

Preorder

50 25 10 5 15

- Visit root node

- Traverse TL



Visit root node

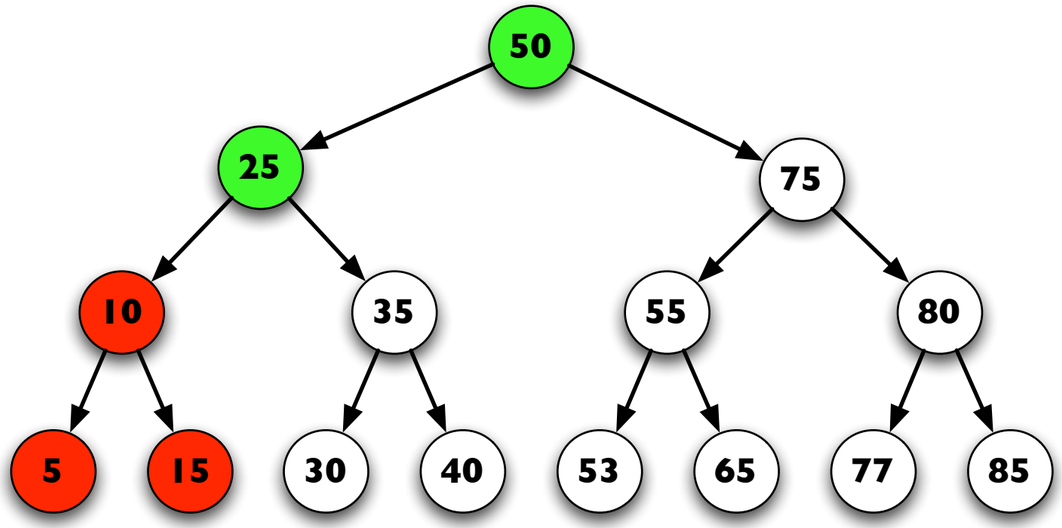
Traverse TL

Traverse TR

Preorder

50 25 10 5 15

- Visit root node
- Traverse TL
- Traverse TR



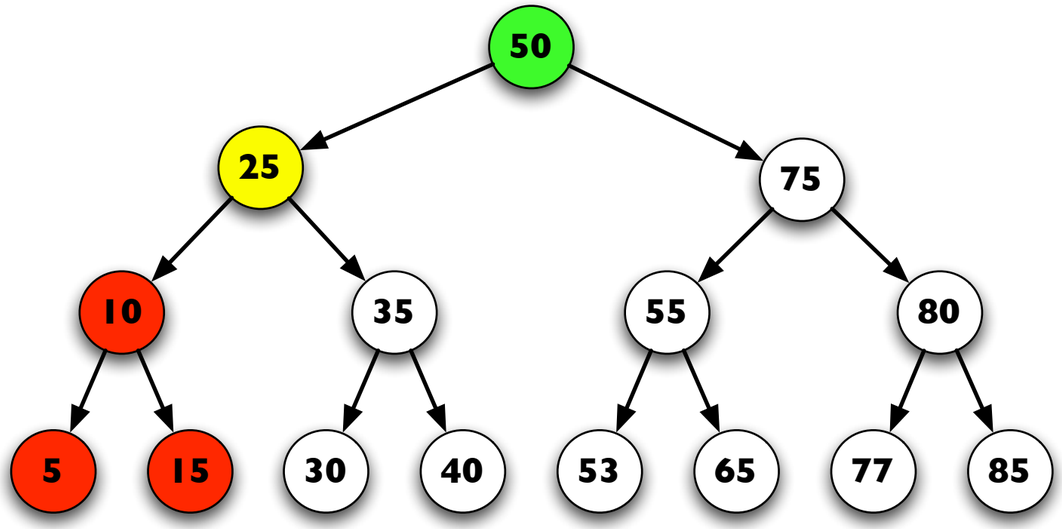
Visit root node

Traverse TL

Traverse TR

Preorder

50 25 10 5 15



Visit root node

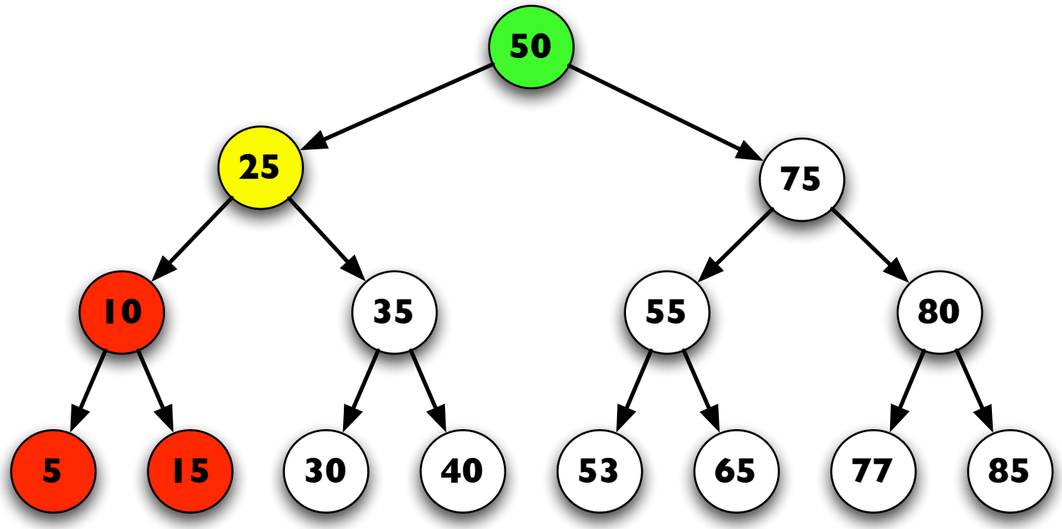
Traverse TL

Traverse TR

Preorder

50 25 10 5 15

- Visit root node



Visit root node

Traverse TL

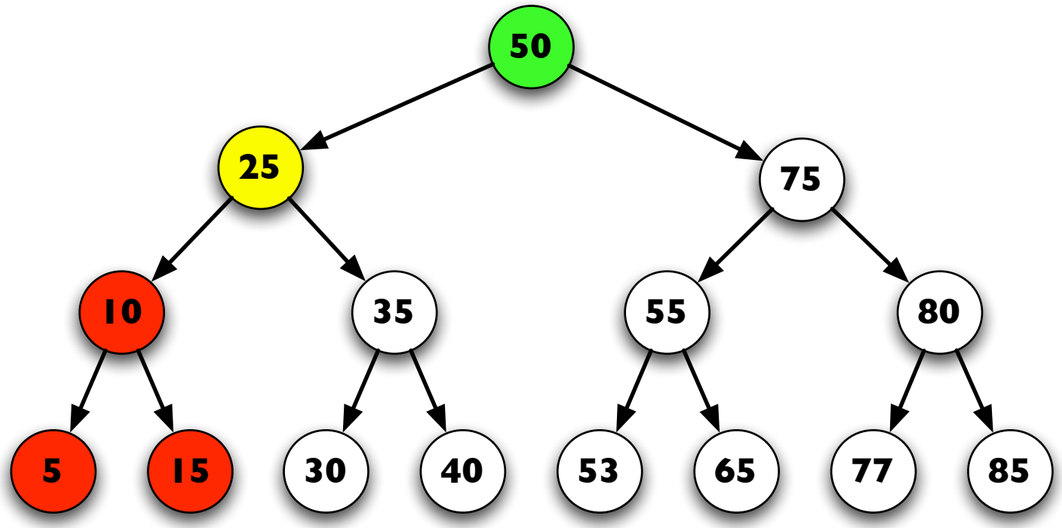
Traverse TR

Preorder

50 25 10 5 15

- Visit root node

- Traverse TL



Visit root node

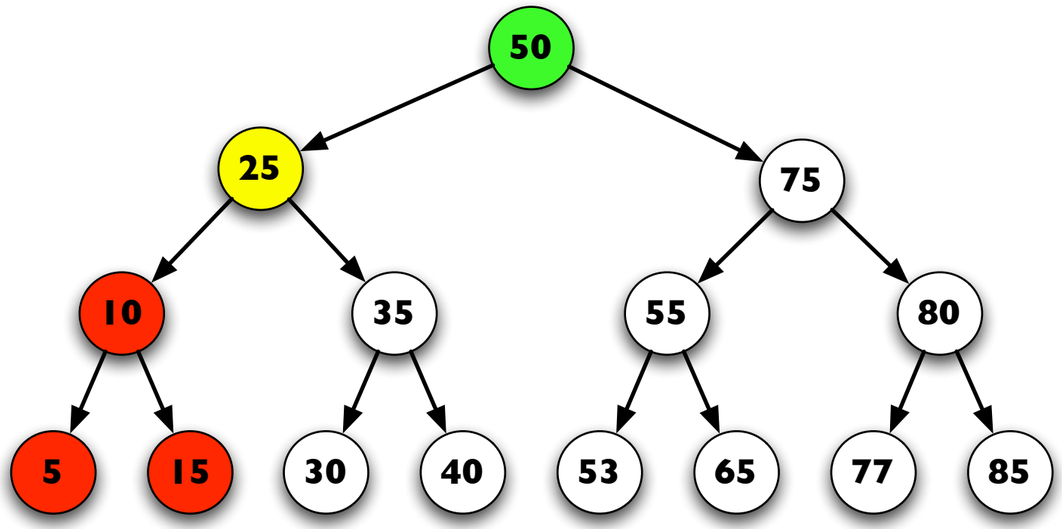
Traverse TL

Traverse TR

Preorder

50 25 10 5 15

- Visit root node
- Traverse TL
- Traverse TR



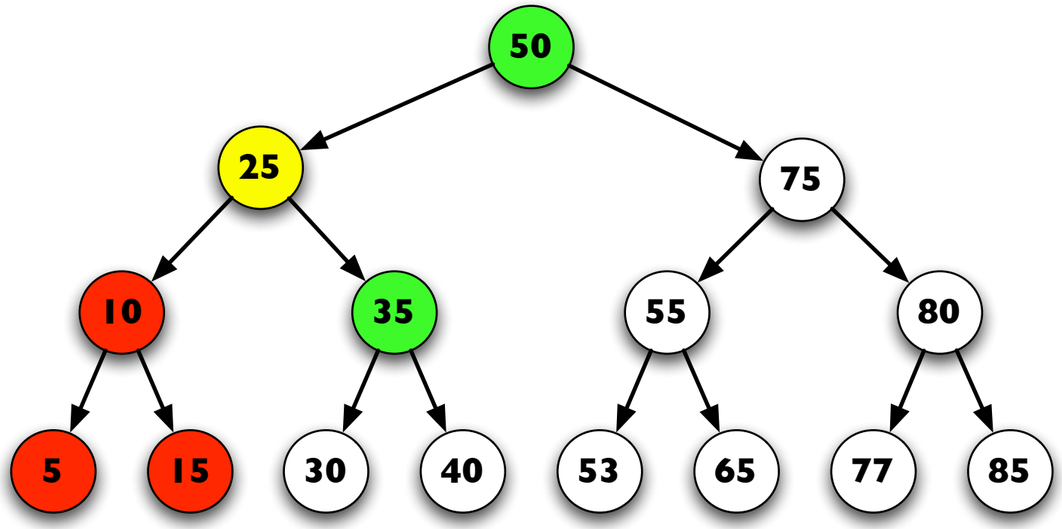
Visit root node

Traverse TL

Traverse TR

Preorder

50 25 10 5 15



Visit root node

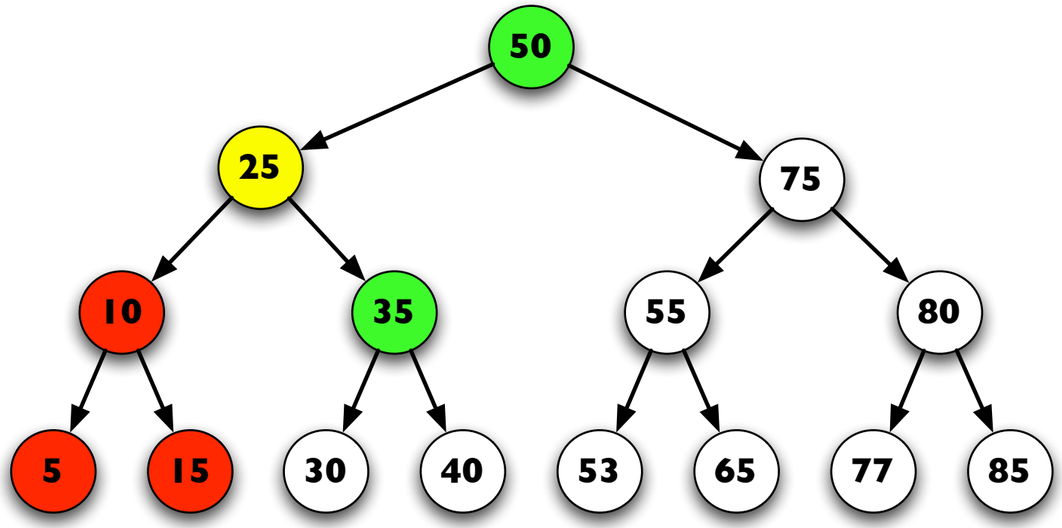
Traverse TL

Traverse TR

Preorder

50 25 10 5 15

- Visit root node



Visit root node

Traverse TL

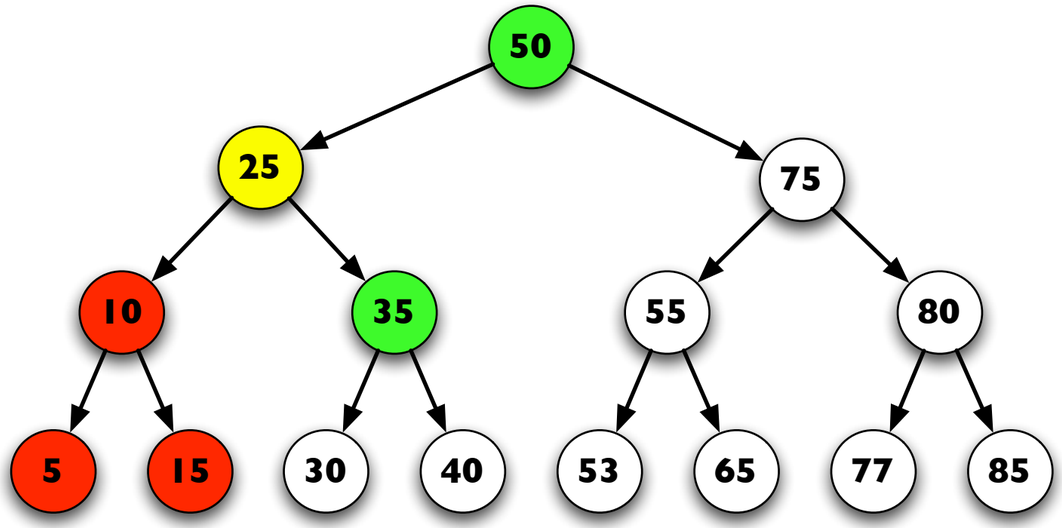
Traverse TR

Preorder

50 25 10 5 15

- Visit root node

- Traverse TL



Visit root node

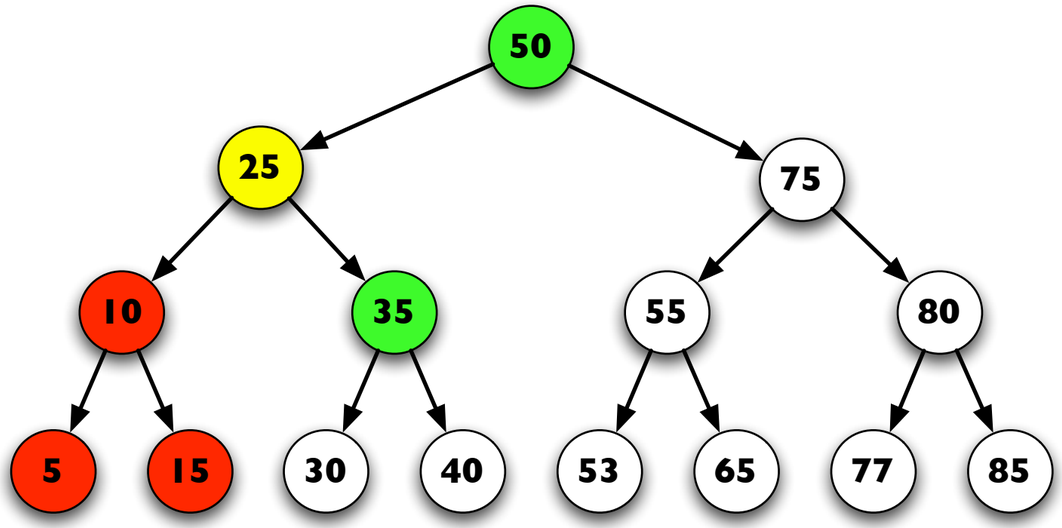
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35

- Visit root node
- Traverse TL
- Traverse TR



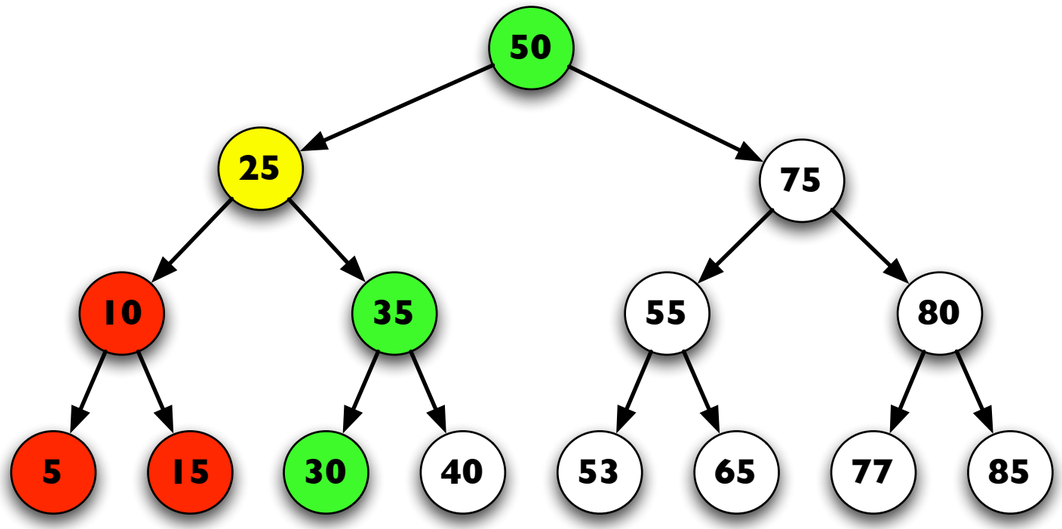
Visit root node

Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35



Visit root node

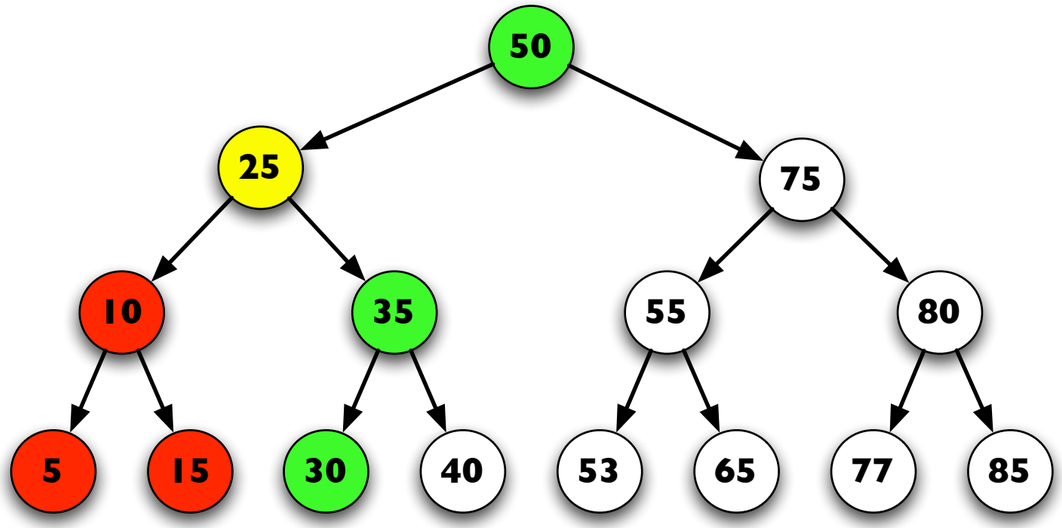
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35

- Visit root node



Visit root node

Traverse TL

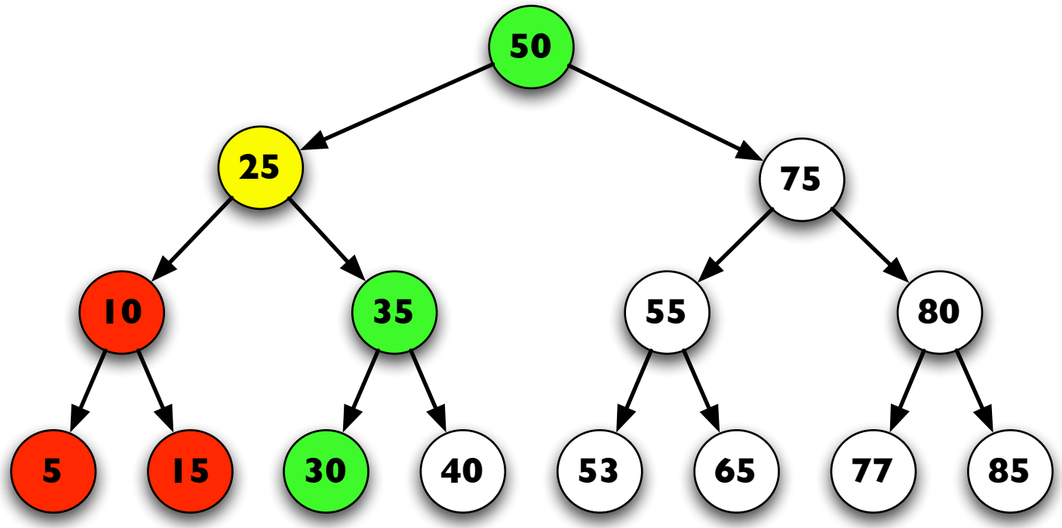
Traverse TR

Preorder

50 25 10 5 15 35

- Visit root node

- Traverse TL



Visit root node

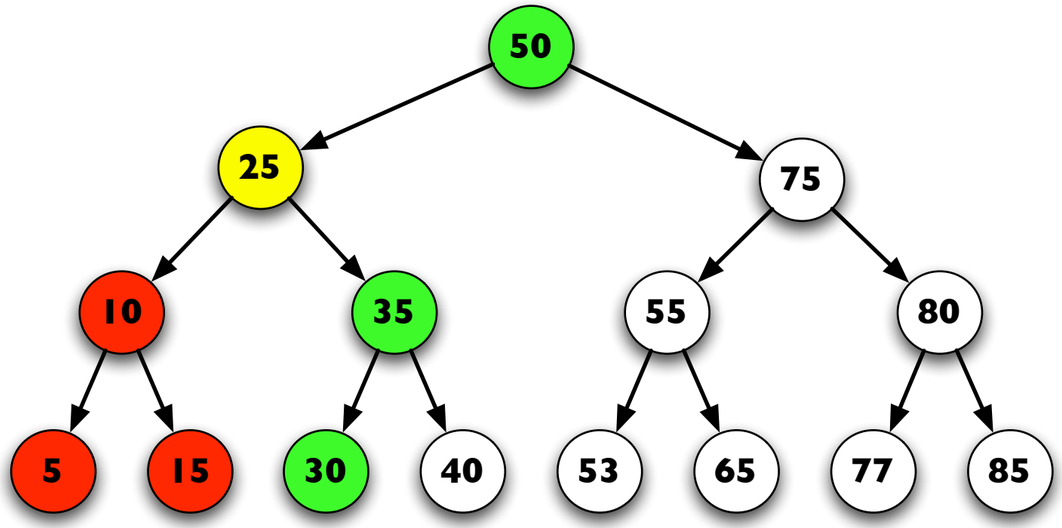
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30

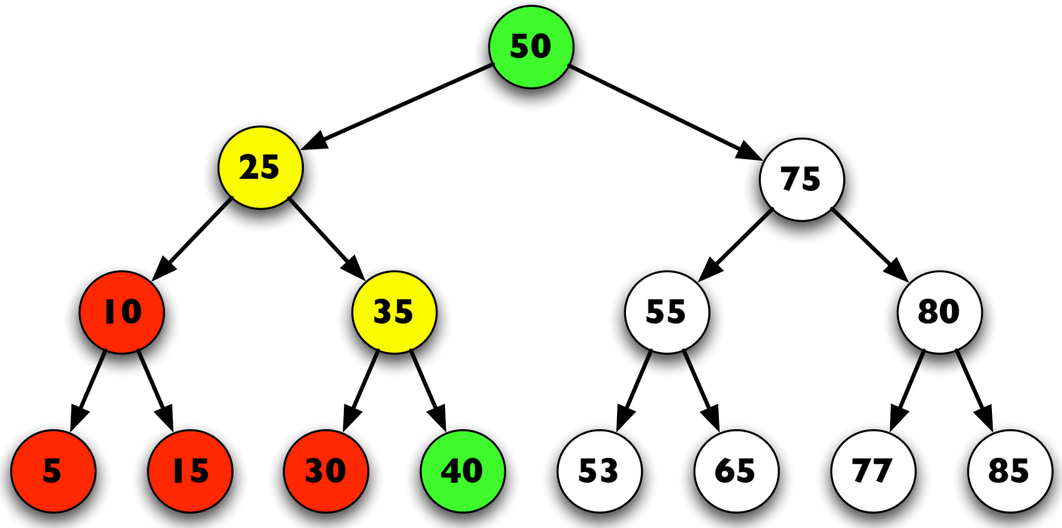
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30



Visit root node

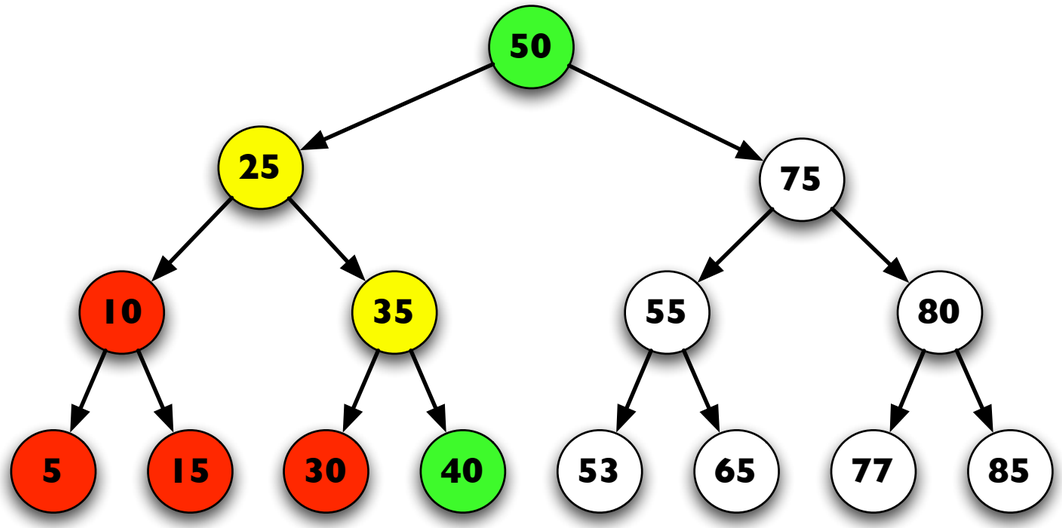
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30

- Visit root node



Visit root node

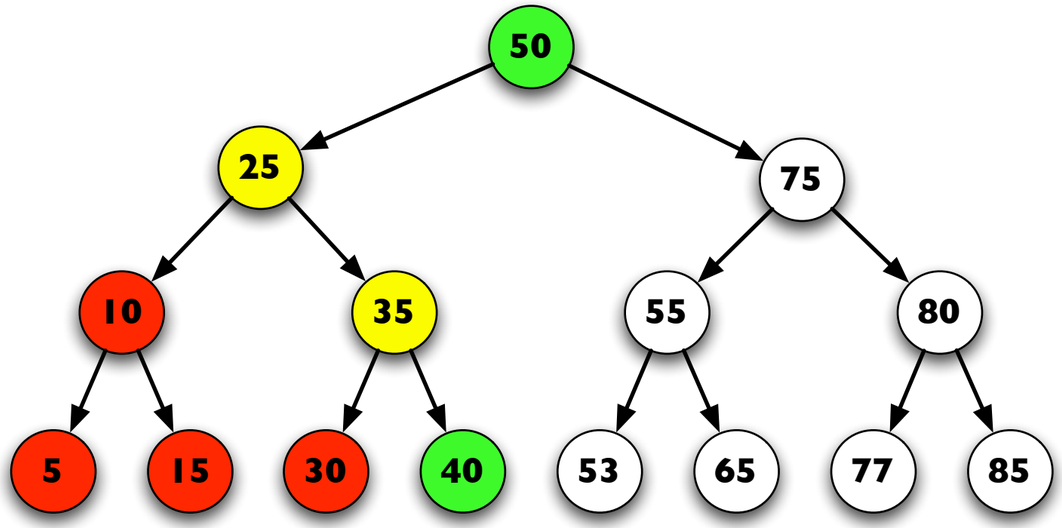
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30

- Visit root node
- Traverse TL
- Traverse TR

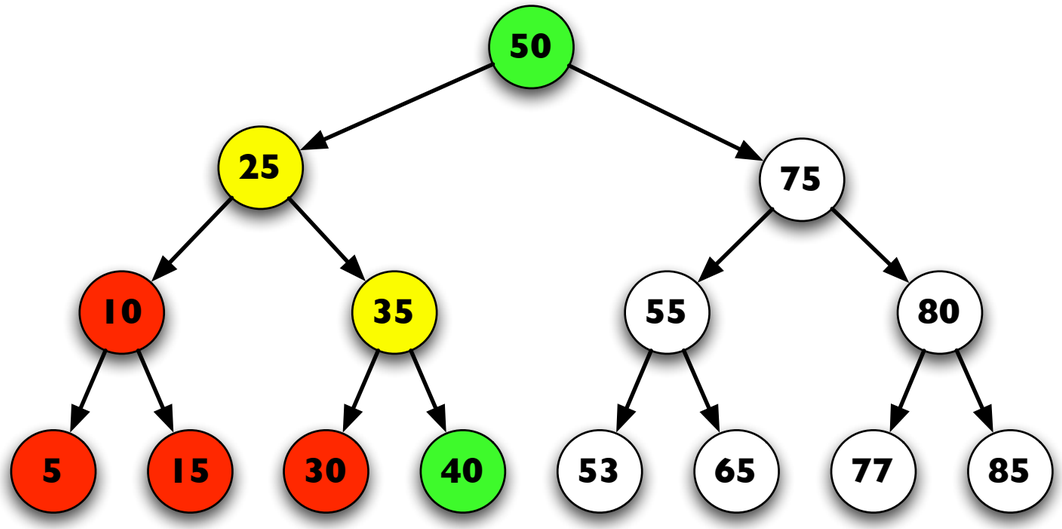


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40

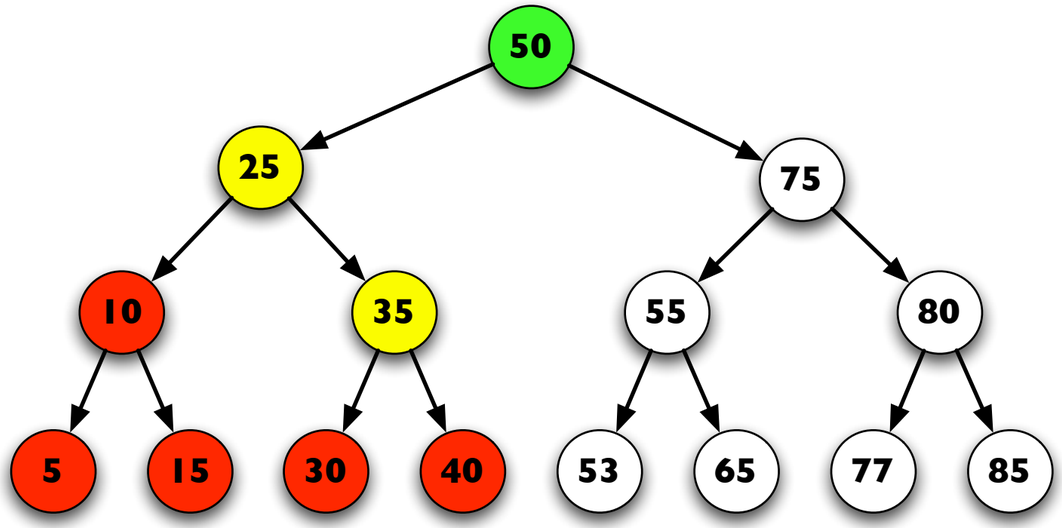
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40



Visit root node

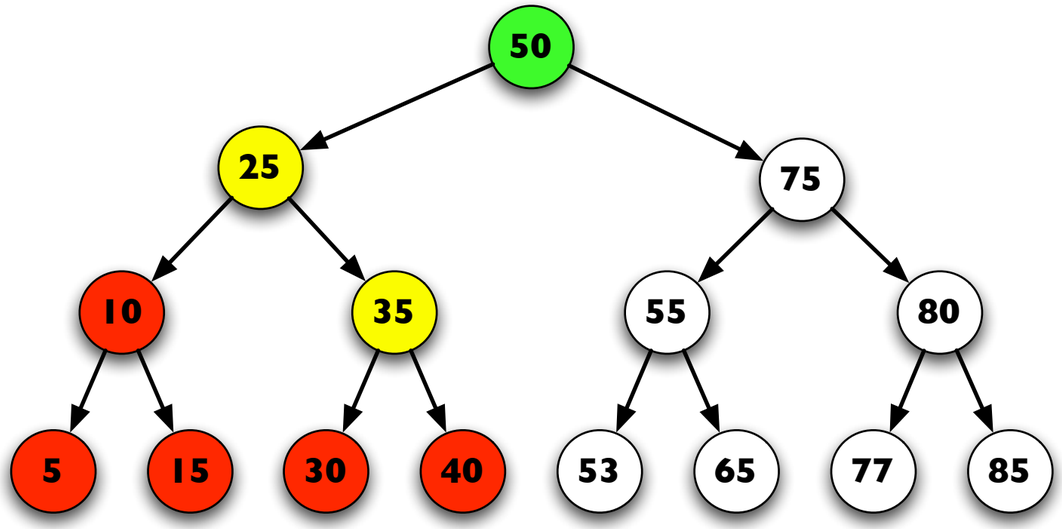
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40

- Visit root node



Visit root node

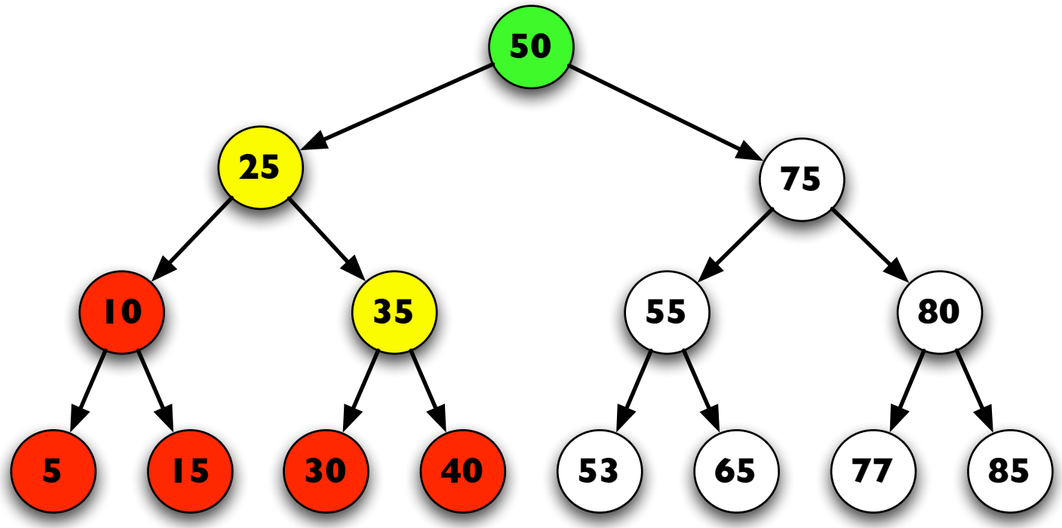
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40

- Visit root node
- Traverse TL
- Traverse TR

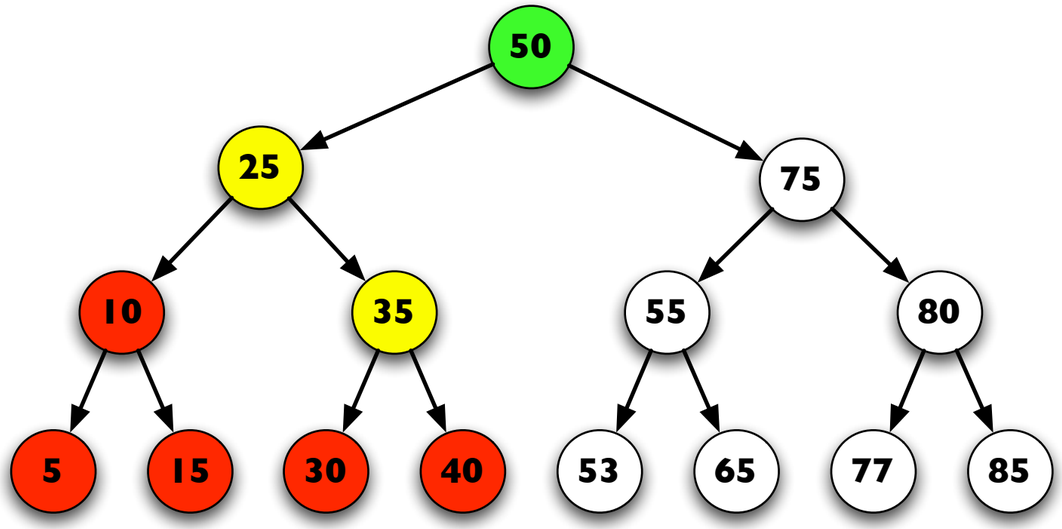


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40

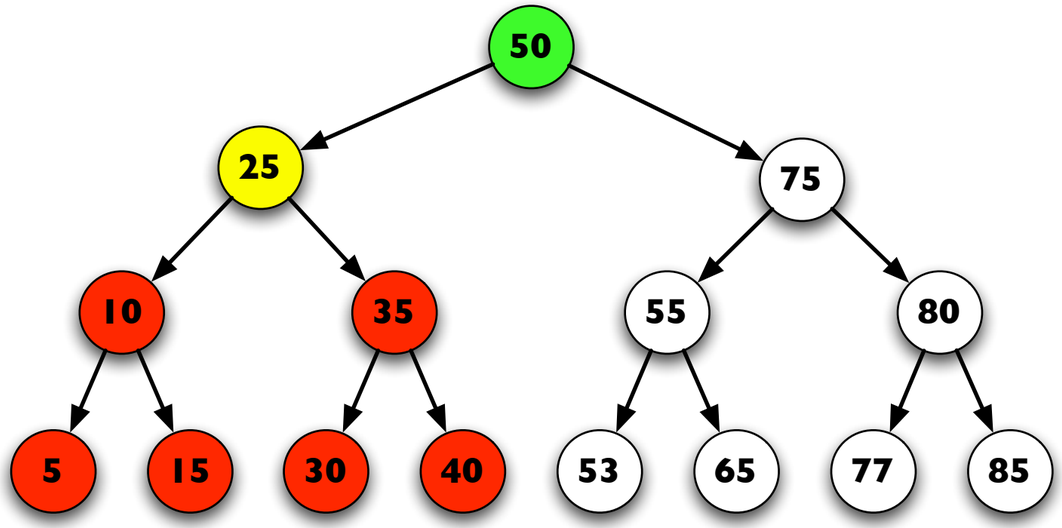
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40



Visit root node

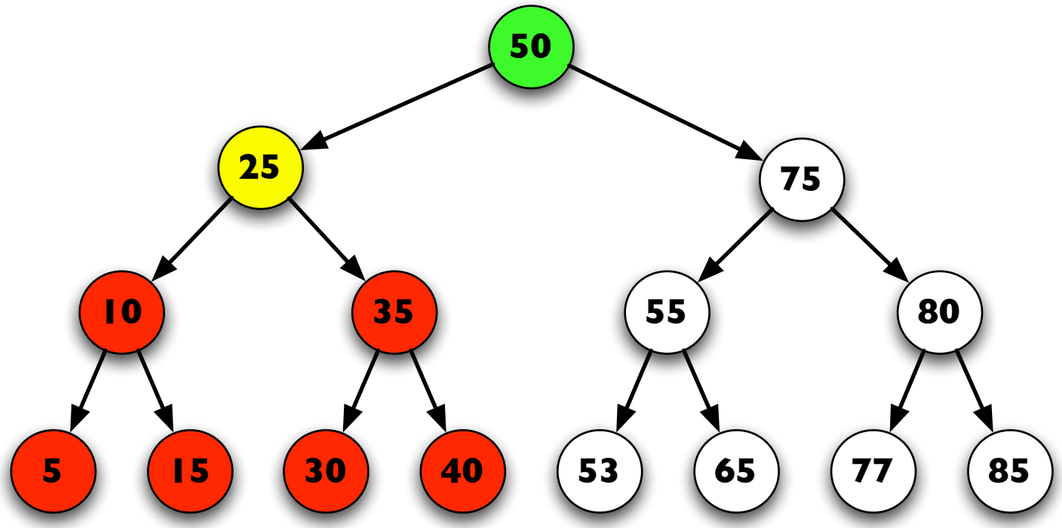
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40

- Visit root node



Visit root node

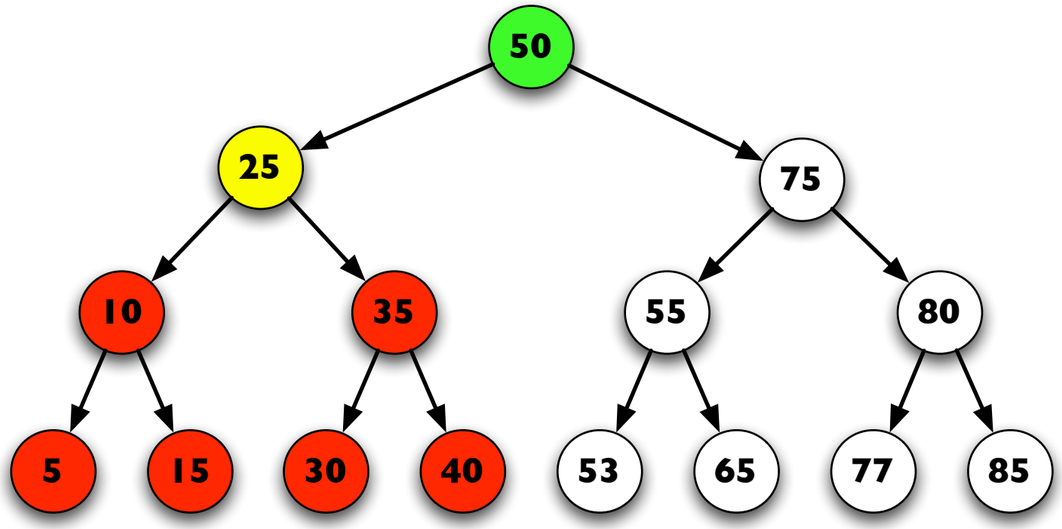
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40

- Visit root node
- Traverse TL
- Traverse TR

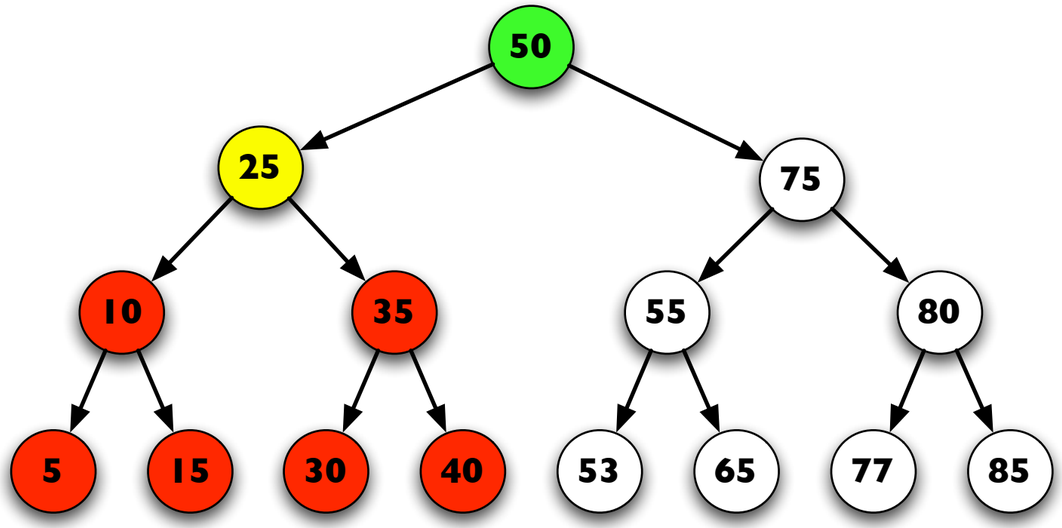


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40

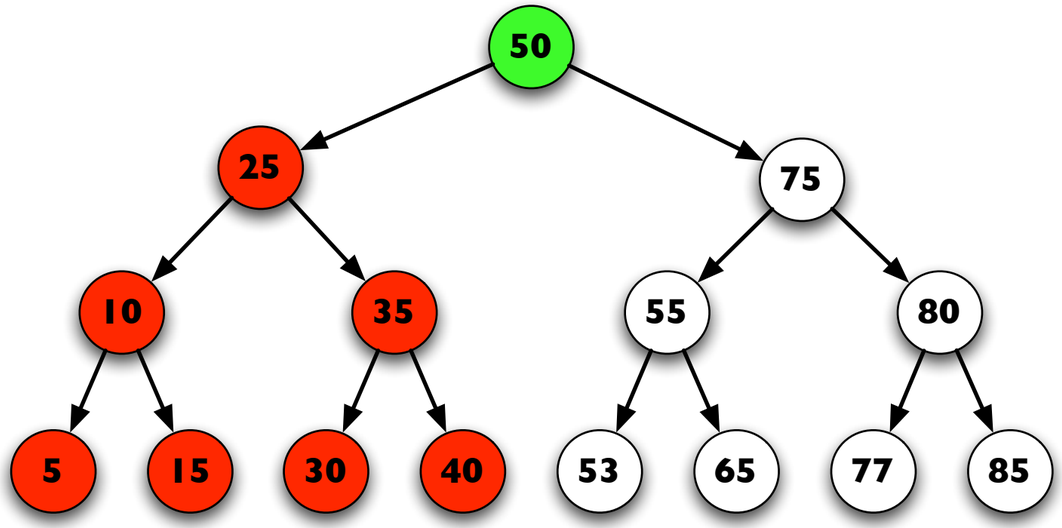
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40



Visit root node

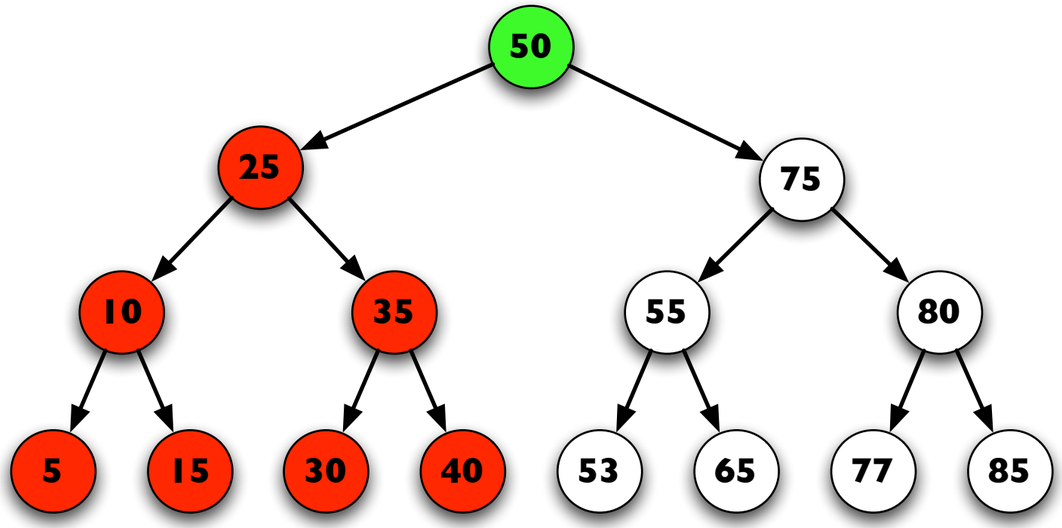
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40

- Visit root node



Visit root node

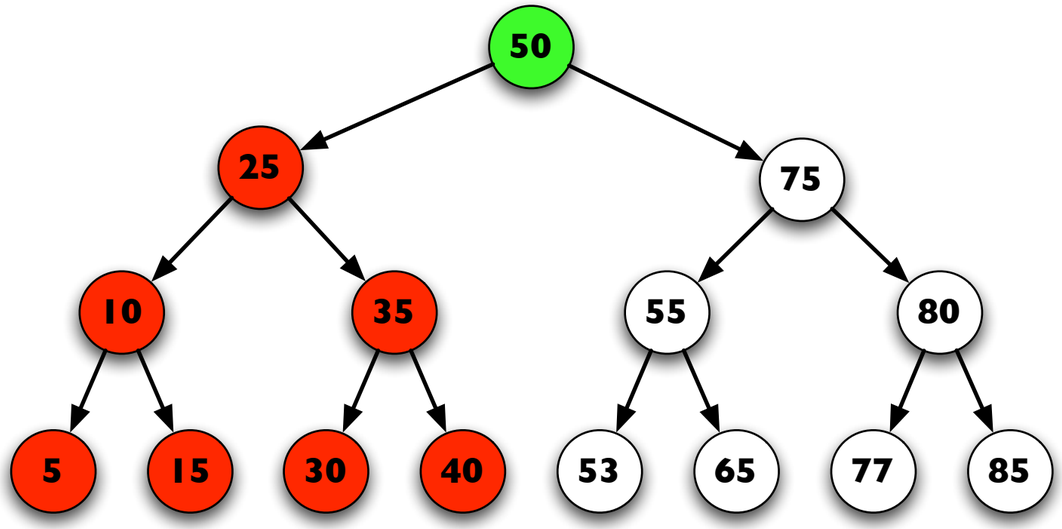
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40

- Visit root node
- Traverse TL
- Traverse TR

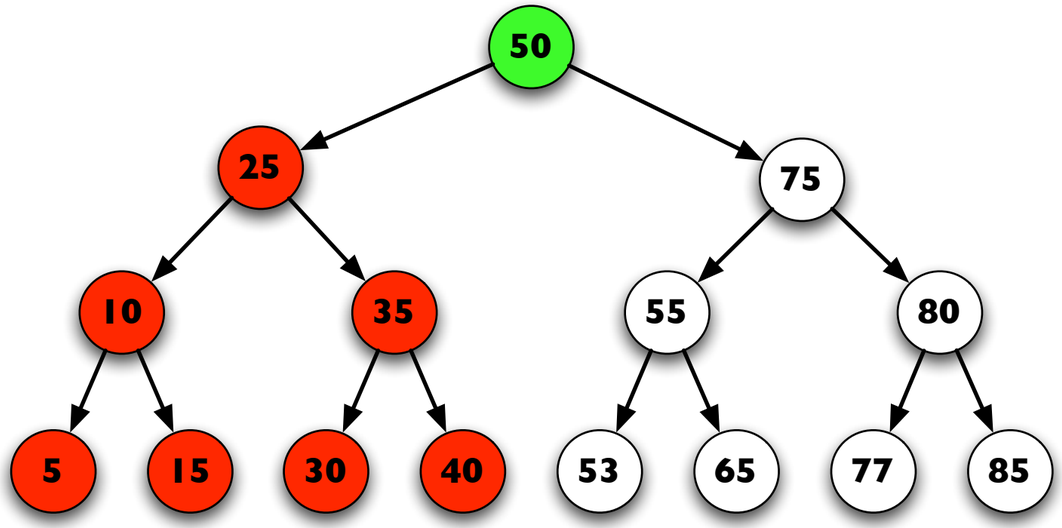


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40

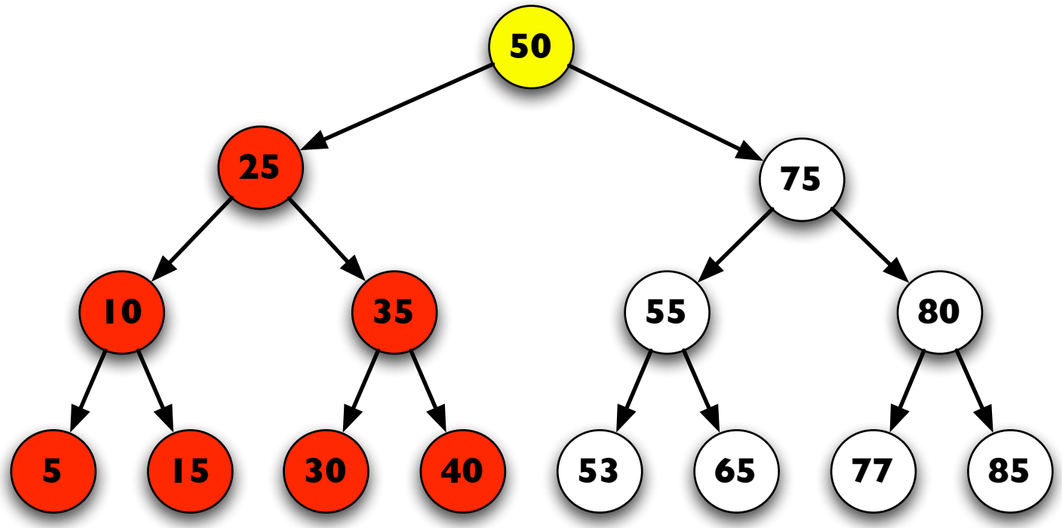
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40



Visit root node

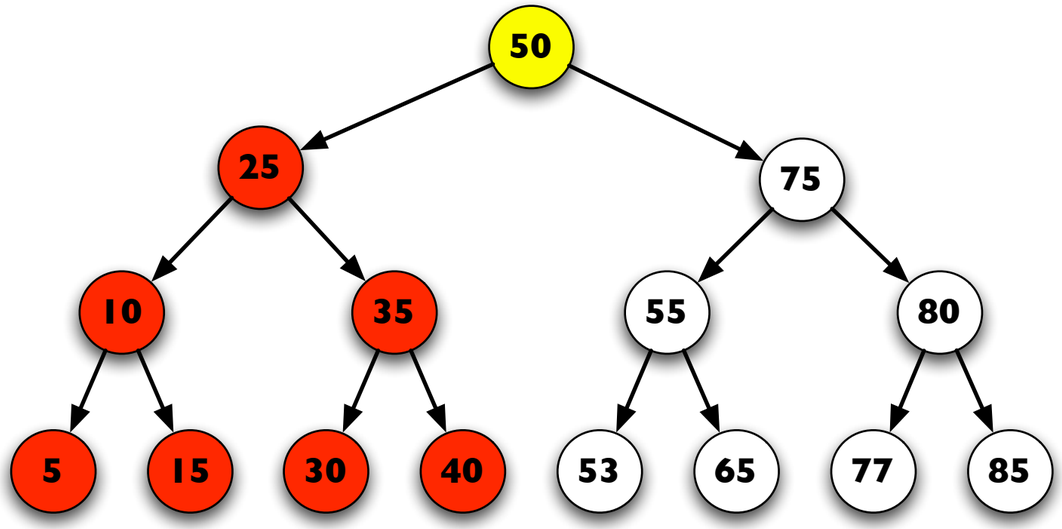
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40

- Visit root node



Visit root node

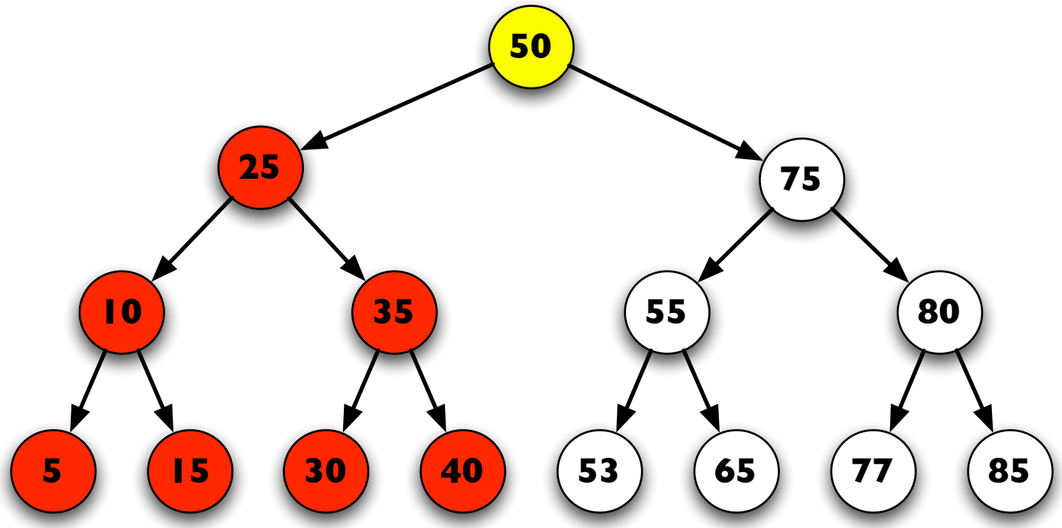
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40

- Visit root node
- Traverse TL
- Traverse TR

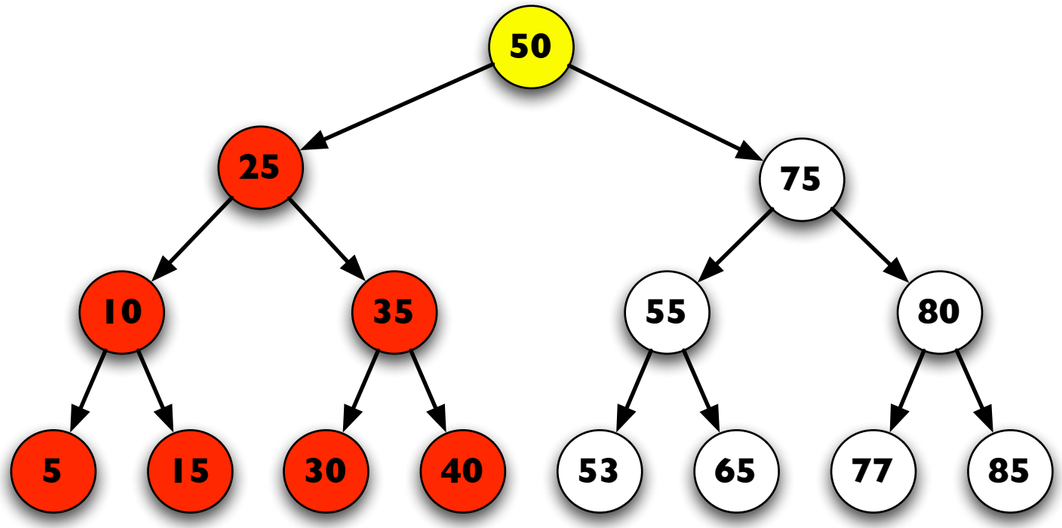


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40

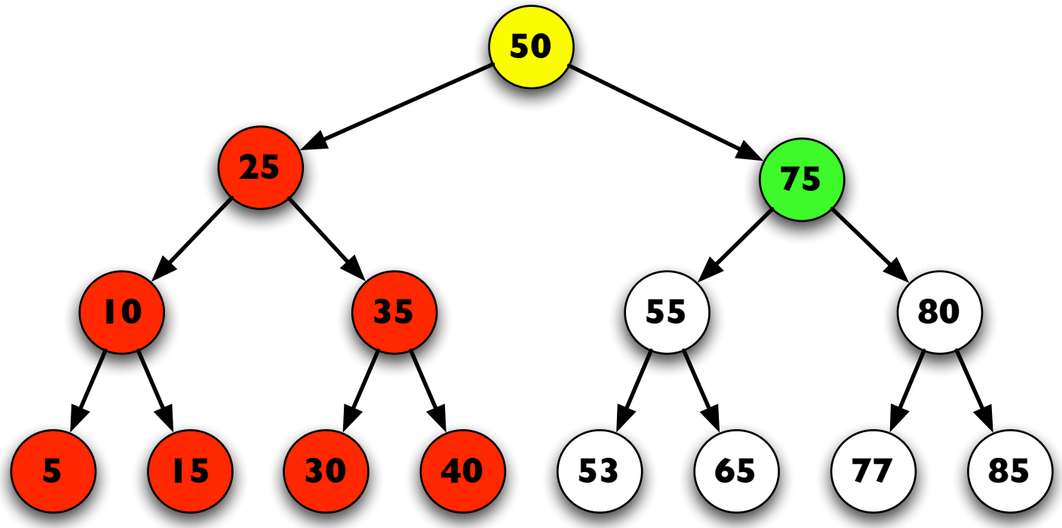
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40



Visit root node

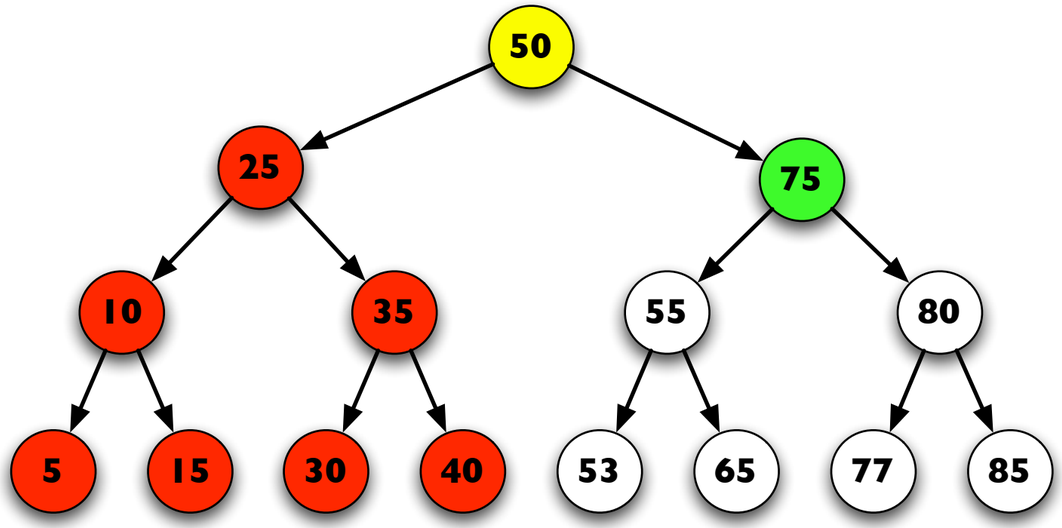
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40

- Visit root node



Visit root node

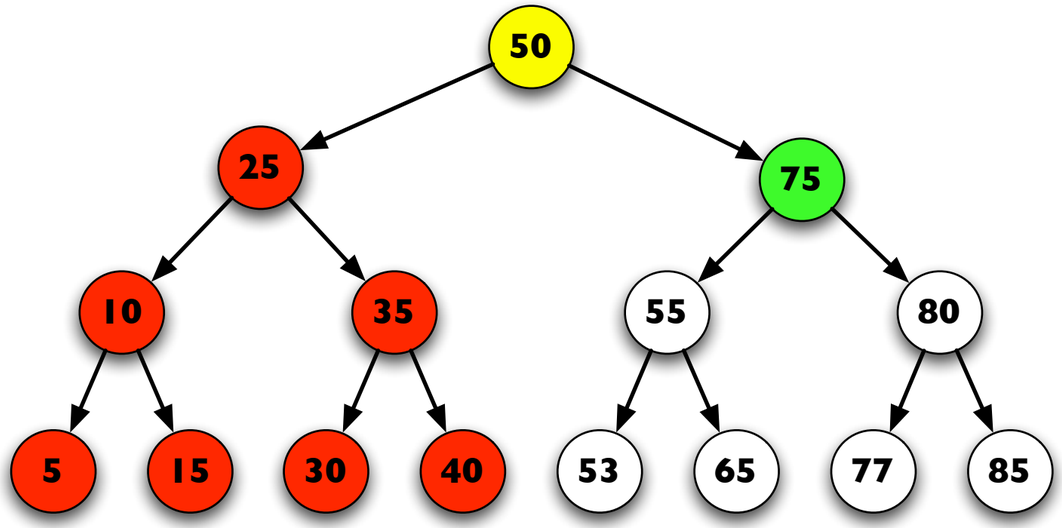
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40

- Visit root node
- Traverse TL
- Traverse TR

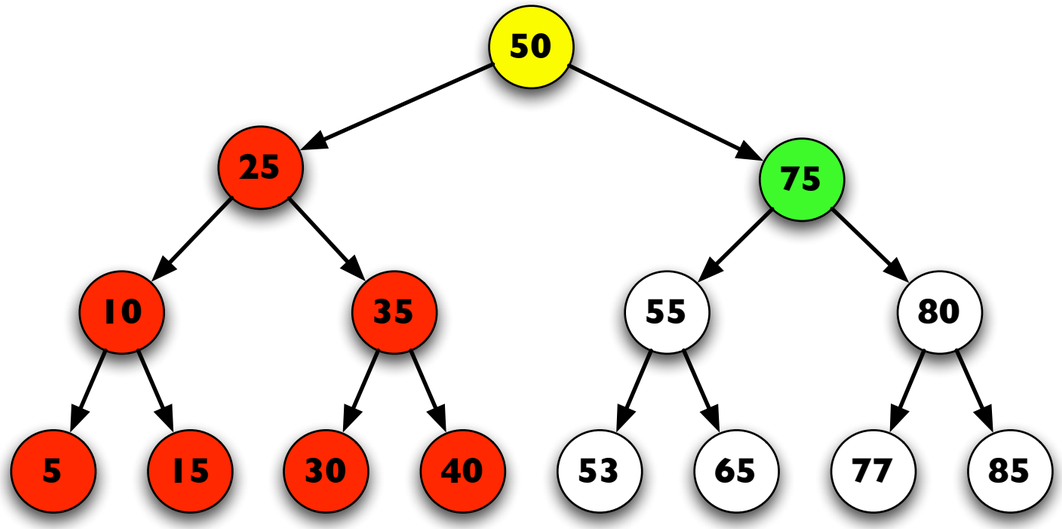


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75

- Visit root node
- Traverse TL
- Traverse TR



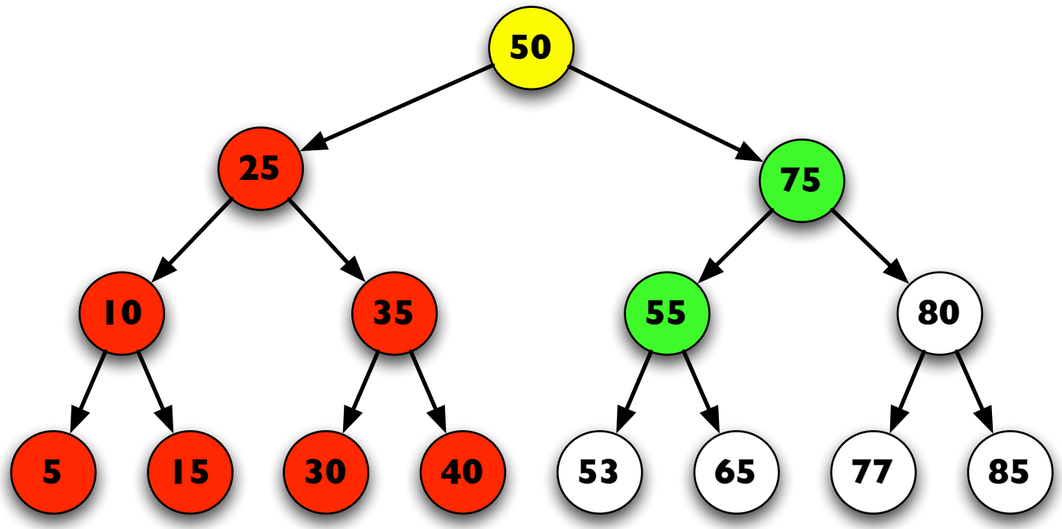
Visit root node

Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75



Visit root node

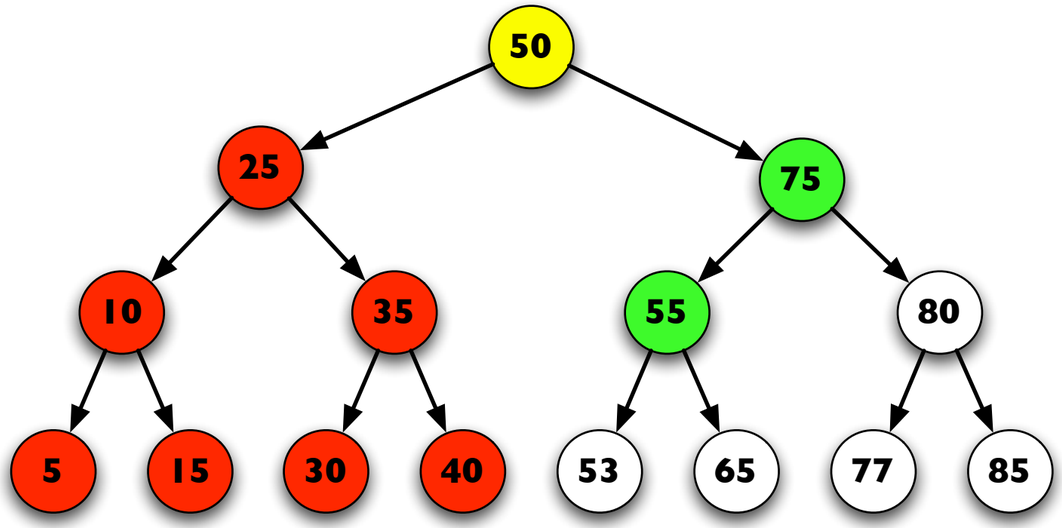
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75

- Visit root node



Visit root node

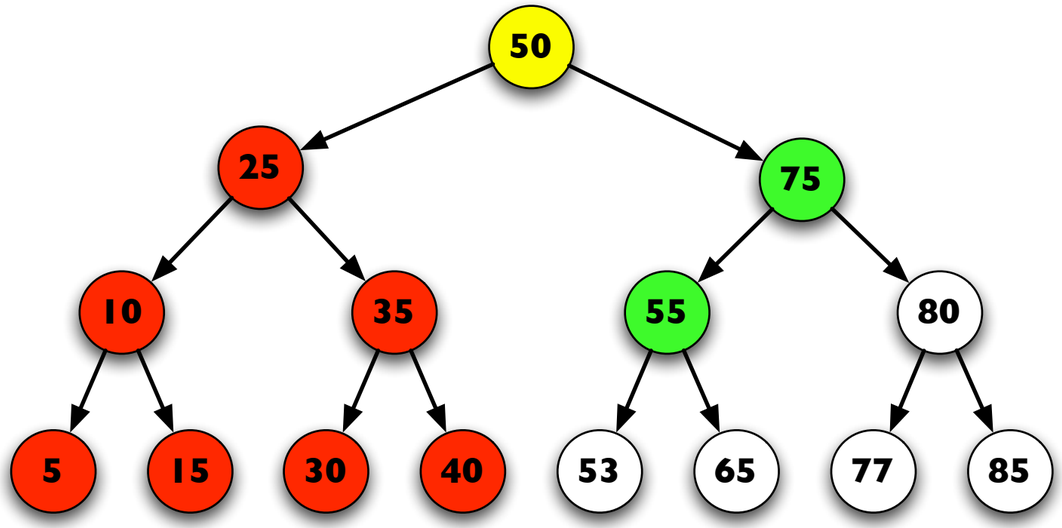
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75

- Visit root node
- Traverse TL
- Traverse TR

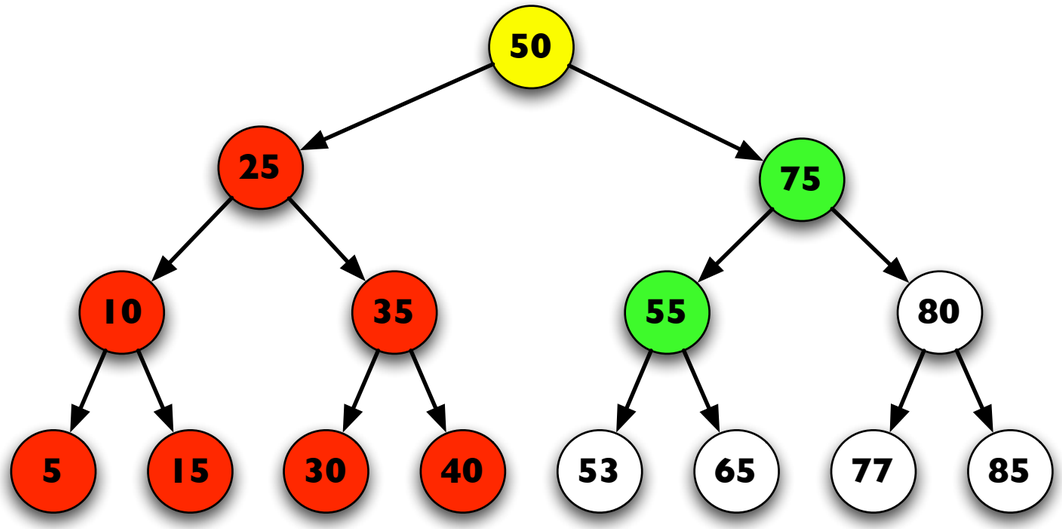


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55

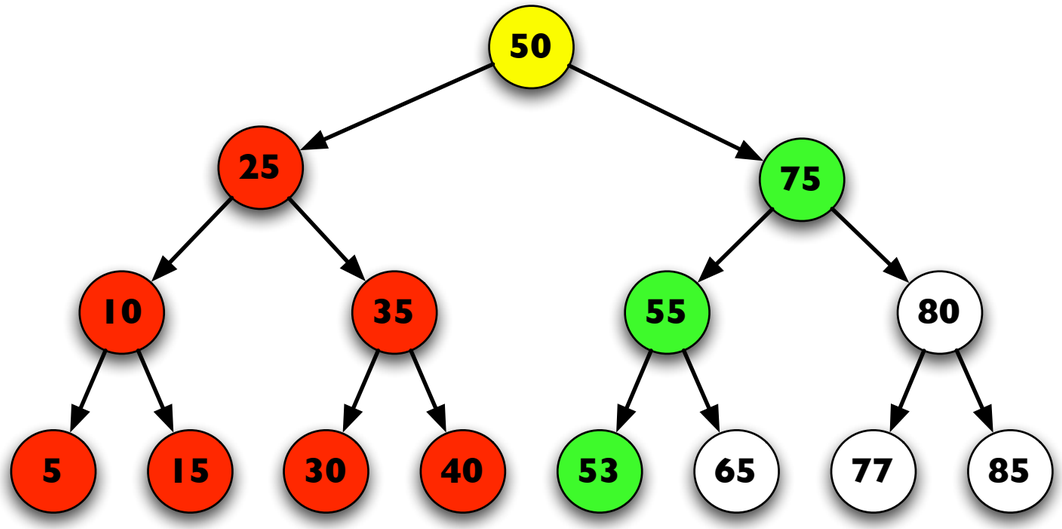
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55



Visit root node

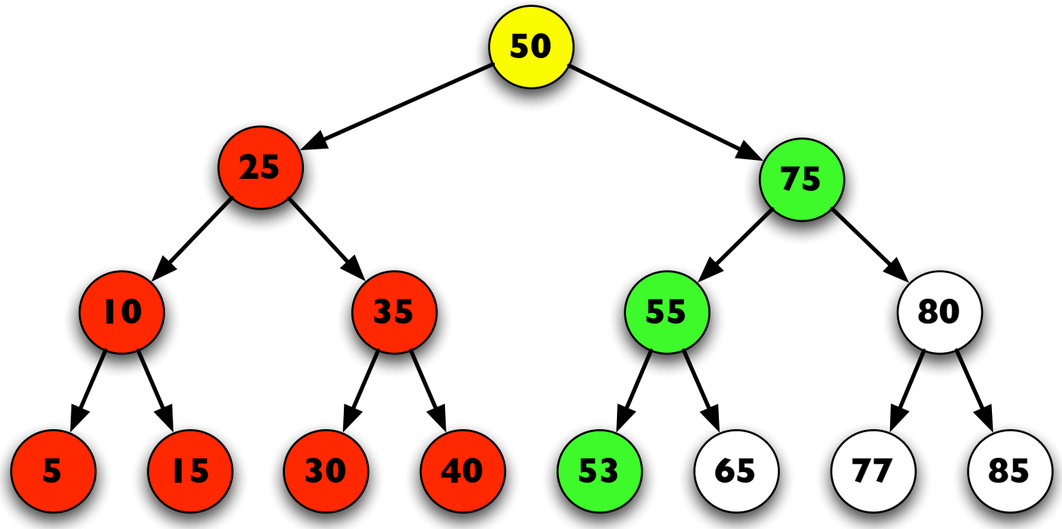
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55

- Visit root node



Visit root node

Traverse TL

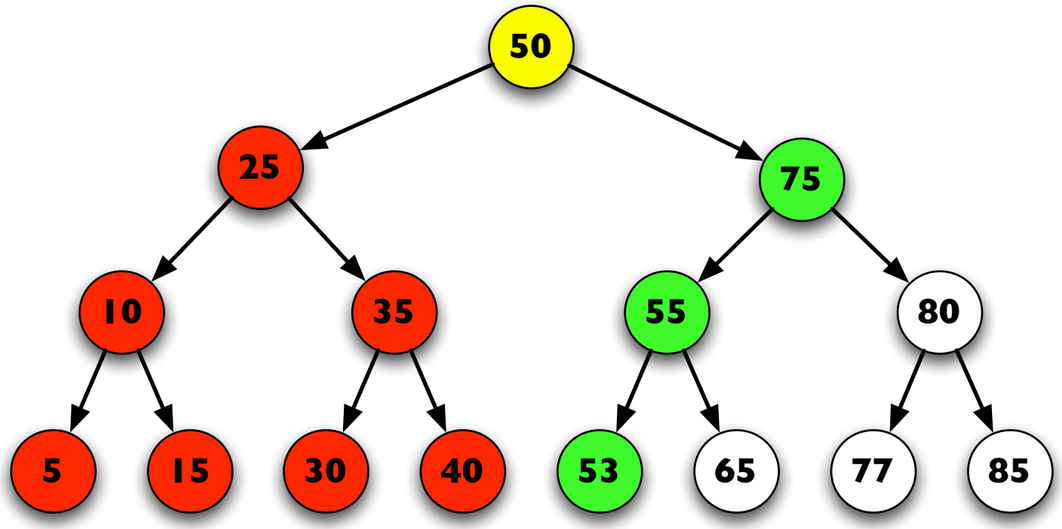
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55

- Visit root node

- Traverse TL



Visit root node

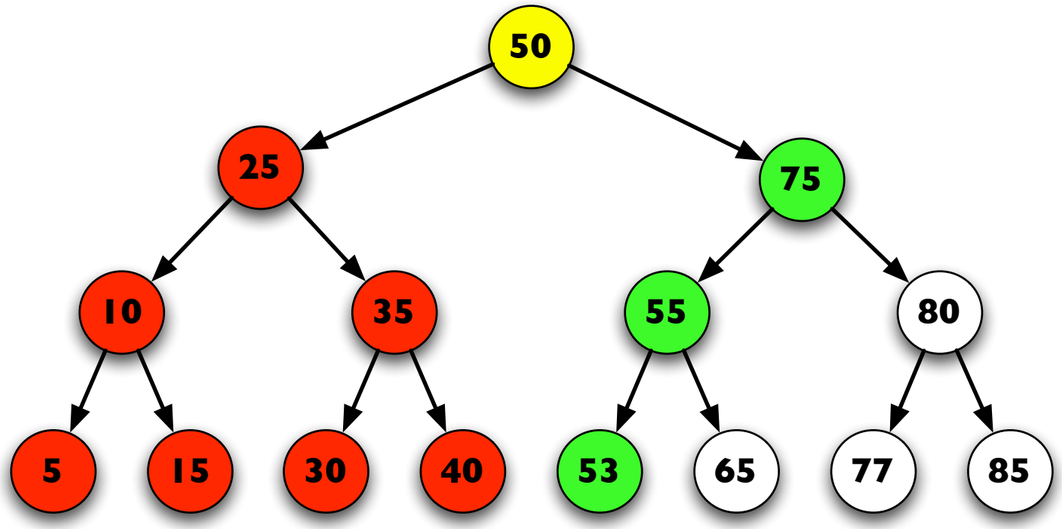
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53

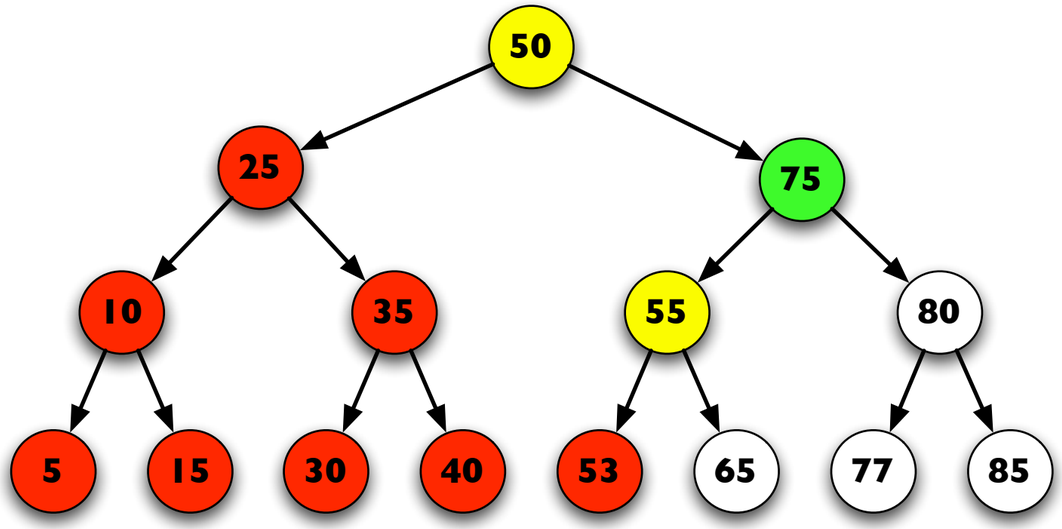
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53



Visit root node

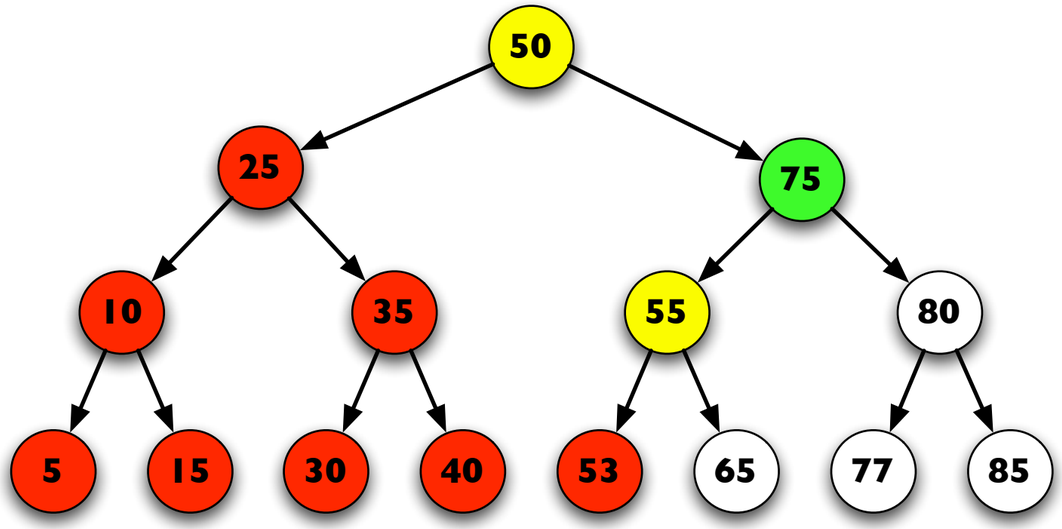
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53

- Visit root node



Visit root node

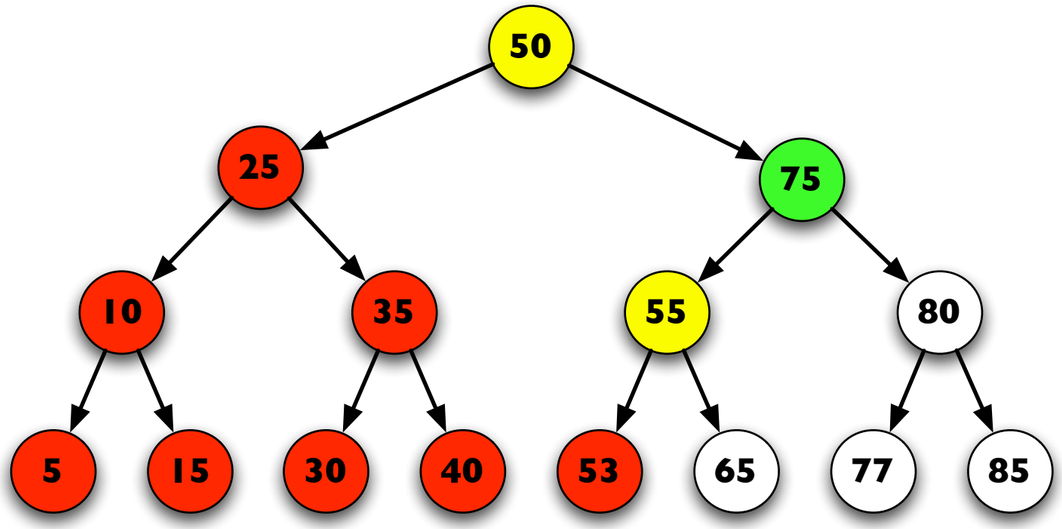
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53

- Visit root node
- Traverse TL
- Traverse TR

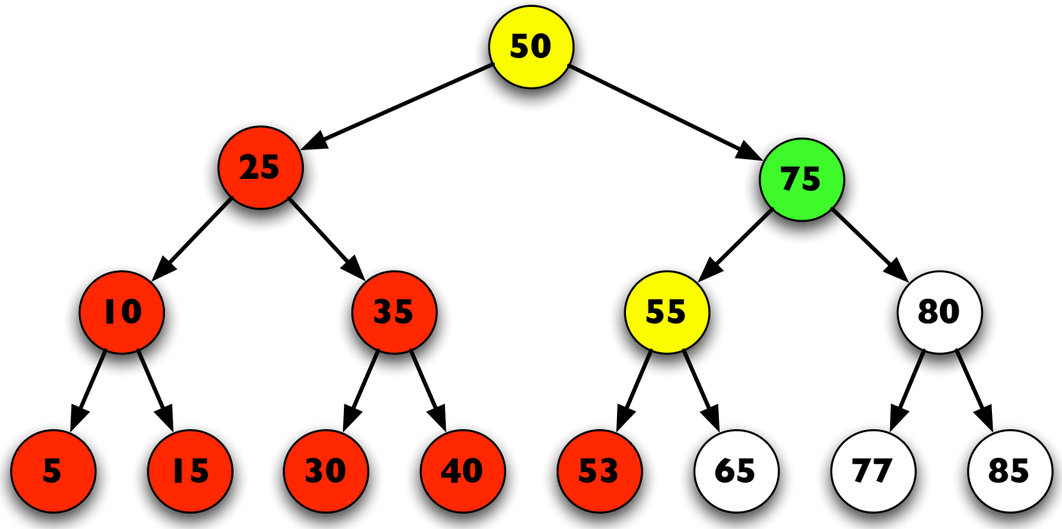


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53

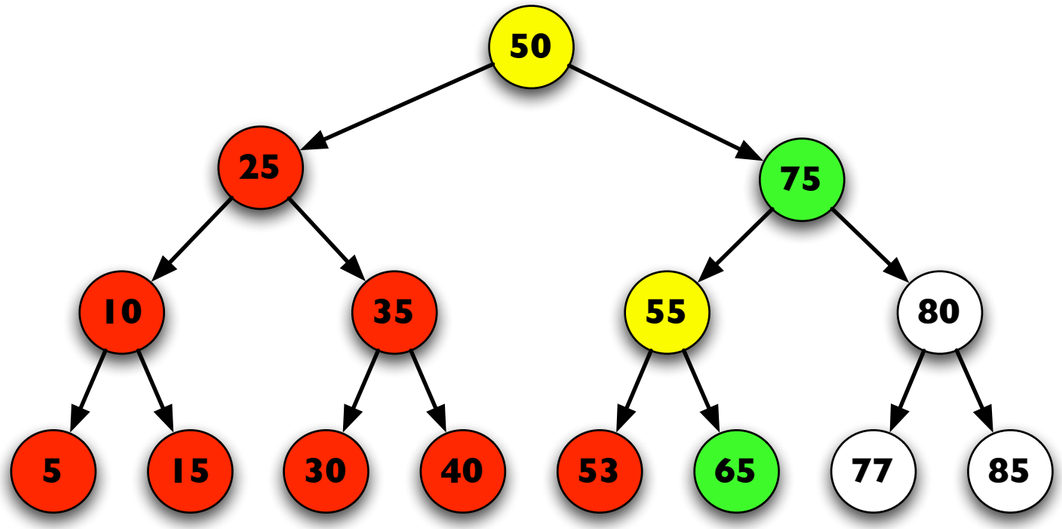
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53



Visit root node

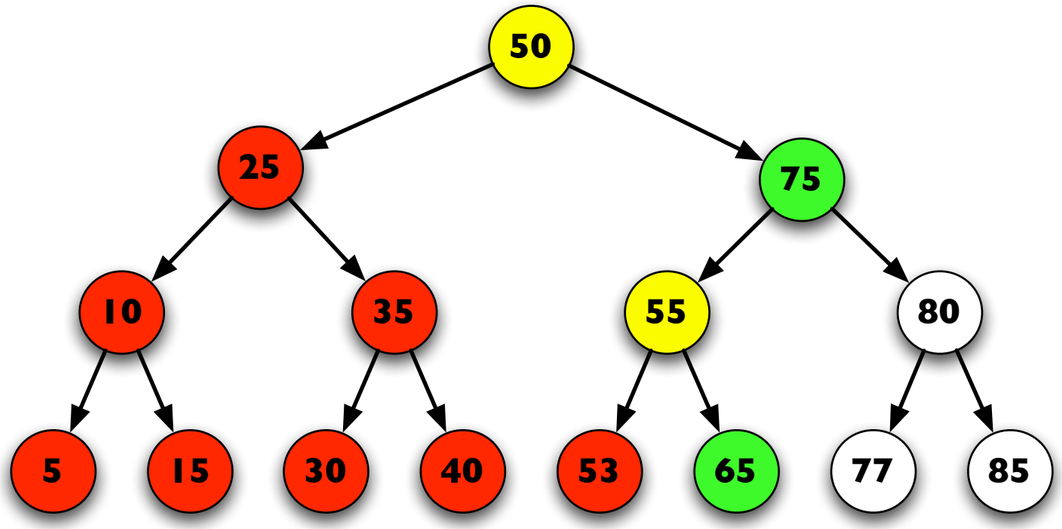
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53

- Visit root node



Visit root node

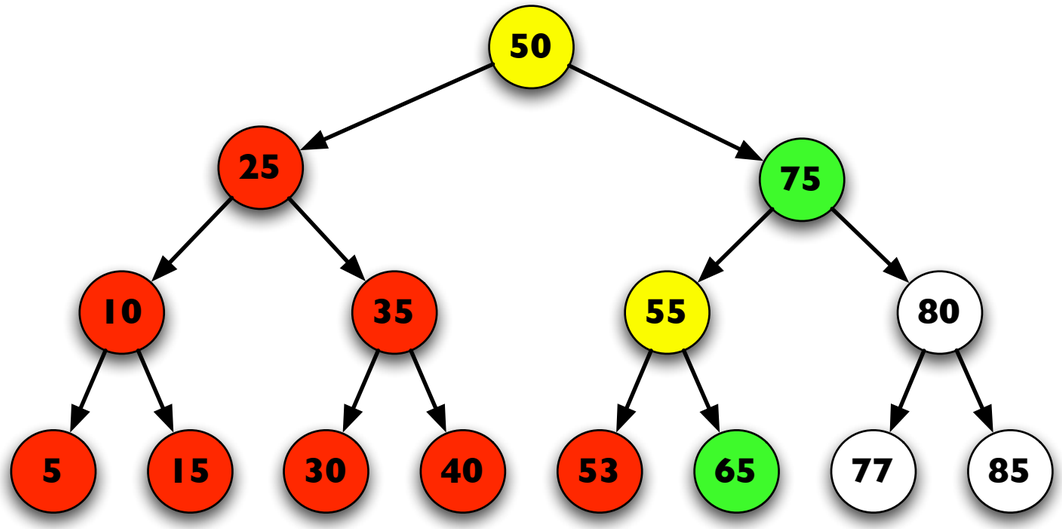
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53

- Visit root node
- Traverse TL
- Traverse TR

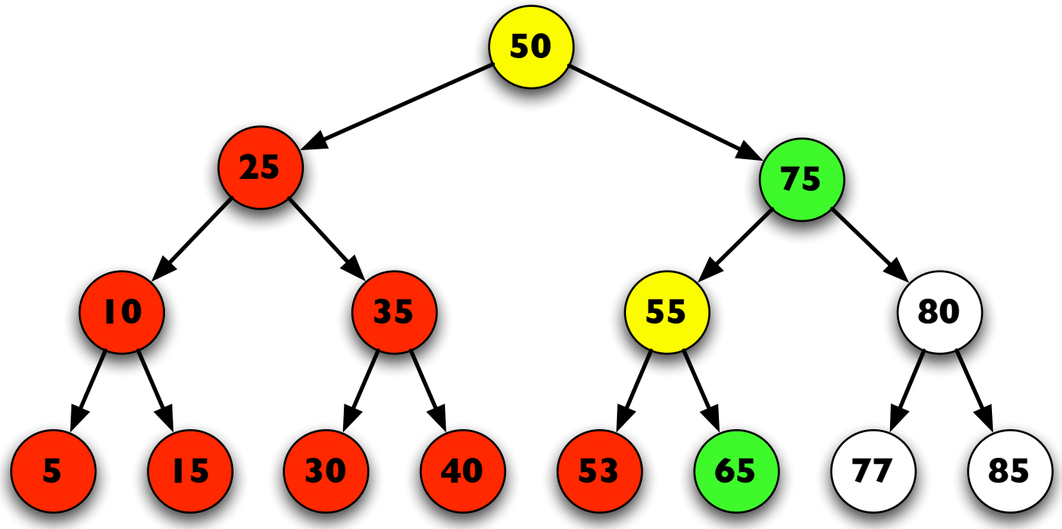


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65

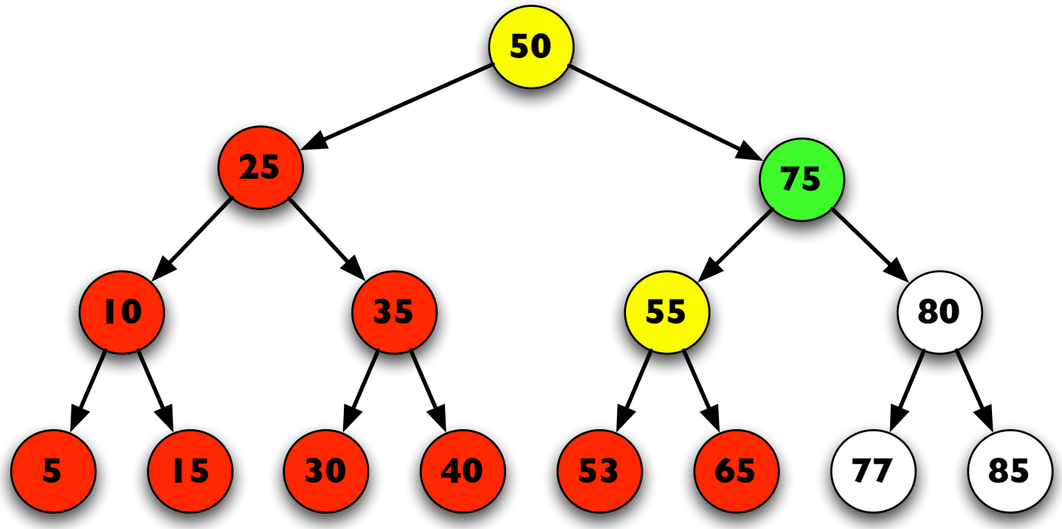
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65



Visit root node

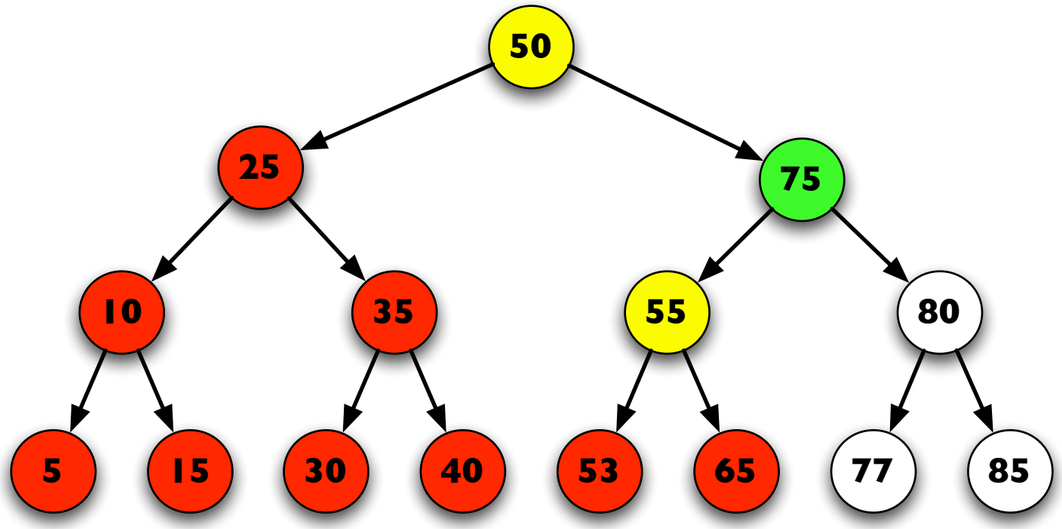
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65

- Visit root node



Visit root node

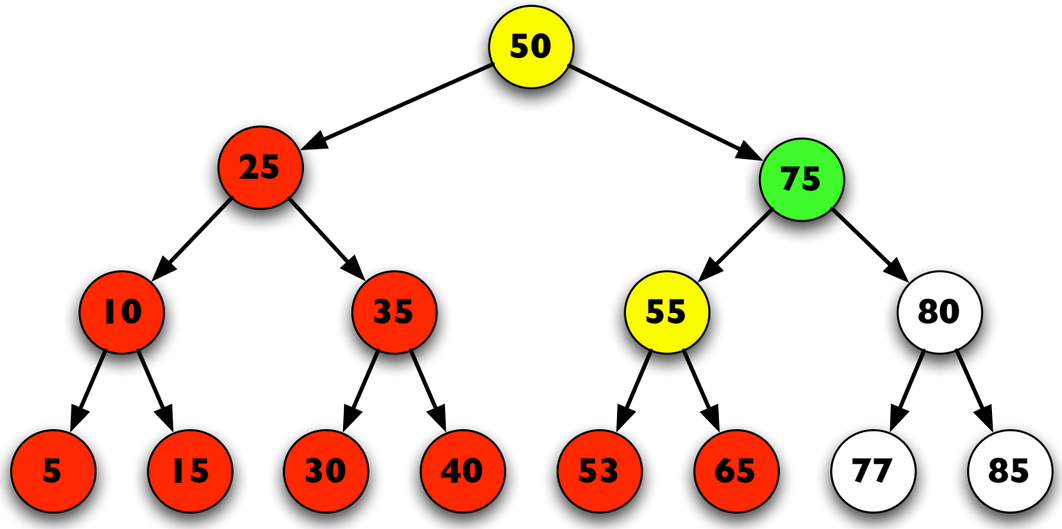
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65

- Visit root node
- Traverse TL
- Traverse TR

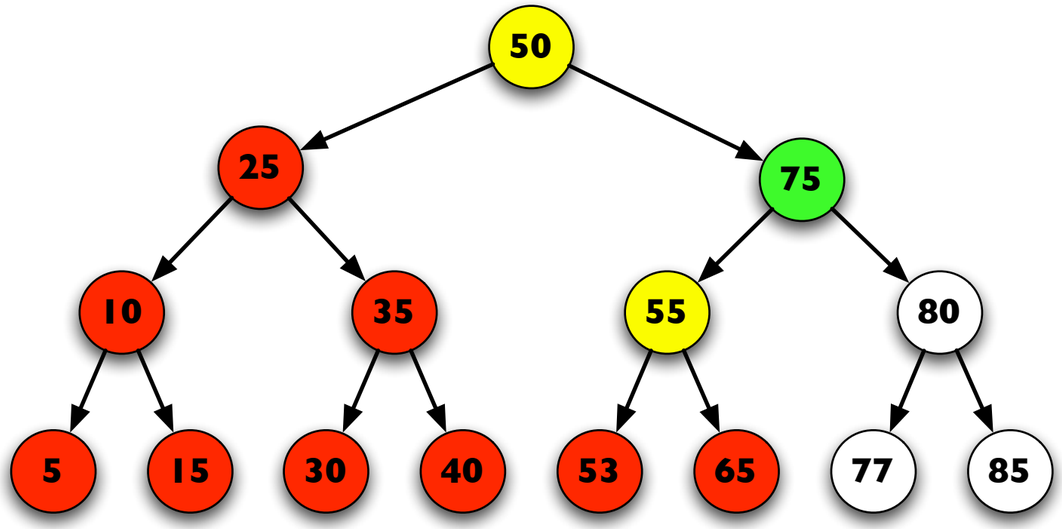


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65

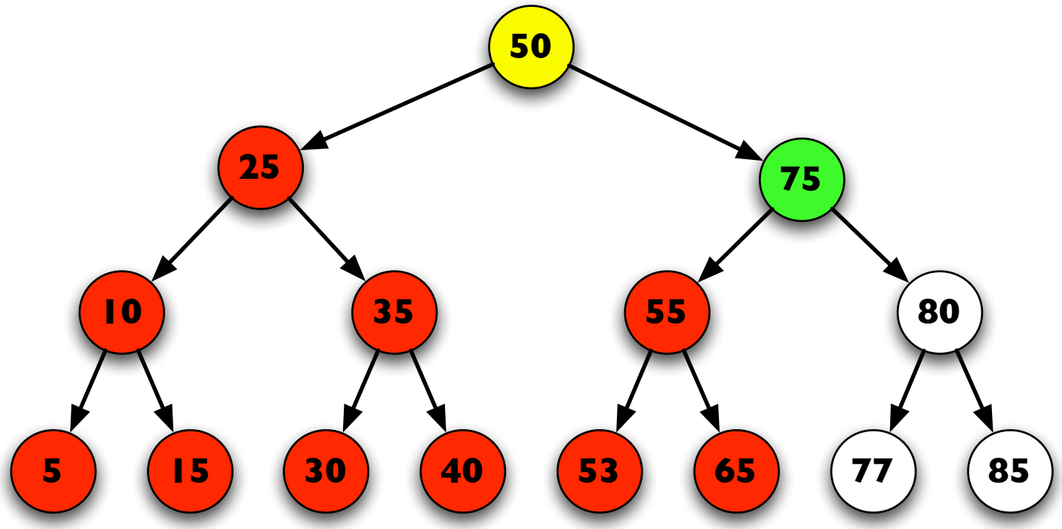
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65



Visit root node

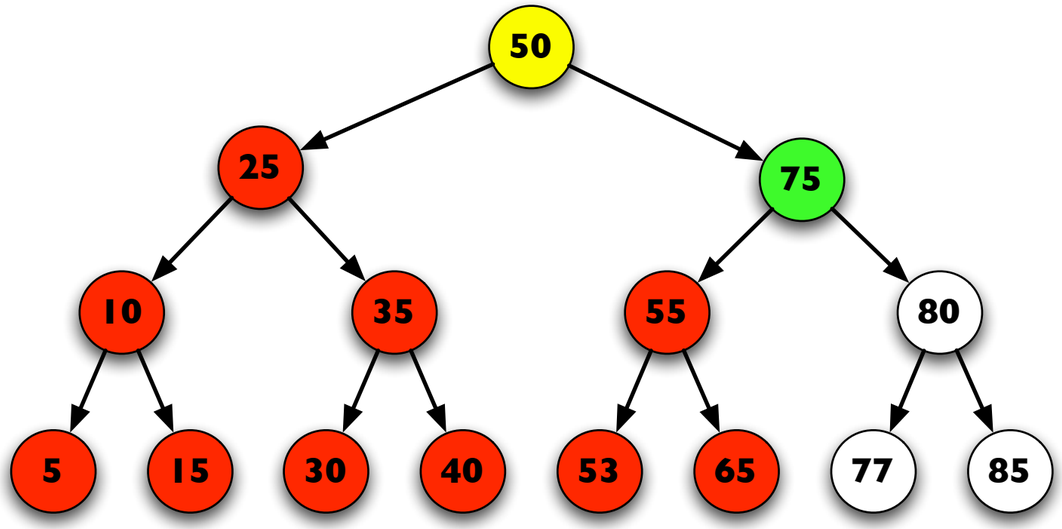
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65

- Visit root node



Visit root node

Traverse TL

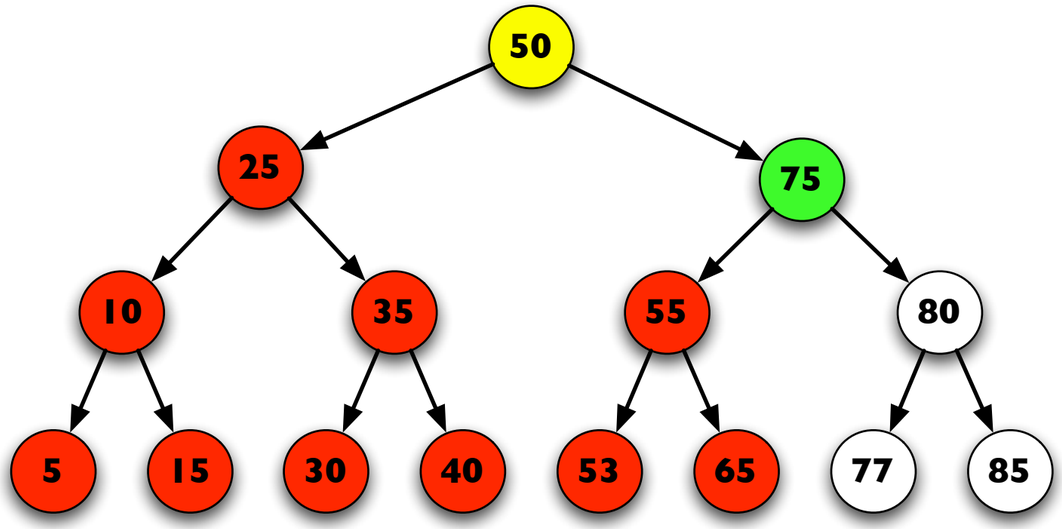
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65

- Visit root node

- Traverse TL



Visit root node

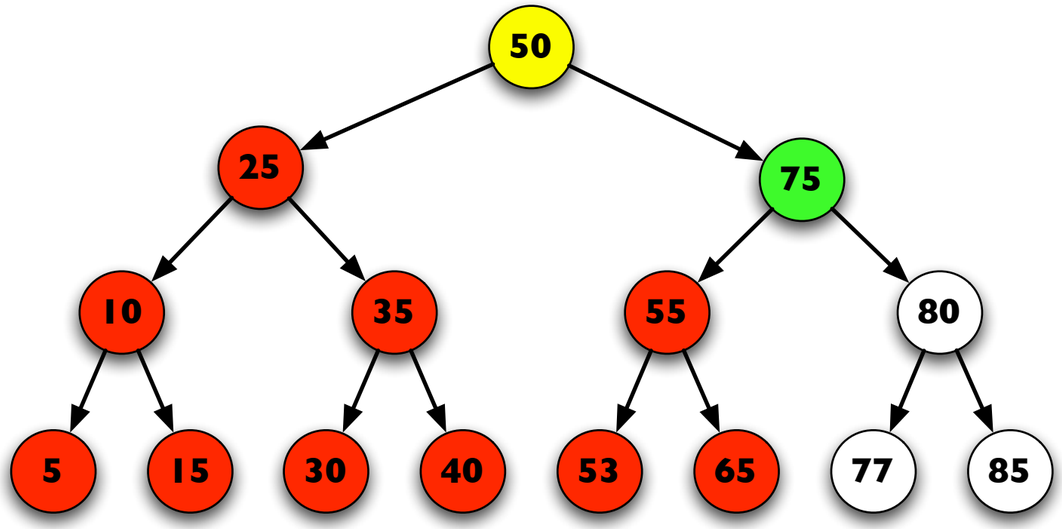
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65

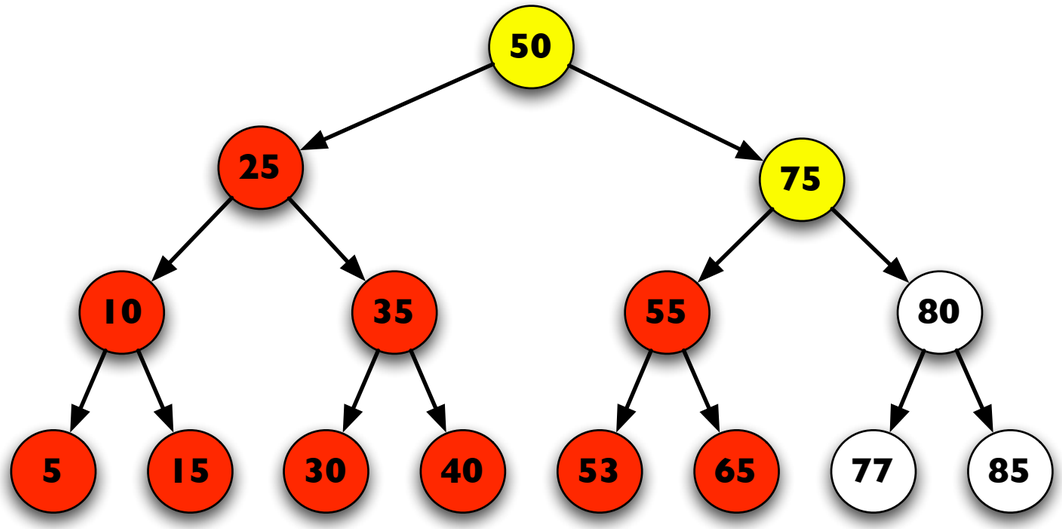
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65



Visit root node

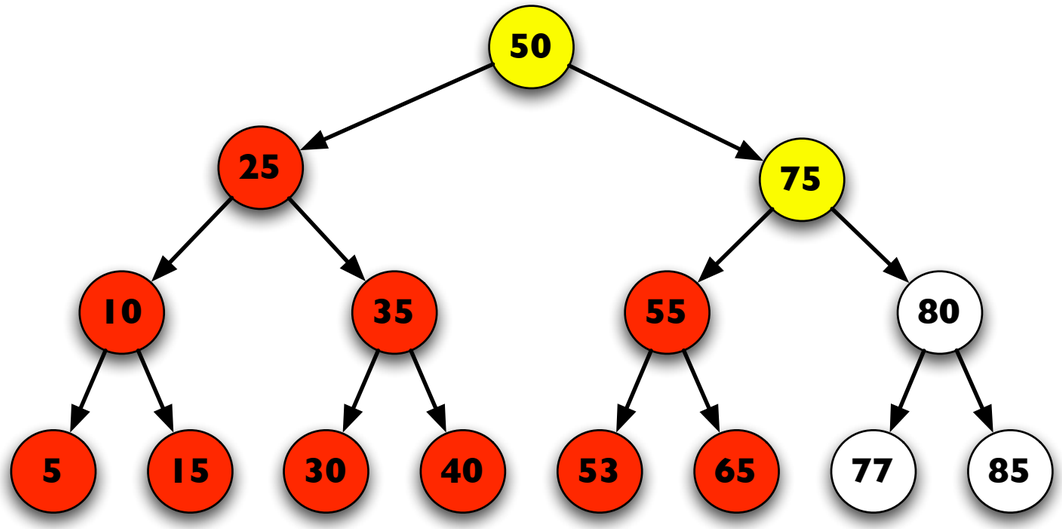
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65

- Visit root node



Visit root node

Traverse TL

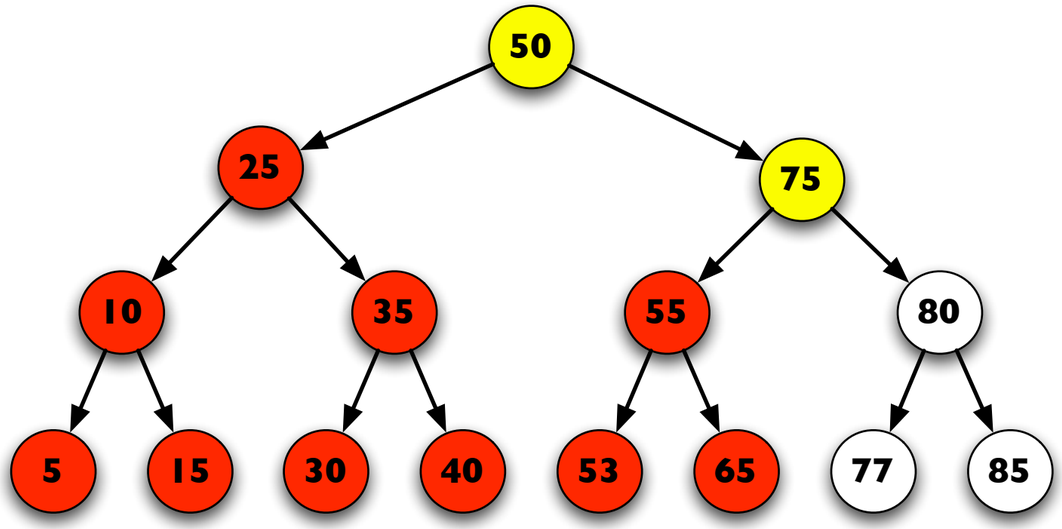
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65

- Visit root node

- Traverse TL



Visit root node

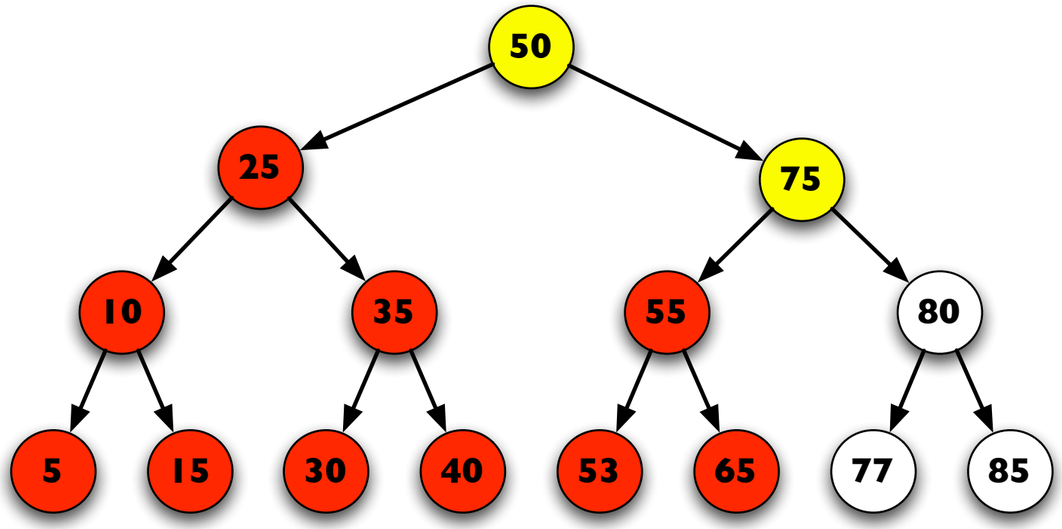
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65

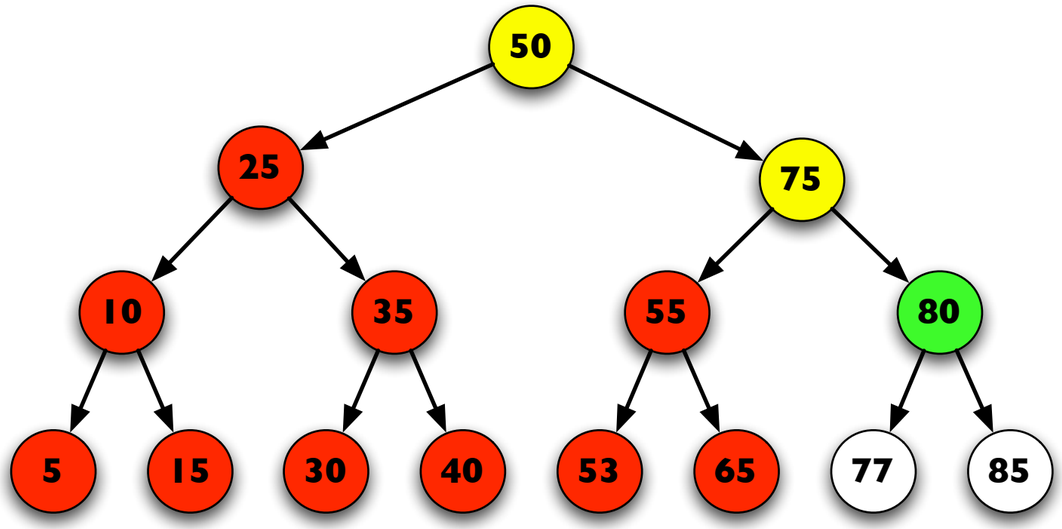
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65



Visit root node

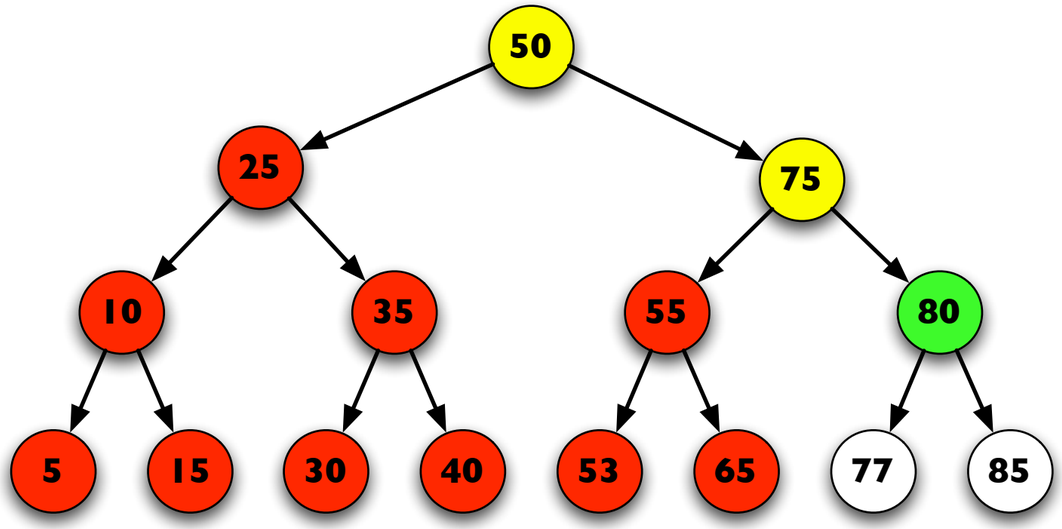
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65

- Visit root node



Visit root node

Traverse TL

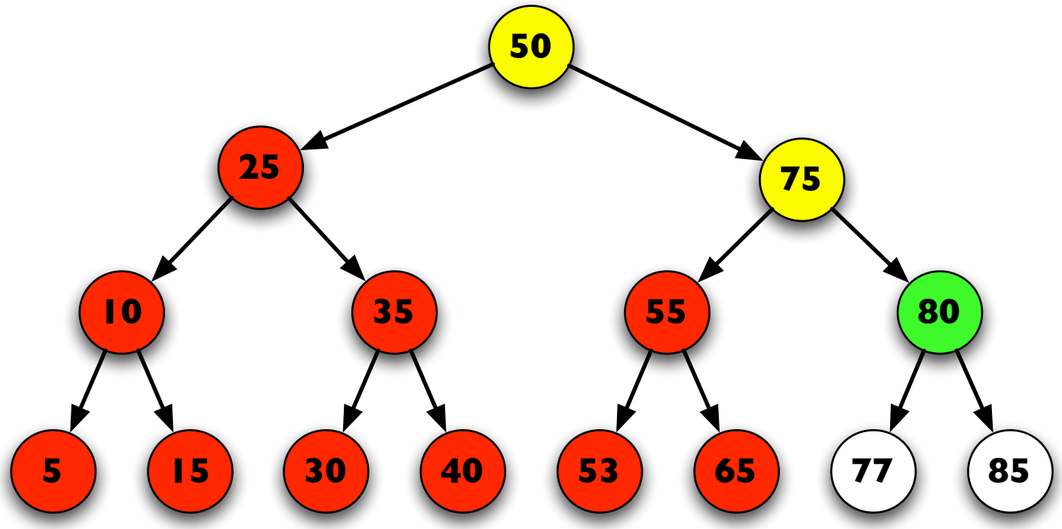
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65

- Visit root node

- Traverse TL



Visit root node

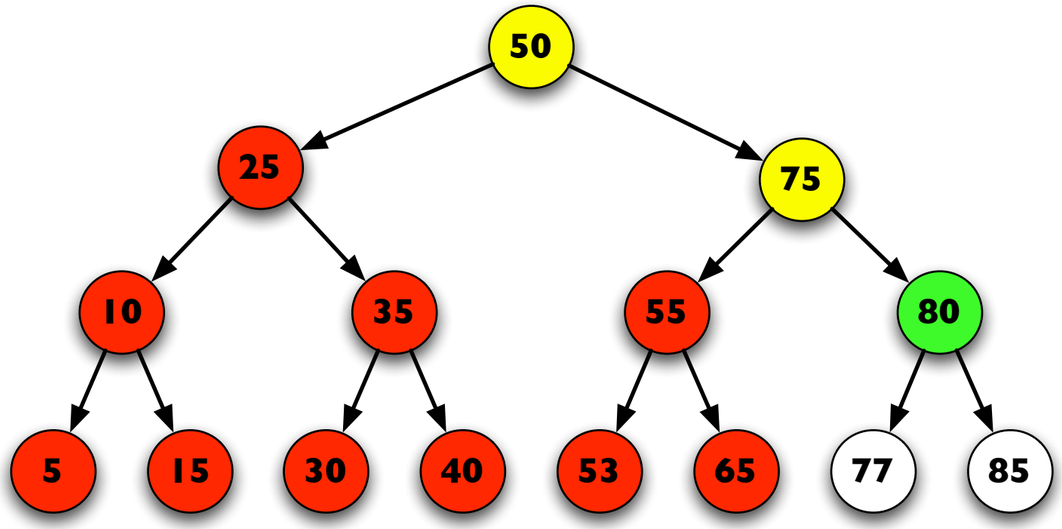
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80

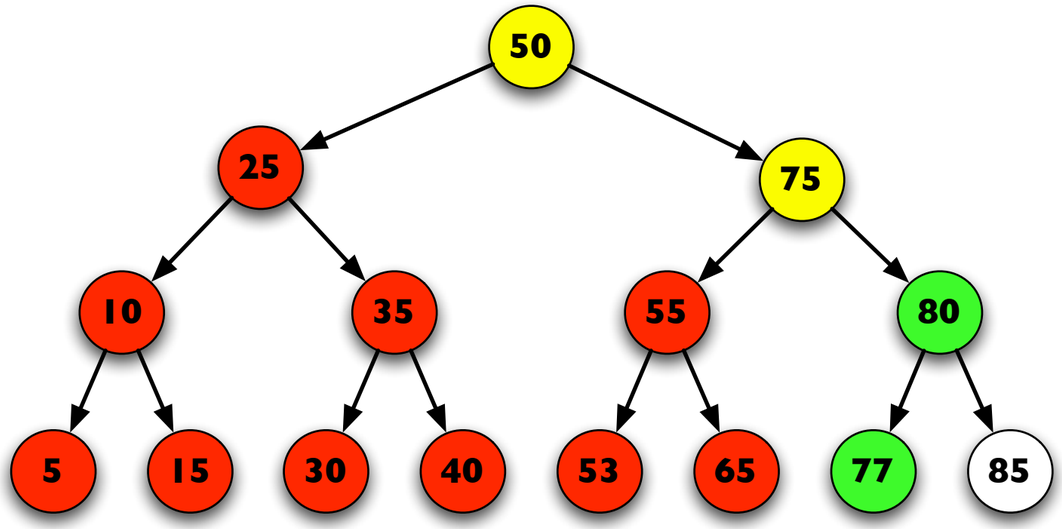
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80



Visit root node

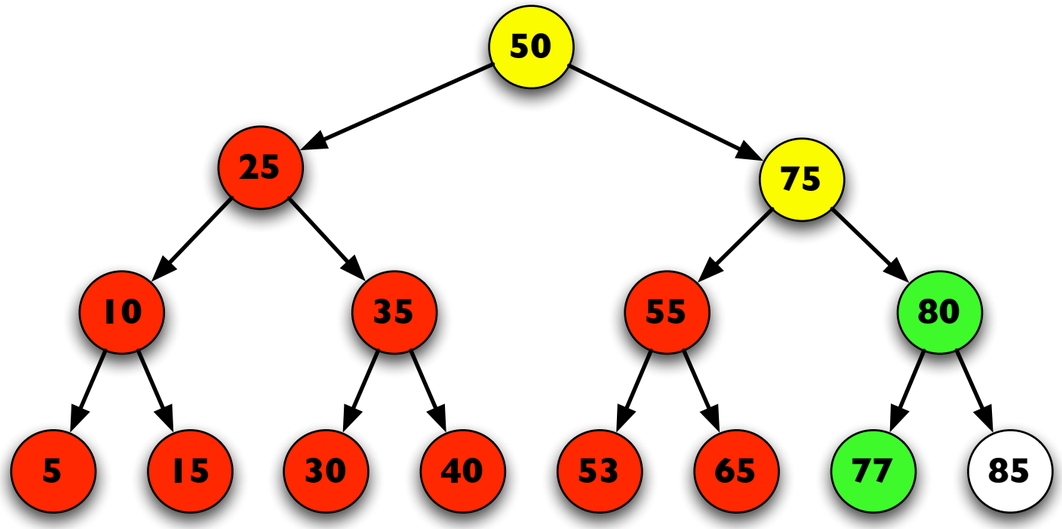
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80

- Visit root node



Visit root node

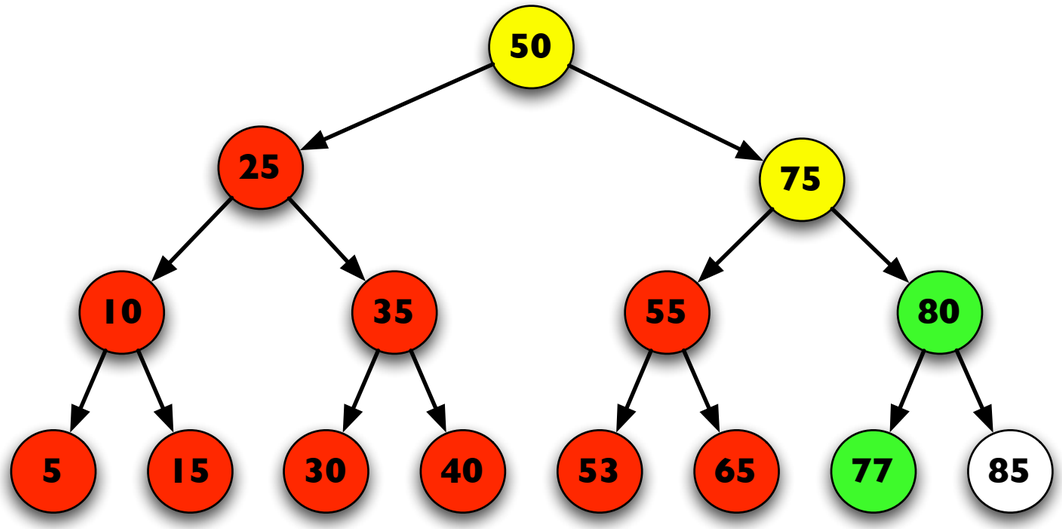
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80

- Visit root node
- Traverse TL
- Traverse TR

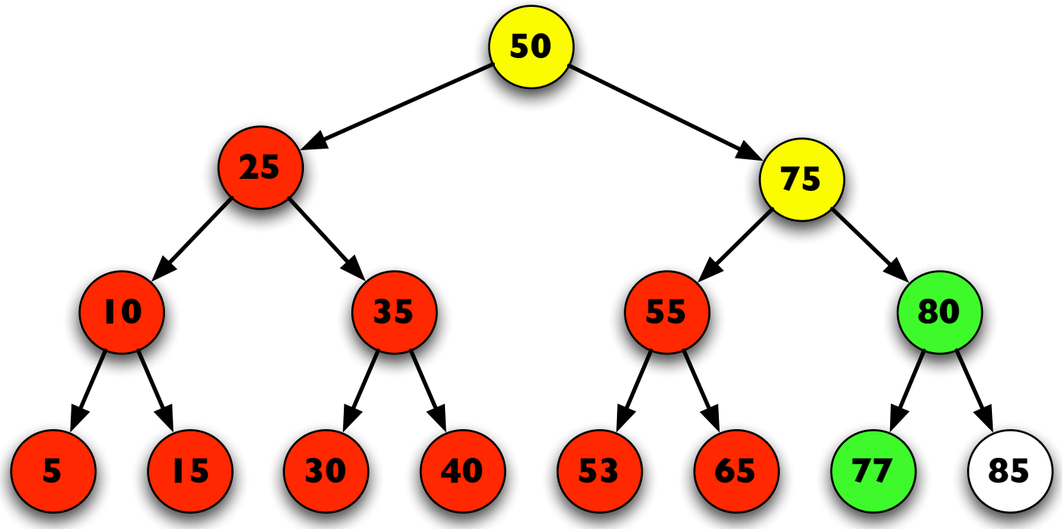


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77

- Visit root node
- Traverse TL
- Traverse TR



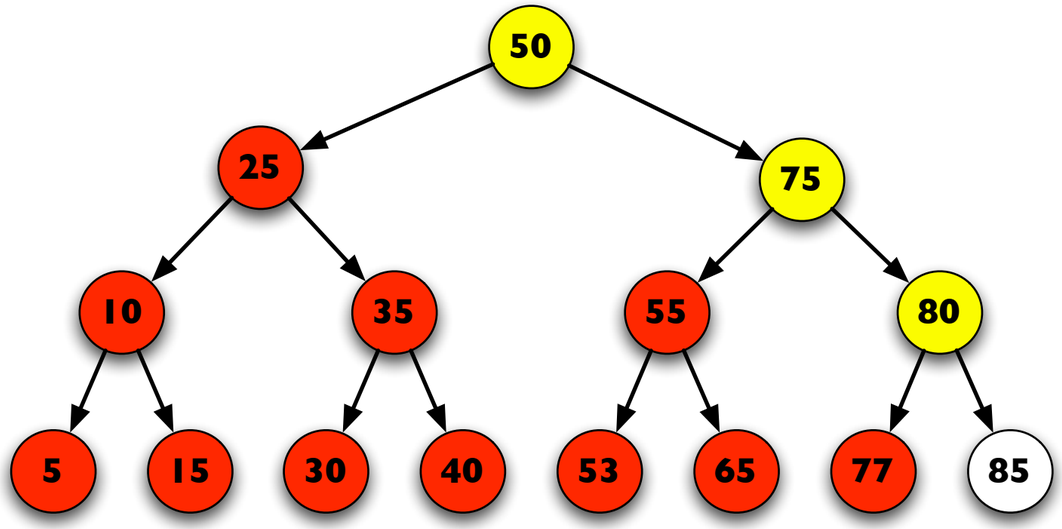
Visit root node

Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77



Visit root node

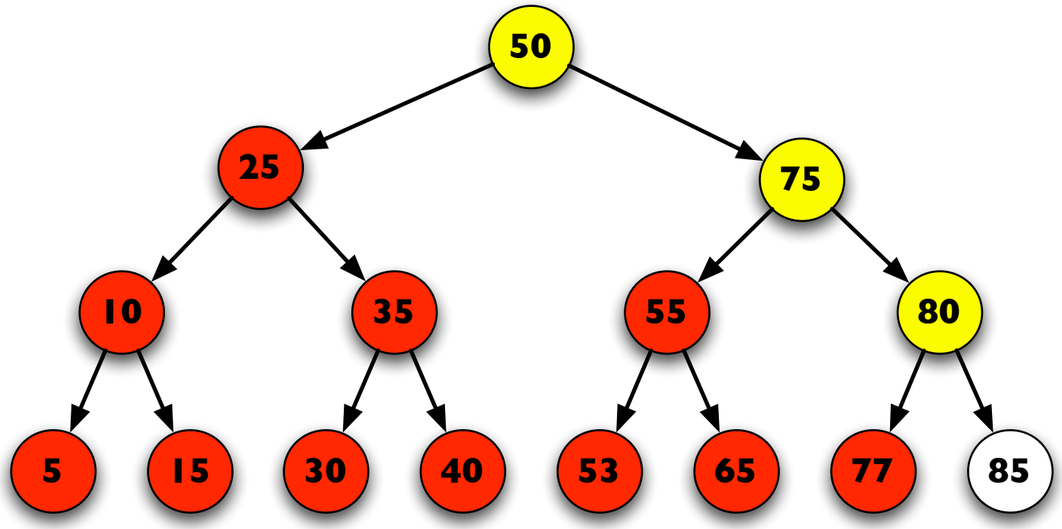
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77

- Visit root node



Visit root node

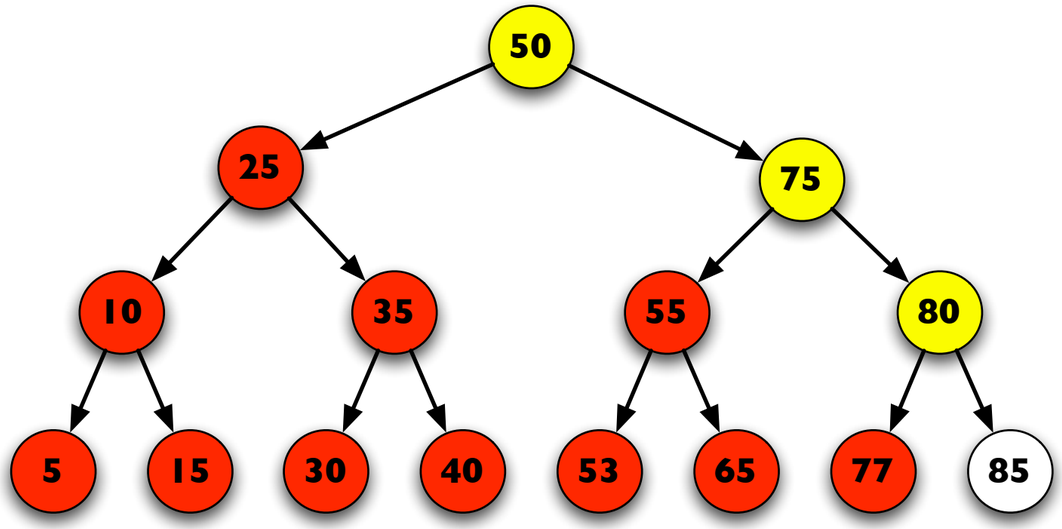
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77

- Visit root node
- Traverse TL
- Traverse TR



Visit root node

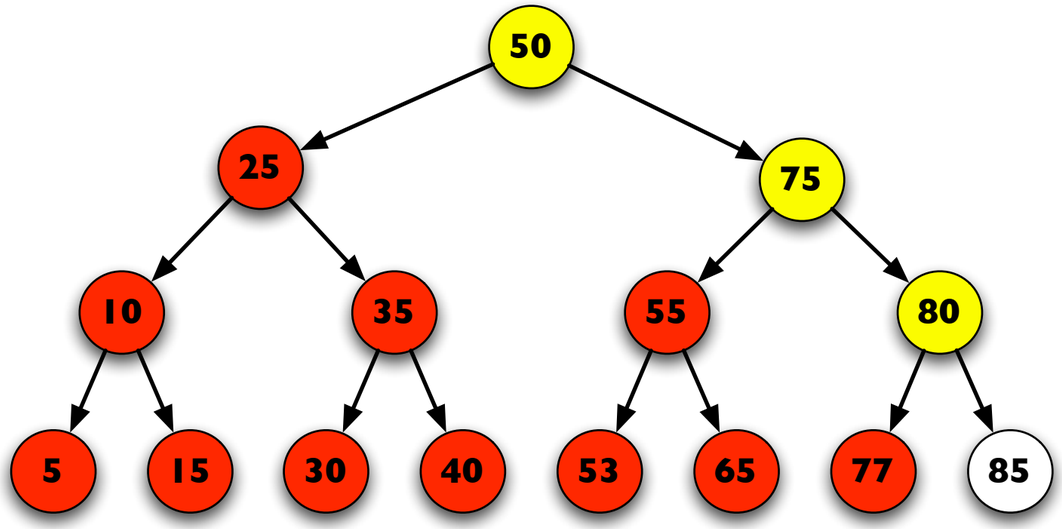
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77

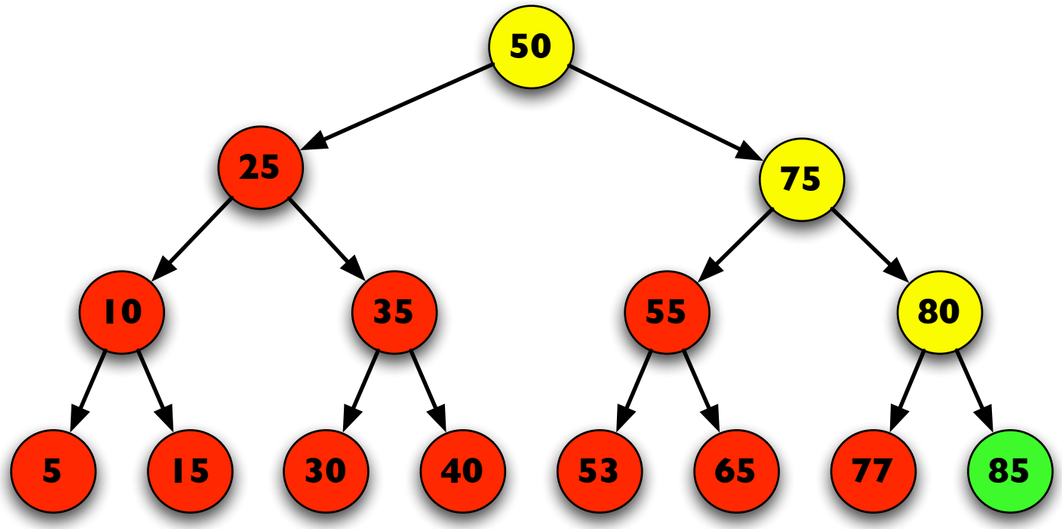
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77



Visit root node

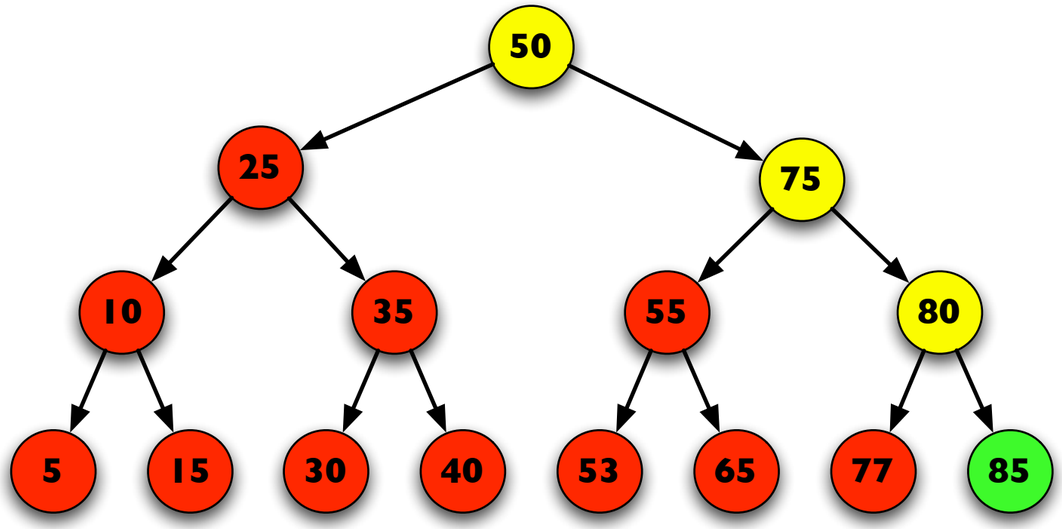
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77

- Visit root node



Visit root node

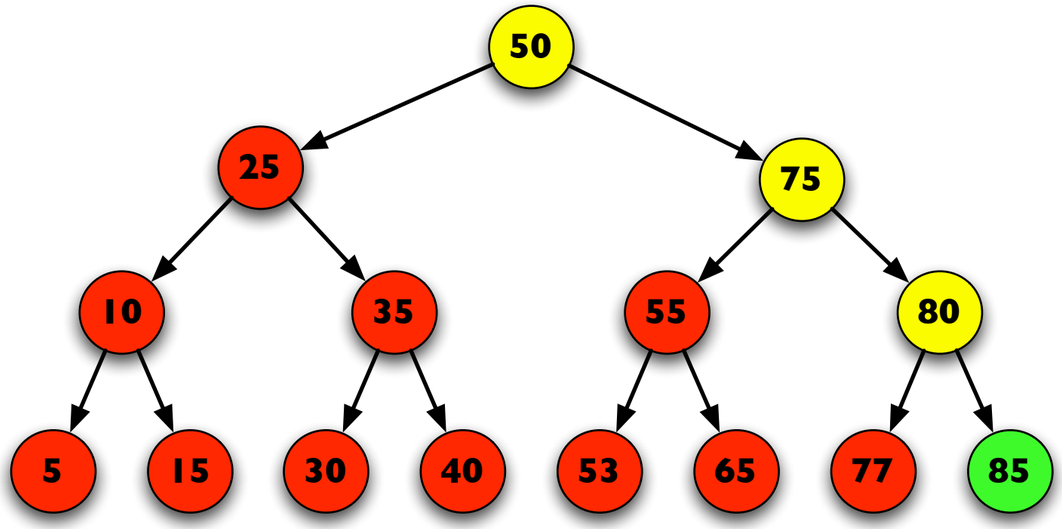
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77

- Visit root node
- Traverse TL
- Traverse TR

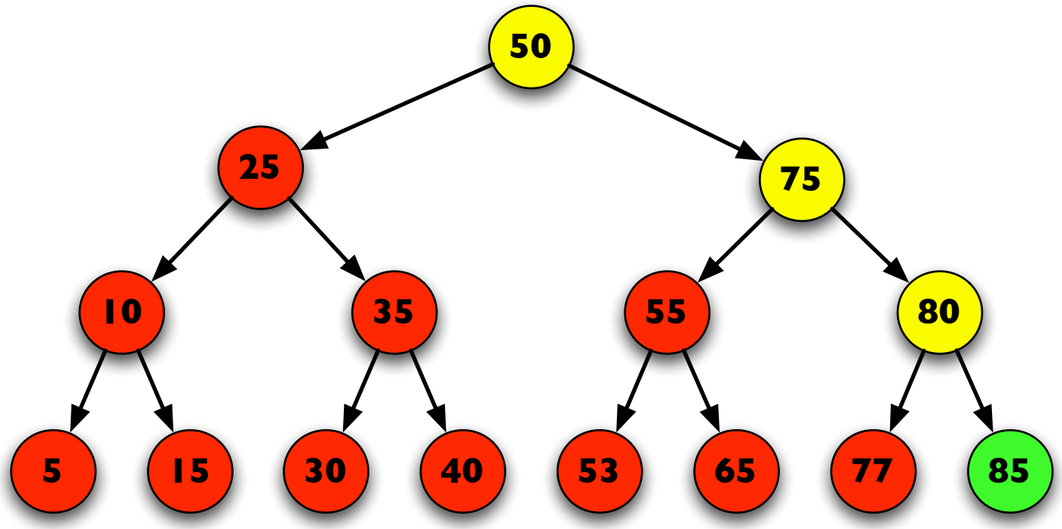


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

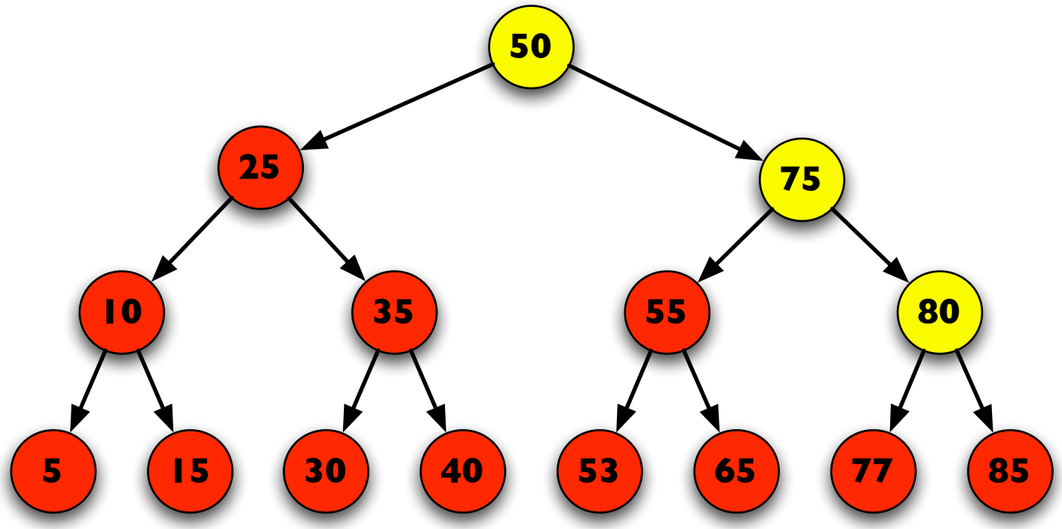
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85



Visit root node

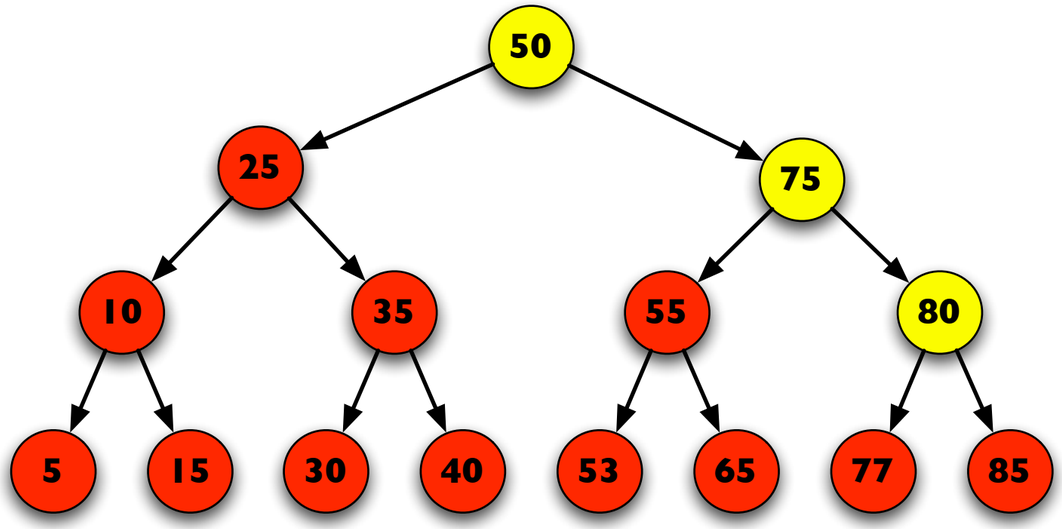
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

- Visit root node



Visit root node

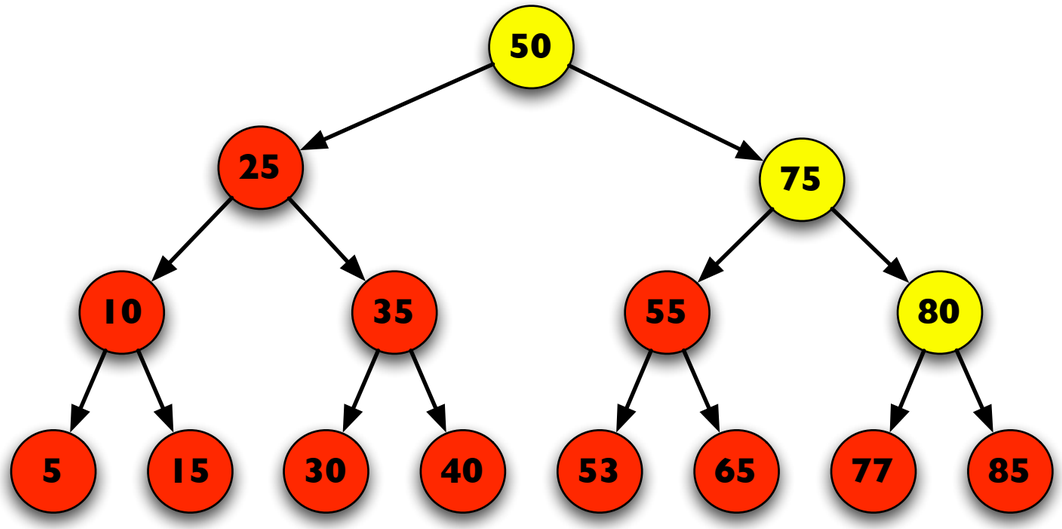
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

- Visit root node
- Traverse TL
- Traverse TR

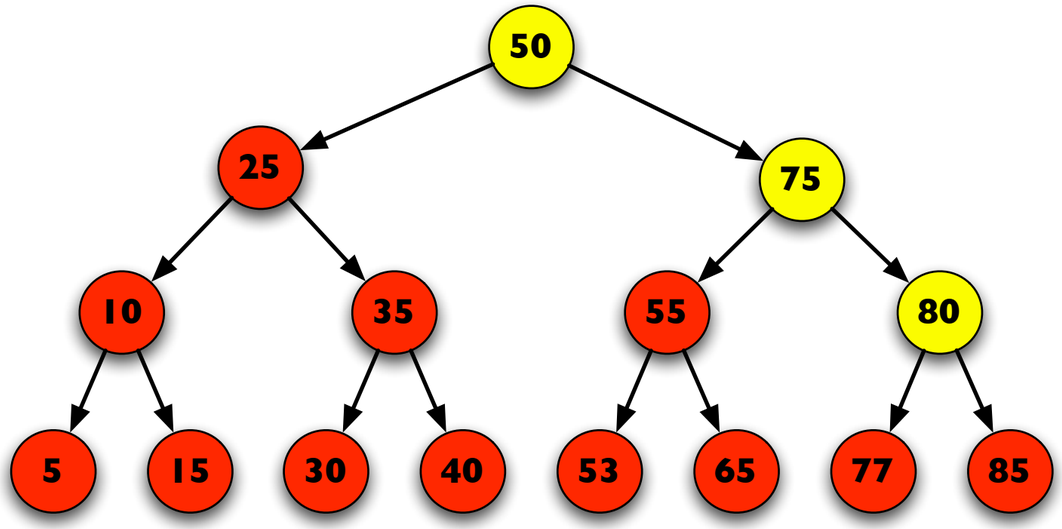


Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

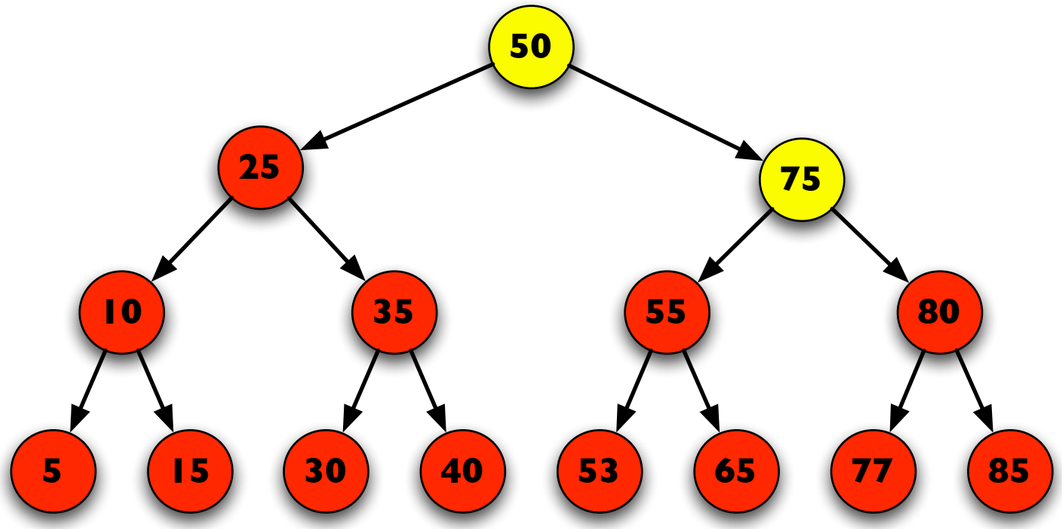
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85



Visit root node

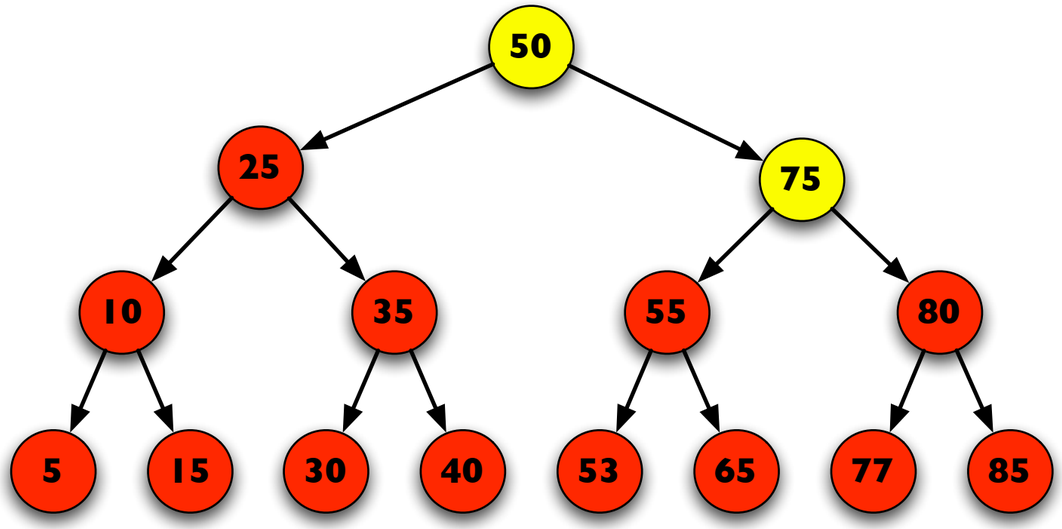
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

- Visit root node



Visit root node

Traverse TL

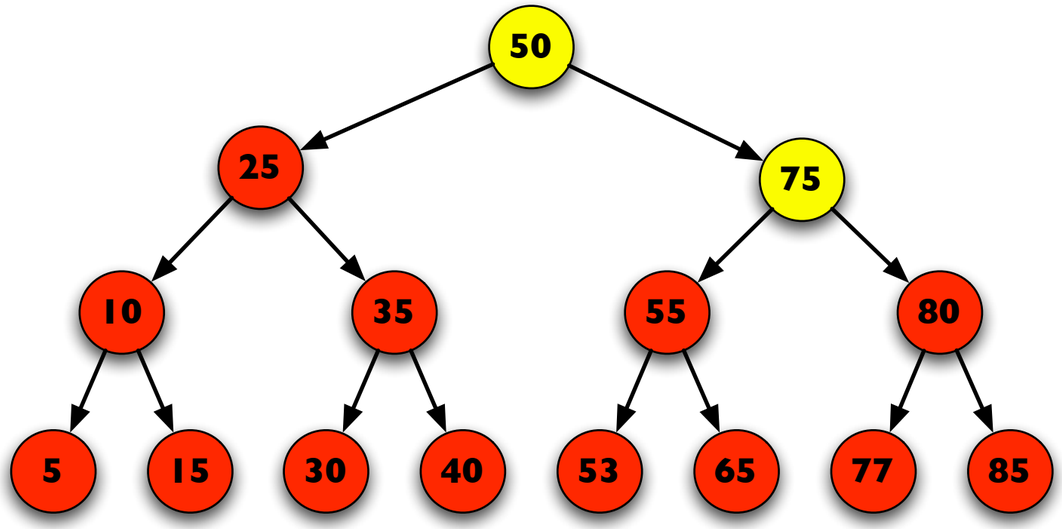
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

- Visit root node

- Traverse TL



Visit root node

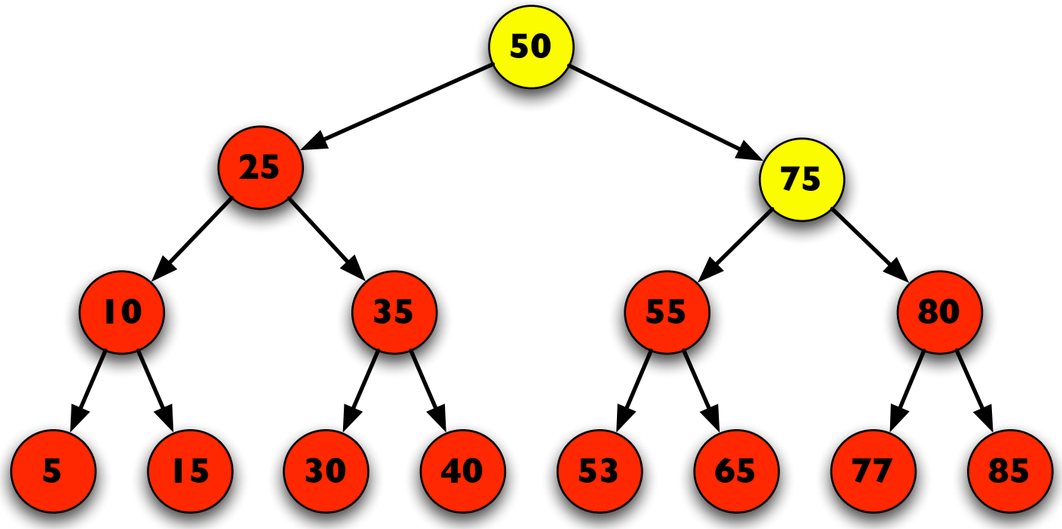
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

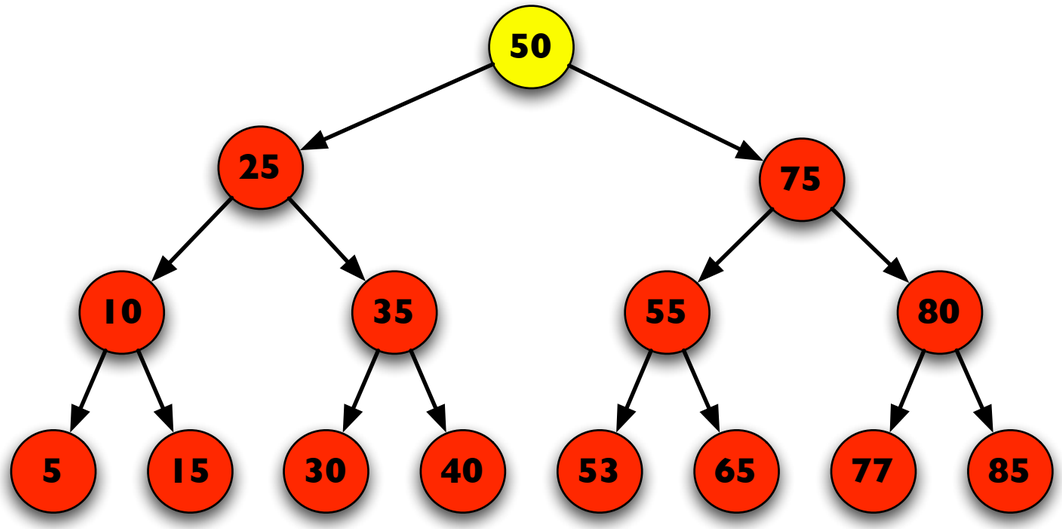
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85



Visit root node

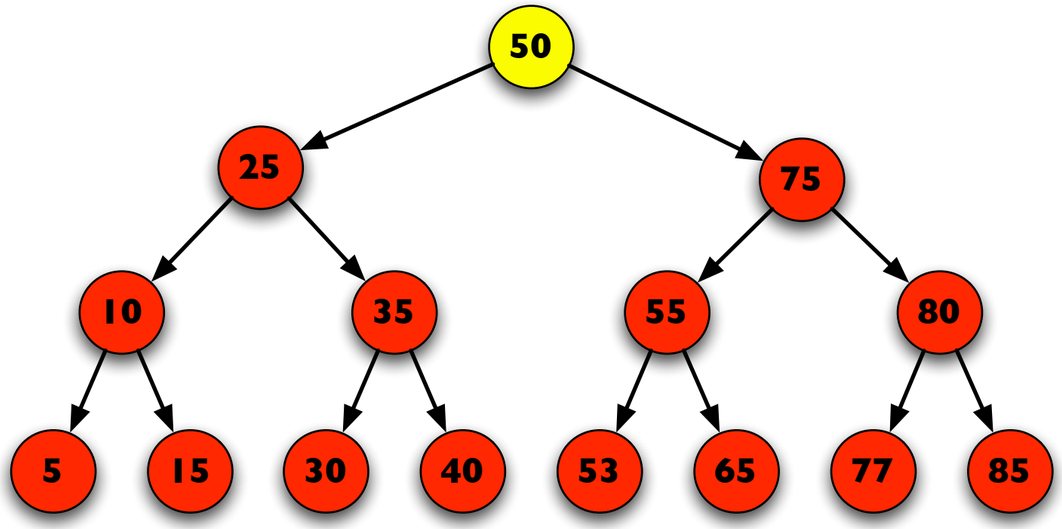
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

- Visit root node



Visit root node

Traverse TL

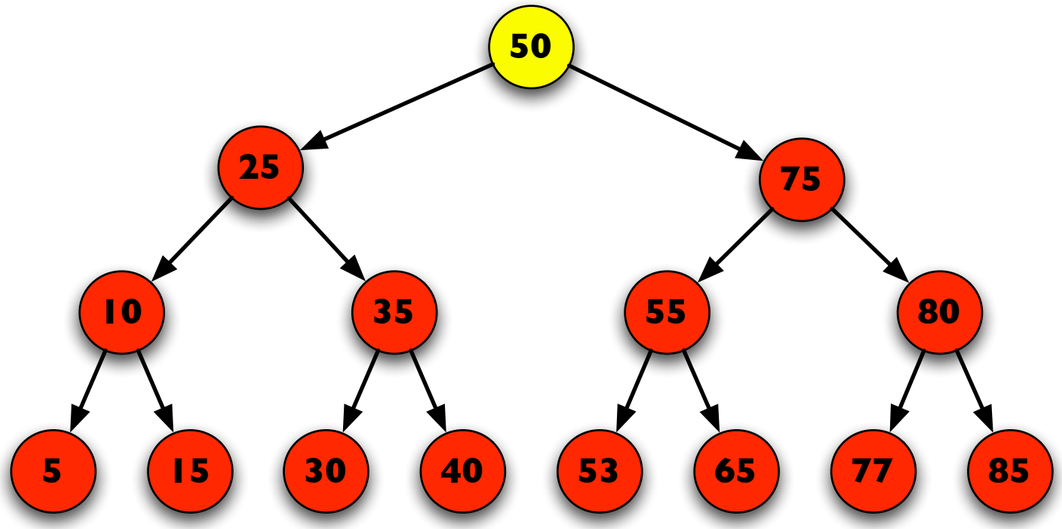
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

- Visit root node

- Traverse TL



Visit root node

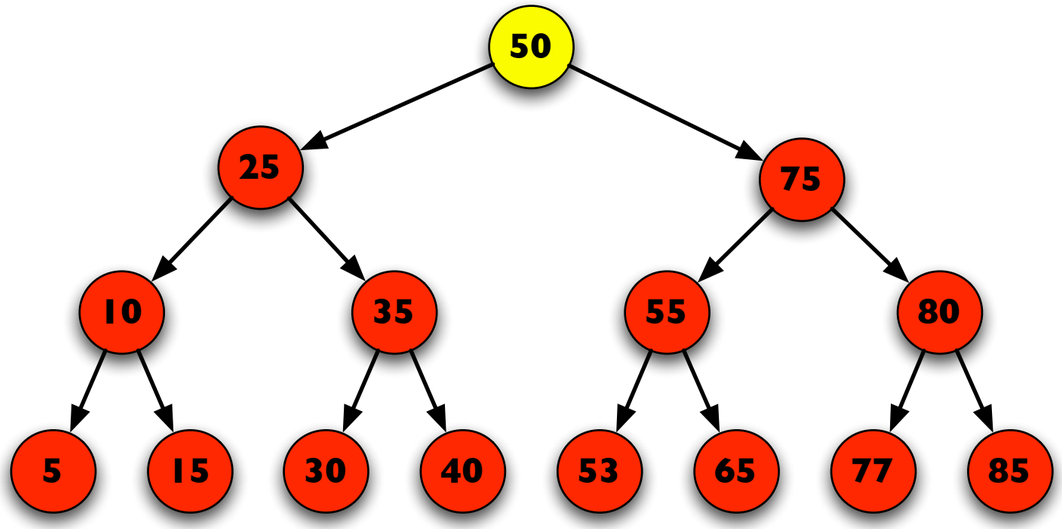
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

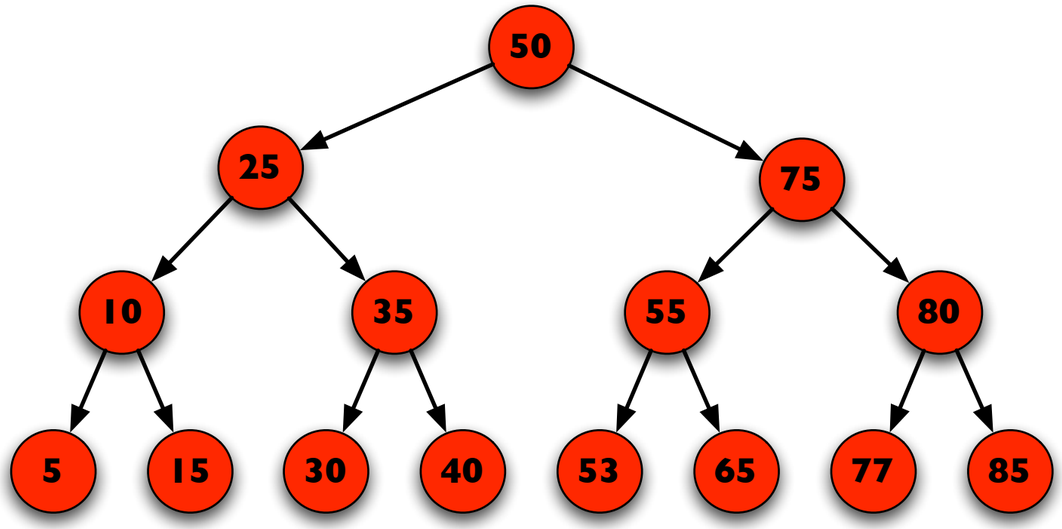
- Visit root node
- Traverse TL
- Traverse TR



Visit root node
Traverse TL
Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85



Visit root node

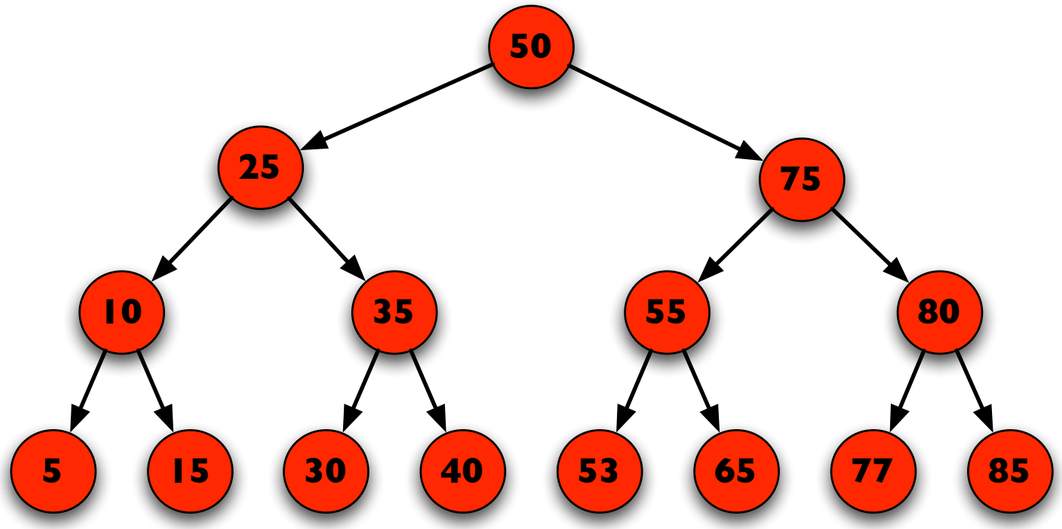
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

- Visit root node



Visit root node

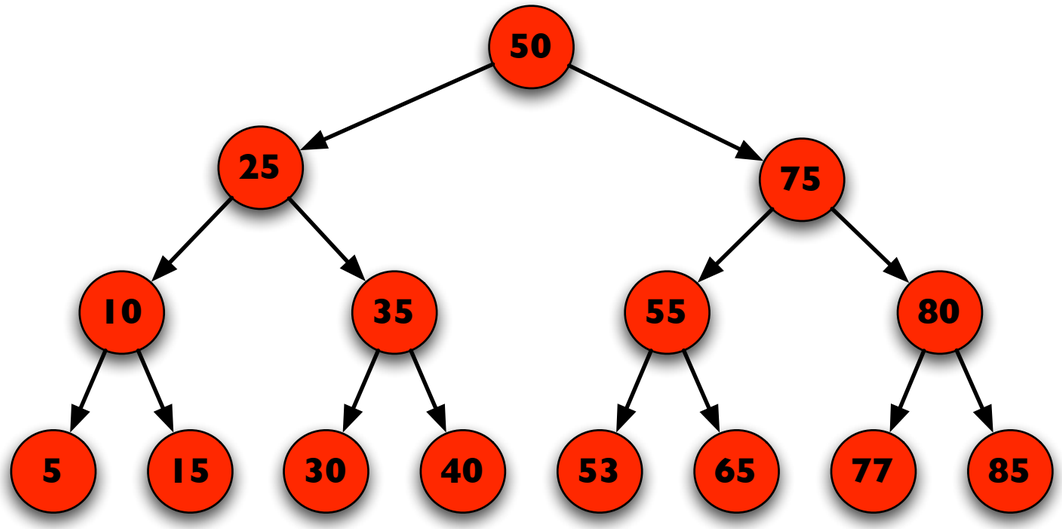
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

- Visit root node
- Traverse TL



Visit root node

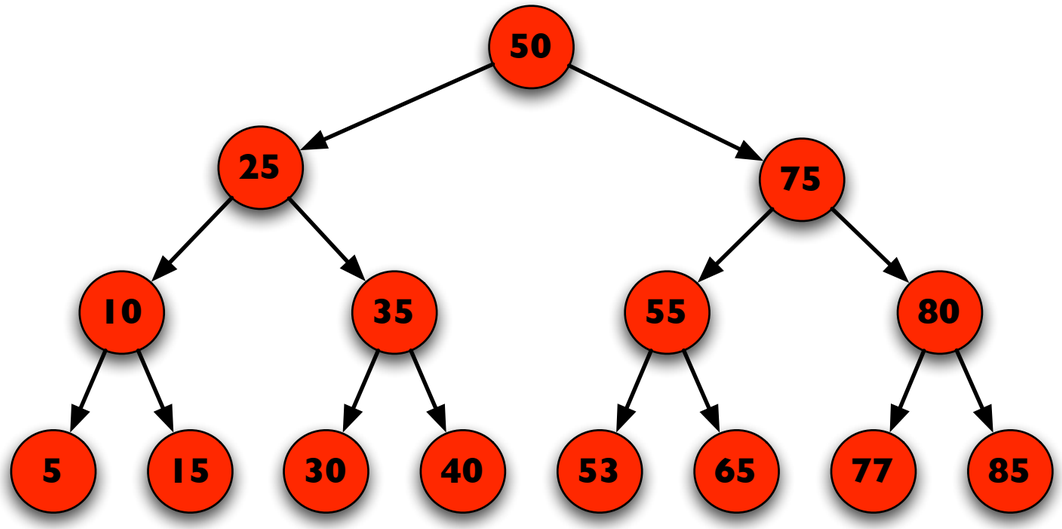
Traverse TL

Traverse TR

Preorder

50 25 10 5 15 35 30 40 75 55 53 65 80 77 85

- Visit root node
- Traverse TL
- Traverse TR



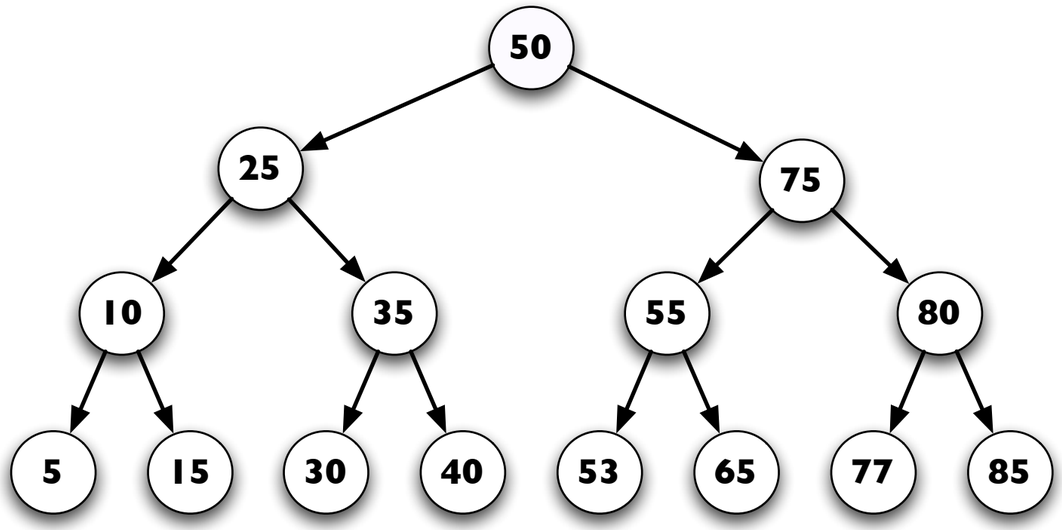
Visit root node

Traverse TL

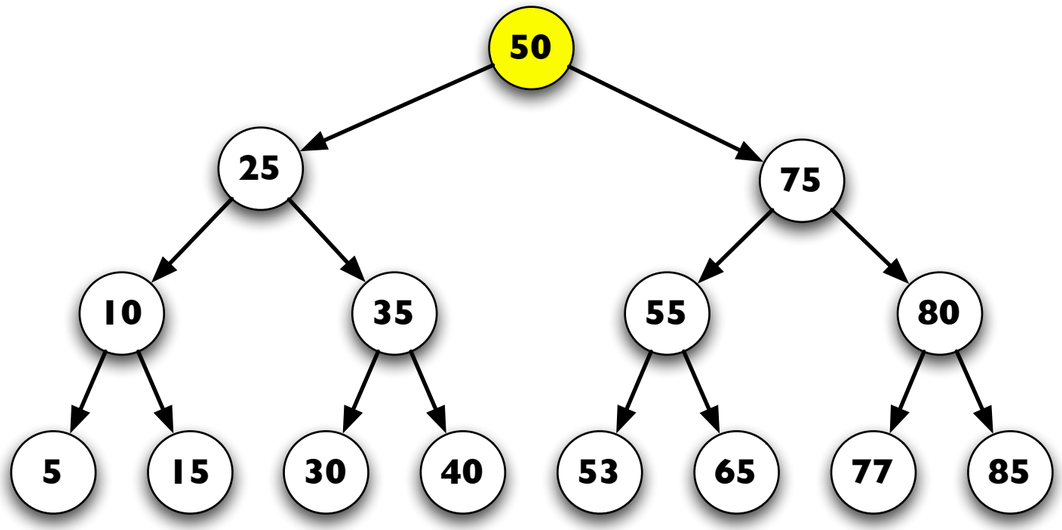
Traverse TR

Inorder Traversal

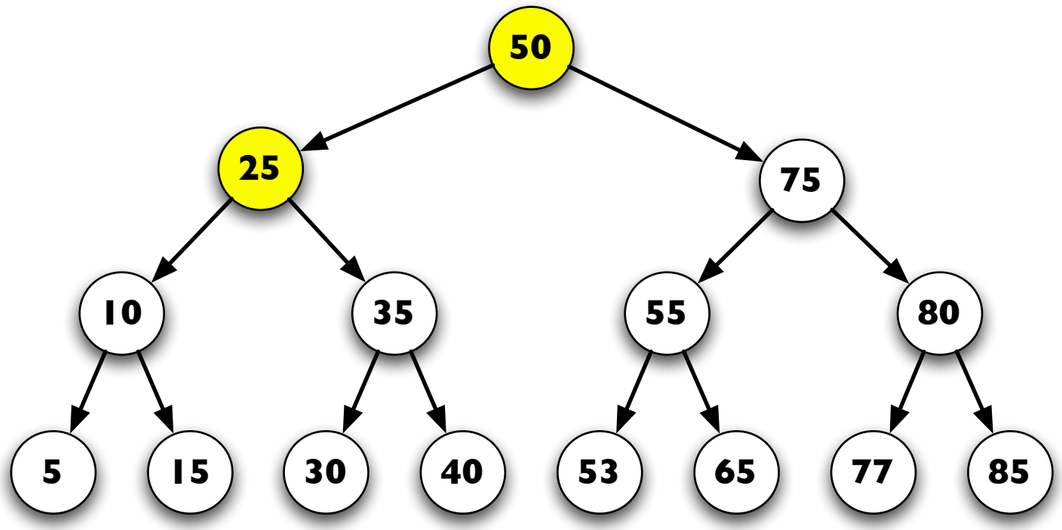
Inorder Traversal



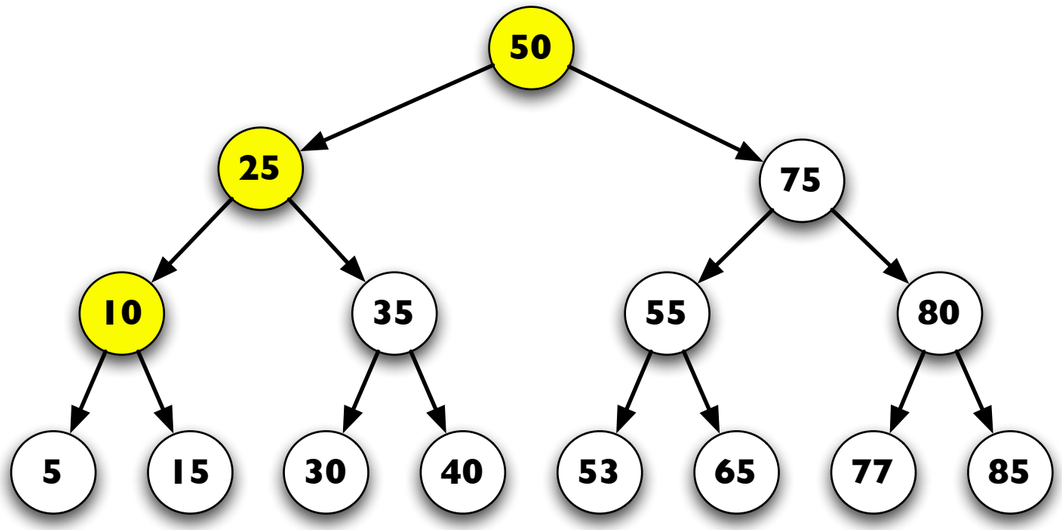
Inorder Traversal



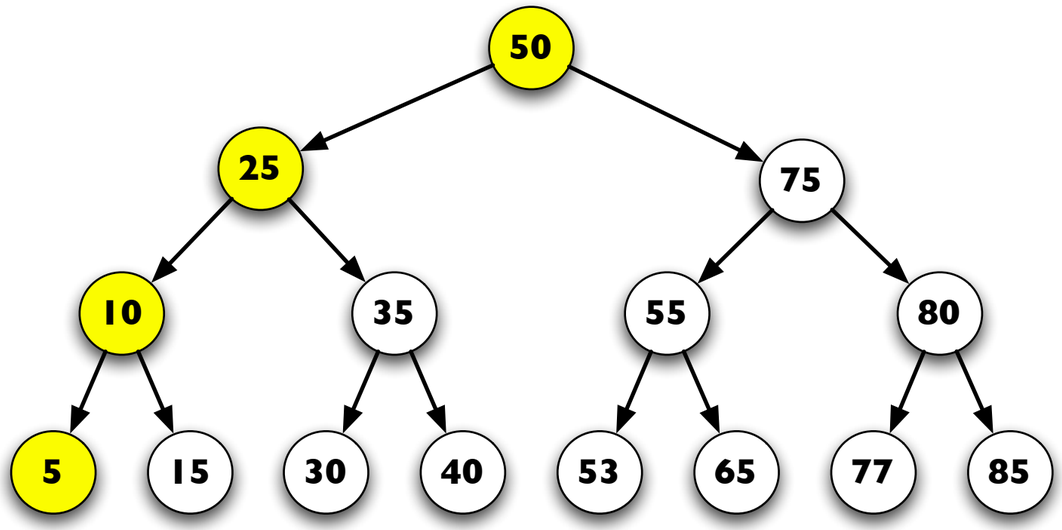
Inorder Traversal



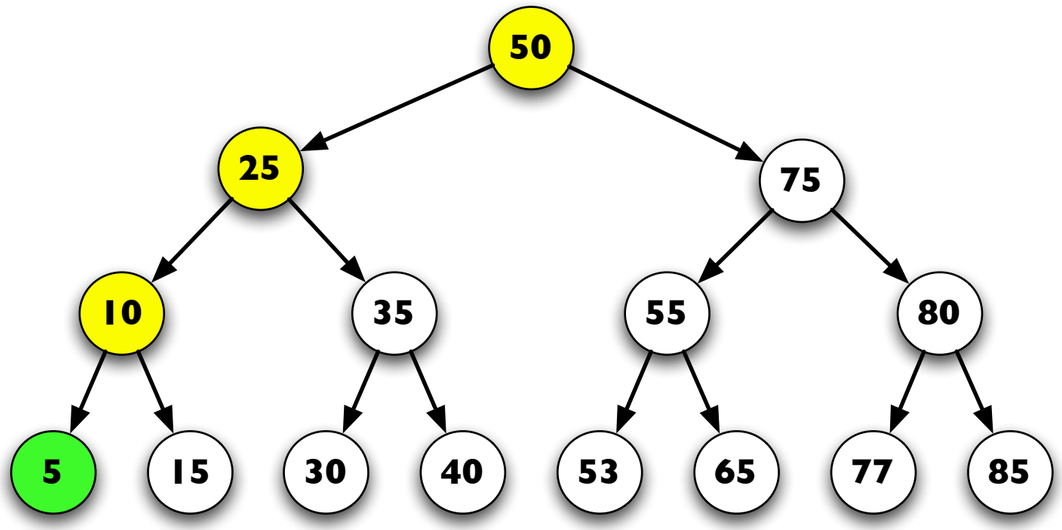
Inorder Traversal



Inorder Traversal

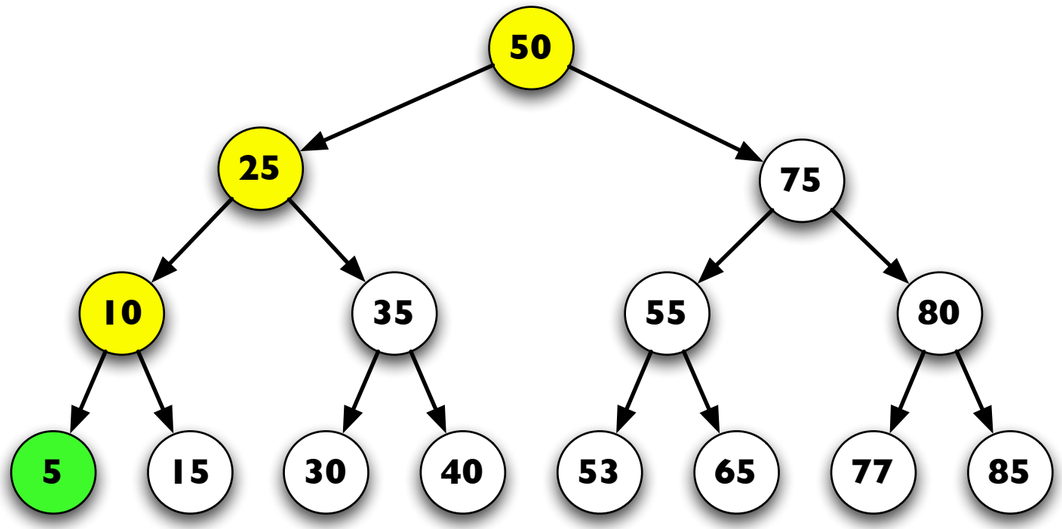


Inorder Traversal



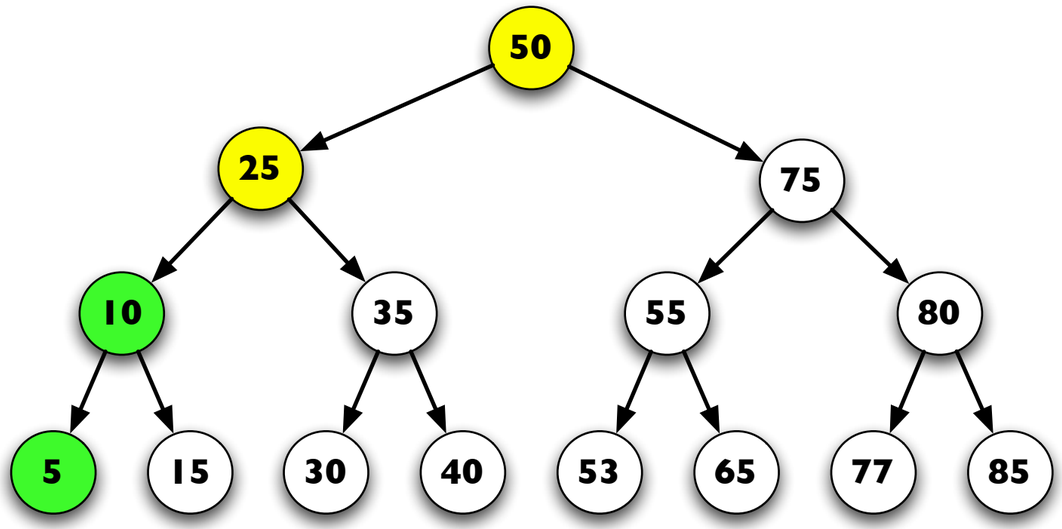
Inorder Traversal

5



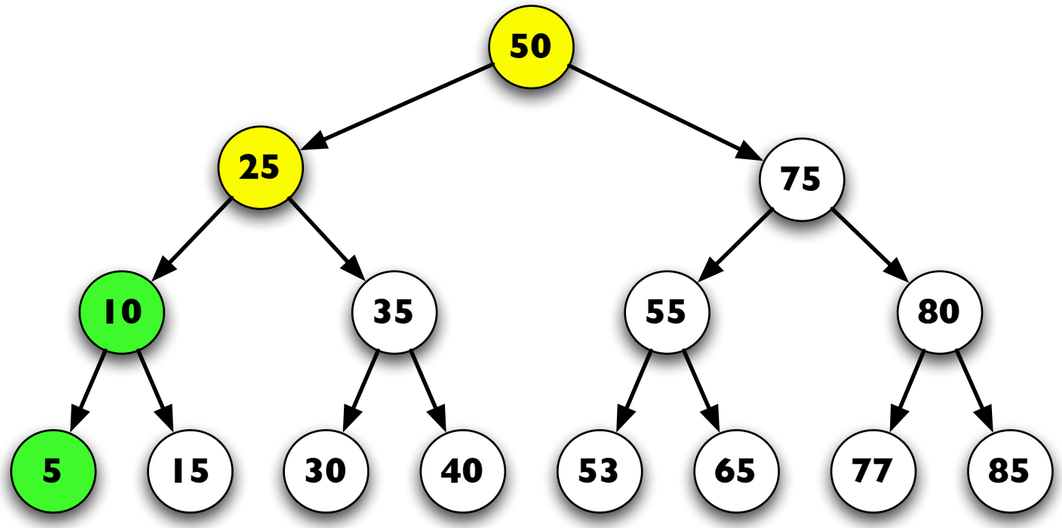
Inorder Traversal

5



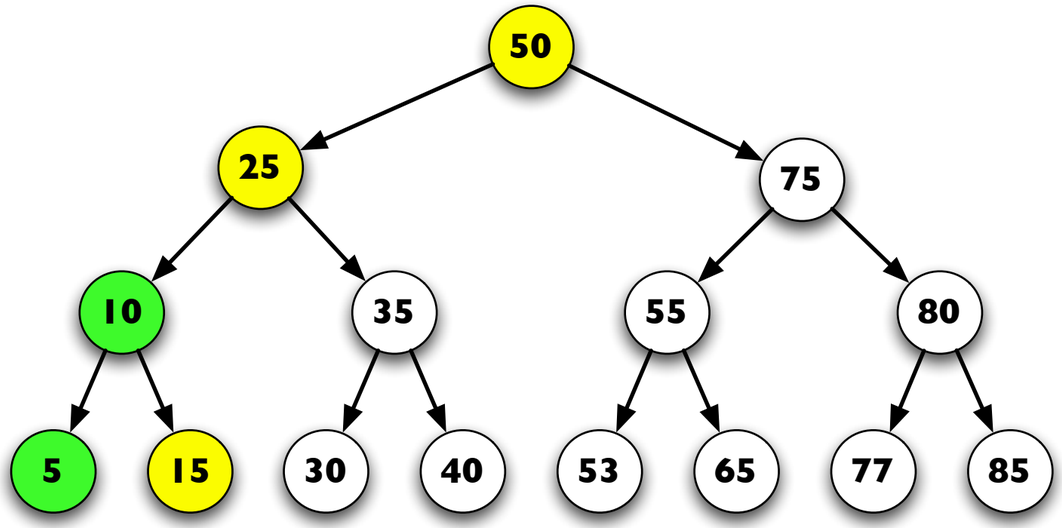
Inorder Traversal

5 10



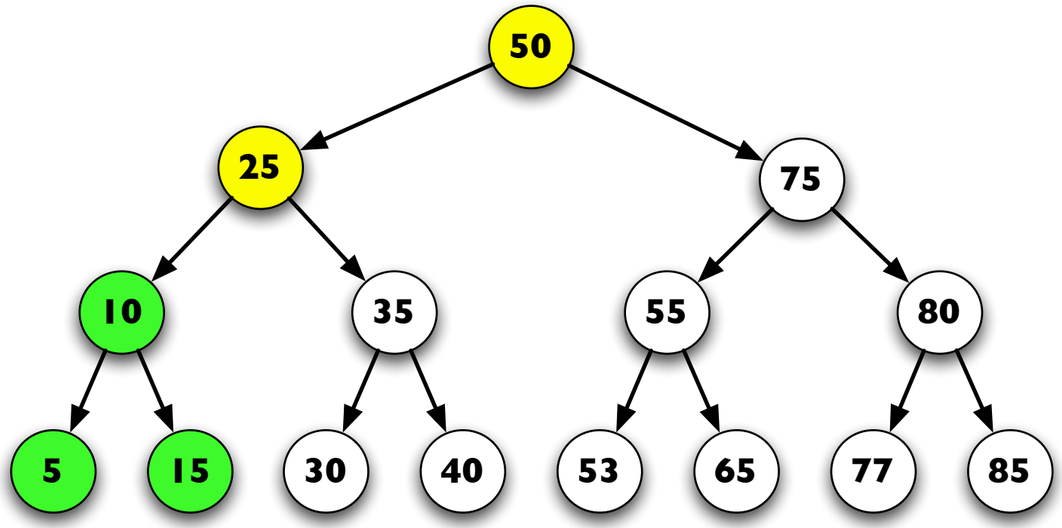
Inorder Traversal

5 10



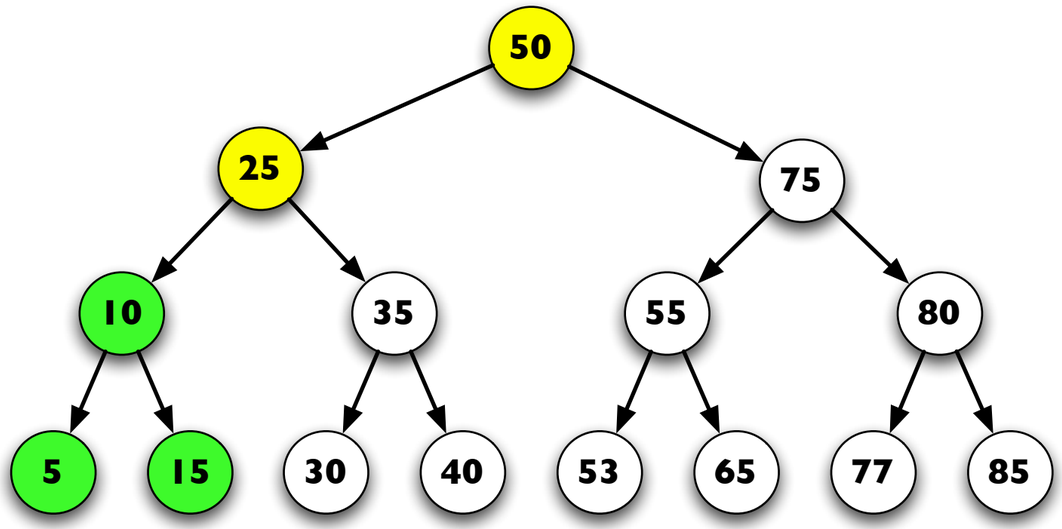
Inorder Traversal

5 10



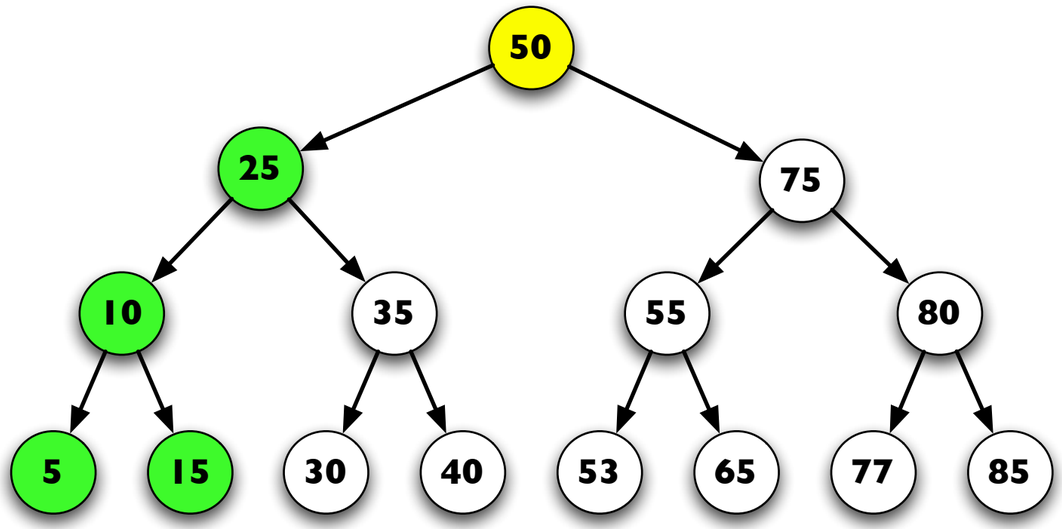
Inorder Traversal

5 10 15



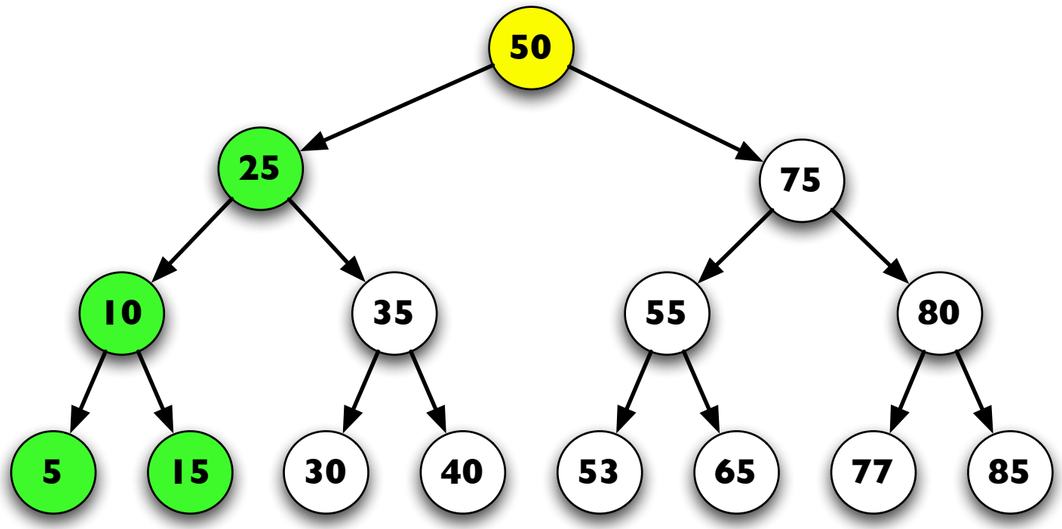
Inorder Traversal

5 10 15



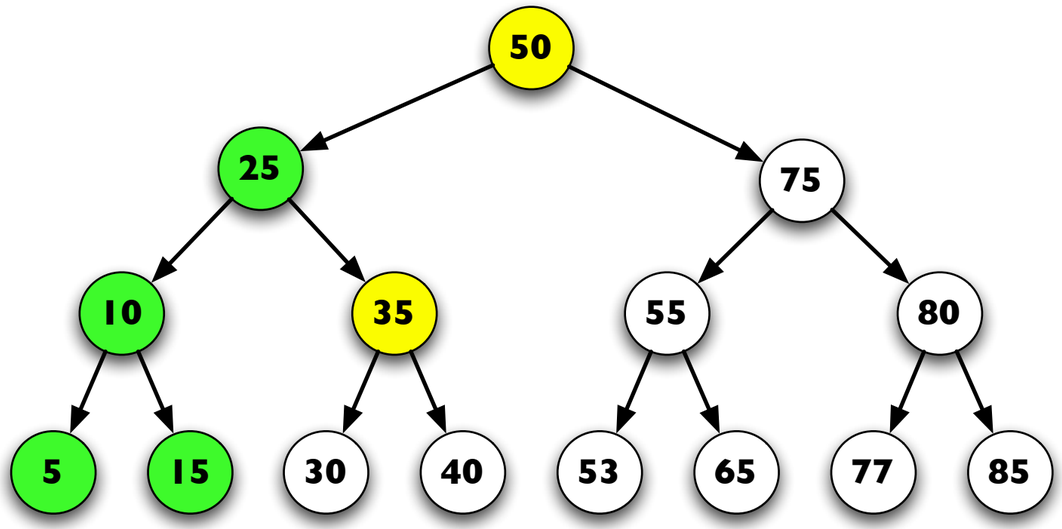
Inorder Traversal

5 10 15 25



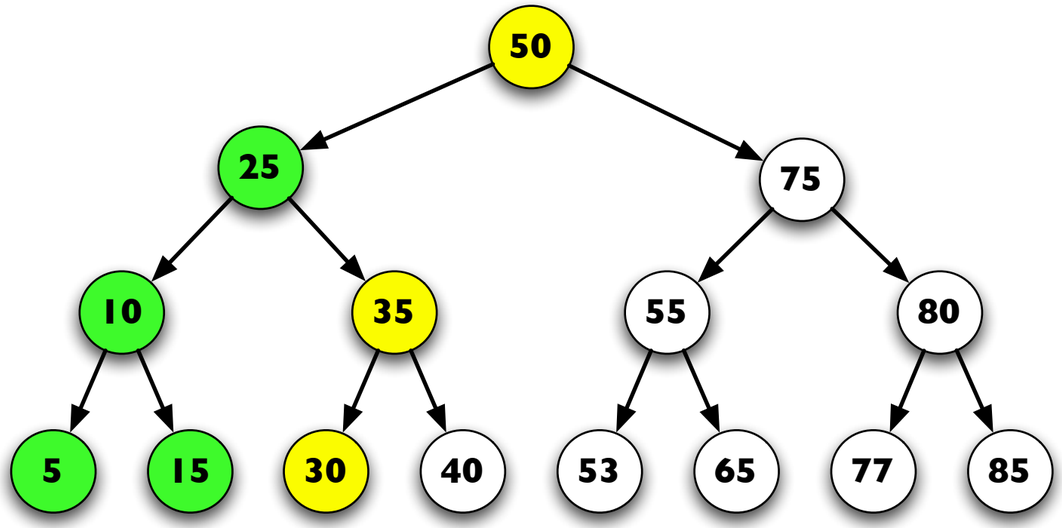
Inorder Traversal

5 10 15 25



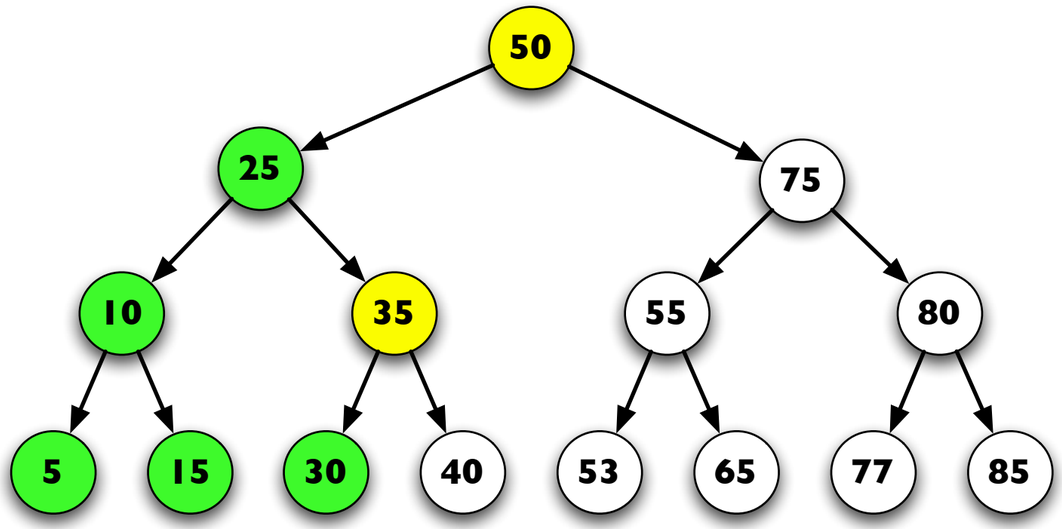
Inorder Traversal

5 10 15 25



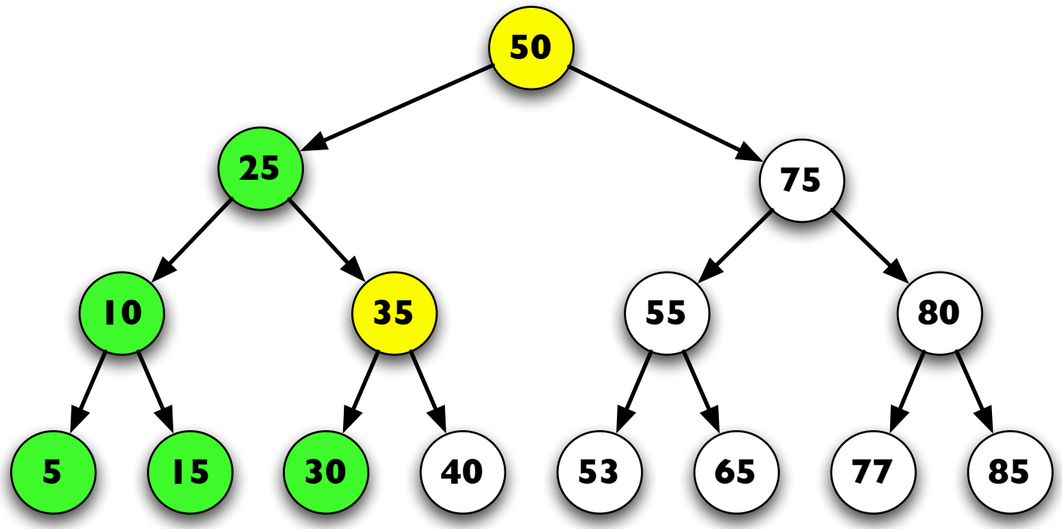
Inorder Traversal

5 10 15 25

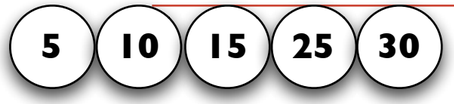


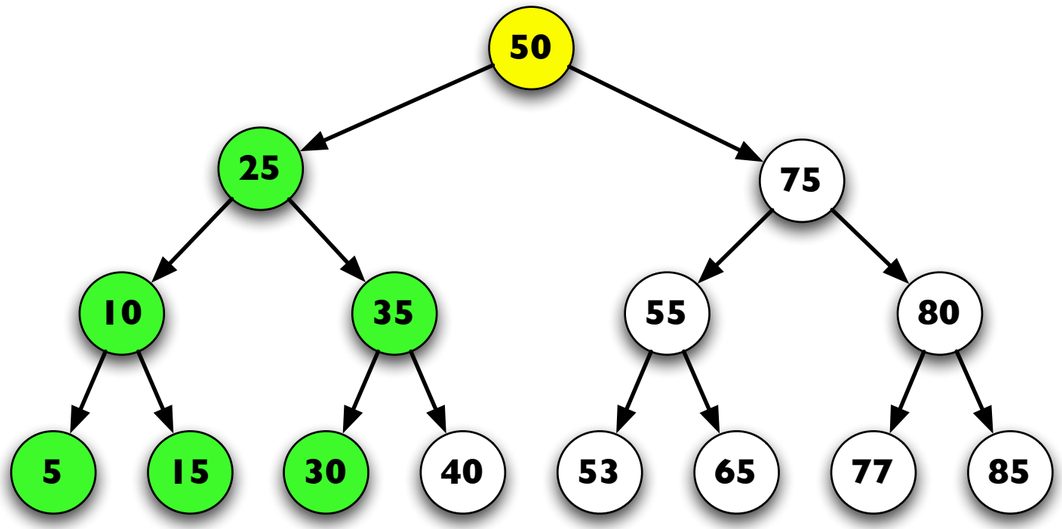
Inorder Traversal

5 10 15 25 30



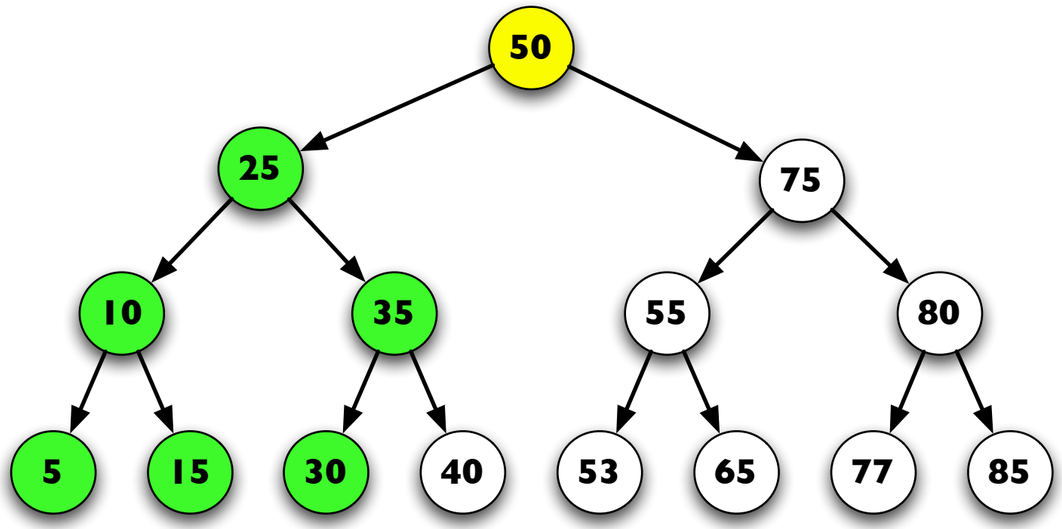
Inorder Traversal





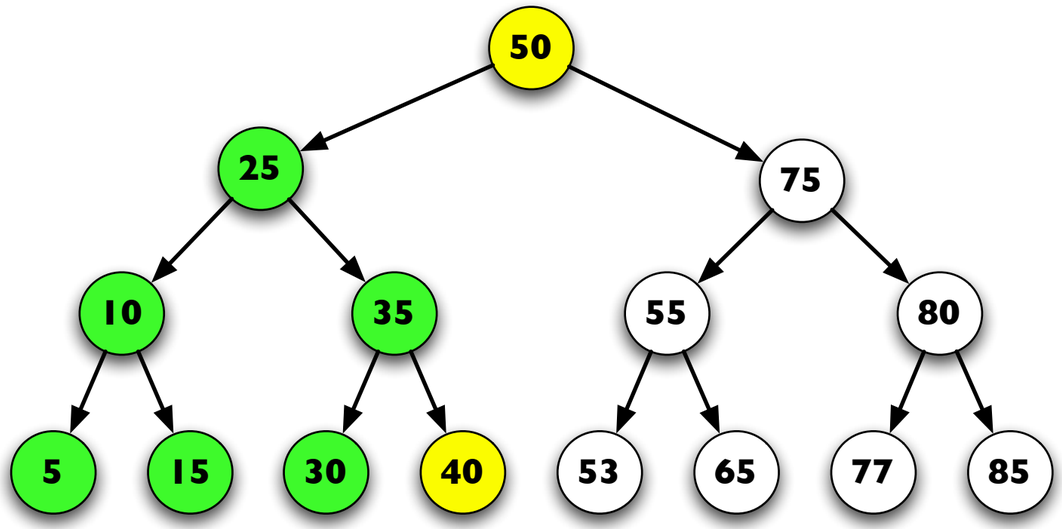
Inorder Traversal

5 10 15 25 30 35



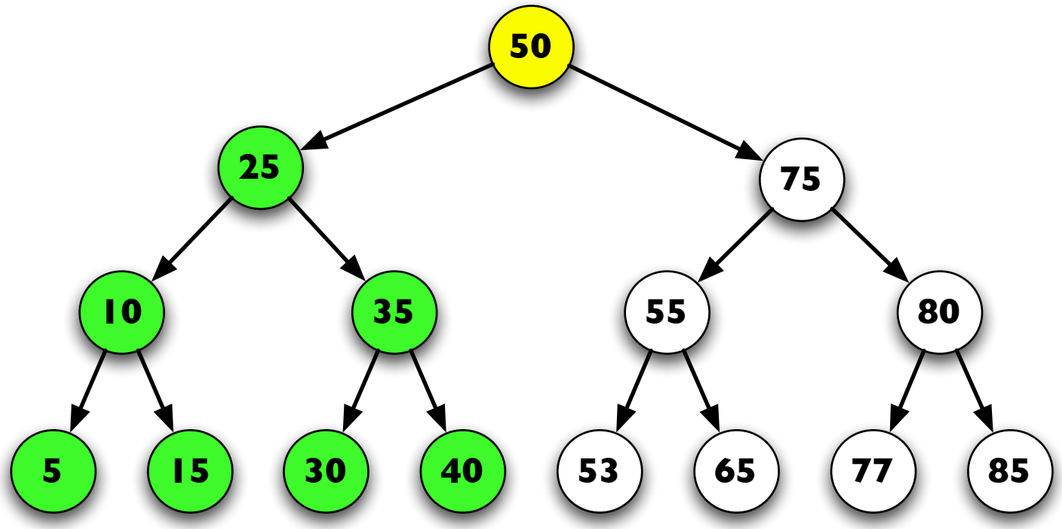
Inorder Traversal

5 10 15 25 30 35

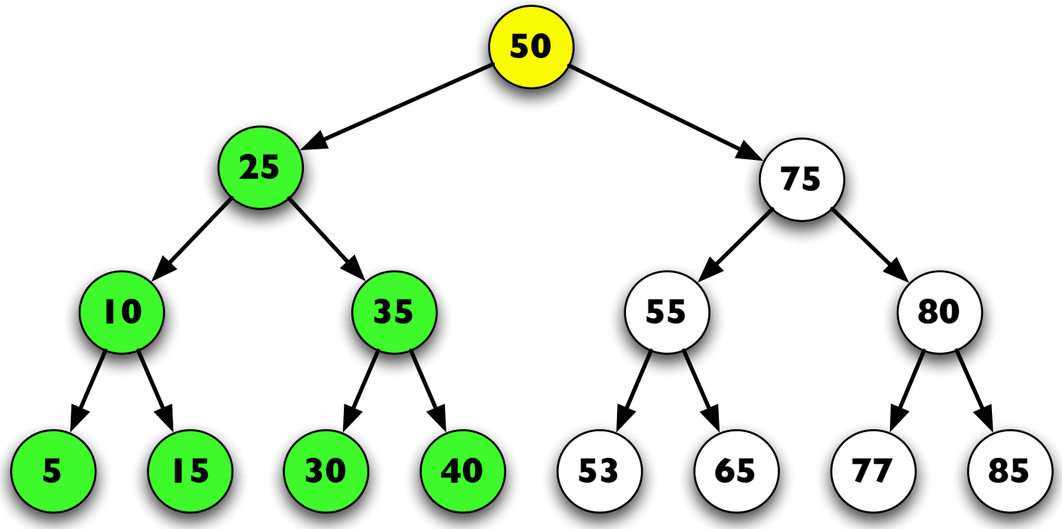


Inorder Traversal

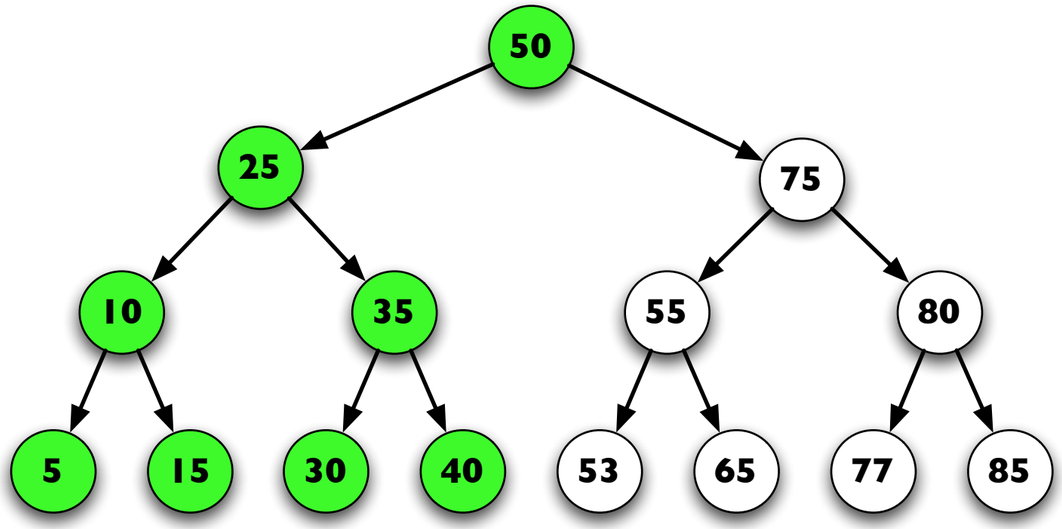
5 10 15 25 30 35



Inorder Traversal

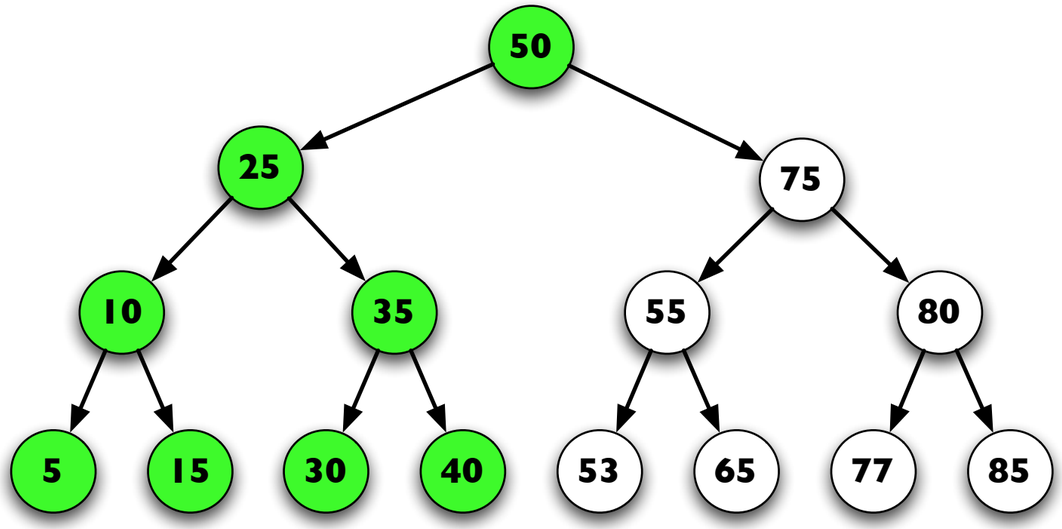


Inorder Traversal



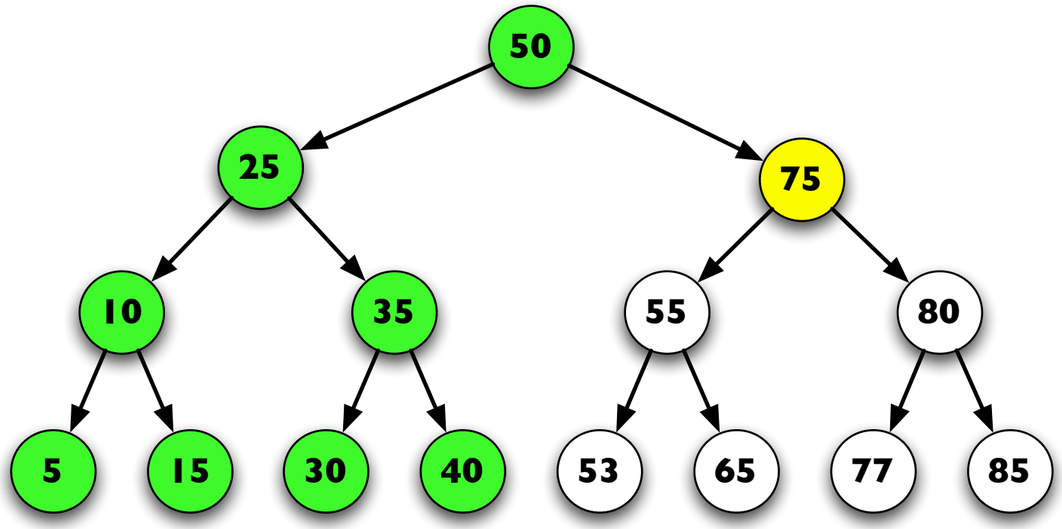
Inorder Traversal

5 10 15 25 30 35 40 50



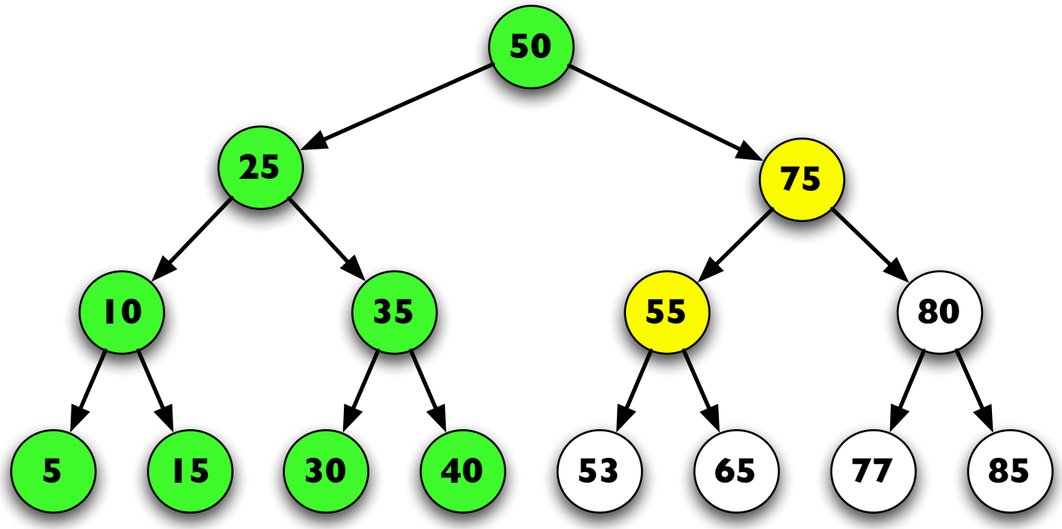
Inorder Traversal

5 10 15 25 30 35 40 50



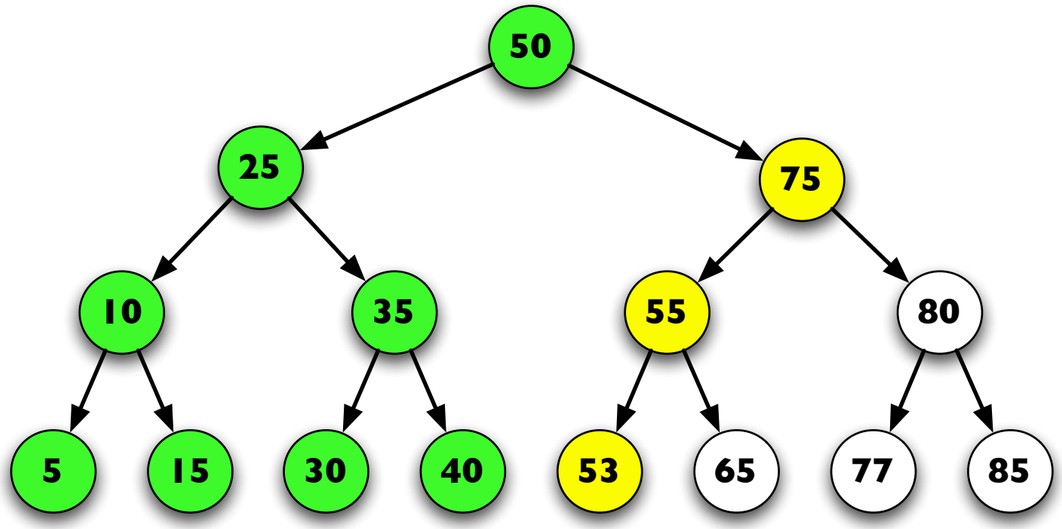
Inorder Traversal

5 10 15 25 30 35 40 50



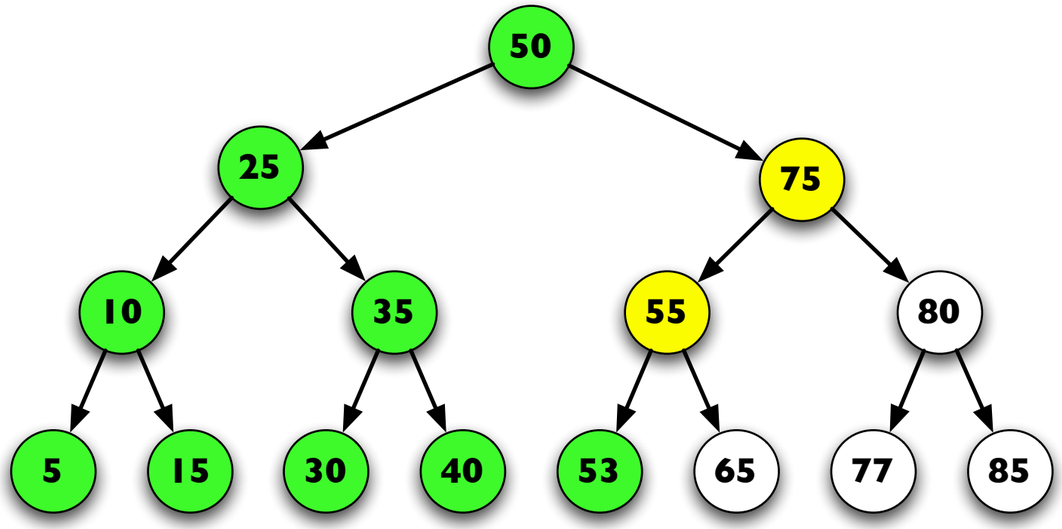
Inorder Traversal

5 10 15 25 30 35 40 50



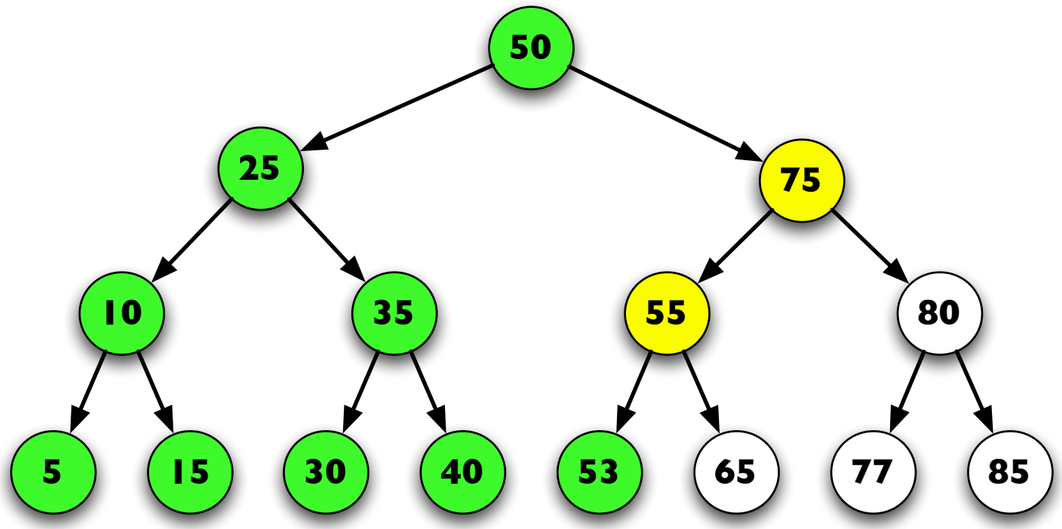
Inorder Traversal

5 10 15 25 30 35 40 50



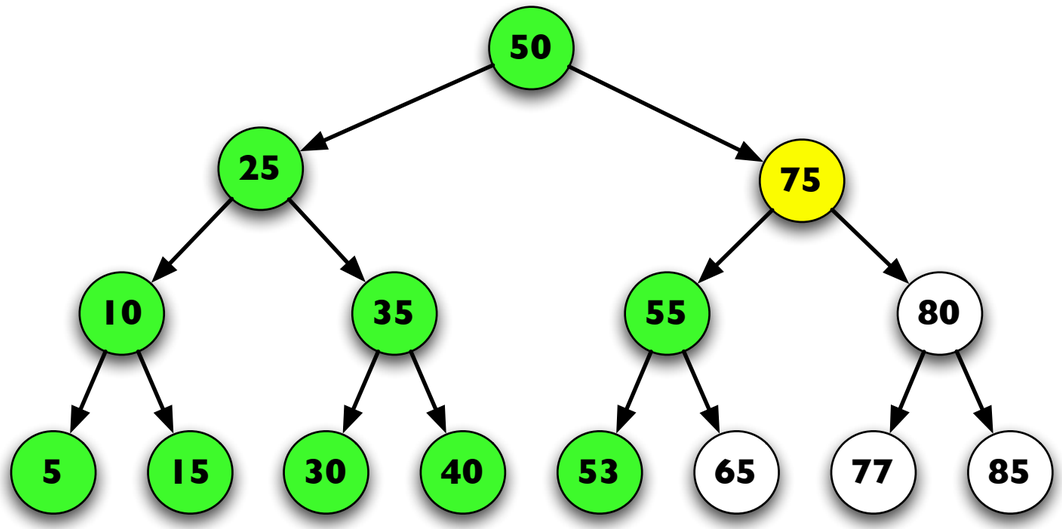
Inorder Traversal

5 10 15 25 30 35 40 50 53



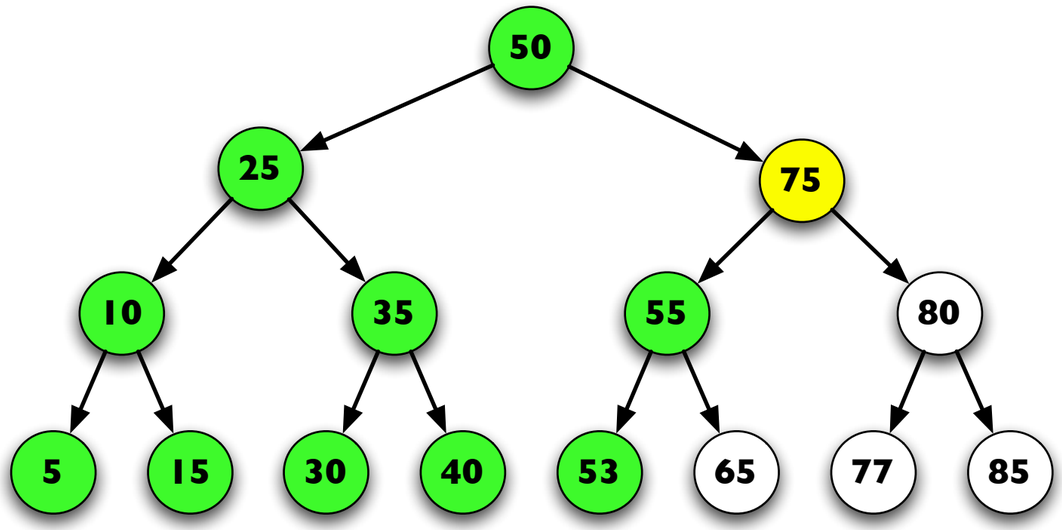
Inorder Traversal

5 10 15 25 30 35 40 50 53



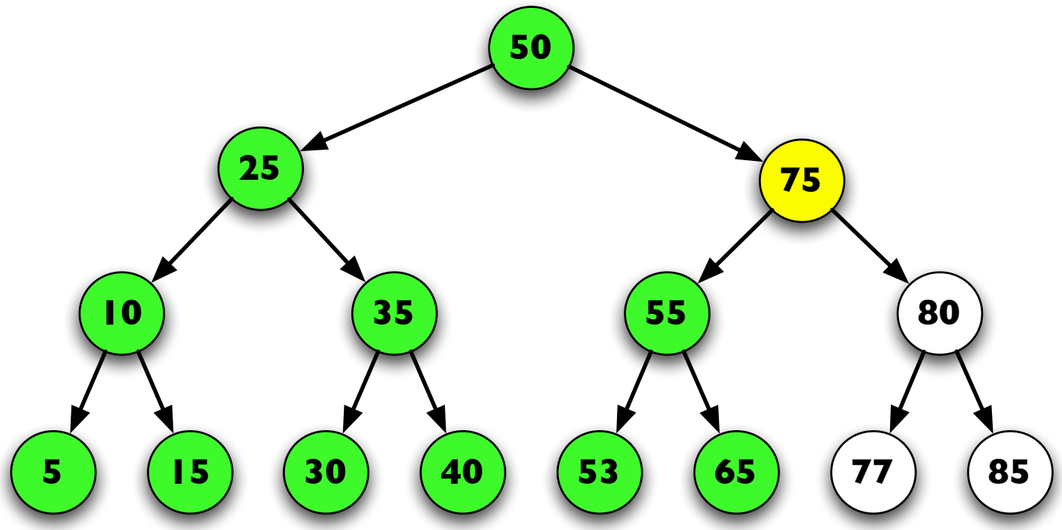
Inorder Traversal

5 10 15 25 30 35 40 50 53 55



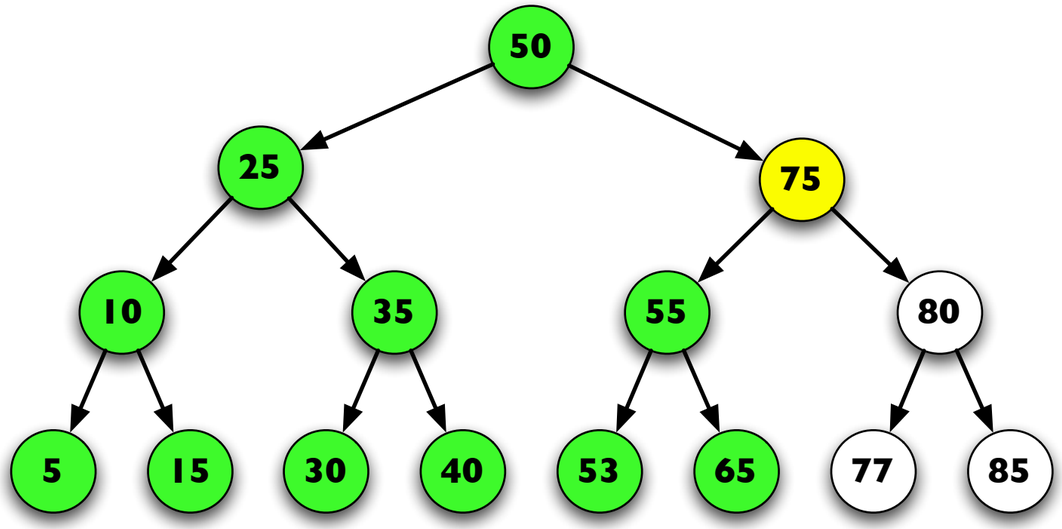
Inorder Traversal

5 10 15 25 30 35 40 50 53 55



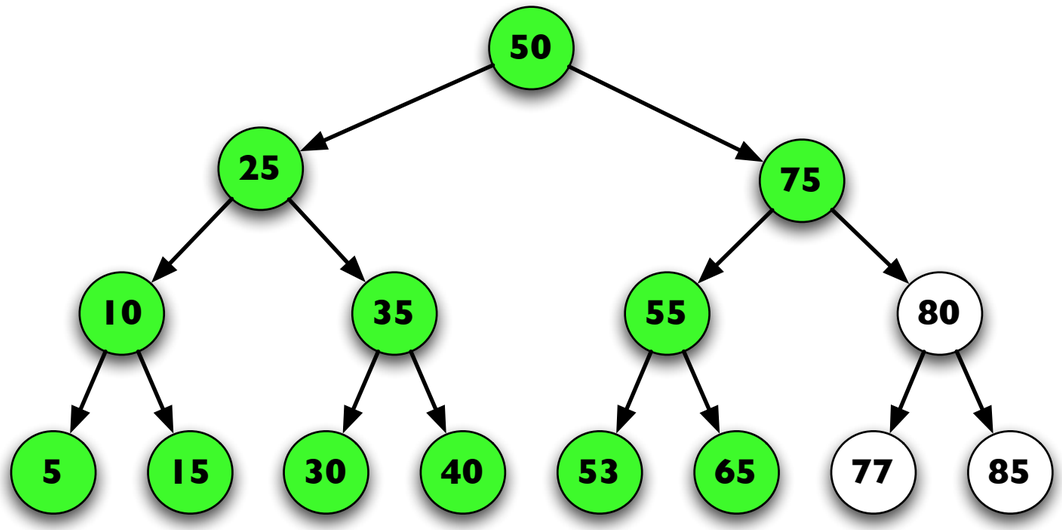
Inorder Traversal

5 10 15 25 30 35 40 50 53 55 65



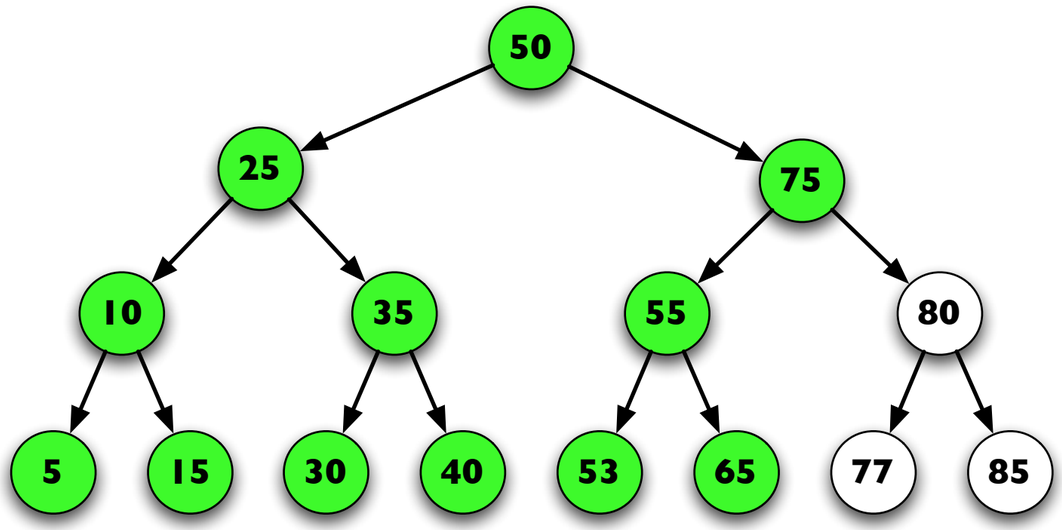
Inorder Traversal

5 10 15 25 30 35 40 50 53 55 65



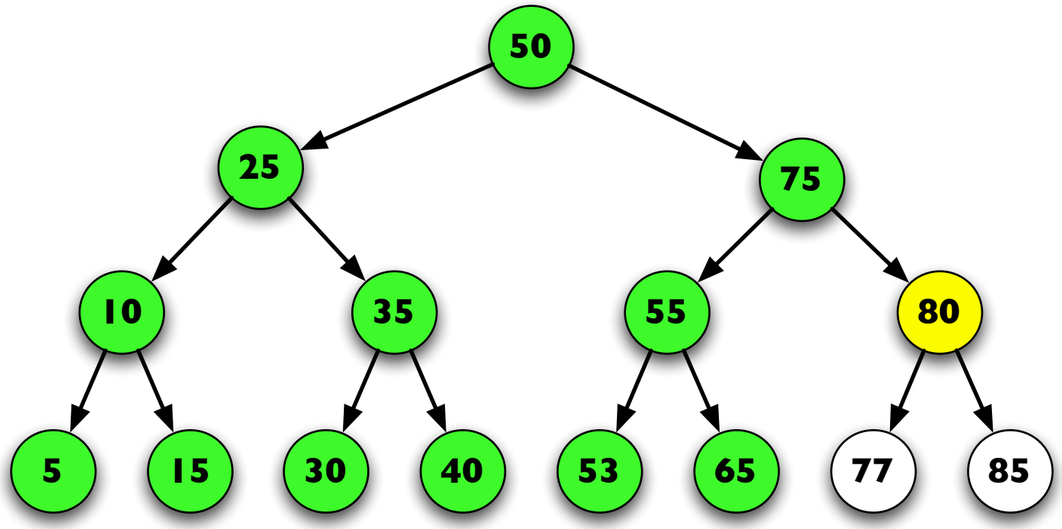
Inorder Traversal

5 10 15 25 30 35 40 50 53 55 65 75



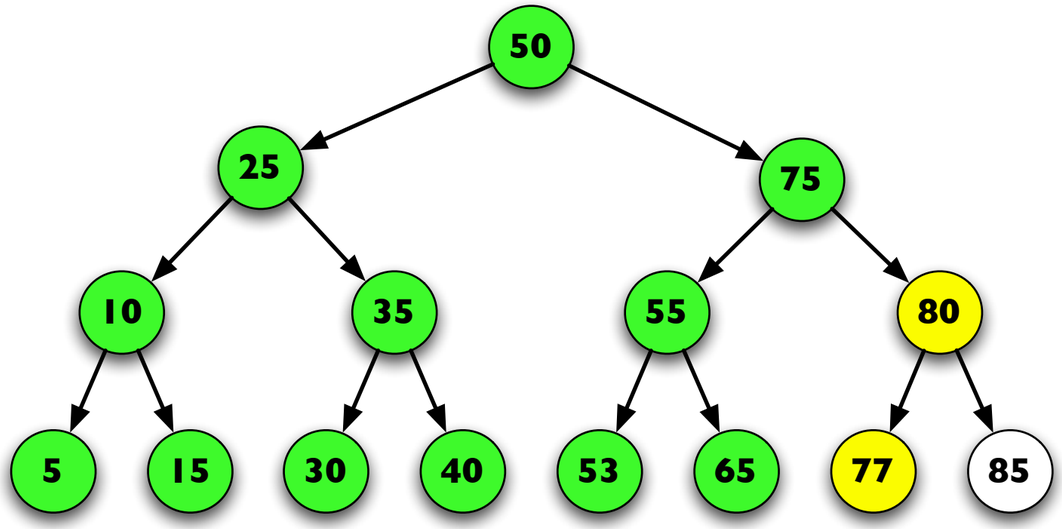
Inorder Traversal

5 10 15 25 30 35 40 50 53 55 65 75



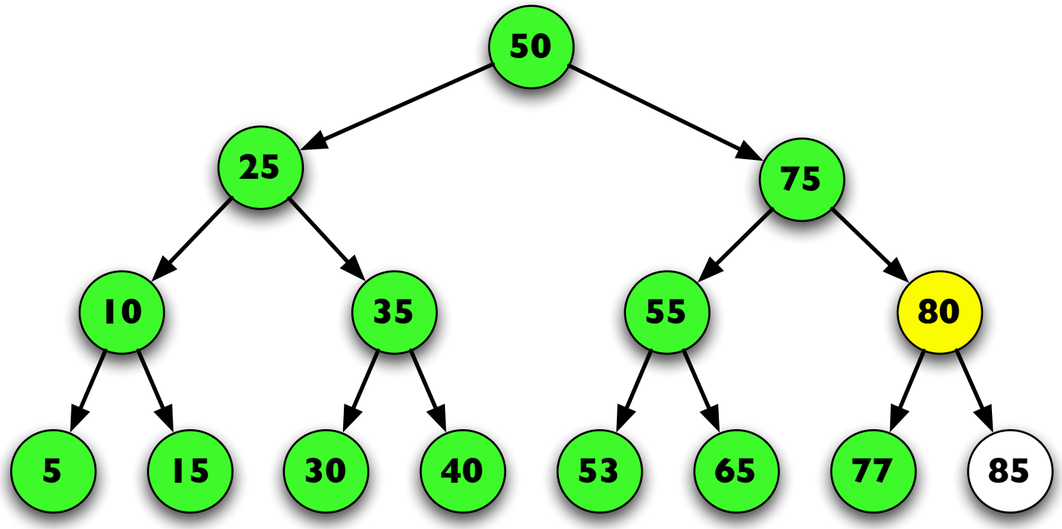
Inorder Traversal

5 10 15 25 30 35 40 50 53 55 65 75

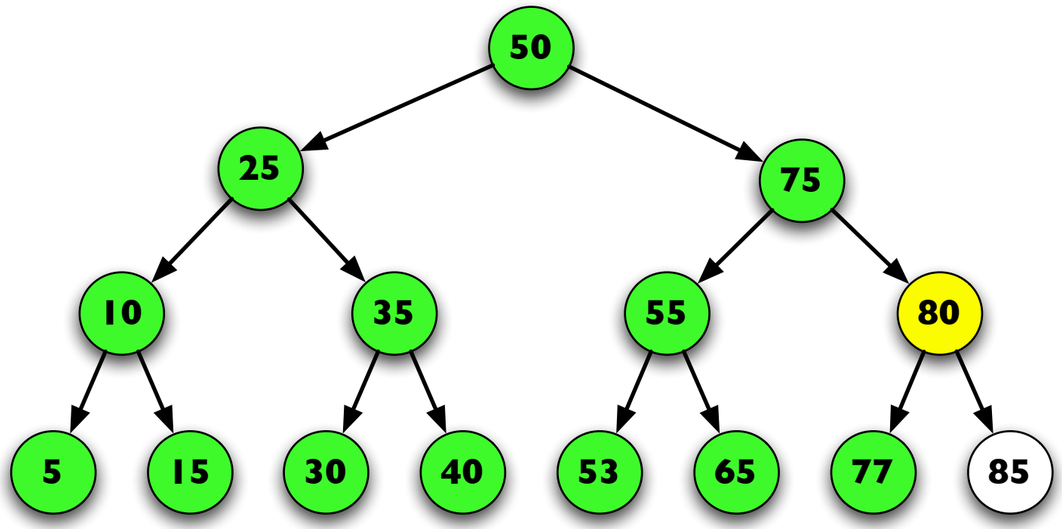


Inorder Traversal

5 10 15 25 30 35 40 50 53 55 65 75

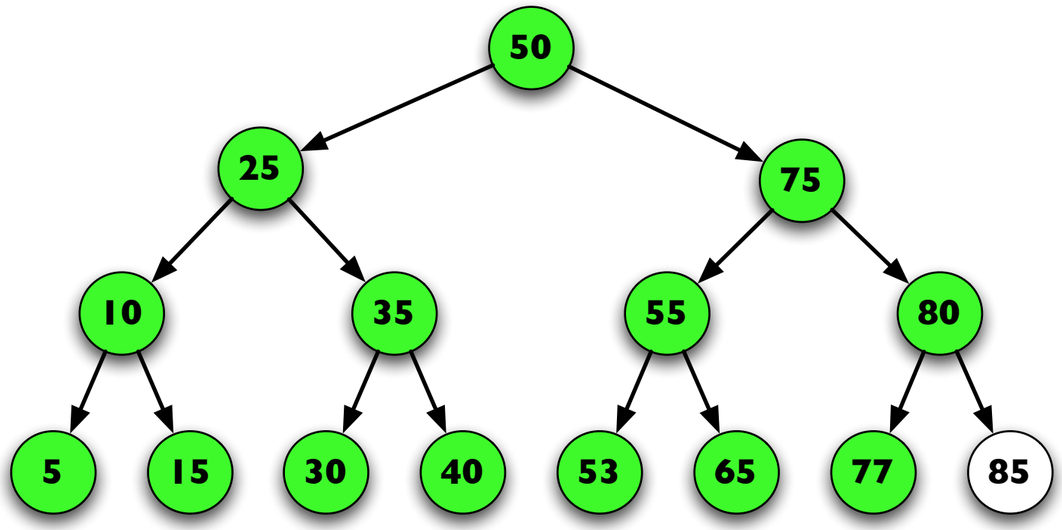


Inorder Traversal

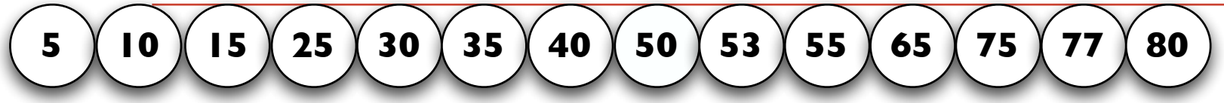


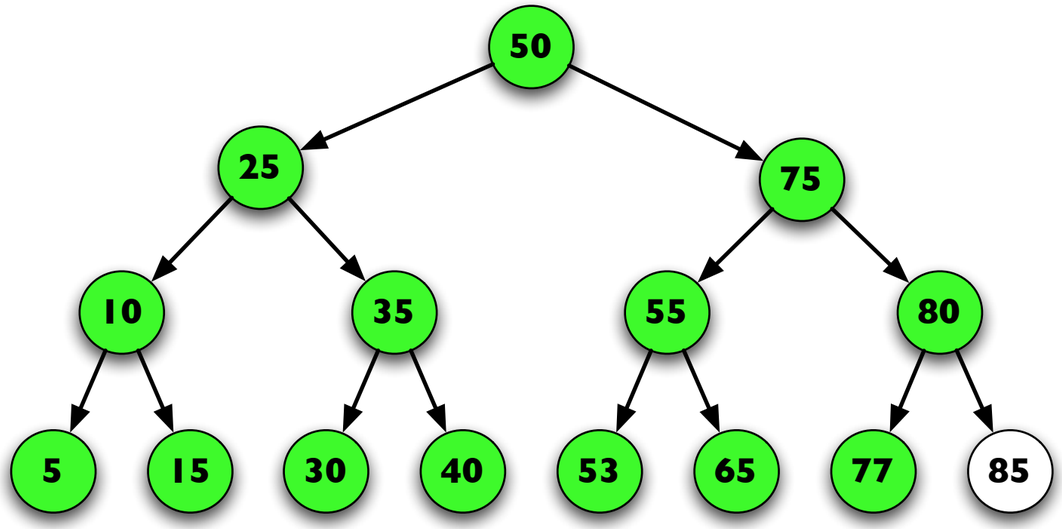
Inorder Traversal

5 10 15 25 30 35 40 50 53 55 65 75 77

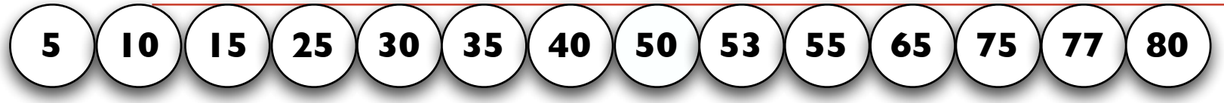


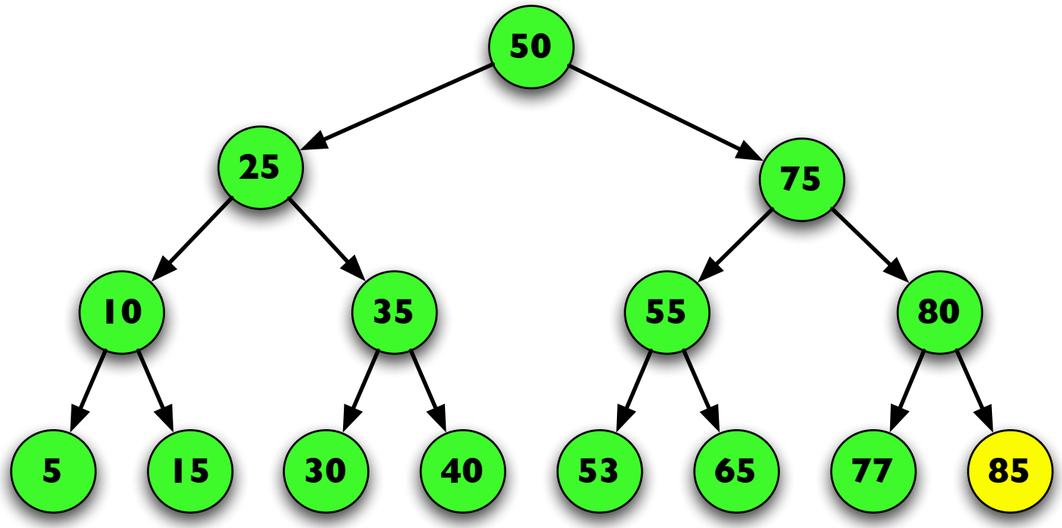
Inorder Traversal



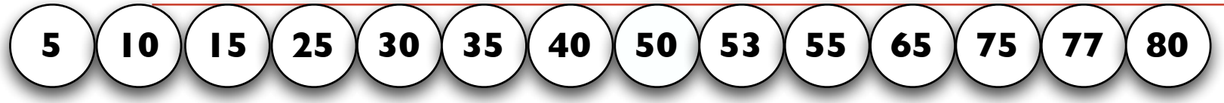


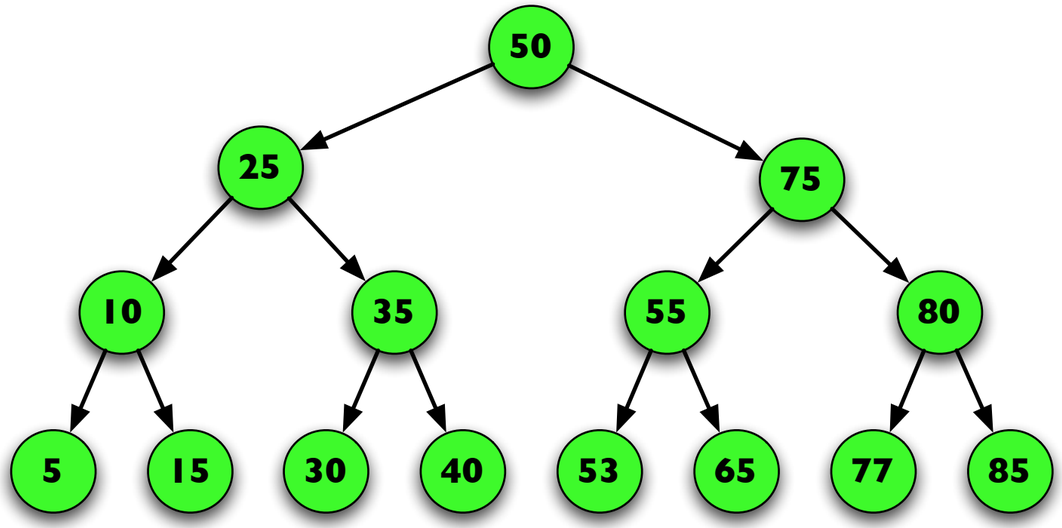
Inorder Traversal



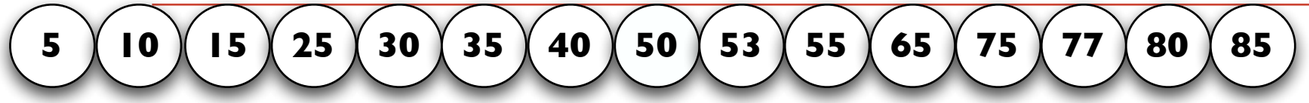


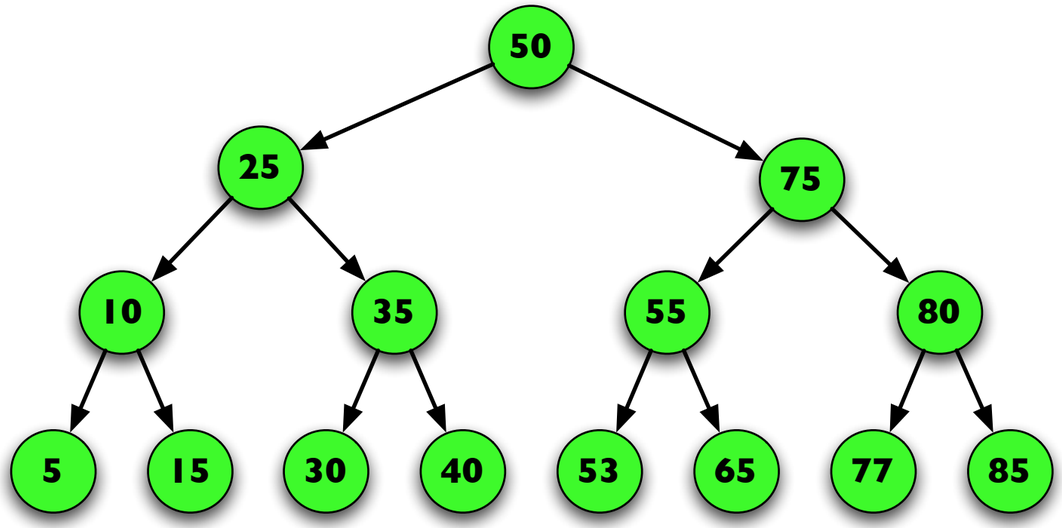
Inorder Traversal





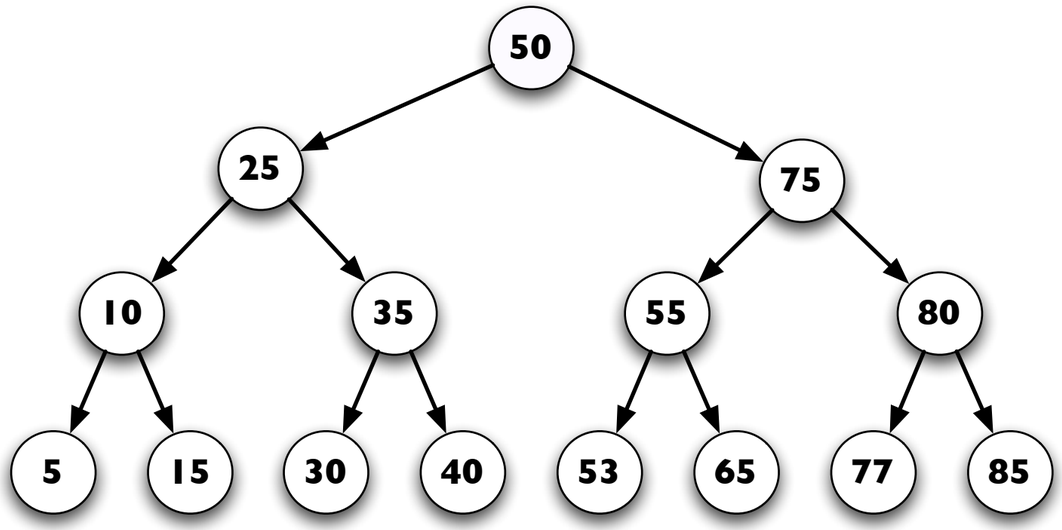
Inorder Traversal



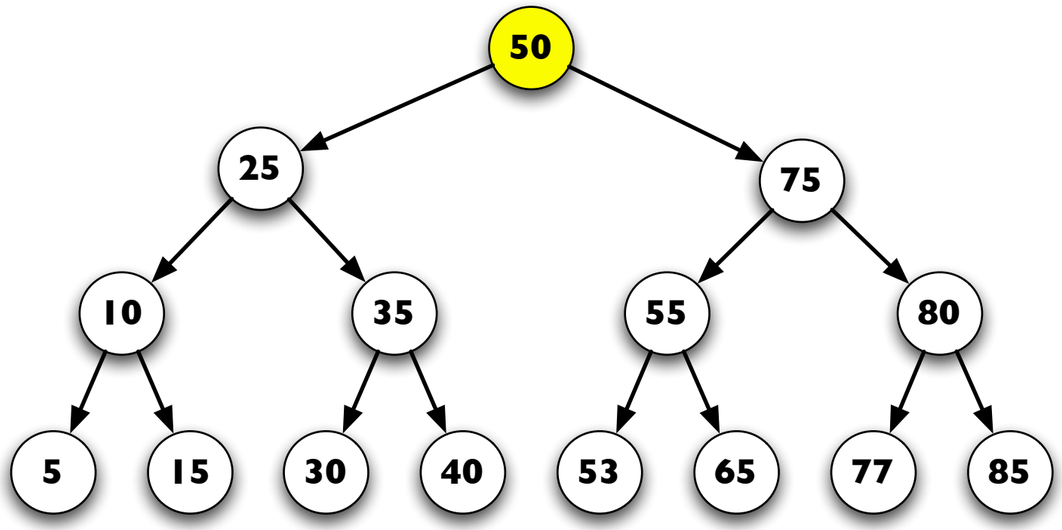


Postorder Traversal

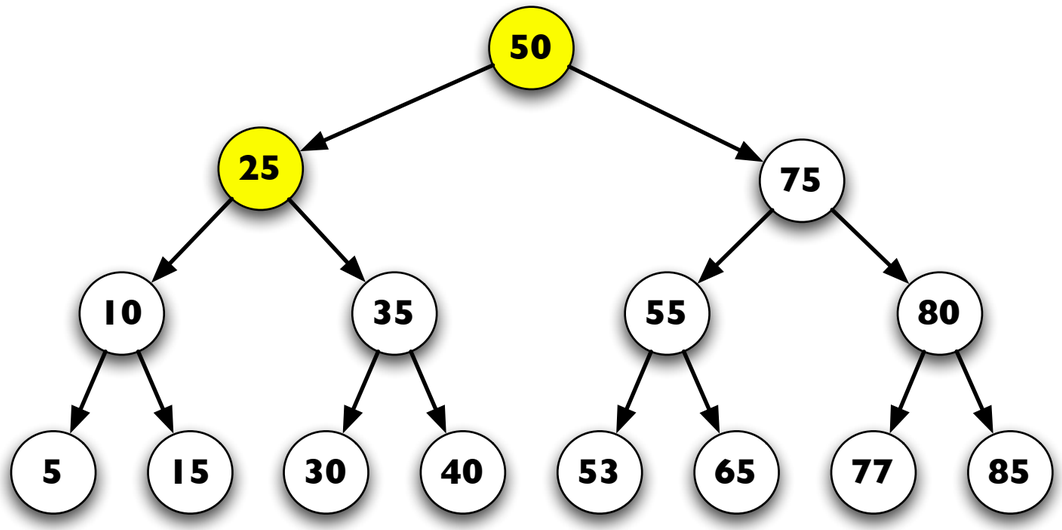
Postorder Traversal



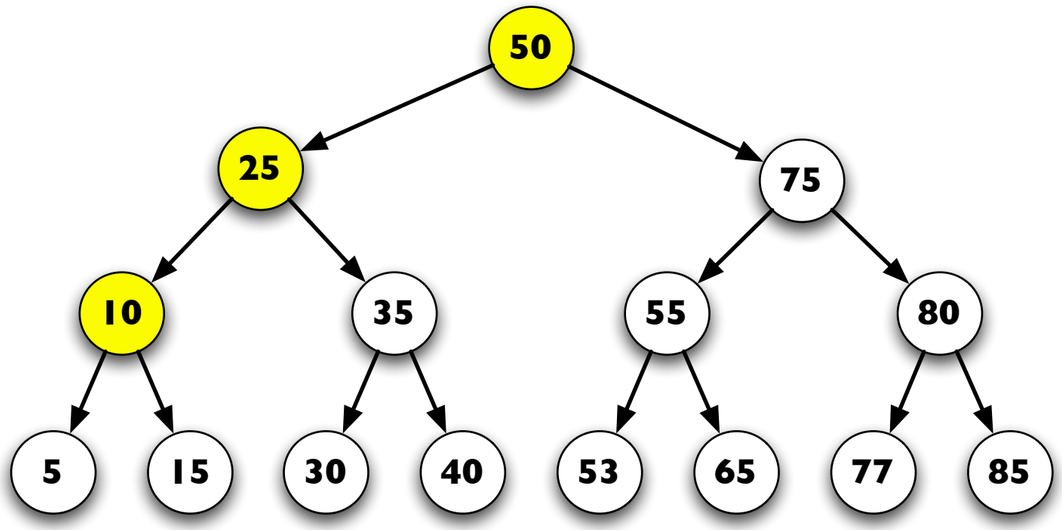
Postorder Traversal



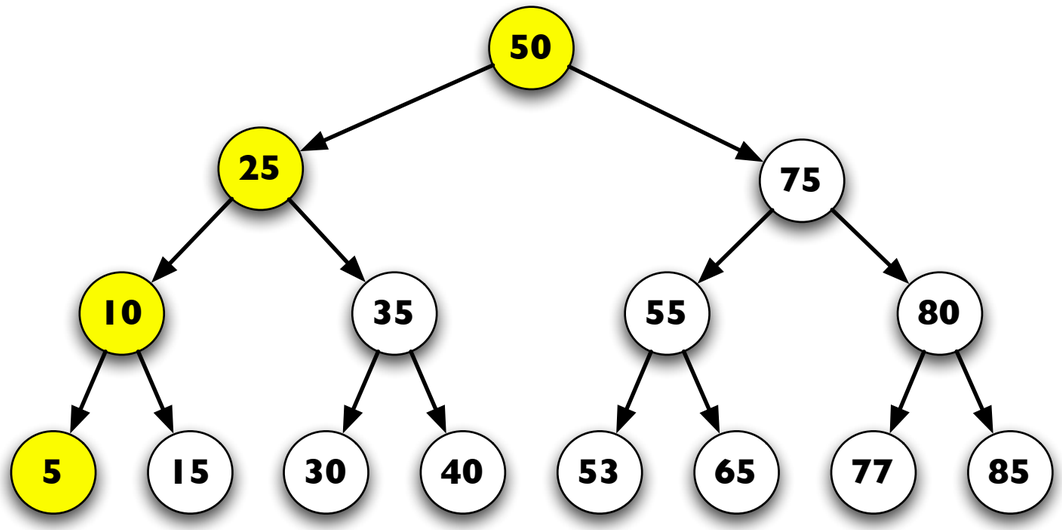
Postorder Traversal



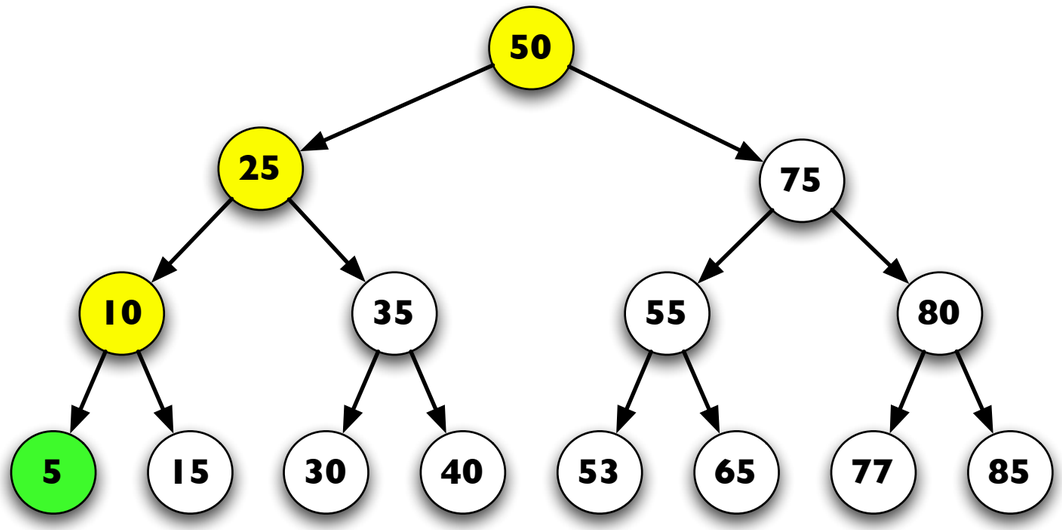
Postorder Traversal



Postorder Traversal

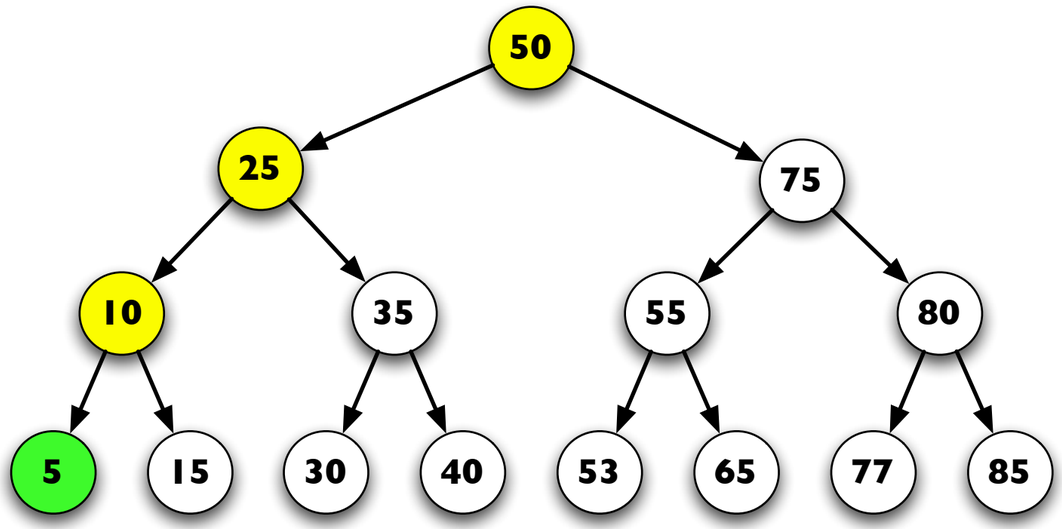


Postorder Traversal



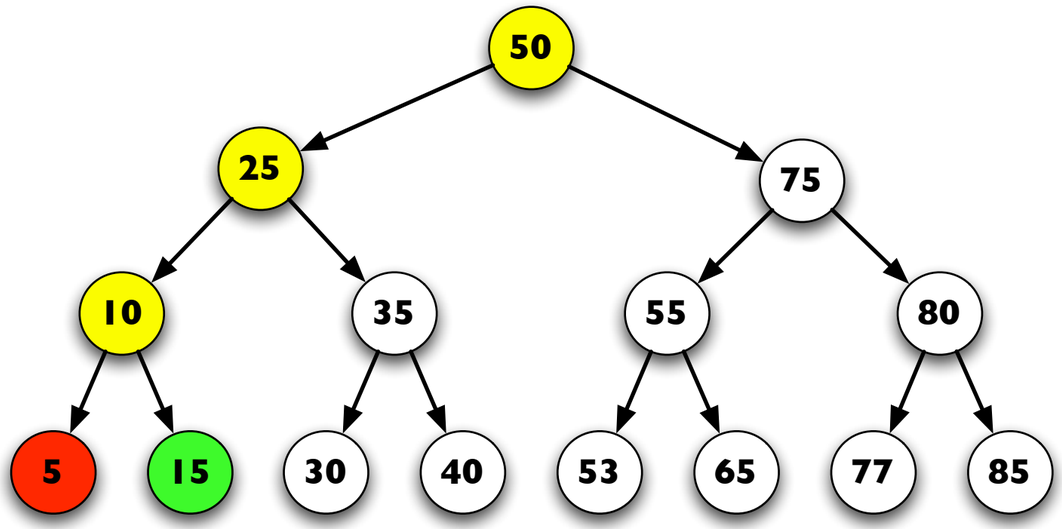
Postorder Traversal

5



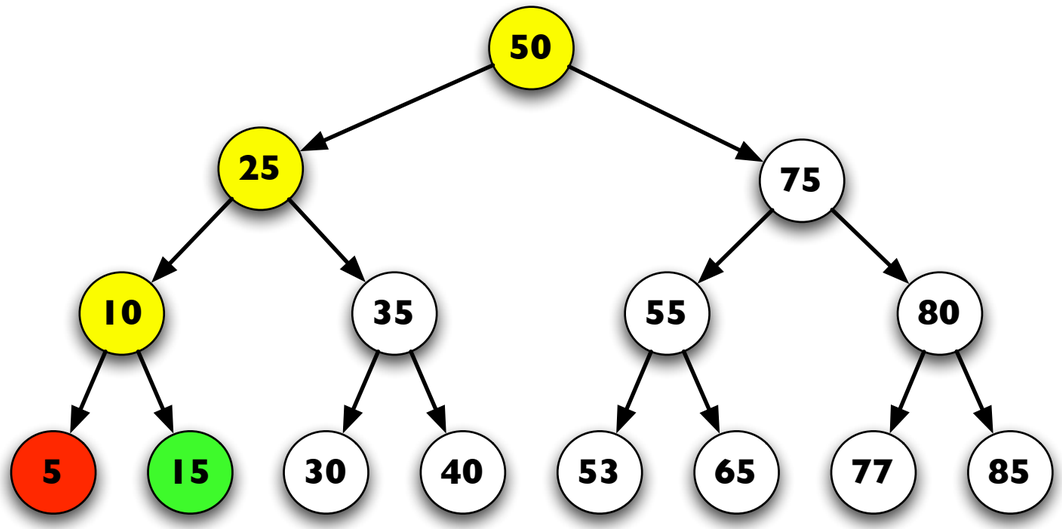
Postorder Traversal

5



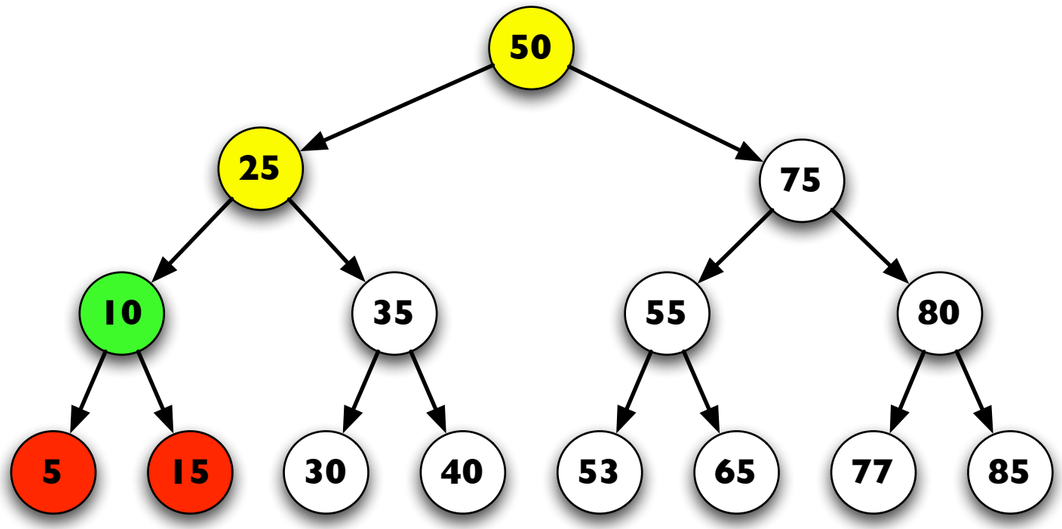
Postorder Traversal

5 15



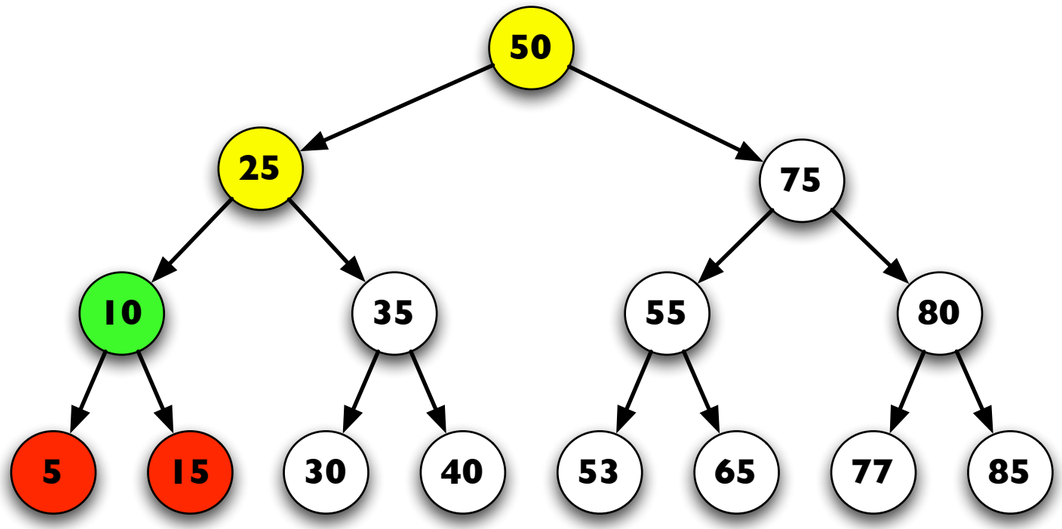
Postorder Traversal

5 15



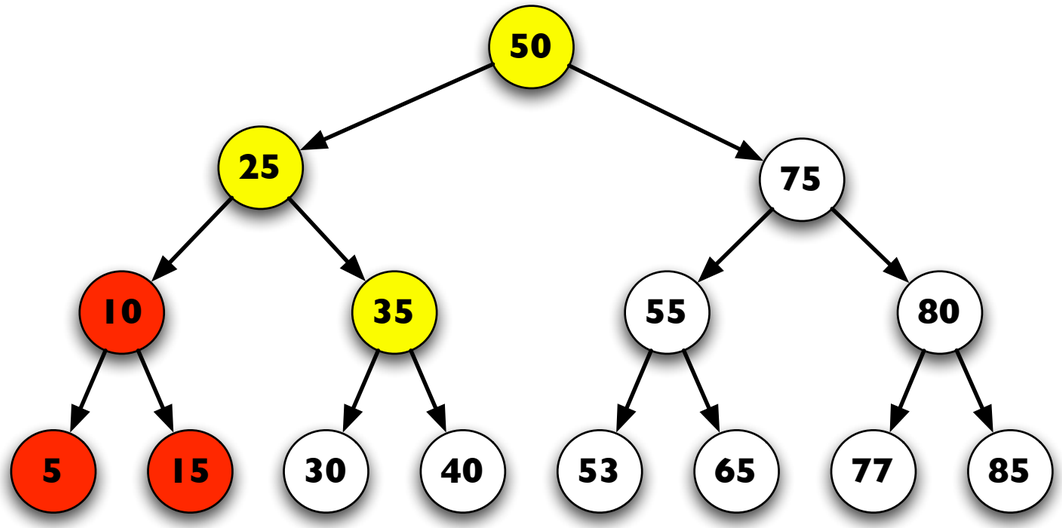
Postorder Traversal

5 15 10



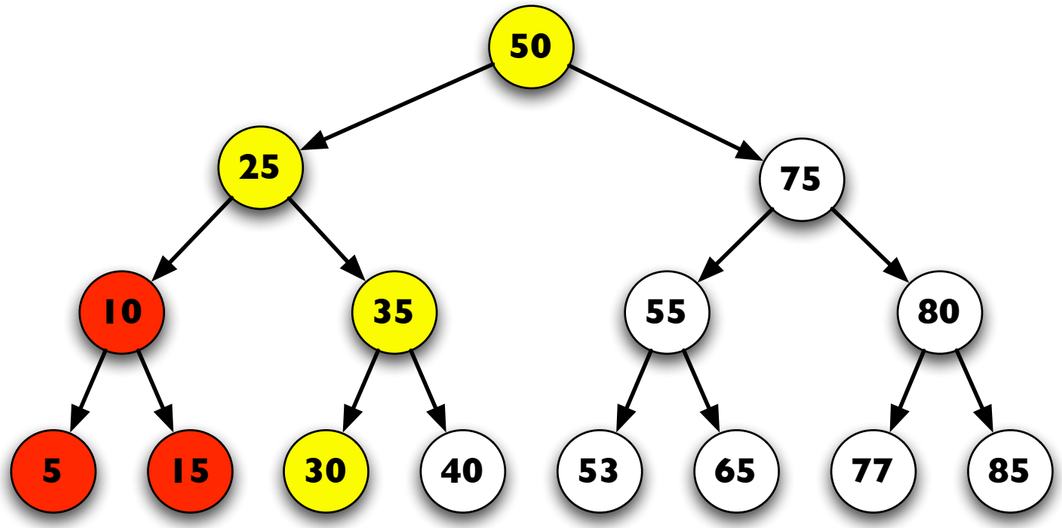
Postorder Traversal

5 15 10



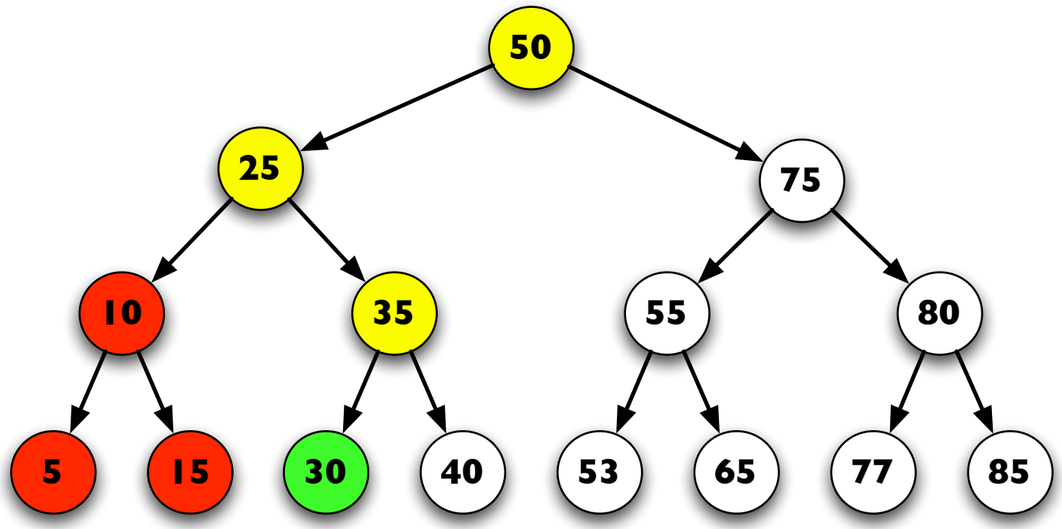
Postorder Traversal

5 15 10



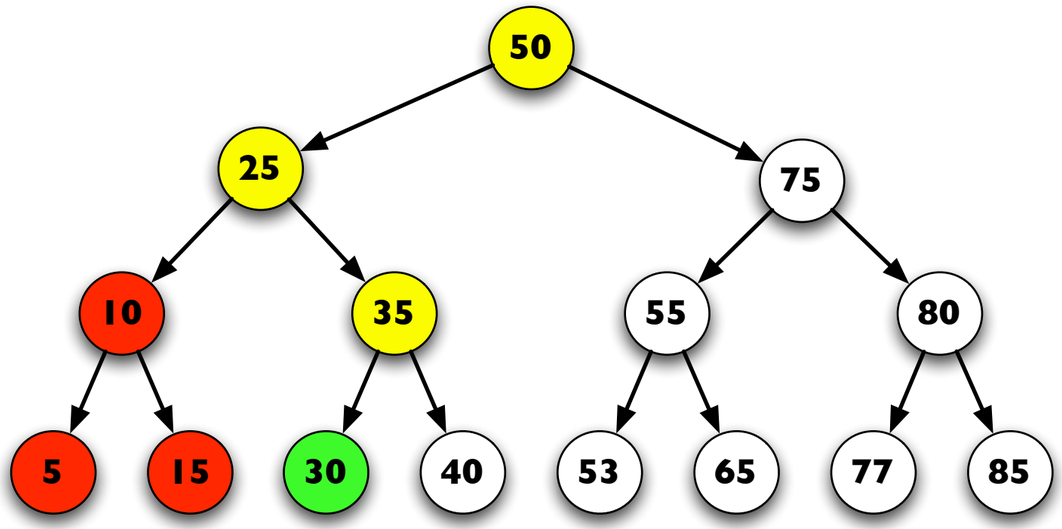
Postorder Traversal

5 15 10



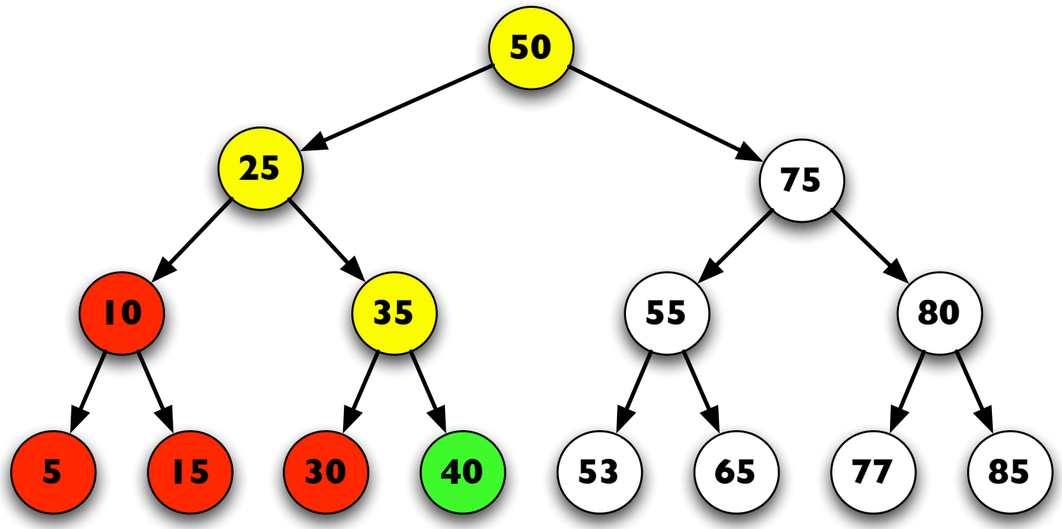
Postorder Traversal

5 15 10 30



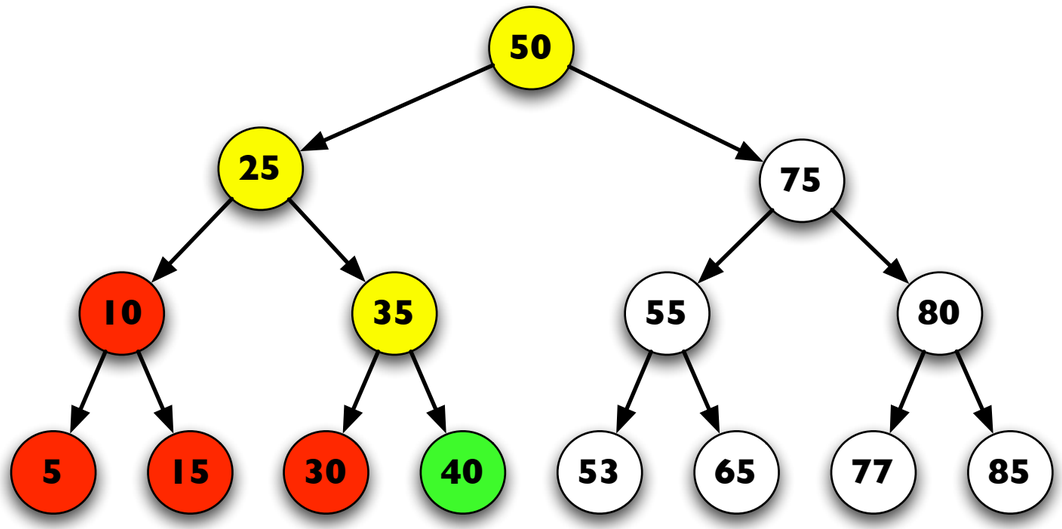
Postorder Traversal

5 15 10 30



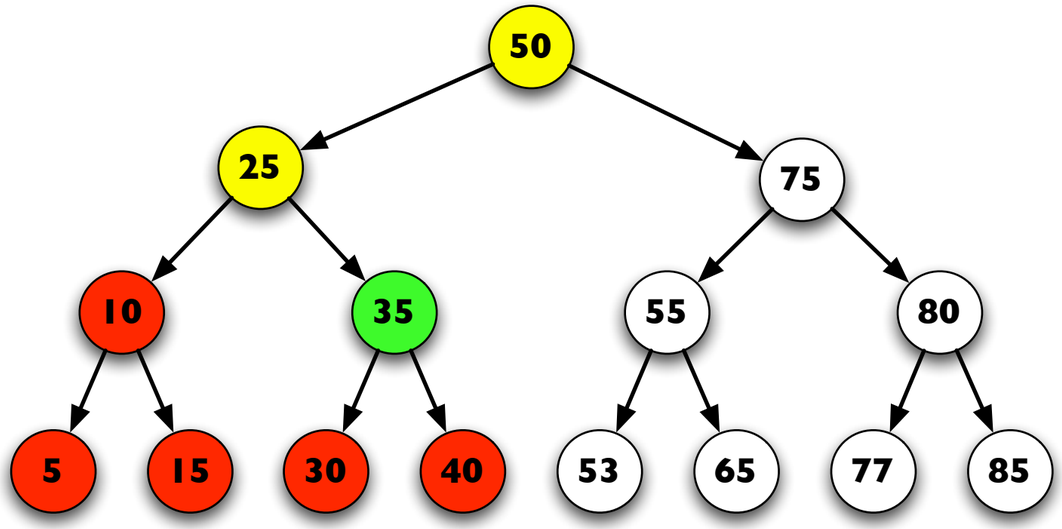
Postorder Traversal

5 15 10 30 40



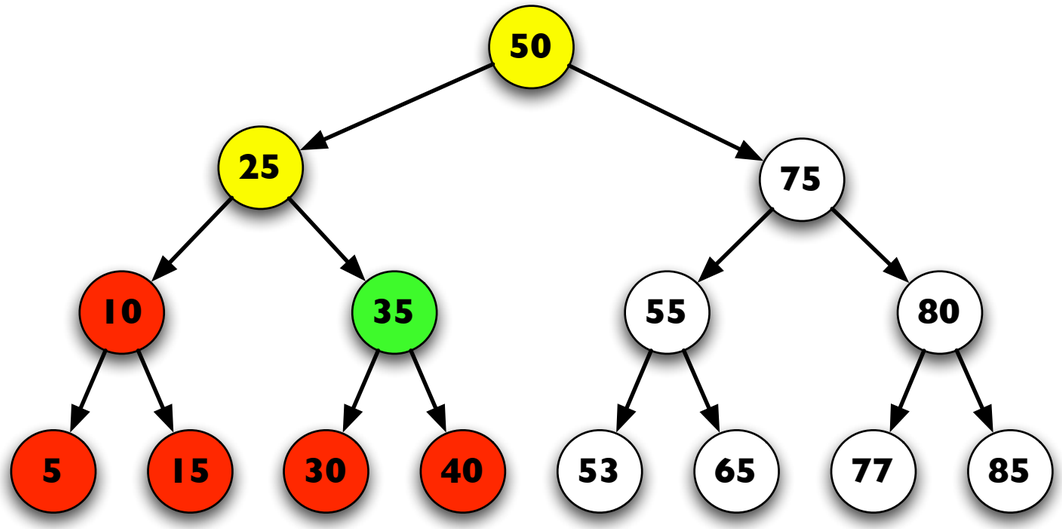
Postorder Traversal

5 15 10 30 40



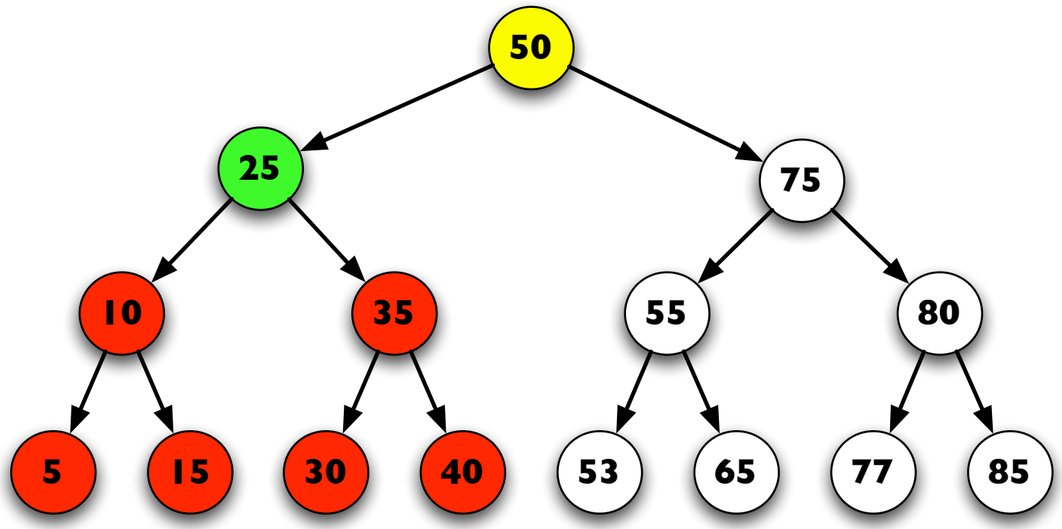
Postorder Traversal

5 15 10 30 40 35



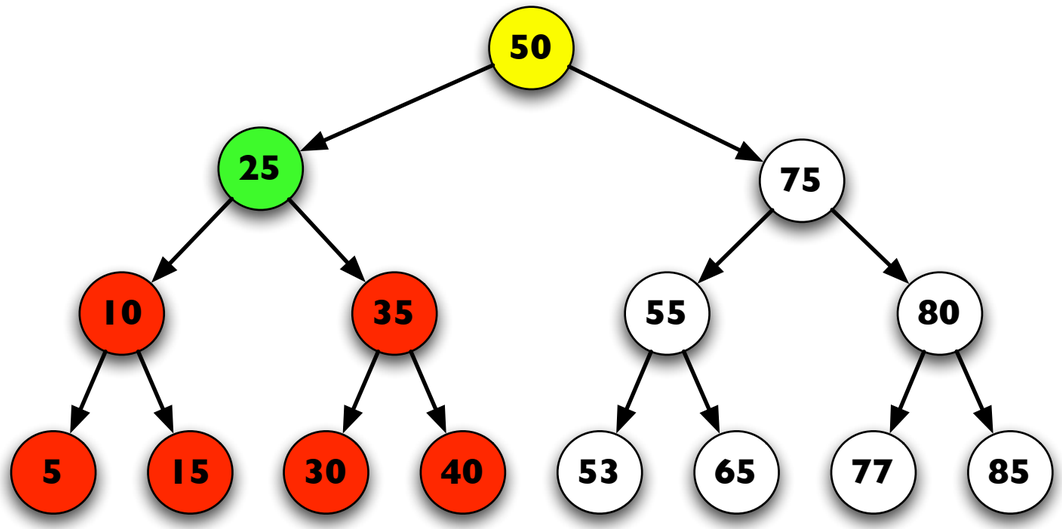
Postorder Traversal

5 15 10 30 40 35



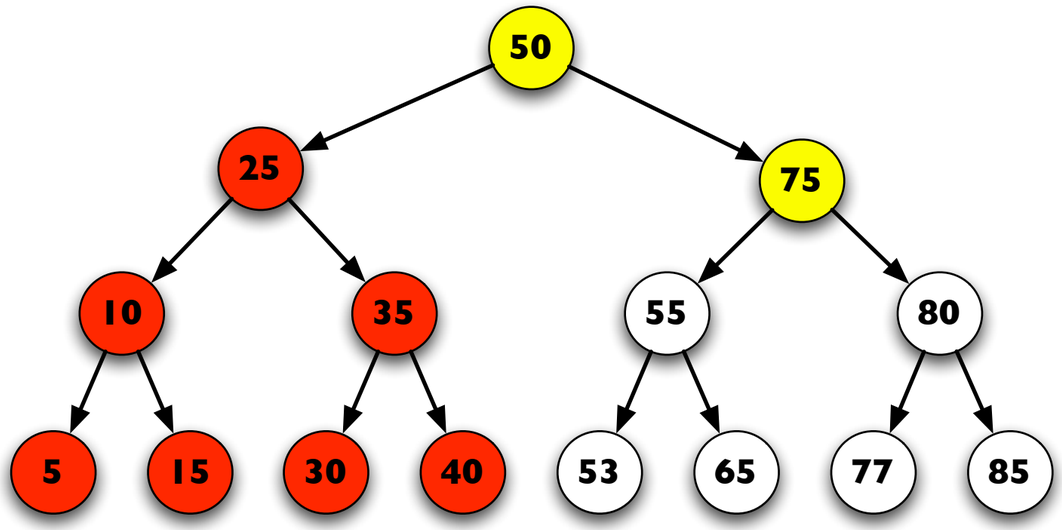
Postorder Traversal

5 15 10 30 40 35 25



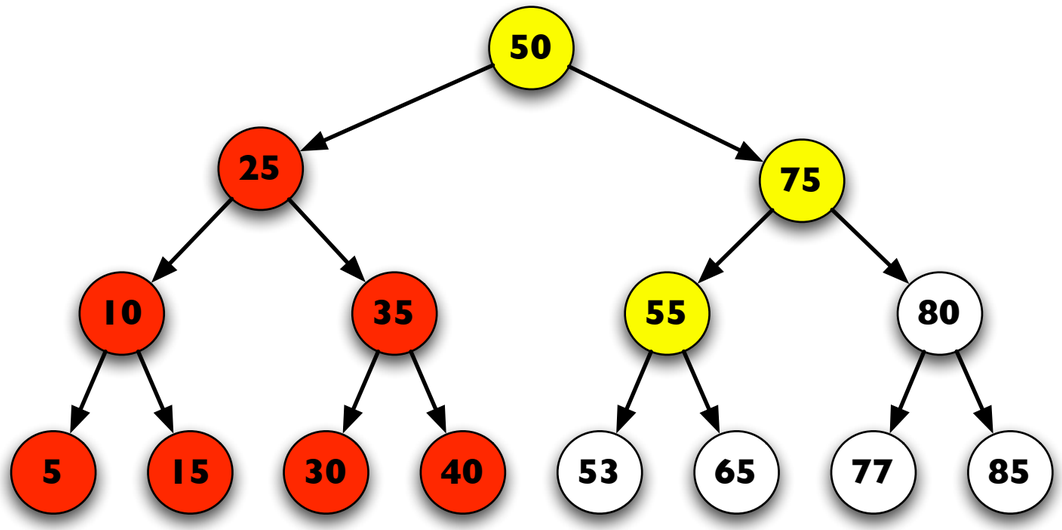
Postorder Traversal

5 15 10 30 40 35 25



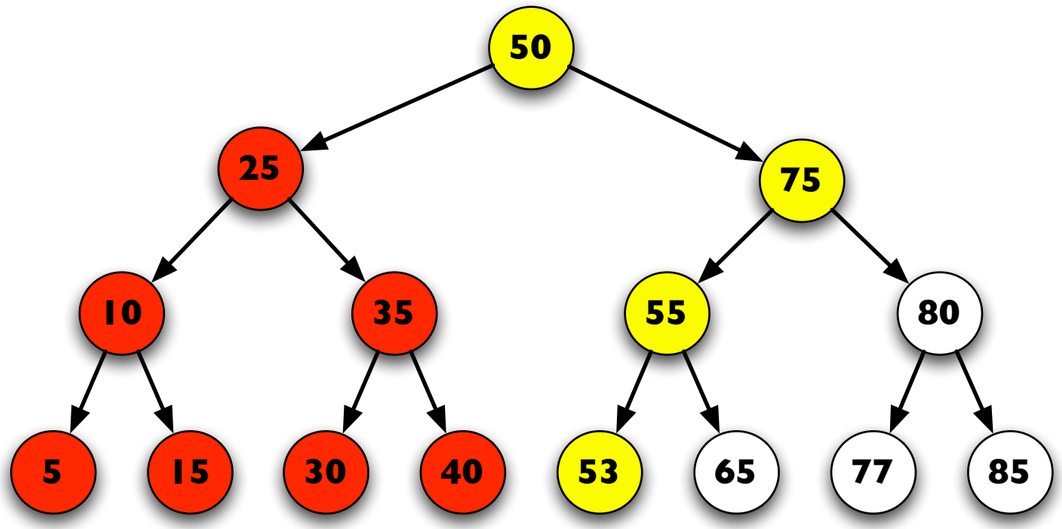
Postorder Traversal

5 15 10 30 40 35 25



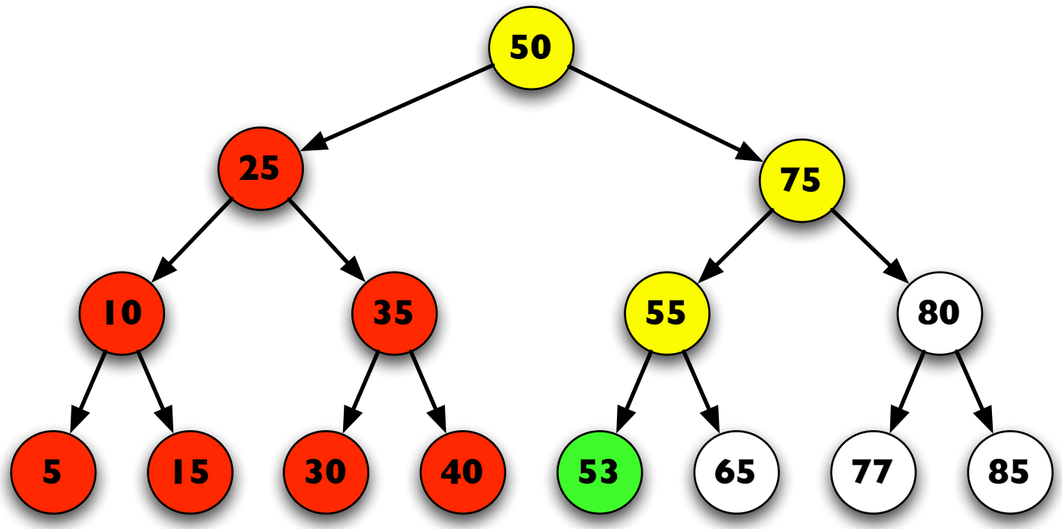
Postorder Traversal

5 15 10 30 40 35 25



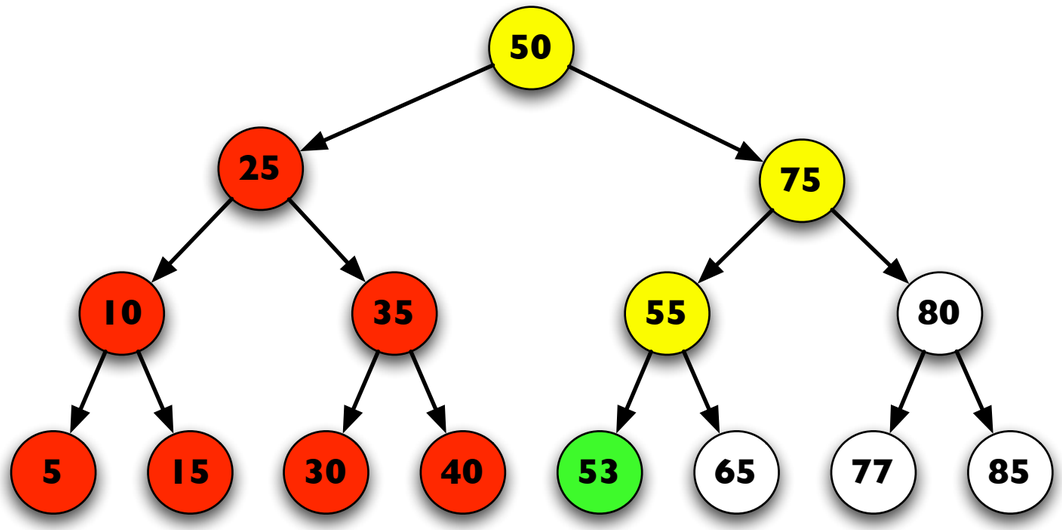
Postorder Traversal

5 15 10 30 40 35 25



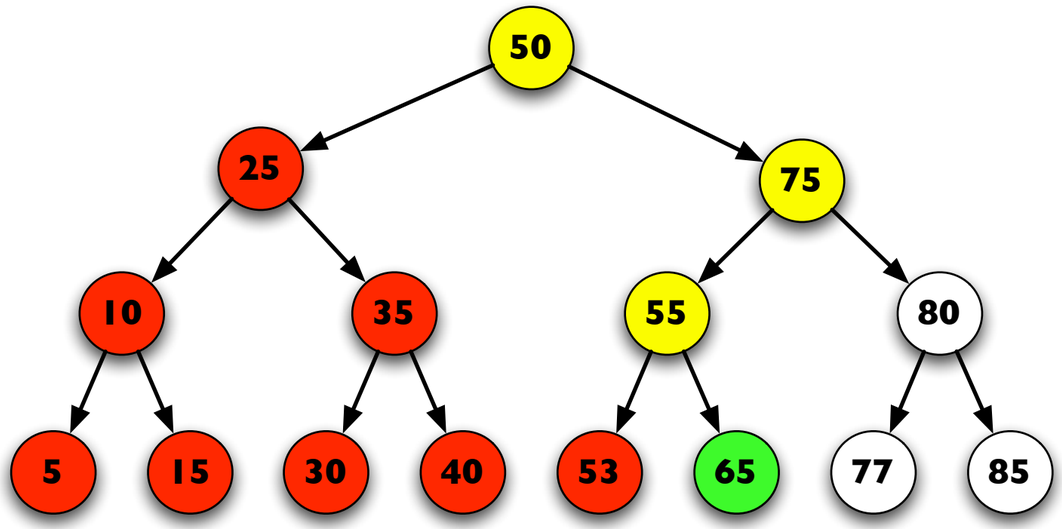
Postorder Traversal

5 15 10 30 40 35 25 53



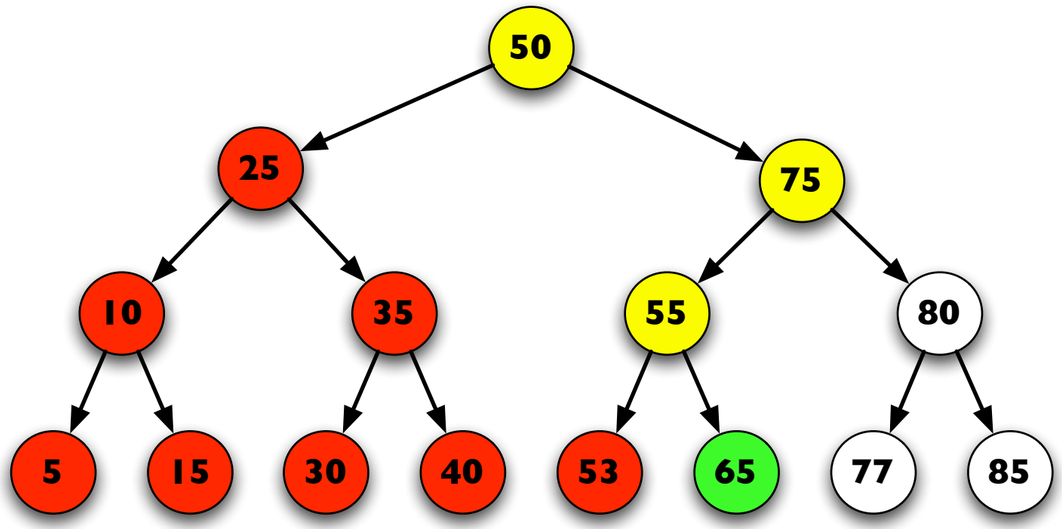
Postorder Traversal

5 15 10 30 40 35 25 53



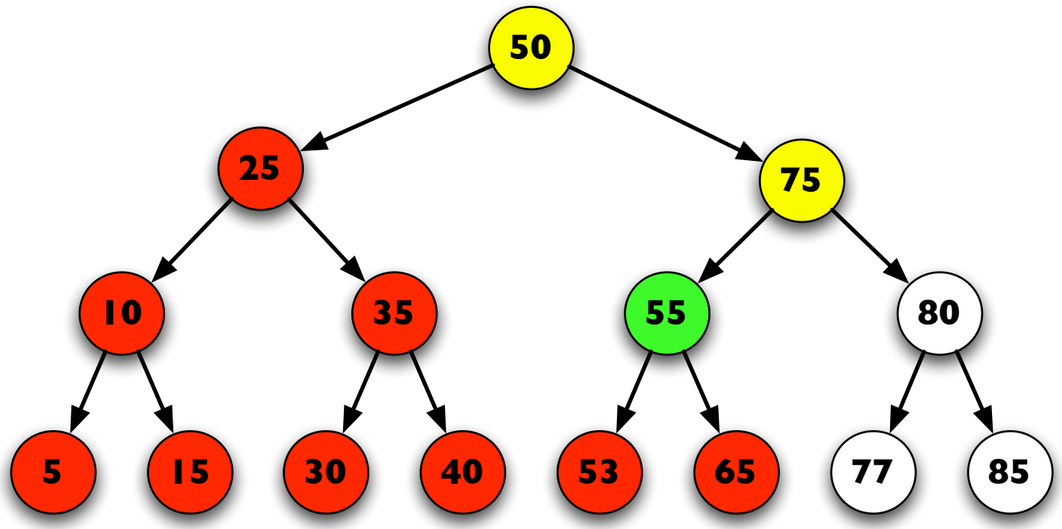
Postorder Traversal

5 15 10 30 40 35 25 53 65



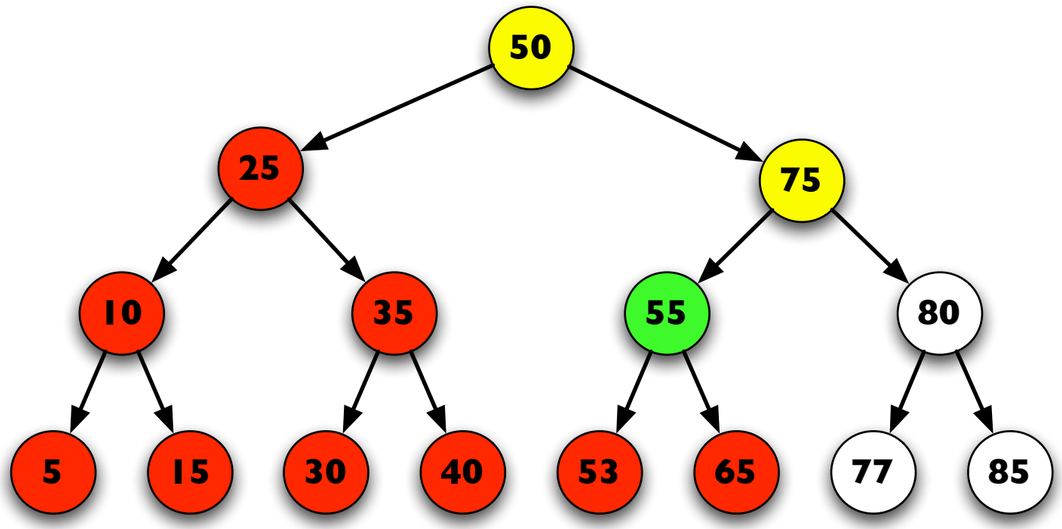
Postorder Traversal

5 15 10 30 40 35 25 53 65



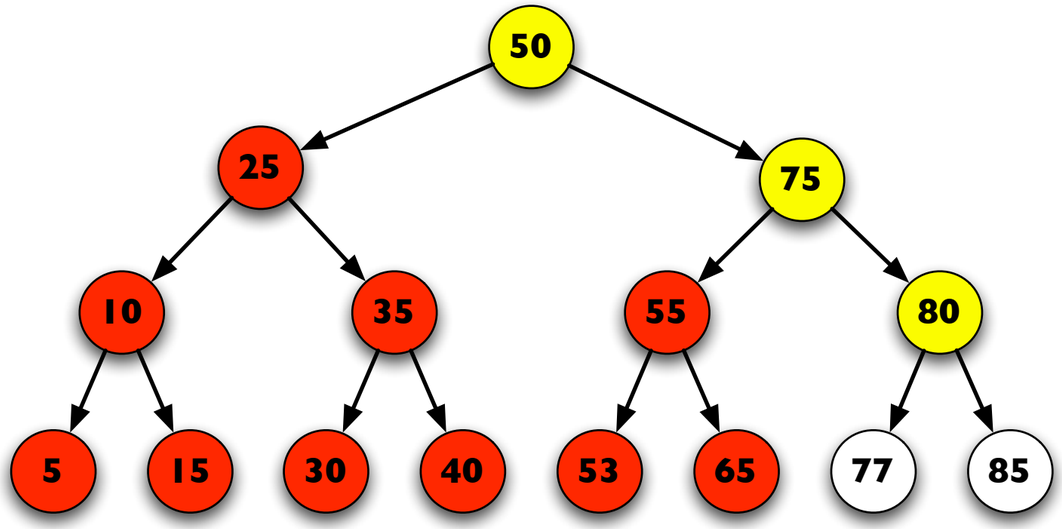
Postorder Traversal

5 15 10 30 40 35 25 53 65 55



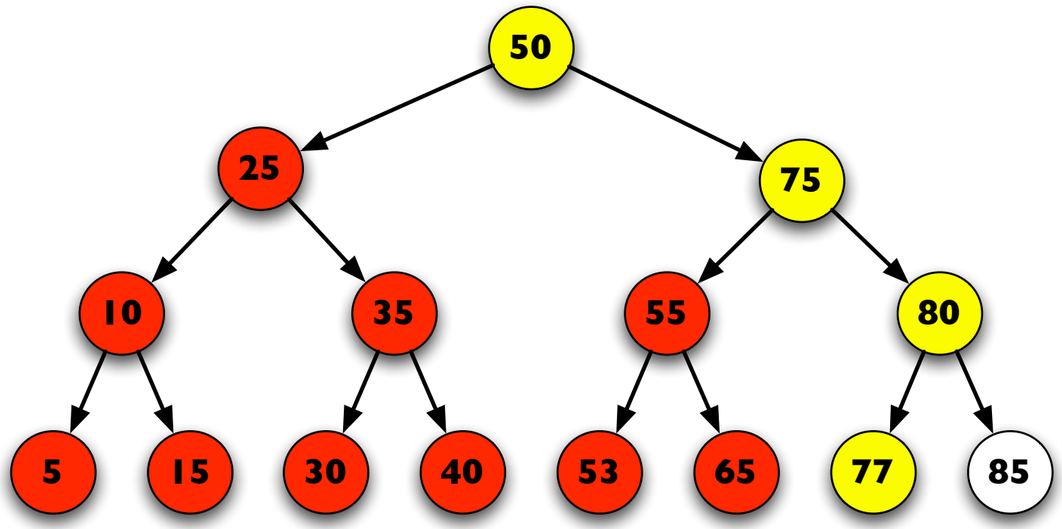
Postorder Traversal

5 15 10 30 40 35 25 53 65 55



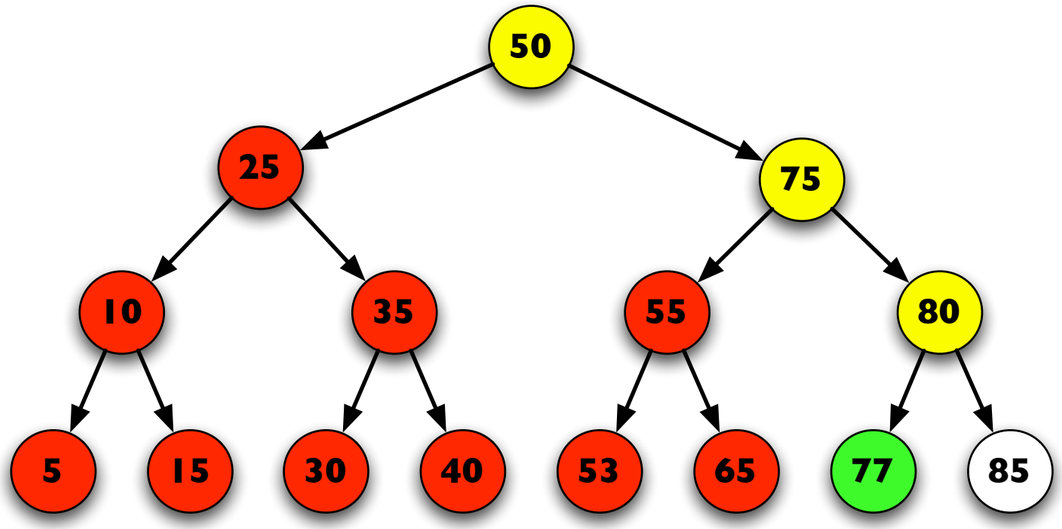
Postorder Traversal

5 15 10 30 40 35 25 53 65 55



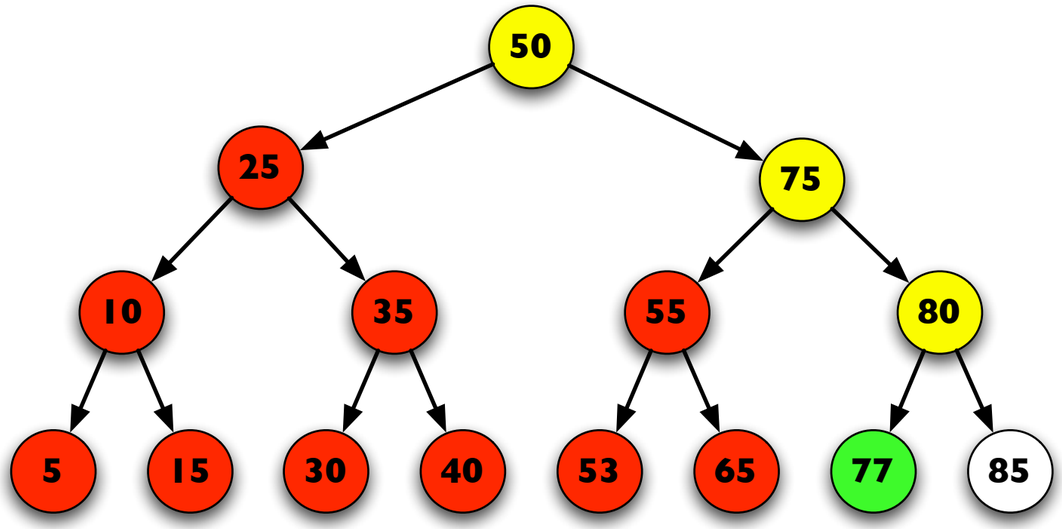
Postorder Traversal

5 15 10 30 40 35 25 53 65 55



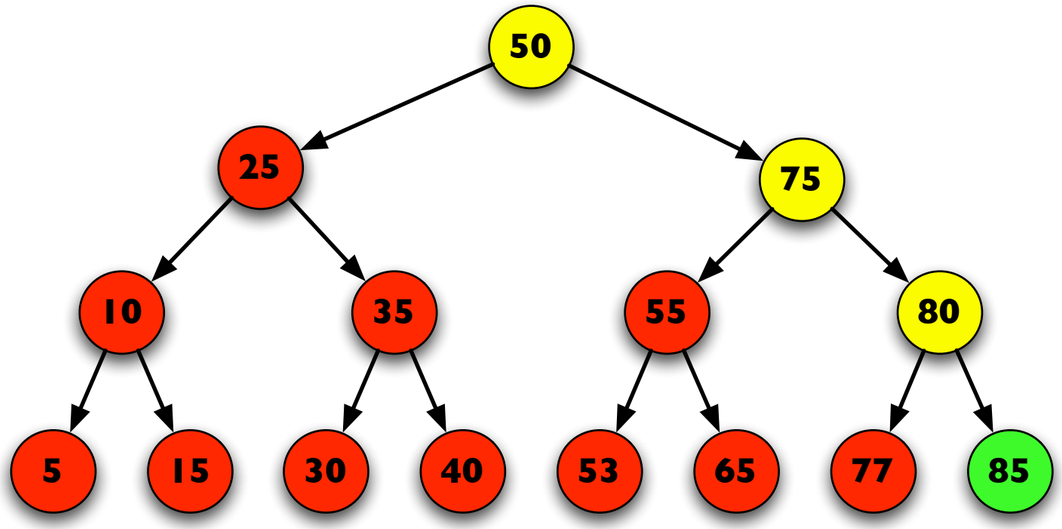
Postorder Traversal

5 15 10 30 40 35 25 53 65 55 77



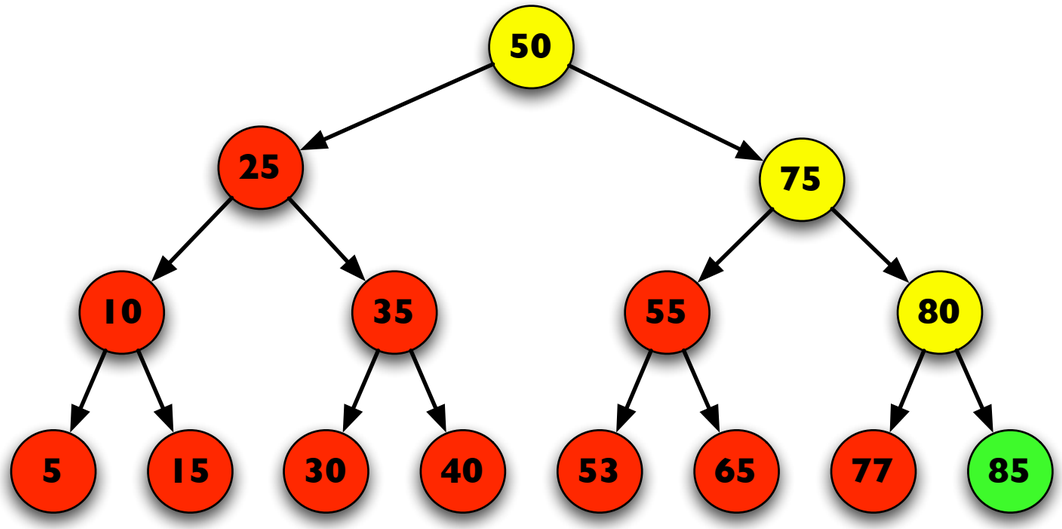
Postorder Traversal

5 15 10 30 40 35 25 53 65 55 77



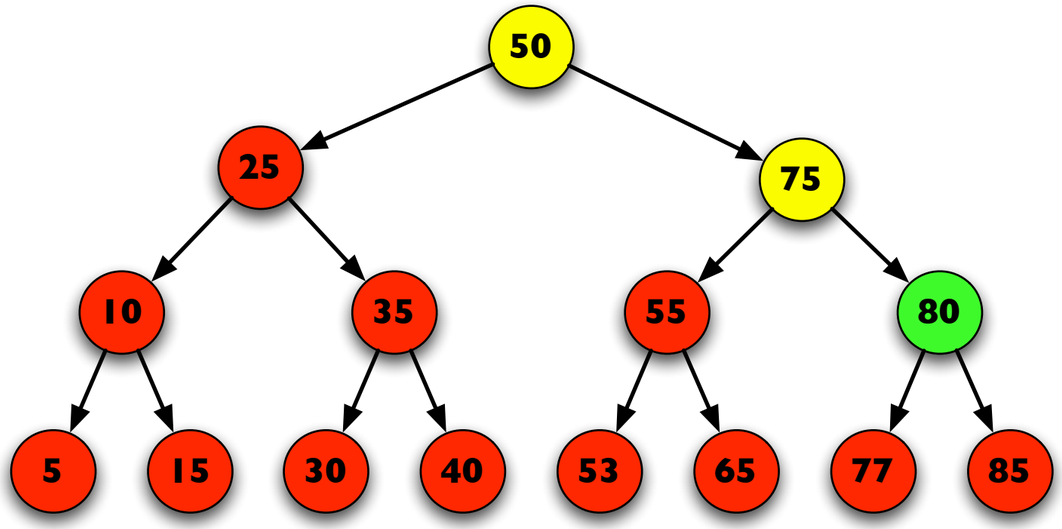
Postorder Traversal

5 15 10 30 40 35 25 53 65 55 77 85



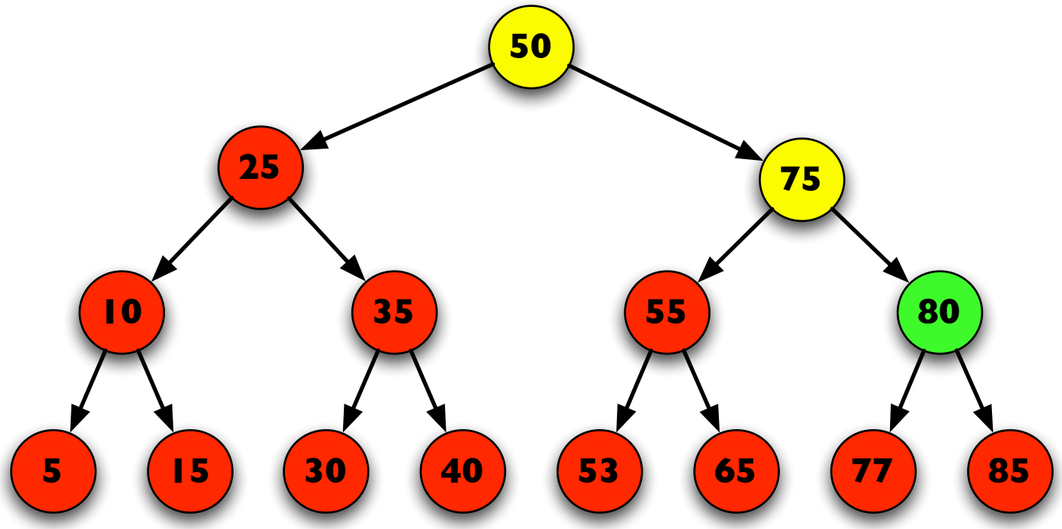
Postorder Traversal

5 15 10 30 40 35 25 53 65 55 77 85



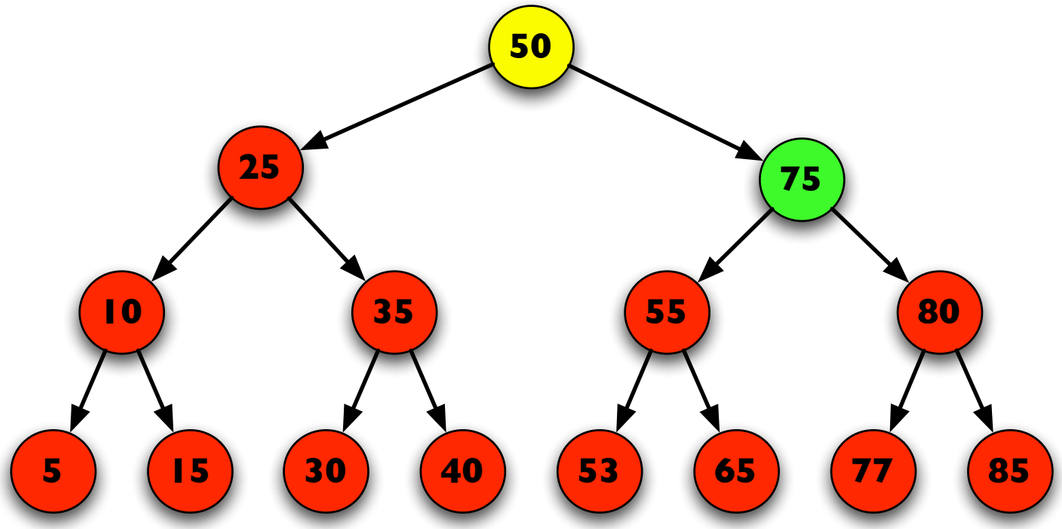
Postorder Traversal

5 15 10 30 40 35 25 53 65 55 77 85 80



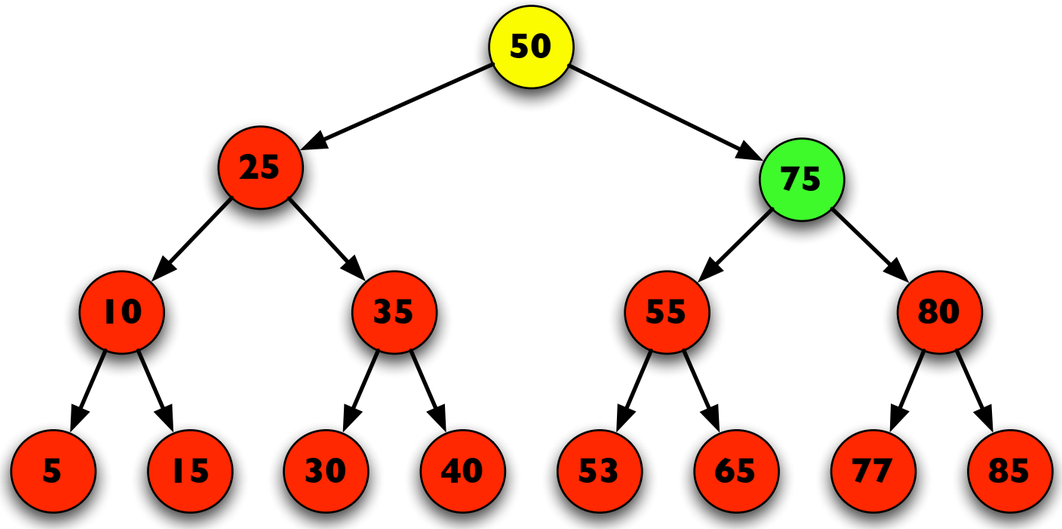
Postorder Traversal

5 15 10 30 40 35 25 53 65 55 77 85 80



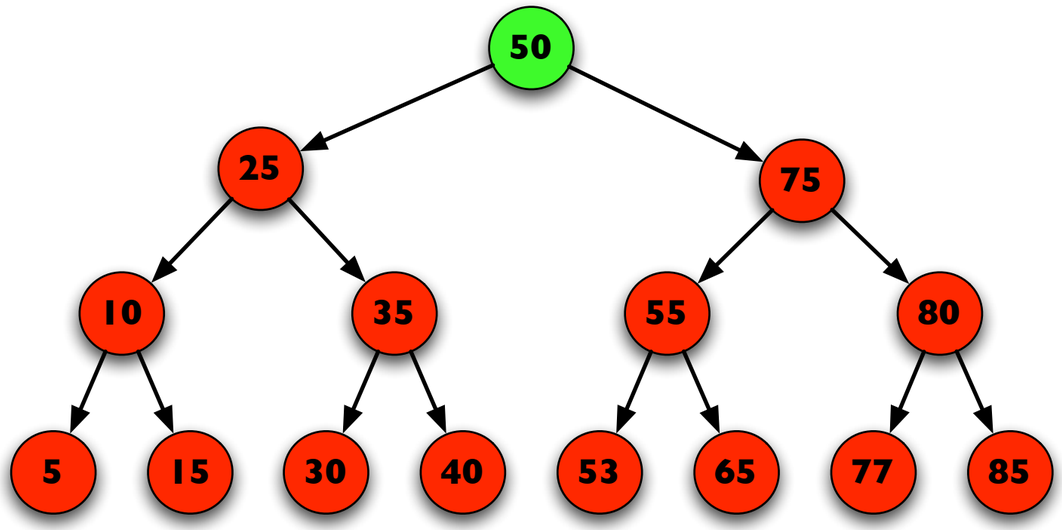
Postorder Traversal

5 15 10 30 40 35 25 53 65 55 77 85 80 75



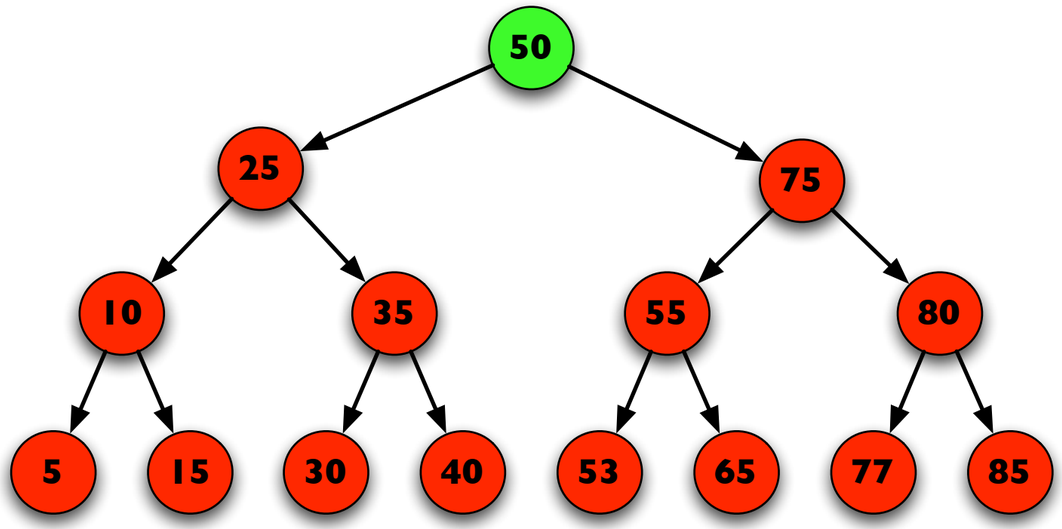
Postorder Traversal

5 15 10 30 40 35 25 53 65 55 77 85 80 75



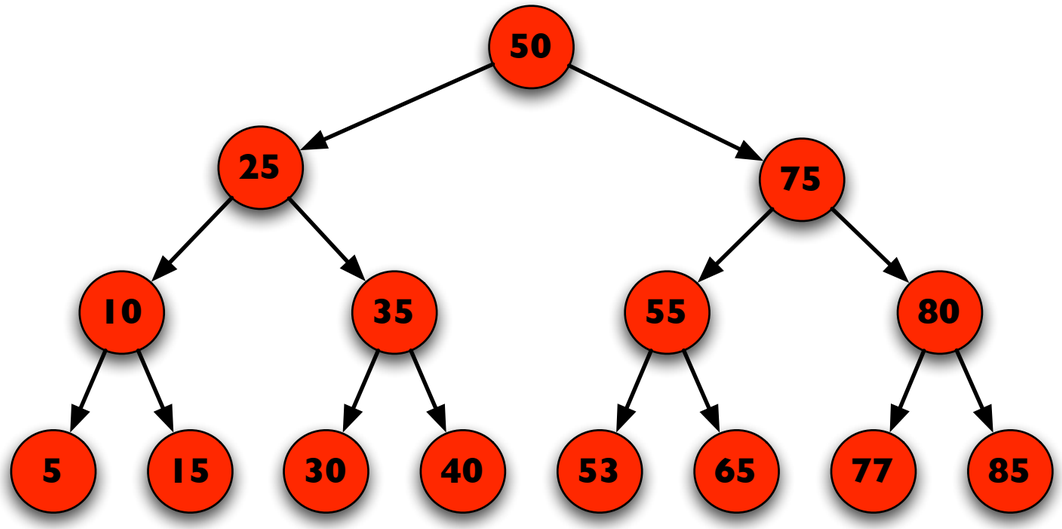
Postorder Traversal

5 15 10 30 40 35 25 53 65 55 77 85 80 75 50



Postorder Traversal

5 15 10 30 40 35 25 53 65 55 77 85 80 75 50

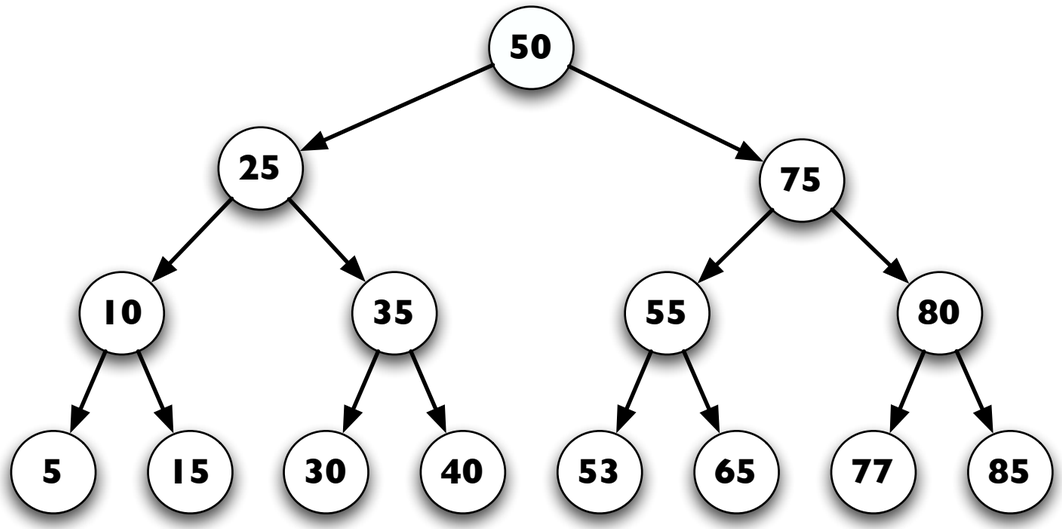


Comparison

Preorder

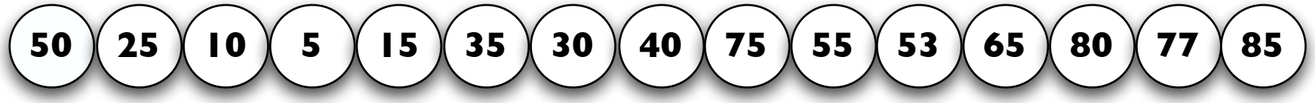
Inorder

Postorder



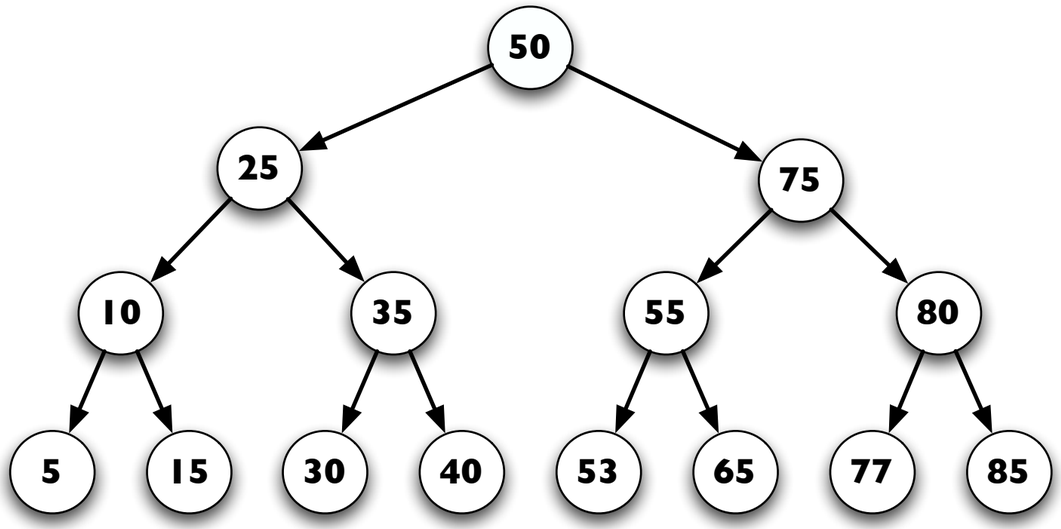
Comparison

Preorder



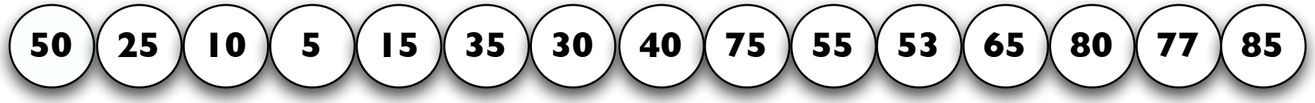
Inorder

Postorder

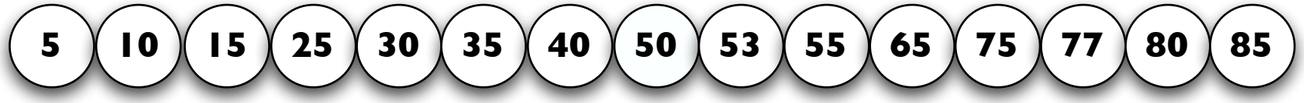


Comparison

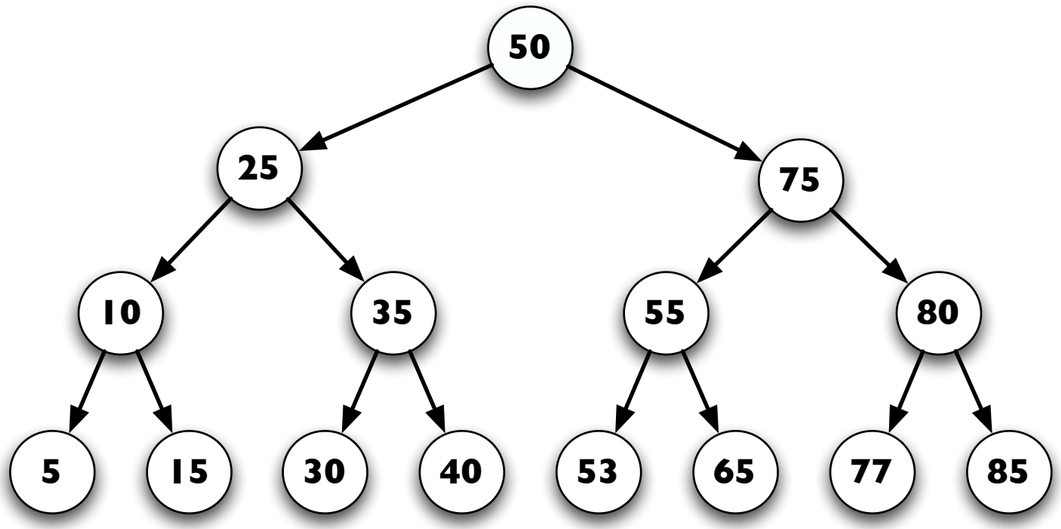
Preorder



Inorder

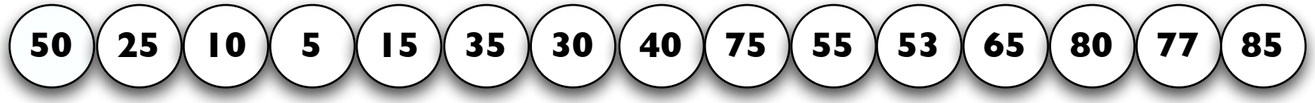


Postorder

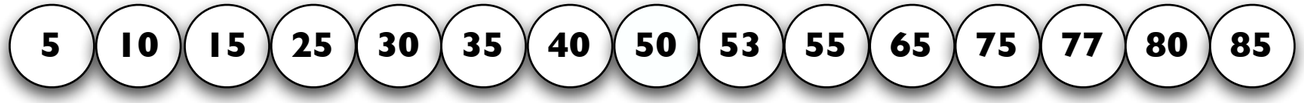


Comparison

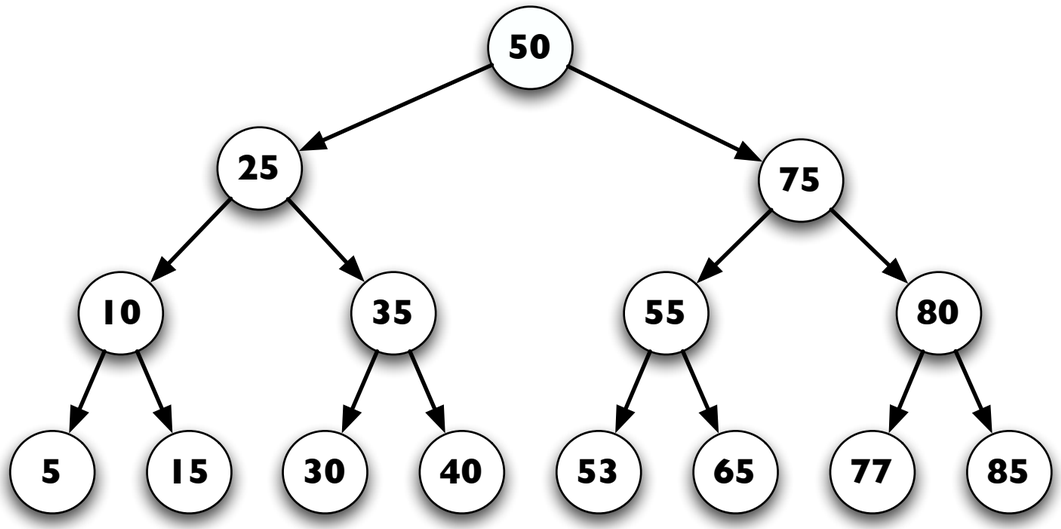
Preorder



Inorder



Postorder



MIAMI UNIVERSITY

Java API

Java API

- Java contains a binary tree implementation

Java API

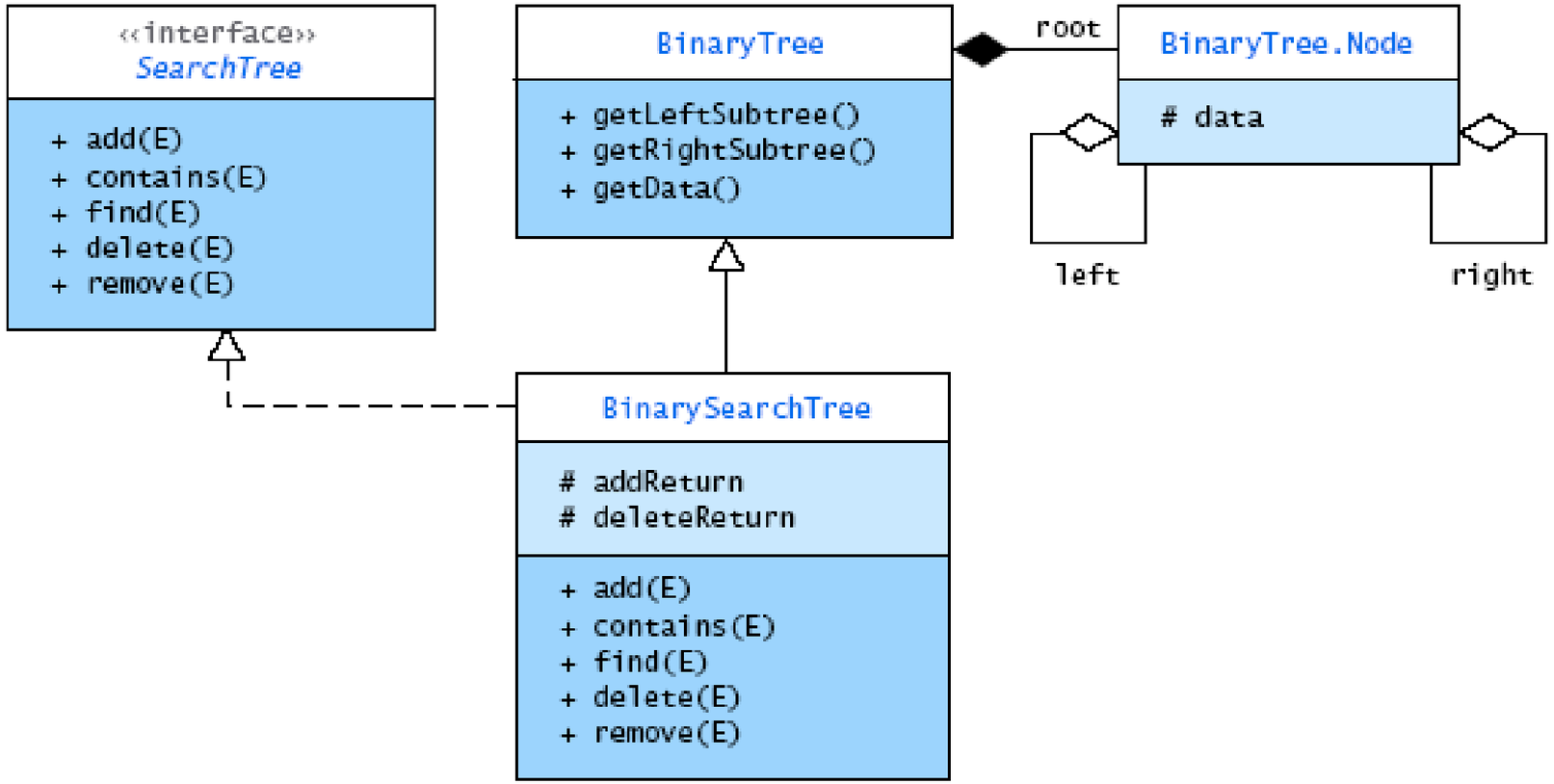
- Java contains a binary tree implementation
    - TreeSet<E>

Java API

- Java contains a binary tree implementation
    - TreeSet<E>
- http://java.sun.com/j2se/1.5.0/docs/api/java/util/TreeSet.html

Roll your own

FIGURE 8.15
UML Diagram of BinarySearchTree



Insert Code

```
public class BinarySearchTree<E> {

   Node<E> root = null;
   boolean addReturn = false;

   /** other methods **/

   public boolean add( E item ) {
      root = add( root, item );
   }

   private Node<E> add( Node<E> localRoot, E item ) {
      if ( localRoot == null ) {
         addReturn = true;
         return new Node<E>(item);
      } else if ( item.compareTo(localRoot.data) == 0 ) {
         addReturn = false;
         return localRoot;
      } else if ( item.compareTo(localRoot.data) < 0 ) {
         localRoot.left = add( localRoot.left, item );
         return localRoot;
      } else {
         localRoot.right = add( localRoot.right, item );
         return localRoot;
      }
   }

}
```

Insert Code

```
public class BinarySearchTree<E> {

   Node<E> root = null;
   boolean addReturn = false;

   /** other methods **/

   public boolean add( E item ) {
      root = add( root, item );
   }

   private Node<E> add( Node<E> localRoot, E item ) {
      if ( localRoot == null ) {
         addReturn = true;
         return new Node<E>(item);
      } else if ( item.compareTo(localRoot.data) == 0 ) {
         addReturn = false;
         return localRoot;
      } else if ( item.compareTo(localRoot.data) < 0 ) {
         localRoot.left = add( localRoot.left, item );
         return localRoot;
      } else {
         localRoot.right = add( localRoot.right, item );
         return localRoot;
      }
   }

}
```

Insert Code

```
public class BinarySearchTree<E> {

   Node<E> root = null;
   boolean addReturn = false;

   /** other methods **/

   public boolean add( E item ) {
      root = add( root, item );
   }

   private Node<E> add( Node<E> localRoot, E item ) {
      if ( localRoot == null ) {
         addReturn = true;
         return new Node<E>(item);
      } else if ( item.compareTo(localRoot.data) == 0 ) {
         addReturn = false;
         return localRoot;
      } else if ( item.compareTo(localRoot.data) < 0 ) {
         localRoot.left = add( localRoot.left, item );
         return localRoot;
      } else {
         localRoot.right = add( localRoot.right, item );
         return localRoot;
      }
   }

}
```

Insert Code

```
public class BinarySearchTree<E> {

   Node<E> root = null;
   boolean addReturn = false;

   /** other methods **/

   public boolean add( E item ) {
      root = add( root, item );
   }

   private Node<E> add( Node<E> localRoot, E item ) {
      if ( localRoot == null ) {
         addReturn = true;
         return new Node<E>(item);
      } else if ( item.compareTo(localRoot.data) == 0 ) {
         addReturn = false;
         return localRoot;
      } else if ( item.compareTo(localRoot.data) < 0 ) {
         localRoot.left = add( localRoot.left, item );
         return localRoot;
      } else {
         localRoot.right = add( localRoot.right, item );
         return localRoot;
      }
   }

}
```

Removal from BST

Removal from BST

- Removing a leaf

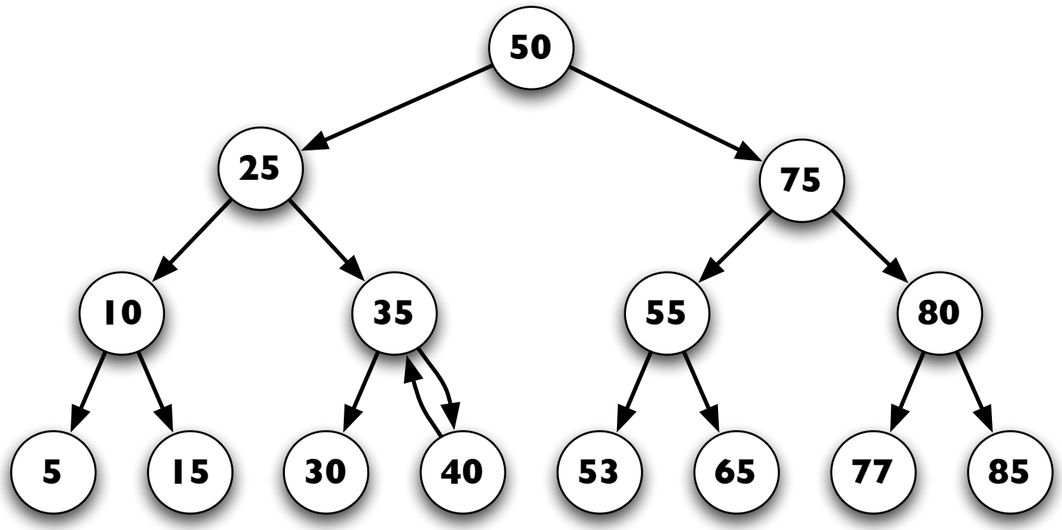
Removal from BST

- Removing a leaf
    - set it's parent's reference to it to null

Removal from BST

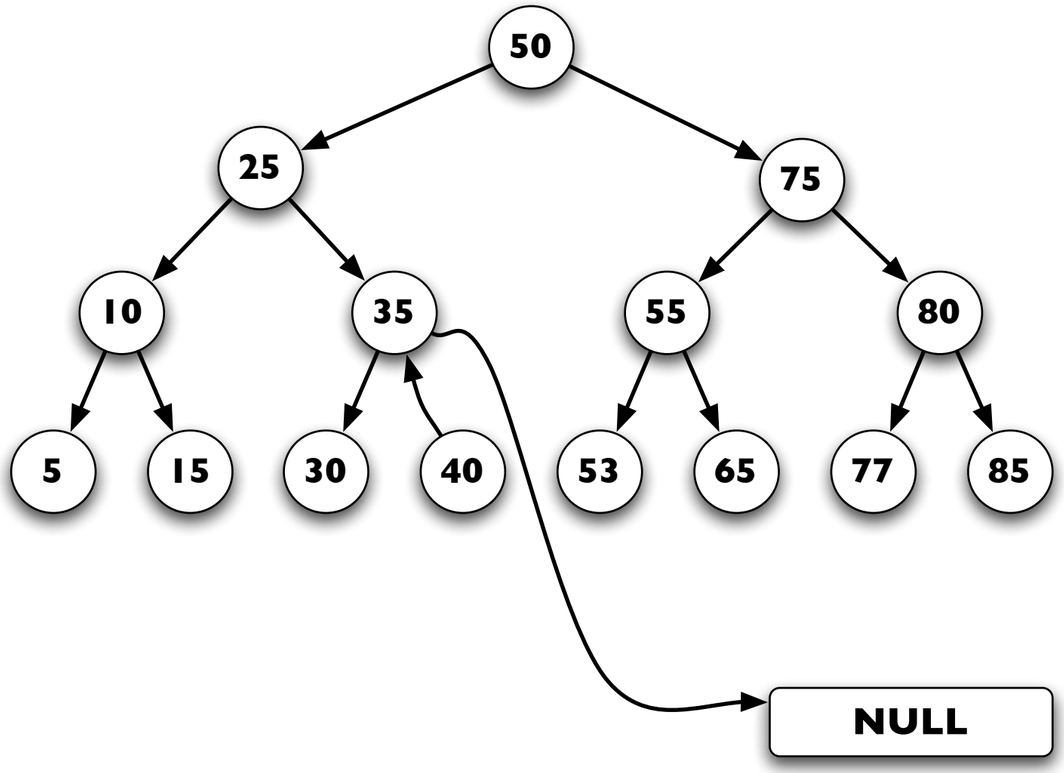
- Removing a leaf
    - set it's parent's reference to it to null
    - set it's reference to it's parent to null

Remove Leaf



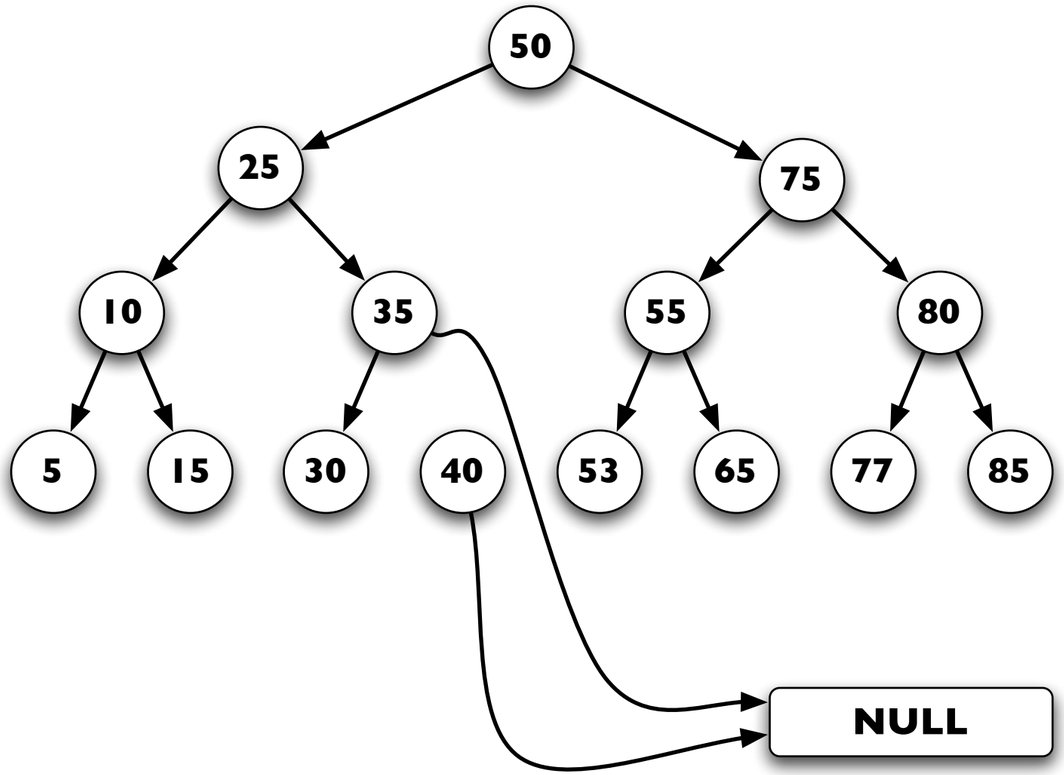
remove 40 (a leaf node)

Remove Leaf



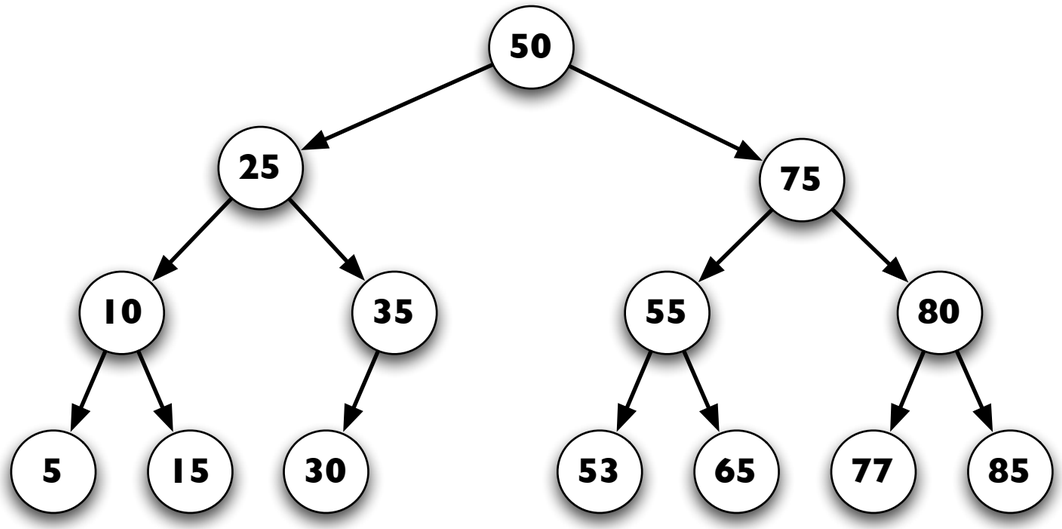
remove 40 (a leaf node)

Remove Leaf



remove 40 (a leaf node)

Remove Leaf



after 40 is removed

Remove Leaf

```
private void remove( Node<E> node ) {

   if ( node.isLeaf() ) {
      if ( node.parent.left == node ) {
         node.parent.left = null;
      } else if ( node.parent.right == node ) {
         node.parent.right = null;
      }
      node.parent = null;
   }

}
```

Remove (leaf only)

General Removal

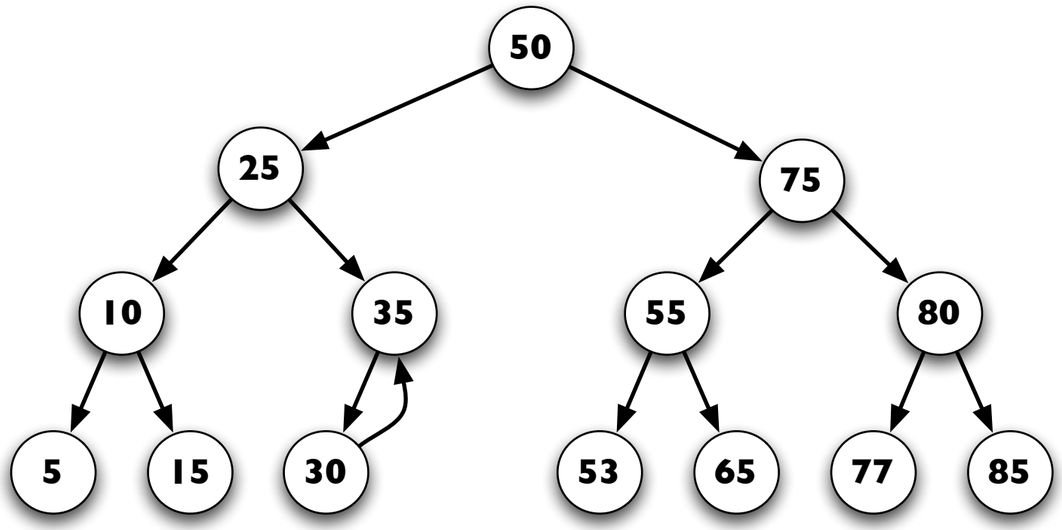
General Removal

- If the node to be removed, only has 1 child

General Removal

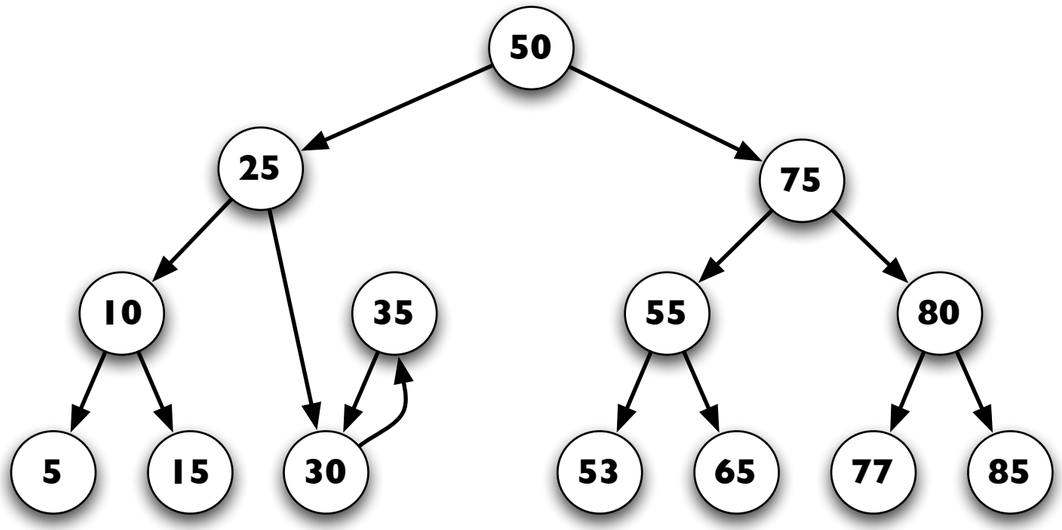
- If the node to be removed, only has 1 child
    - we can just “pull” the child up

Remove 1 Child



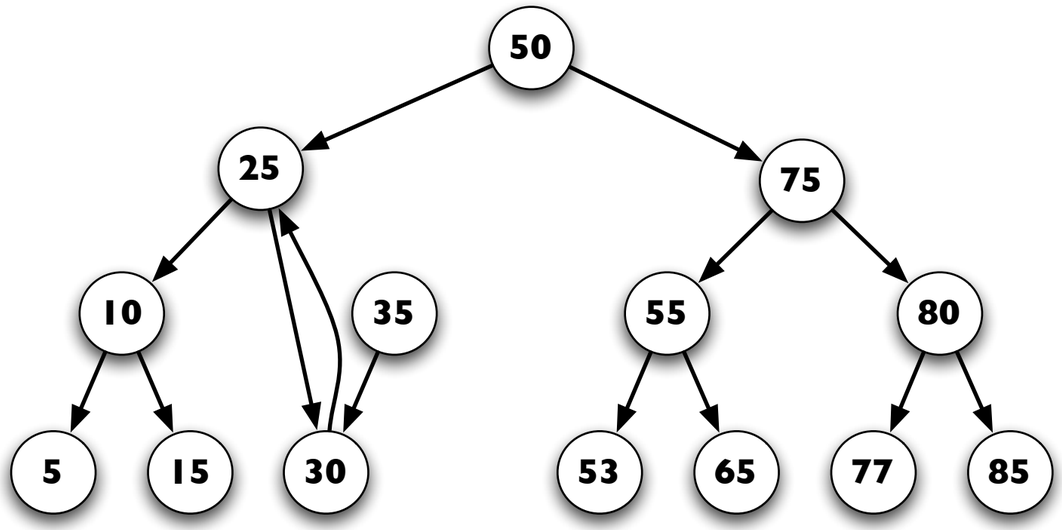
Remove 35

Remove 1 Child



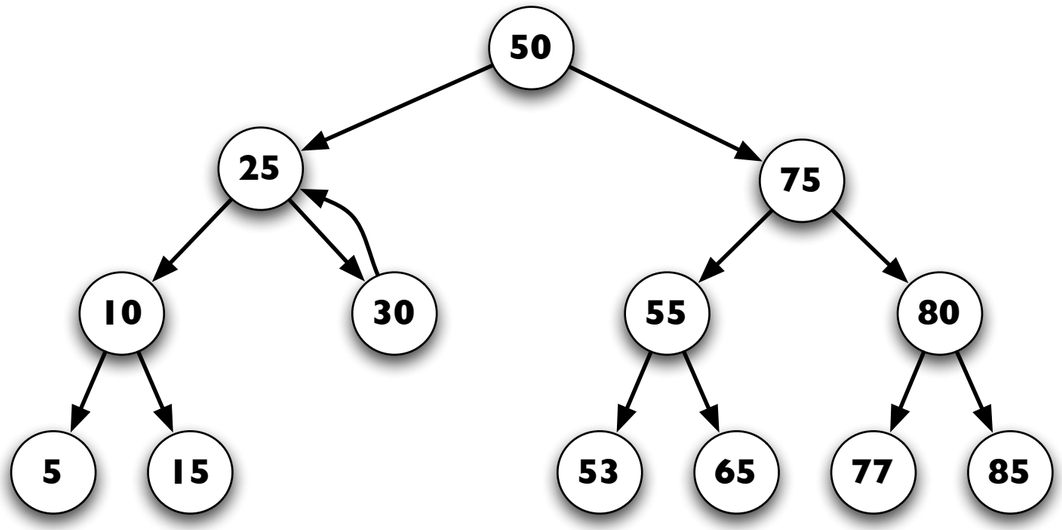
Remove 35

Remove 1 Child



Remove 35

Remove 1 Child



Remove 35

General Removal

General Removal

- More complicated if we have 2 children at the node to be removed

General Removal

- More complicated if we have 2 children at the node to be removed
    - How do we decide which node to pull up?

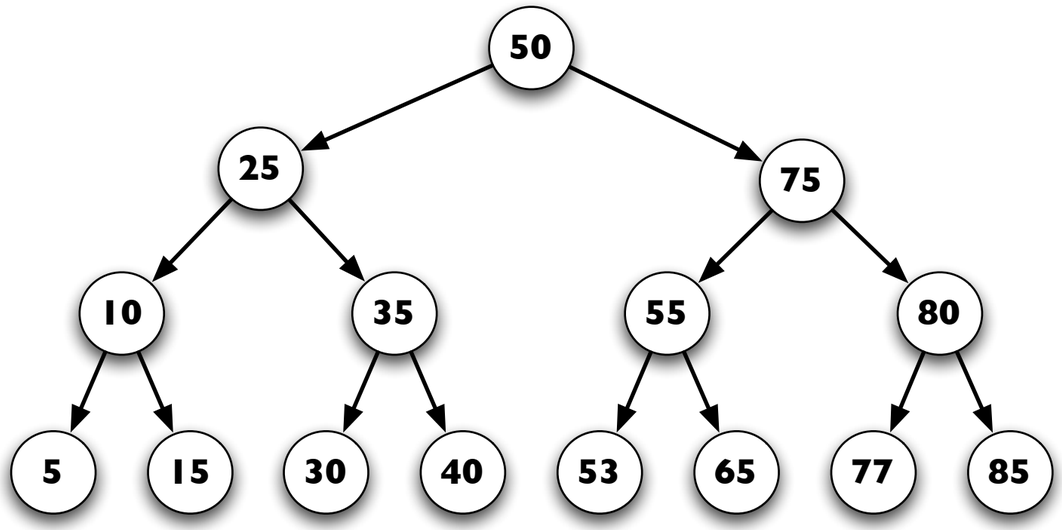
General Removal

- More complicated if we have 2 children at the node to be removed
  - How do we decide which node to pull up?
  - We need to keep the properties of the binary tree

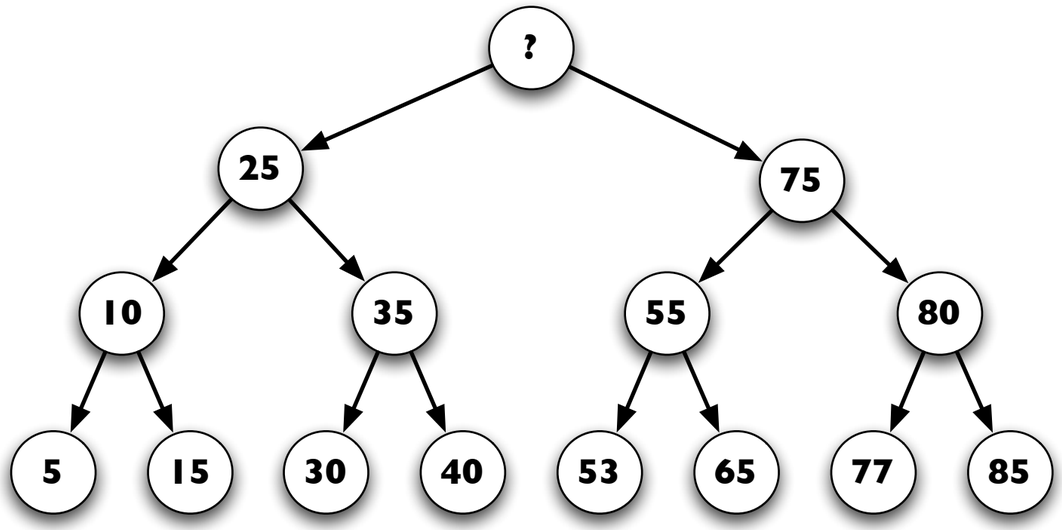
General Removal

- More complicated if we have 2 children at the node to be removed
  - How do we decide which node to pull up?
  - We need to keep the properties of the binary tree
    - node pulled up must be greater than everything in the left subtree and less than everything in the right subtree

Remove (2 children)

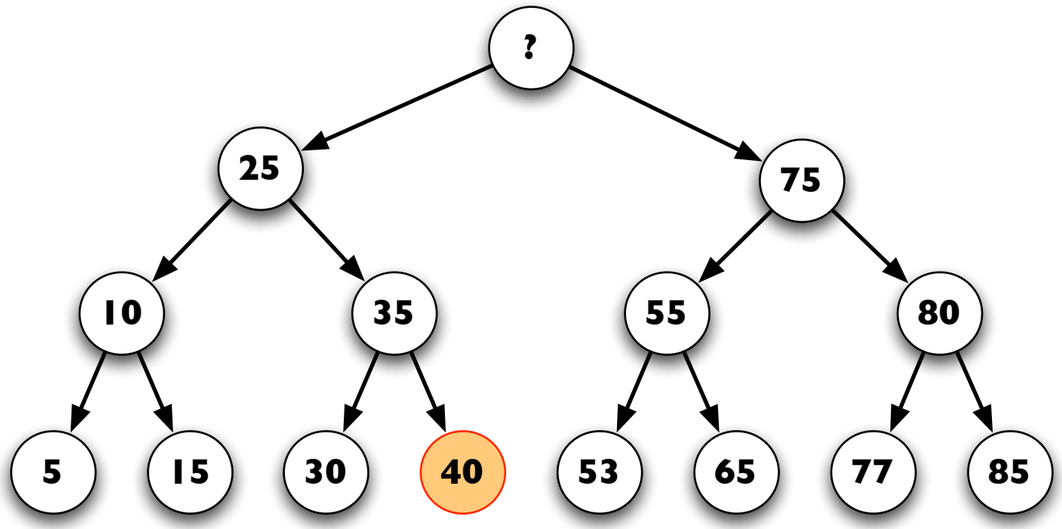


Remove (2 children)



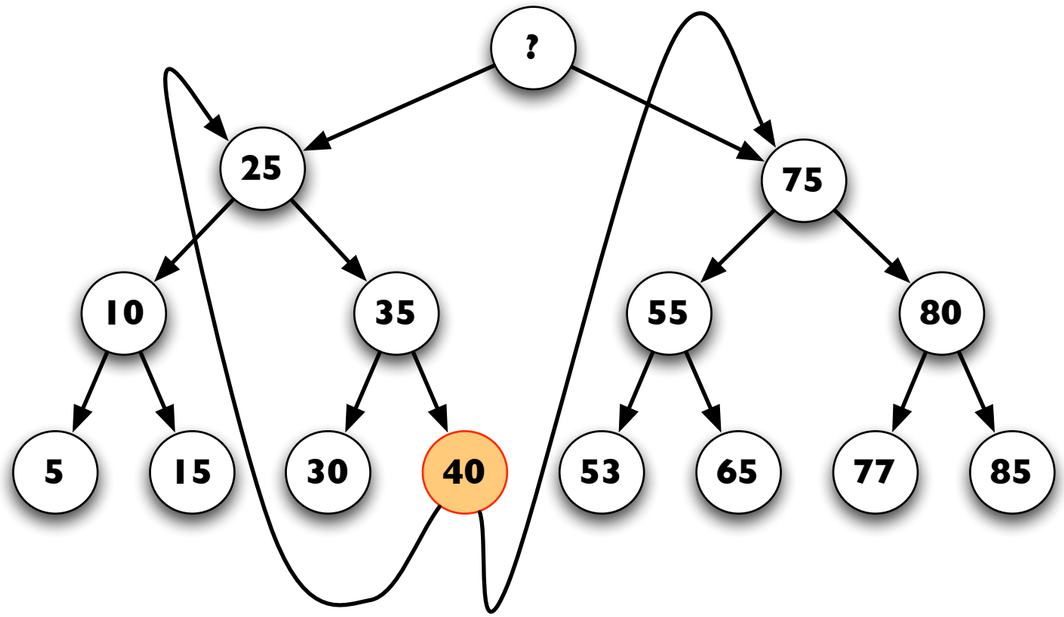
Remove 50

Remove (2 children)



Remove 50
select the largest node from the left subtree
(the inorder predecessor)

Remove (2 children)

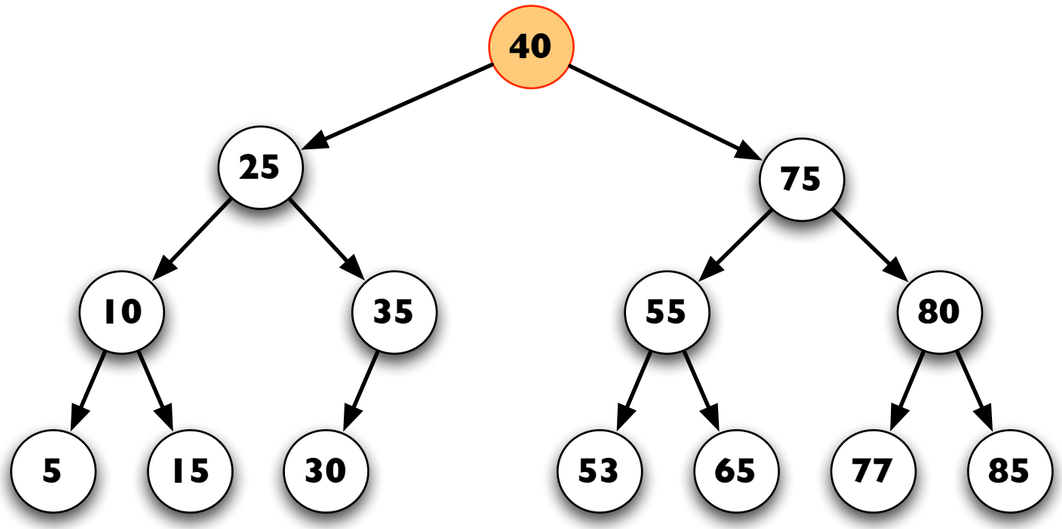


Remove 50

select the largest node from the left subtree

(the inorder predecessor)

Remove (2 children)



Remove 50
select the largest node from the left subtree
(the inorder predecessor)

Remove

```
1  public class BinarySearchTree<E> {
2     private E deleteReturn = null;
3
4     public E delete( E target ) {
5        root = delete( root, target );
6        return deleteReturn;
7     }
```

Remove

```
1  public class BinarySearchTree<E> {
2     private E deleteReturn = null;
3
4     public E delete( E target ) {
5        root = delete( root, target );
6        return deleteReturn;
7     }
```

Remove

```
 1 public class BinarySearchTree<E> {
 2    private E deleteReturn = null;
 3
 4    public E delete( E target ) {
 5       root = delete( root, target );
 6       return deleteReturn;
 7    }
 8
 9    private Node<E> delete( Node<E> localRoot, E item ) {…
49
50    private E findLargestChild( Node<E> parent ) {…
59
60 }
```

Remove

```
1  public class BinarySearchTree<E> {
2     private E deleteReturn = null;
3
4     public E delete( E target ) {
5        root = delete( root, target );
6        return deleteReturn;
7     }
8
9     private Node<E> delete( Node<E> localRoot, E item ) {...
49
50    private E findLargestChild( Node<E> parent ) {...
59
60 }
```

Remove

```
 1 public class BinarySearchTree<E> {
 2    private E deleteReturn = null;
 3
 4    public E delete( E target ) {
 5       root = delete( root, target );
 6       return deleteReturn;
 7    }
 8
 9    private Node<E> delete( Node<E> localRoot, E item ) {…
49
50    private E findLargestChild( Node<E> parent ) {…
59
60 }
```

Remove

```
1  public class BinarySearchTree<E> {
2     private E deleteReturn = null;
3
4     public E delete( E target ) {...
8
9     private Node<E> delete( Node<E> localRoot, E item ) {
10       if ( localRoot == null ) {
11          // item not found
12          deleteReturn = null;
13          return localRoot;
14       }
15
16       int compare = item.compareTo( localRoot.data );
17       if ( compare < 0 ) {...
47
48    }
49
50    private E findLargestChild( Node<E> parent ) {...
59
60 }
```

Remove

```
1  public class BinarySearchTree<E> {
2     private E deleteReturn = null;
3
4     public E delete( E target ) {...
8
9     private Node<E> delete( Node<E> localRoot, E item ) {
10       if ( localRoot == null ) {
11          // item not found
12          deleteReturn = null;
13          return localRoot;
14       }
15
16       int compare = item.compareTo( localRoot.data );
17       if ( compare < 0 ) {...
47
48    }
49
50    private E findLargestChild( Node<E> parent ) {...
59
60 }
```

Remove

```
1  public class BinarySearchTree<E> {
2     private E deleteReturn = null;
3
4     public E delete( E target ) {...
8
9     private Node<E> delete( Node<E> localRoot, E item ) {
10       if ( localRoot == null ) {
11          // item not found
12          deleteReturn = null;
13          return localRoot;
14       }
15
16       int compare = item.compareTo( localRoot.data );
17       if ( compare < 0 ) {...
47
48    }
49
50    private E findLargestChild( Node<E> parent ) {...
59
60 }
```

Remove

```
1  public class BinarySearchTree<E> {
2     private E deleteReturn = null;
3
4     public E delete( E target ) {...
8
9     private Node<E> delete( Node<E> localRoot, E item ) {
10       if ( localRoot == null ) {...
15
16       int compare = item.compareTo( localRoot.data );
17       if ( compare < 0 ) {
18          localRoot.left = delete( localRoot.left, item );
19          return localRoot;
20       } else if ( compare > 0 ) {
21          localRoot.right = delete( localRoot.right, item );
22          return localRoot;
23       } else {
24          // item to be removed is the local root
25          if ( localRoot.left == null ) {...
45
46       }
47
48    }
49
50    private E findLargestChild( Node<E> parent ) {...
59
60 }
```

Remove

```
1  public class BinarySearchTree<E> {
2     private E deleteReturn = null;
3
4     public E delete( E target ) {...
8
9     private Node<E> delete( Node<E> localRoot, E item ) {
10       if ( localRoot == null ) {...
15
16       int compare = item.compareTo( localRoot.data );
17       if ( compare < 0 ) {
18          localRoot.left = delete( localRoot.left, item );
19          return localRoot;
20       } else if ( compare > 0 ) {
21          localRoot.right = delete( localRoot.right, item );
22          return localRoot;
23       } else {
24          // item to be removed is the local root
25          if ( localRoot.left == null ) {...
45
46       }
47
48    }
49
50    private E findLargestChild( Node<E> parent ) {...
59
60 }
```

Remove

```
1  public class BinarySearchTree<E> {
2     private E deleteReturn = null;
3
4     public E delete( E target ) {...
8
9     private Node<E> delete( Node<E> localRoot, E item ) {
10       if ( localRoot == null ) {...
15
16       int compare = item.compareTo( localRoot.data );
17       if ( compare < 0 ) {
18          localRoot.left = delete( localRoot.left, item );
19          return localRoot;
20       } else if ( compare > 0 ) {
21          localRoot.right = delete( localRoot.right, item );
22          return localRoot;
23       } else {
24          // item to be removed is the local root
25          if ( localRoot.left == null ) {...
45
46       }
47
48    }
49
50    private E findLargestChild( Node<E> parent ) {...
59
60 }
```

Remove

```
23        } else {
24           // item to be removed is the local root
25           if ( localRoot.left == null ) {
26              // no left child, pull up right
27              return localRoot.right;
28           } else if ( localRoot.right == null ) {
29              // no right child, pull pu left
30              return localRoot.left;
31           } else {
32              // node has 2 children - need to reorganize
33              if ( localNode.left.right == null ) {
34                 // left child has no right child, so it
35                 // is the largest node on our left
36                 localRoot.data = localRoot.left.data;
37                 localRoot.left = localRoot.left.left;
38                 return localRoot;
39              } else {
40                 // in the inorder predecessor
41                 localRoot.data = findLargestChild( localRoot.left );
42                 return localRoot;
43              }
44           }
45
46        }
```

Remove

```
23      } else {
24         // item to be removed is the local root
25         if ( localRoot.left == null ) {
26            // no left child, pull up right
27            return localRoot.right;
28         } else if ( localRoot.right == null ) {
29            // no right child, pull pu left
30            return localRoot.left;
31         } else {
32            // node has 2 children - need to reorganize
33            if ( localNode.left.right == null ) {
34               // left child has no right child, so it
35               // is the largest node on our left
36               localRoot.data = localRoot.left.data;
37               localRoot.left = localRoot.left.left;
38               return localRoot;
39            } else {
40               // in the inorder predecessor
41               localRoot.data = findLargestChild( localRoot.left );
42               return localRoot;
43            }
44         }
45
46      }
```

Remove

```
23      } else {
24         // item to be removed is the local root
25         if ( localRoot.left == null ) {
26            // no left child, pull up right
27            return localRoot.right;
28         } else if ( localRoot.right == null ) {
29            // no right child, pull pu left
30            return localRoot.left;
31         } else {
32            // node has 2 children - need to reorganize
33            if ( localNode.left.right == null ) {
34               // left child has no right child, so it
35               // is the largest node on our left
36               localRoot.data = localRoot.left.data;
37               localRoot.left = localRoot.left.left;
38               return localRoot;
39            } else {
40               // in the inorder predecessor
41               localRoot.data = findLargestChild( localRoot.left );
42               return localRoot;
43            }
44         }
45
46      }
```

Remove

```
23        } else {
24           // item to be removed is the local root
25           if ( localRoot.left == null ) {
26              // no left child, pull up right
27              return localRoot.right;
28           } else if ( localRoot.right == null ) {
29              // no right child, pull pu left
30              return localRoot.left;
31           } else {
32              // node has 2 children - need to reorganize
33              if ( localNode.left.right == null ) {
34                 // left child has no right child, so it
35                 // is the largest node on our left
36                 localRoot.data = localRoot.left.data;
37                 localRoot.left = localRoot.left.left;
38                 return localRoot;
39              } else {
40                 // in the inorder predecessor
41                 localRoot.data = findLargestChild( localRoot.left );
42                 return localRoot;
43              }
44           }
45
46        }
```

Remove

```
50    private E findLargestChild( Node<E> parent ) {
51       if ( parent.right.right == null ) {
52          E returnValue = parent.right.data;
53          parent.right = parent.right.left;
54          return returnValue;
55       } else {
56          return findLargestChild( parent.right );
57       }
58    }
```

Remove

```
50    private E findLargestChild( Node<E> parent ) {
51       if ( parent.right.right == null ) {
52          E returnValue = parent.right.data;
53          parent.right = parent.right.left;
54          return returnValue;
55       } else {
56          return findLargestChild( parent.right );
57       }
58    }
```

Remove

```
50    private E findLargestChild( Node<E> parent ) {
51       if ( parent.right.right == null ) {
52          E returnValue = parent.right.data;
53          parent.right = parent.right.left;
54          return returnValue;
55       } else {
56          return findLargestChild( parent.right );
57       }
58    }
```

Index Generator

Index Generator

- Could be used to generate an index for a book or paper

Index Generator

- Could be used to generate an index for a book or paper
    - but we'll see a better way to do this later...