202: Computer Science II

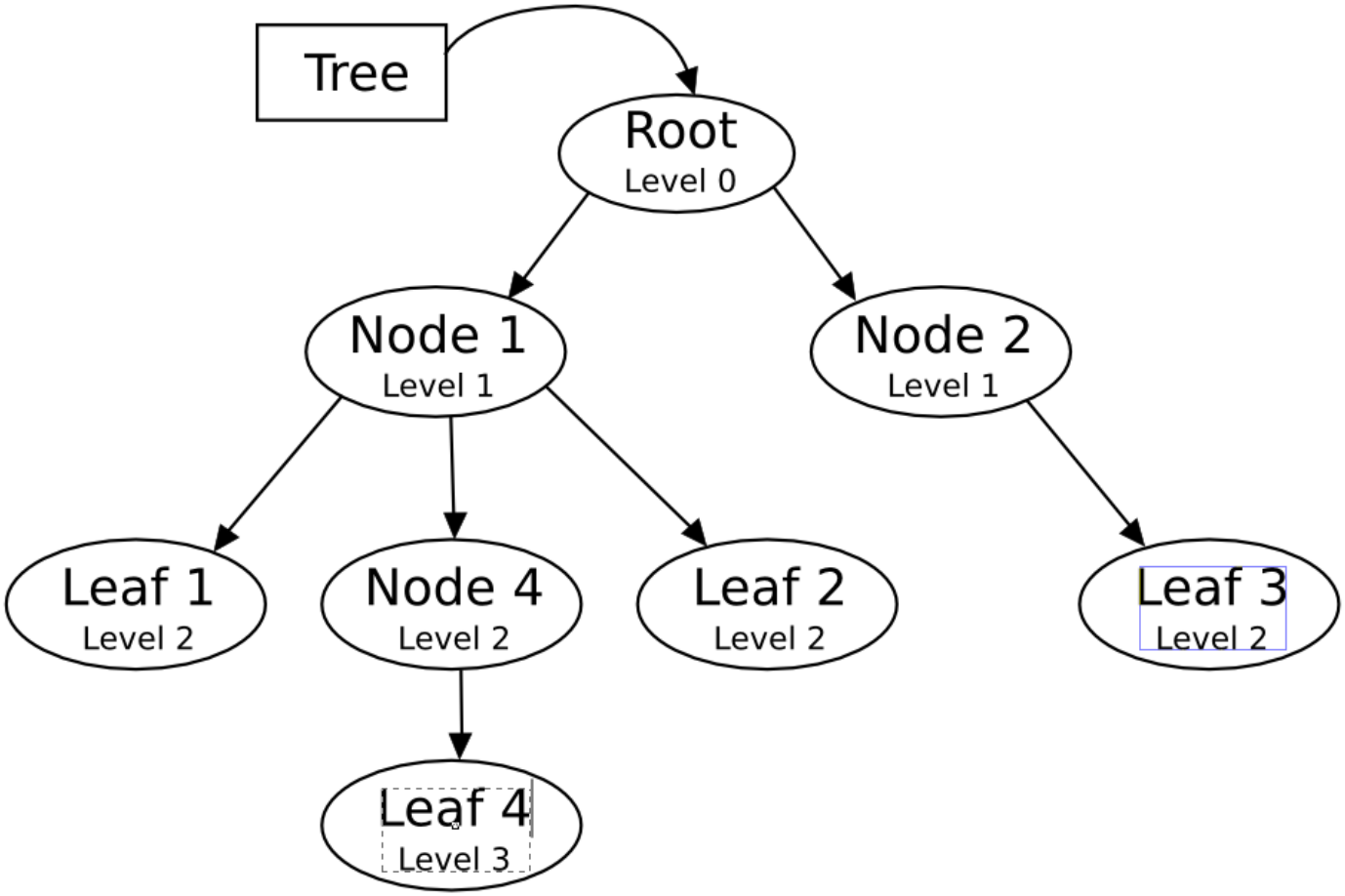
Northern Virginia Community College

Trees

Cody Narber, M.S.
October 28, 2017

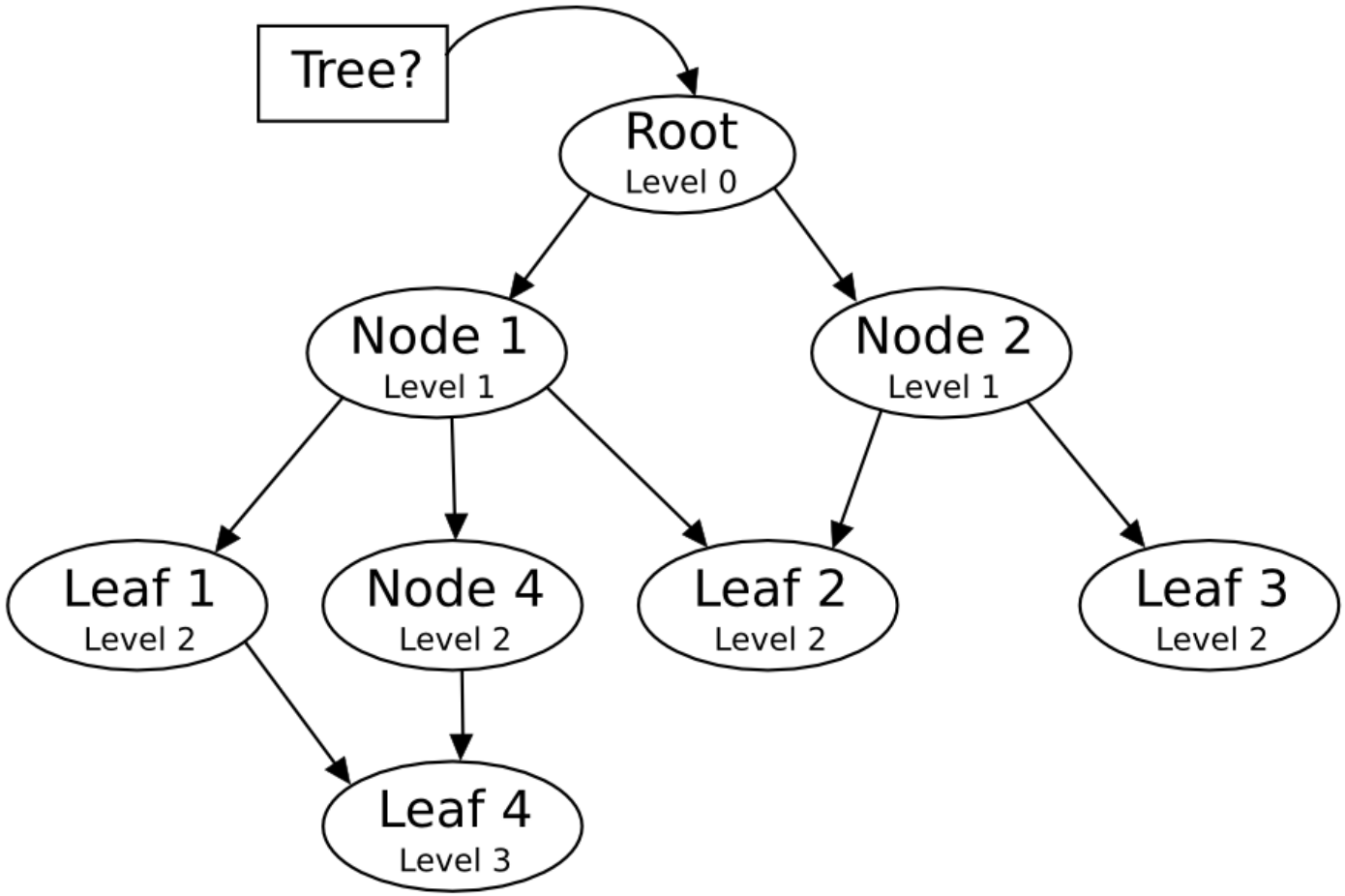
Data Structures - Tree

Tree: A structure (graph) with a unique starting node (**root**), in which each node is capable of having multiple child nodes such that there is a ***unique*** path exists from the root to every other node.



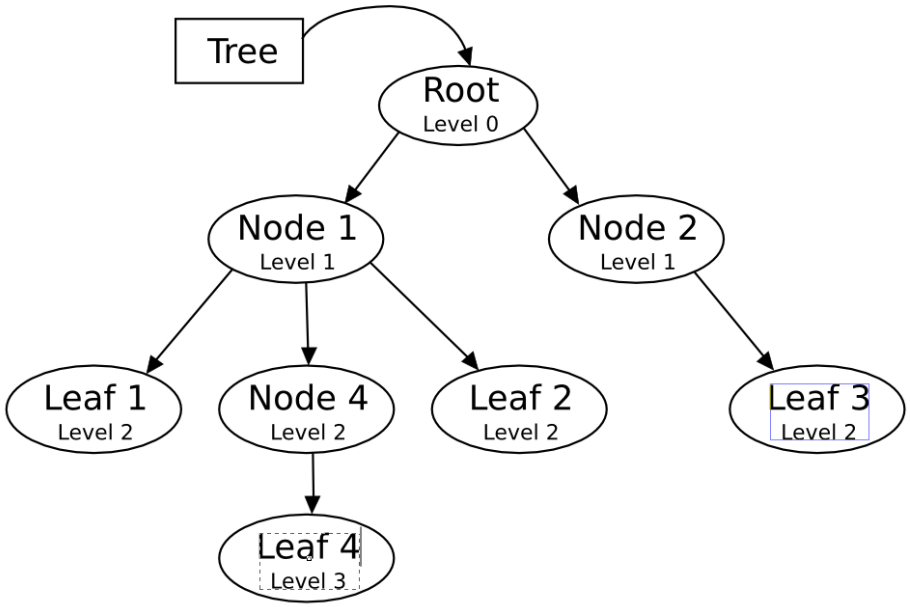
Data Structures - Tree?

The following is **NOT** a tree because for both nodes: **Leaf 2** and **Leaf 4** There exists multiple paths from the root.



Tree Structure

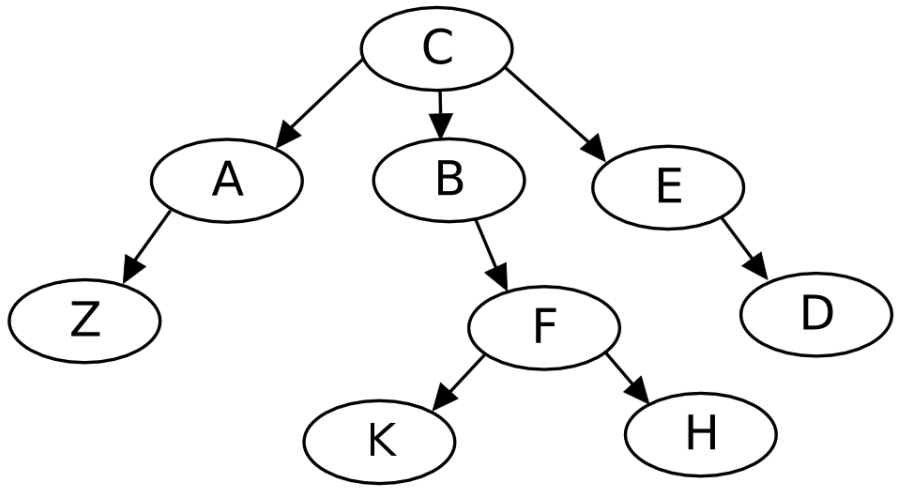
- **Root** - The Top-most node with no parents
- **Leaf** - A bottom node that has NO children
- **Level** - The distance from the tree's root



- **Descendant** - A node where there exists a path from the node to the descendant. (ALL nodes are descendants of the root)
- **Ancestor** - A node where there exists a path from the ancestor to the node. (The root is an ancestor to all other nodes)

Tree Structure

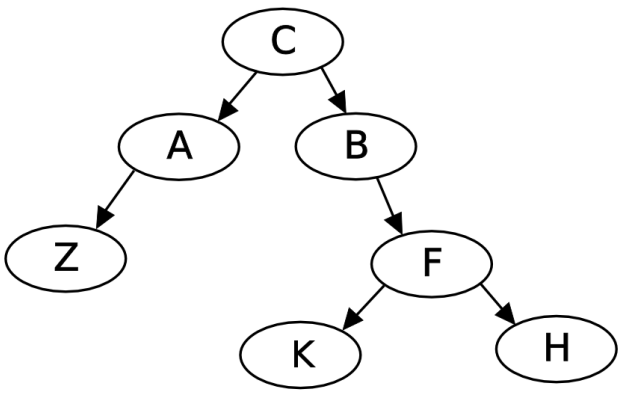
- How many Levels is the tree?
- How many Ancestors does F Have?
- How many Descendants does B Have?
- How many Children does A Have?
- How many Leaves are in the tree?



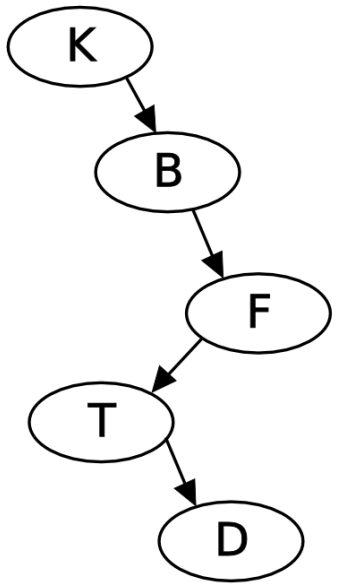
Submit answer as comma-separated: #,#,#,#,#

Binary Tree

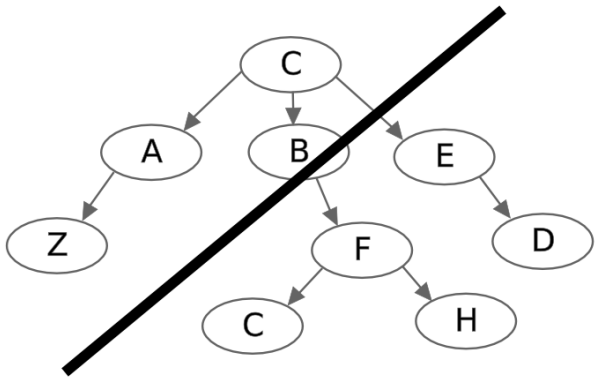
Binary Tree: A tree structure in which every single node can have **AT MOST** 2 children (usually denoted as left child and right child). Some examples of trees:



Binary Tree 1



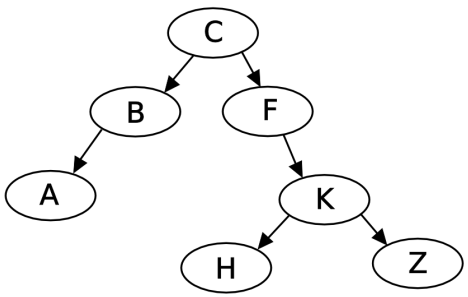
Binary Tree 2



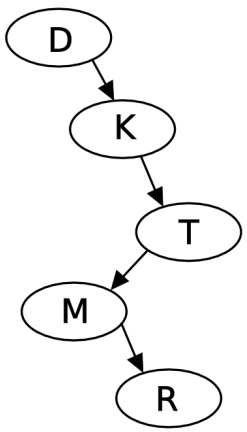
Not a Binary Tree

Binary Search Tree

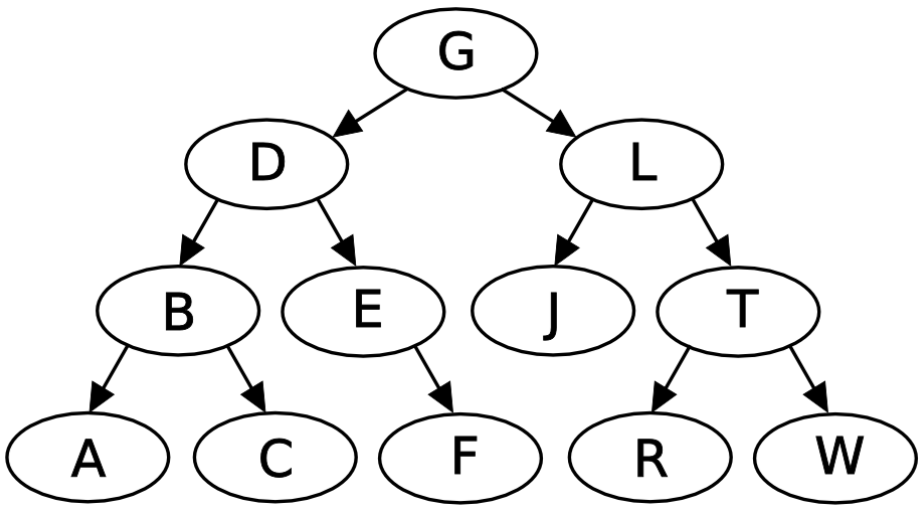
Binary Search Tree (BST): A binary tree in which nodes are organized such that every left child is less than it's parent and every right child is greater than it's parent:



BST 1



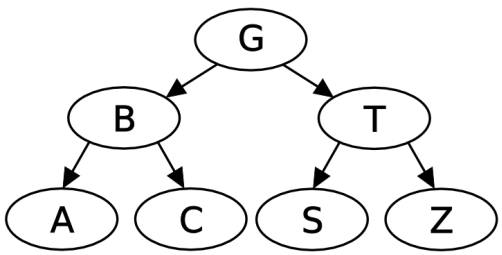
BST 2



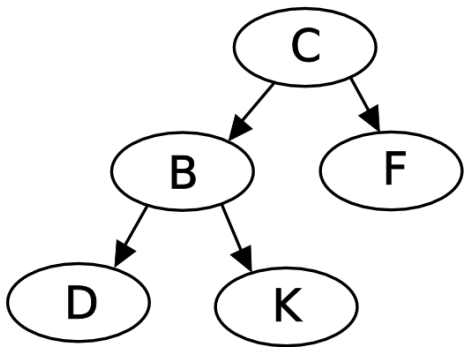
BST 3

Binary Search Tree Terms

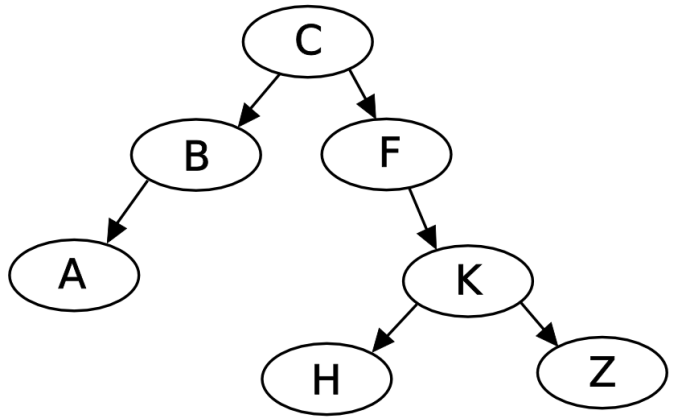
- **Full** – A tree that has **(A)**The leaves are all on the same level **(B)** All other nodes have 2 children.
- **Complete** – A tree where: **(A)** The subtree excluding the last level is full **(B)** The leaves on the last level must fill the left-most nodes first (left-to-right)
- *Note: A Full tree by definition is also complete.*



Full and Complete



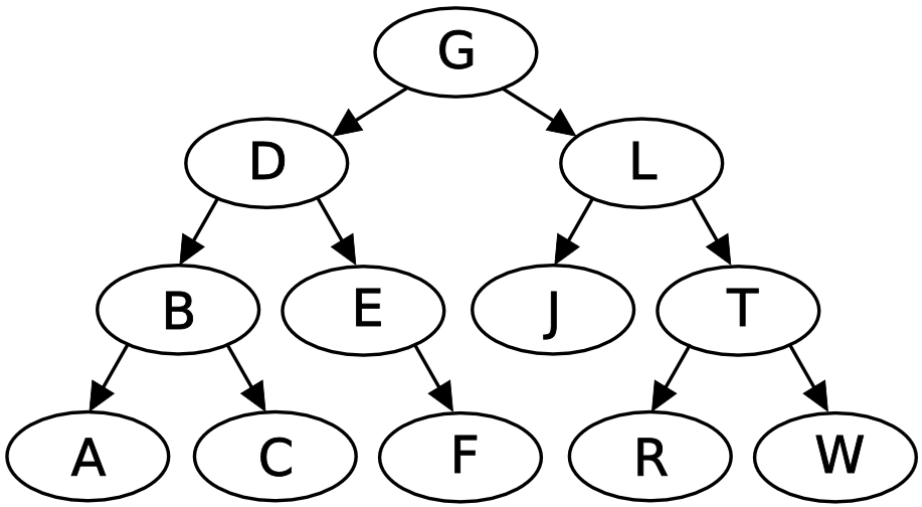
Complete



Neither

Binary Search Tree Traversals

- **Preorder Traversal** – Visit the root, visit the left subtree, and visit the right subtree
- **Inorder Traversal** – Visit the left subtree, visit the root, and visit the right subtree
- **Postorder Traversal** – Visit the left subtree, visit the right subtree, and visit the root

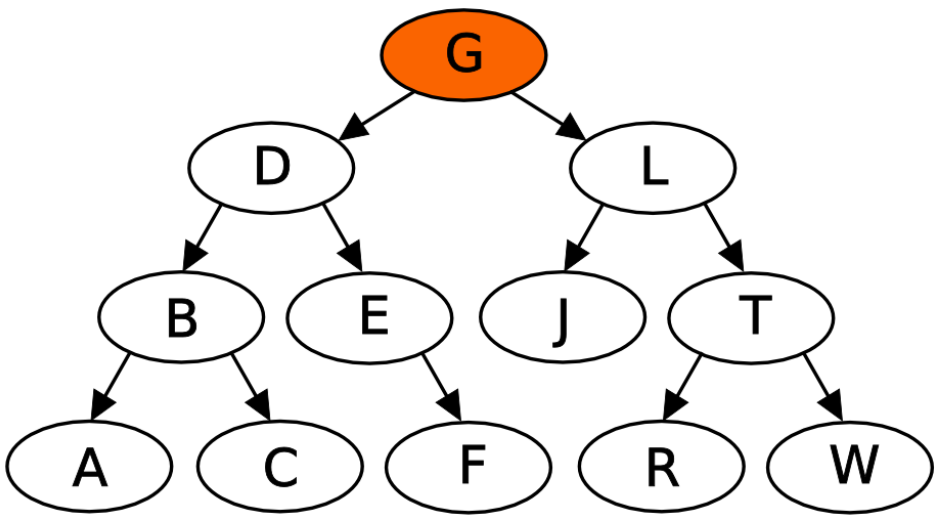


Binary Search Tree Traversals

Preorder Traversal –

- ***Visit the root/current node***
- Visit the left subtree
- Visit the right subtree

OUTPUT: G

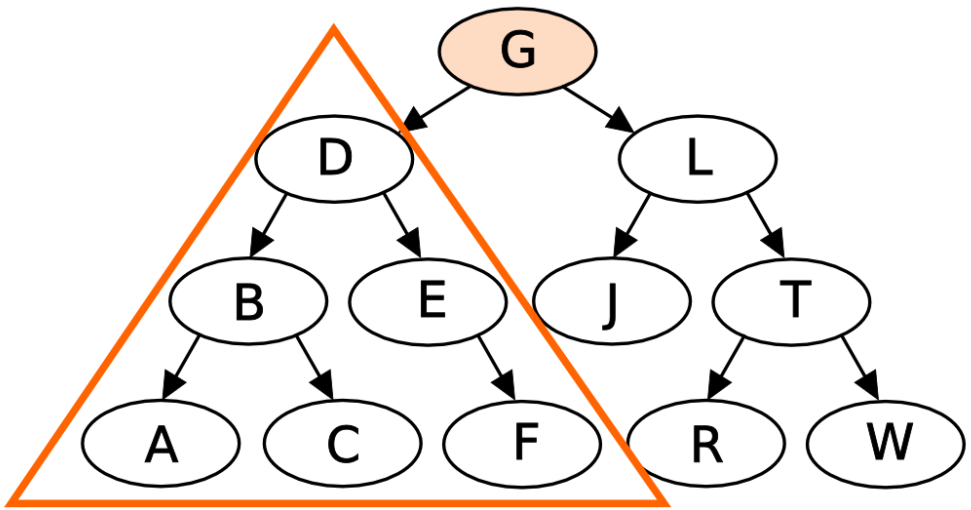


Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- ***Visit the left subtree***
- Visit the right subtree

OUTPUT: G

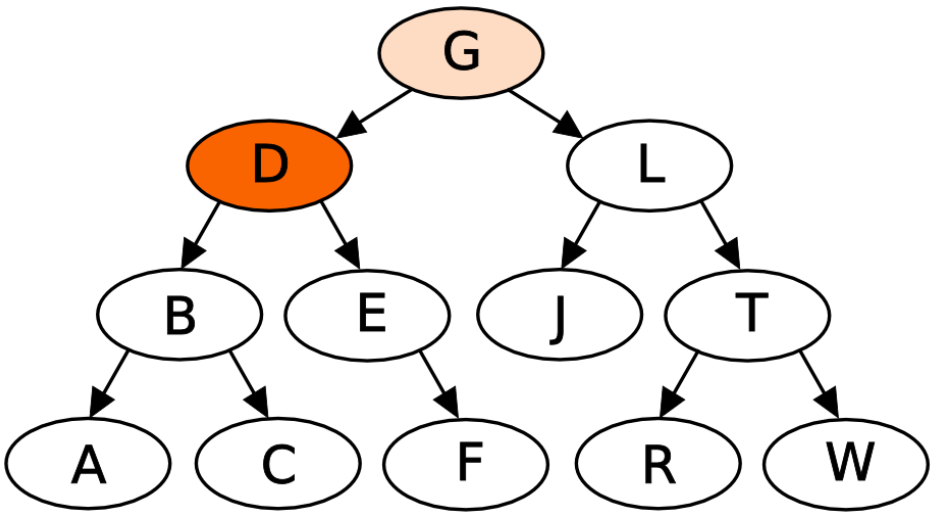


Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- ***Visit the left subtree***
  - ***Visit the root/current node***
  - Visit the left subtree
  - Visit the right subtree
- Visit the right subtree

OUTPUT: G D

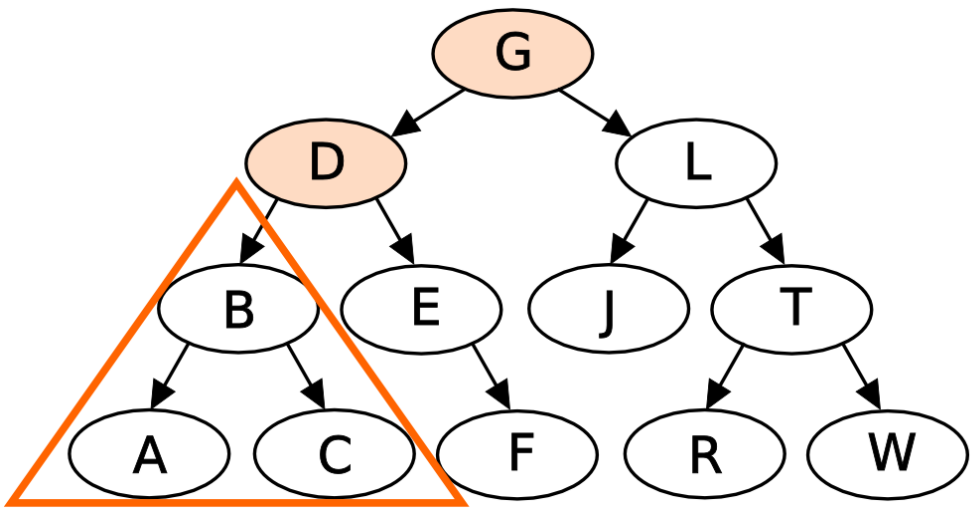


Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- ***Visit the left subtree***
  - Visit the root/current node
  - ***Visit the left subtree***
  - Visit the right subtree
- Visit the right subtree

OUTPUT: G D

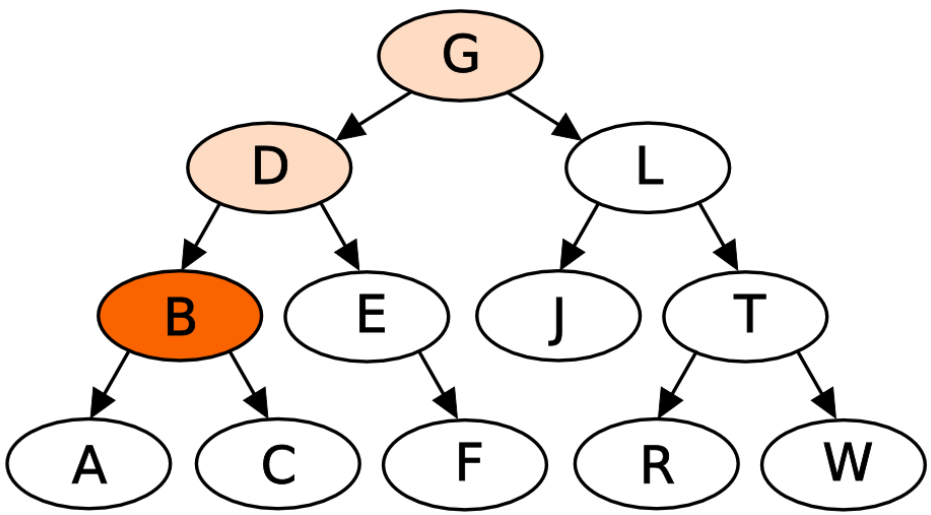


Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- ***Visit the left subtree***
  - Visit the root/current node
  - ***Visit the left subtree***
    - ...
  - Visit the right subtree
- Visit the right subtree

OUTPUT: G D B

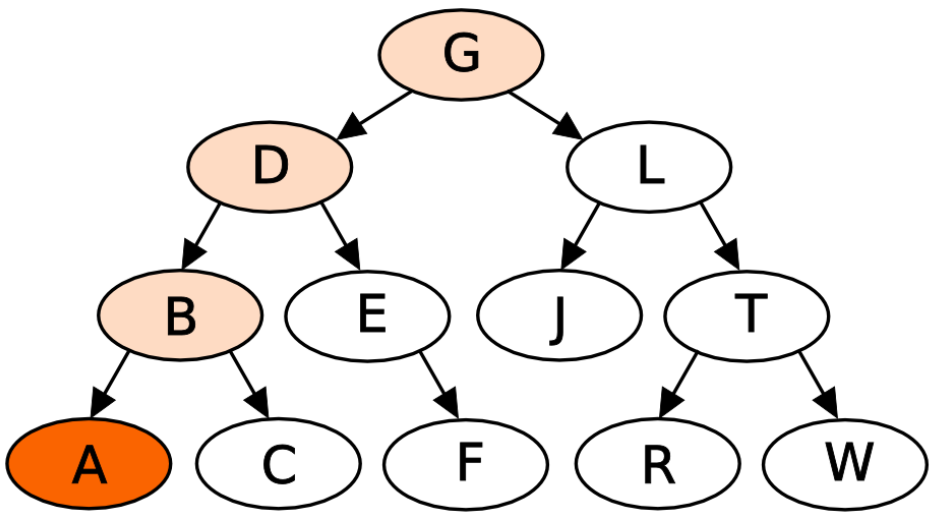


Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- ***Visit the left subtree***
  - Visit the root/current node
  - ***Visit the left subtree***
    - ...
  - Visit the right subtree
- Visit the right subtree

OUTPUT: G D B A

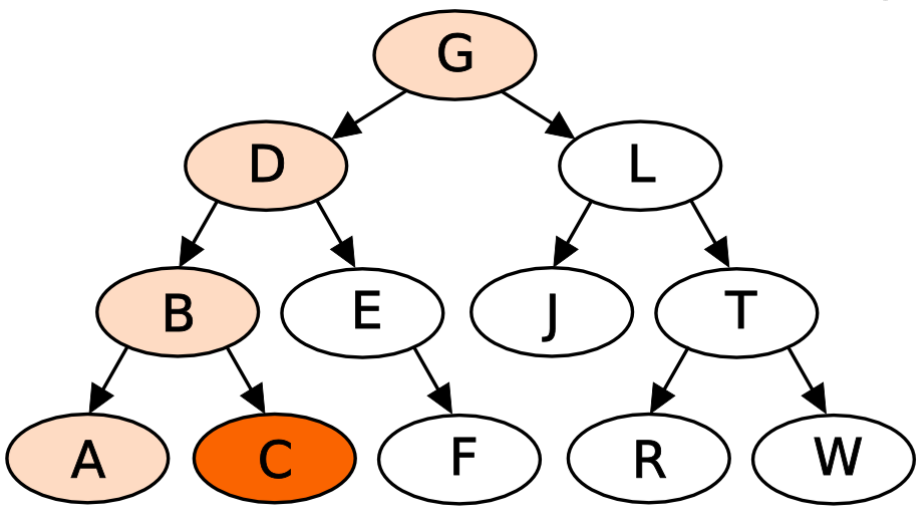


Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- ***Visit the left subtree***
  - Visit the root/current node
  - ***Visit the left subtree***
    - ...
  - Visit the right subtree
- Visit the right subtree

OUTPUT: G D B A C

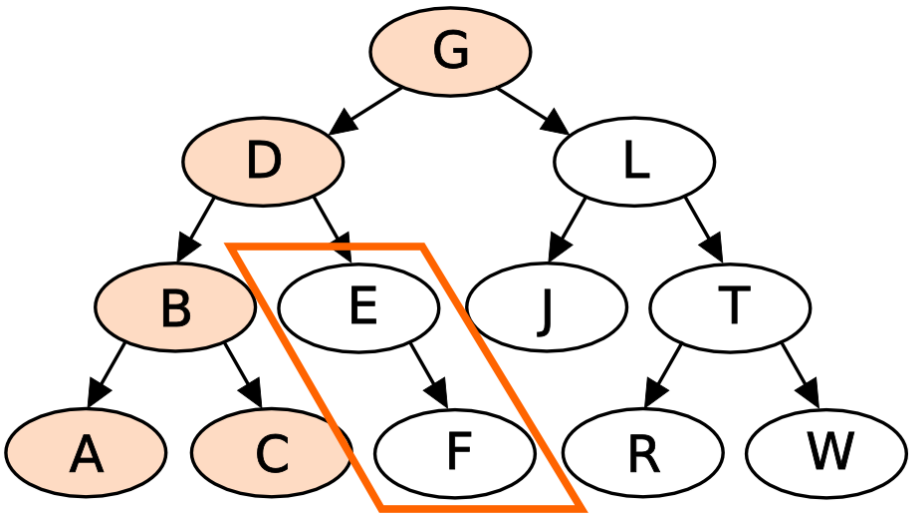


Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- ***Visit the left subtree***
  - Visit the root/current node
  - Visit the left subtree
  - ***Visit the right subtree***
- Visit the right subtree

OUTPUT: G D B A C

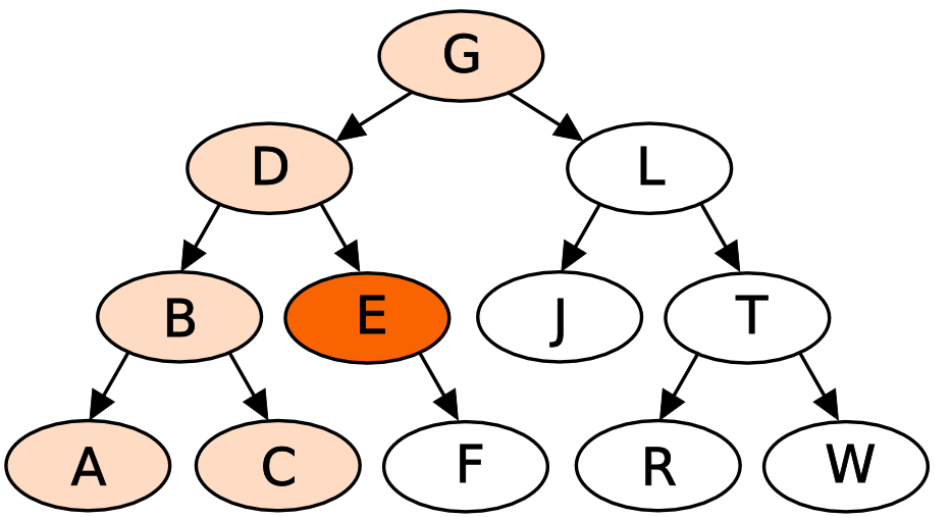


Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- ***Visit the left subtree***
  - Visit the root/current node
  - Visit the left subtree
  - ***Visit the right subtree***
    - ...
- Visit the right subtree

OUTPUT: G D B A C E

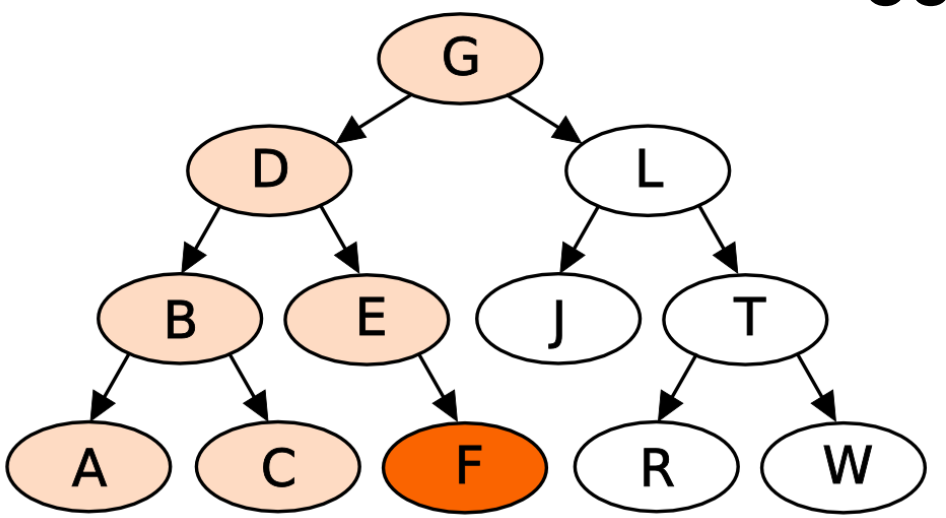


Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- ***Visit the left subtree***
  - Visit the root/current node
  - Visit the left subtree
  - ***Visit the right subtree***
    - ...
- Visit the right subtree

OUTPUT: G D B A C E F

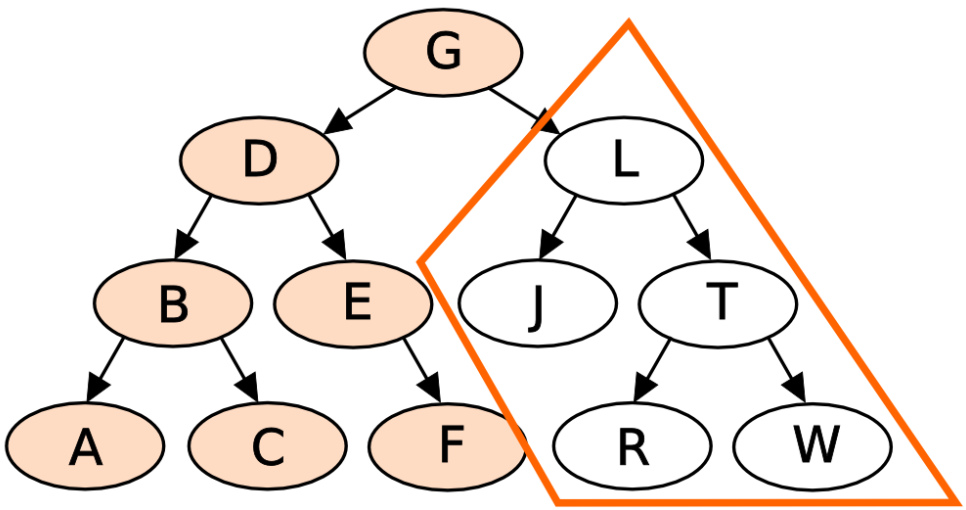


Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- Visit the left subtree
- ***Visit the right subtree***

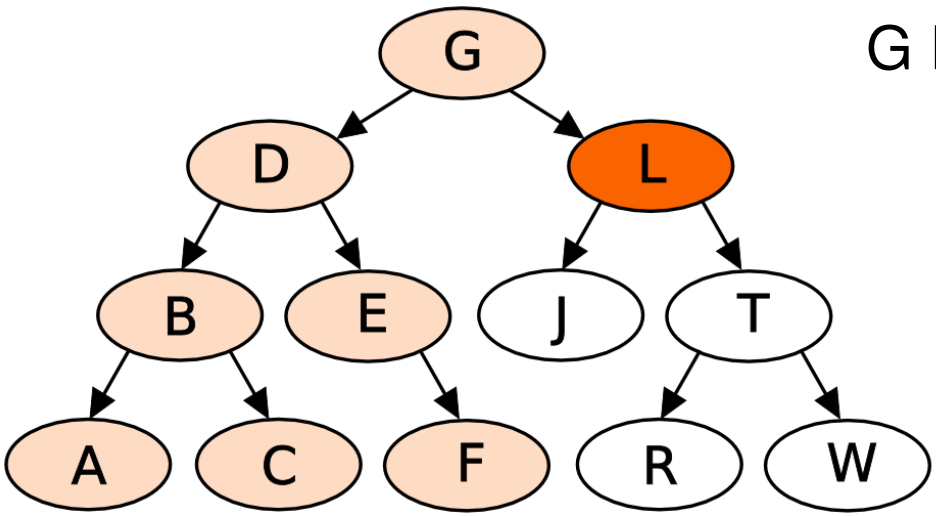
OUTPUT: G D B A C E F



Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- Visit the left subtree
- ***Visit the right subtree***
  - ***Visit the root/current node***
  - Visit the left subtree
  - Visit the right subtree

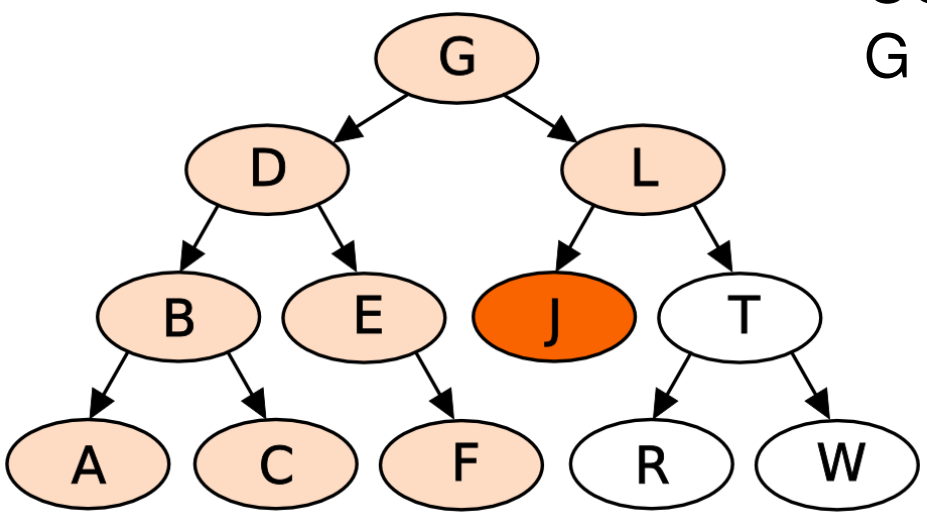


OUTPUT:
G D B A C E F L

Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- Visit the left subtree
- ***Visit the right subtree***
  - Visit the root/current node
  - ***Visit the left subtree...***
  - Visit the right subtree

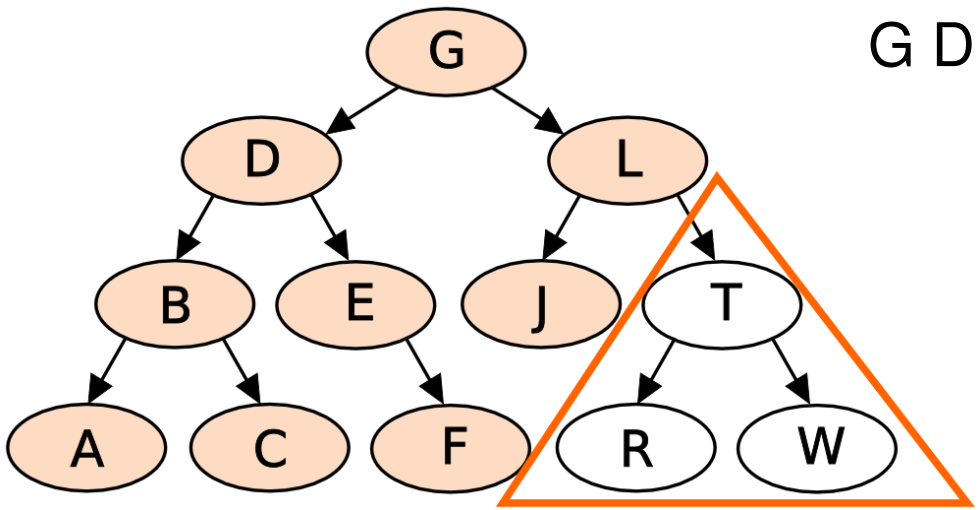


OUTPUT:
G D B A C E F L J

Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- Visit the left subtree
- ***Visit the right subtree***
  - Visit the root/current node
  - Visit the left subtree
  - ***Visit the right subtree***



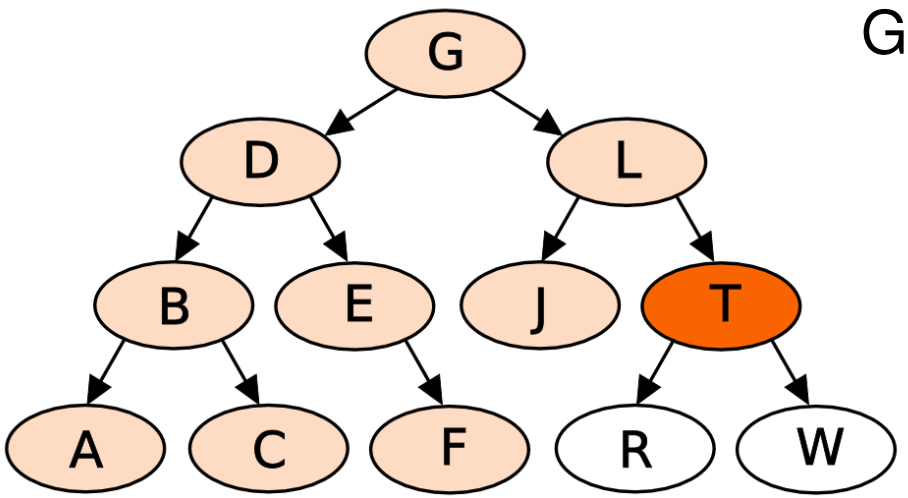
OUTPUT:

G D B A C E F L J

Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- Visit the left subtree
- ***Visit the right subtree***
  - Visit the root/current node
  - Visit the left subtree
  - ***Visit the right subtree***
    - ...



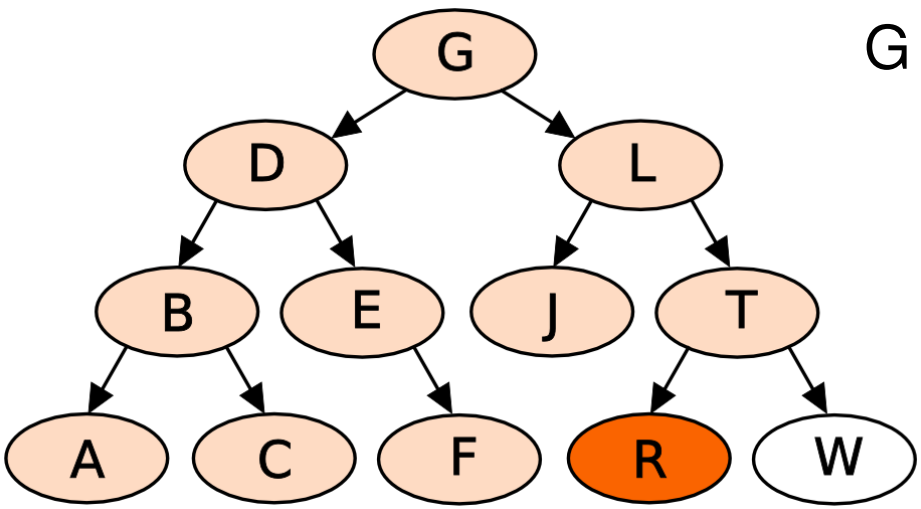
OUTPUT:

G D B A C E F L J T

Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- Visit the left subtree
- ***Visit the right subtree***
  - Visit the root/current node
  - Visit the left subtree
  - ***Visit the right subtree***
    - ...



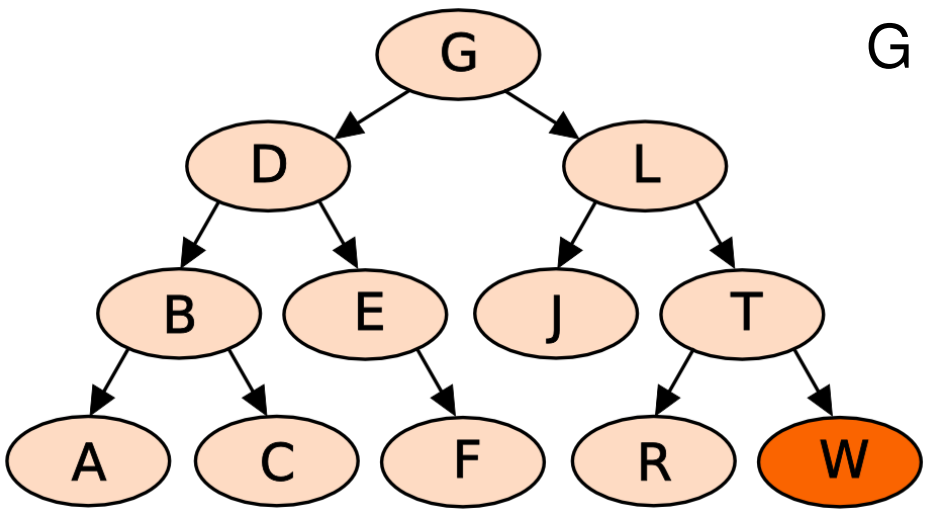
OUTPUT:

G D B A C E F L J T R

Binary Search Tree Traversals

Preorder Traversal –

- Visit the root/current node
- Visit the left subtree
- ***Visit the right subtree***
  - Visit the root/current node
  - Visit the left subtree
  - ***Visit the right subtree***
    - ...



OUTPUT:

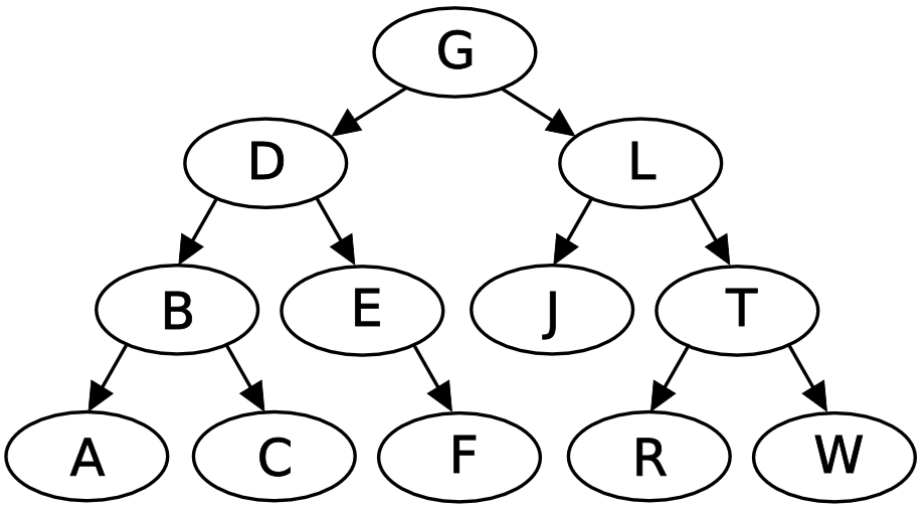
G D B A C E F L J T R W

Binary Search Tree Traversals

Inorder Traversal –

- Visit the left subtree
- Visit the root/current node
- Visit the right subtree

OUTPUT:

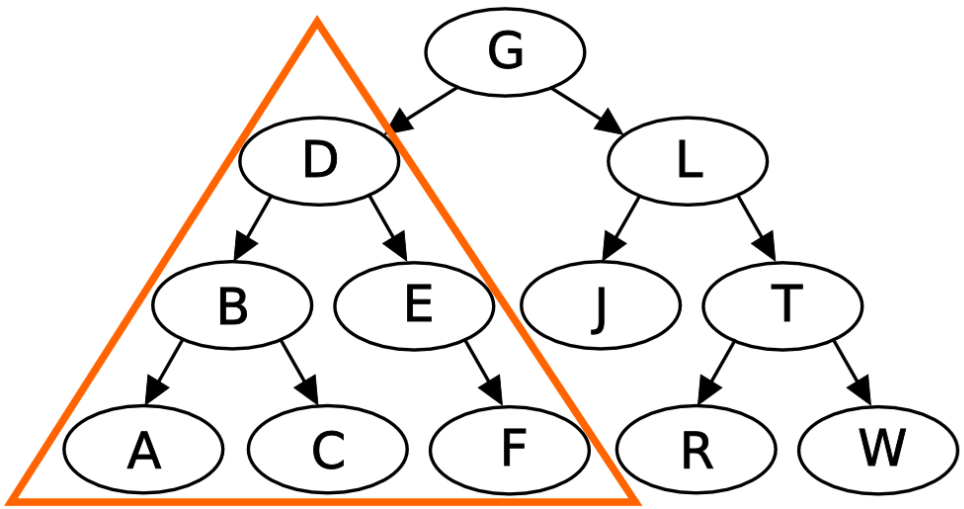


Binary Search Tree Traversals

Inorder Traversal –

- ***Visit the left subtree***
- Visit the root/current node
- Visit the right subtree

OUTPUT:

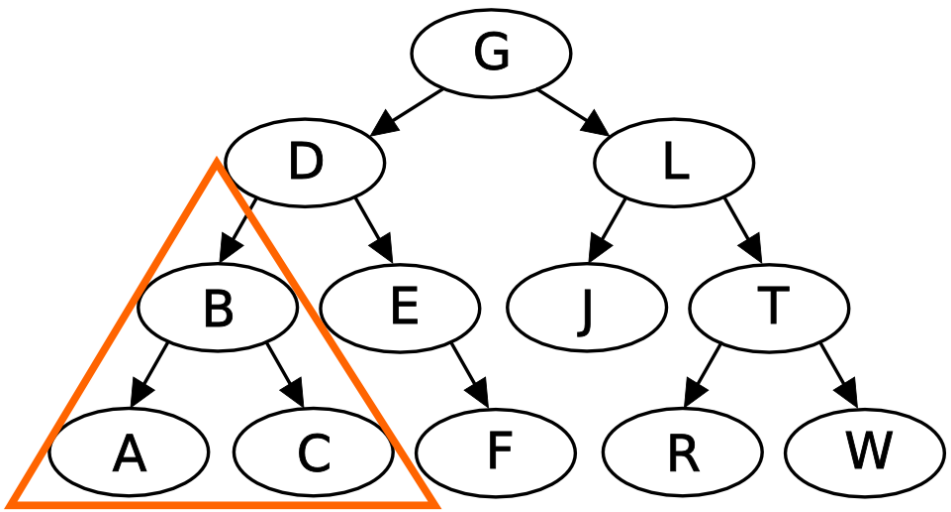


Binary Search Tree Traversals

Inorder Traversal –

- ***Visit the left subtree***
  - ***Visit the left subtree***
  - Visit the root/current node
  - Visit the right subtree
- Visit the root/current node
- Visit the right subtree

OUTPUT:

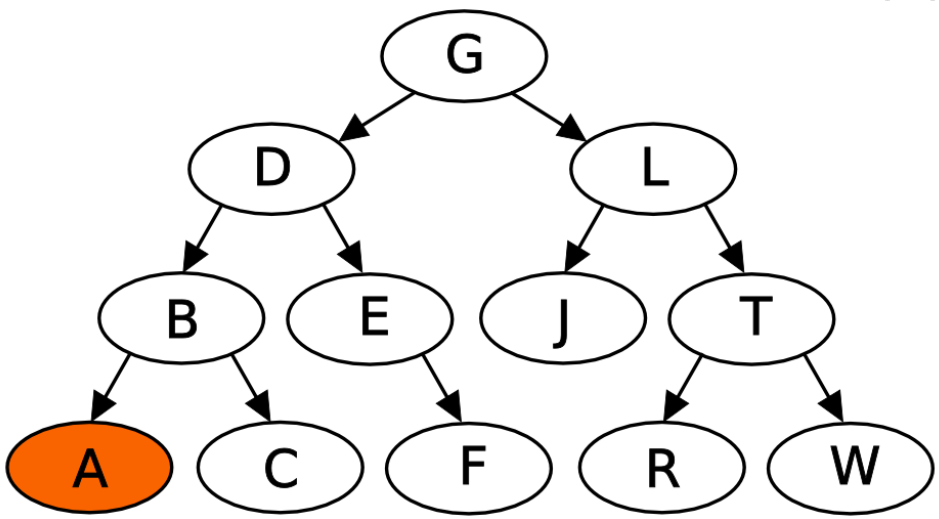


Binary Search Tree Traversals

Inorder Traversal –

- ***Visit the left subtree***
  - ***Visit the left subtree***
    - ...
  - Visit the root/current node
  - Visit the right subtree
- Visit the root/current node
- Visit the right subtree

OUTPUT: A

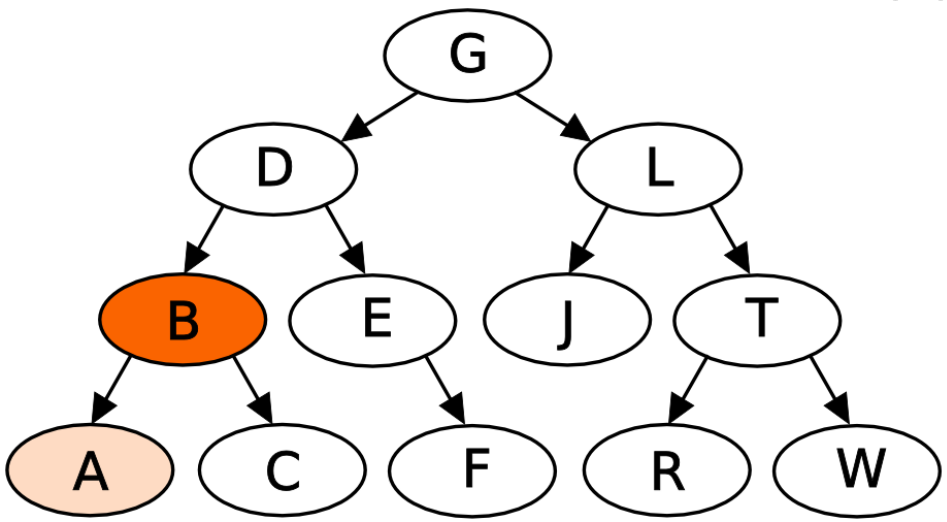


Binary Search Tree Traversals

Inorder Traversal –

- ***Visit the left subtree***
  - ***Visit the left subtree***
    - ...
  - Visit the root/current node
  - Visit the right subtree
- Visit the root/current node
- Visit the right subtree

OUTPUT: A B

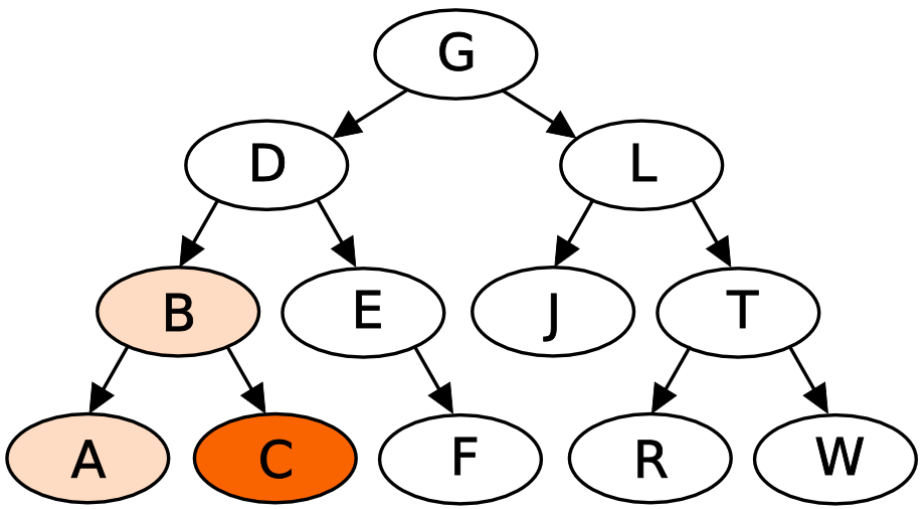


Binary Search Tree Traversals

Inorder Traversal –

- ***Visit the left subtree***
  - ***Visit the left subtree***
    - ...
  - Visit the root/current node
  - Visit the right subtree
- Visit the root/current node
- Visit the right subtree

OUTPUT: A B C

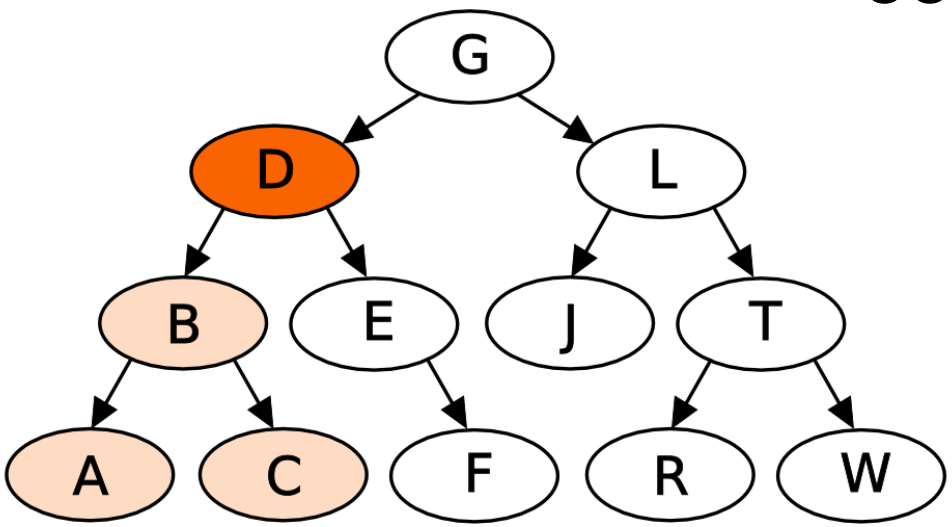


Binary Search Tree Traversals

Inorder Traversal –

- ***Visit the left subtree***
  - Visit the left subtree
  - ***Visit the root/current node***
  - Visit the right subtree
- Visit the root/current node
- Visit the right subtree

OUTPUT: A B C D

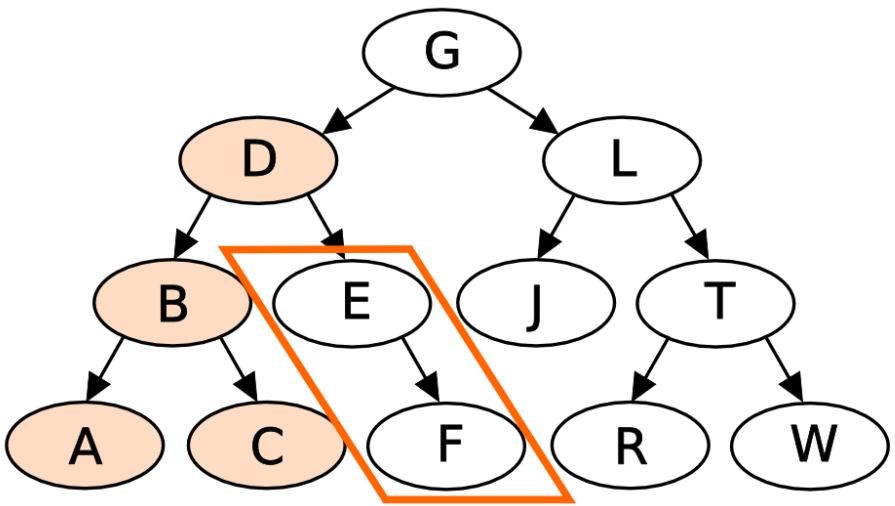


Binary Search Tree Traversals

Inorder Traversal –

- ***Visit the left subtree***
  - Visit the left subtree
  - Visit the root/current node
  - ***Visit the right subtree***
- Visit the root/current node
- Visit the right subtree

OUTPUT: A B C D

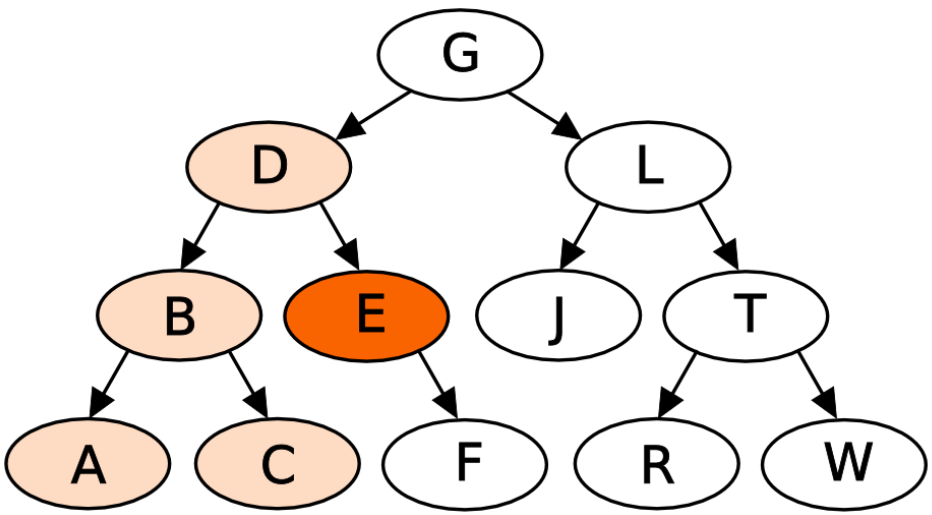


Binary Search Tree Traversals

Inorder Traversal –

- ***Visit the left subtree***
  - Visit the left subtree
  - Visit the root/current node
  - ***Visit the right subtree***
    - ...
- Visit the root/current node
- Visit the right subtree

OUTPUT: A B C D E

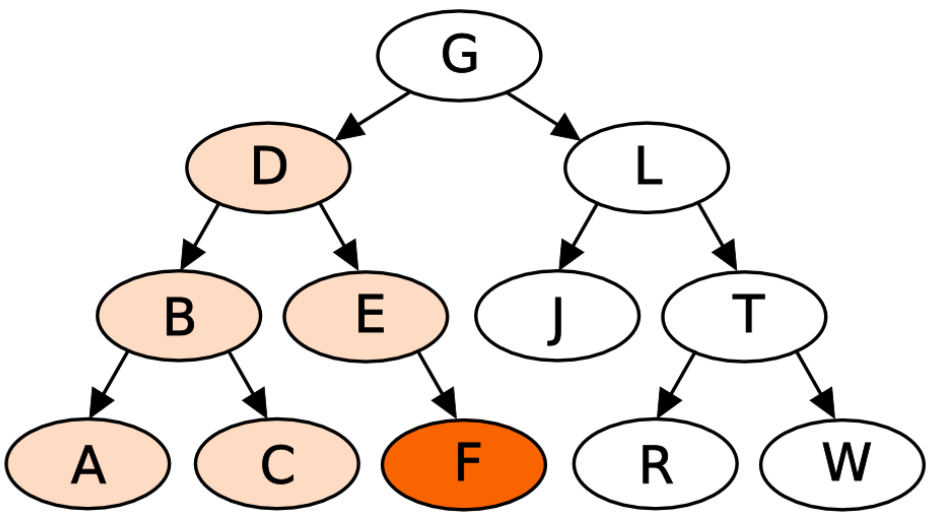


Binary Search Tree Traversals

Inorder Traversal –

- ***Visit the left subtree***
  - Visit the left subtree
  - Visit the root/current node
  - ***Visit the right subtree***
    - ...
- Visit the root/current node
- Visit the right subtree

OUTPUT: A B C D E F

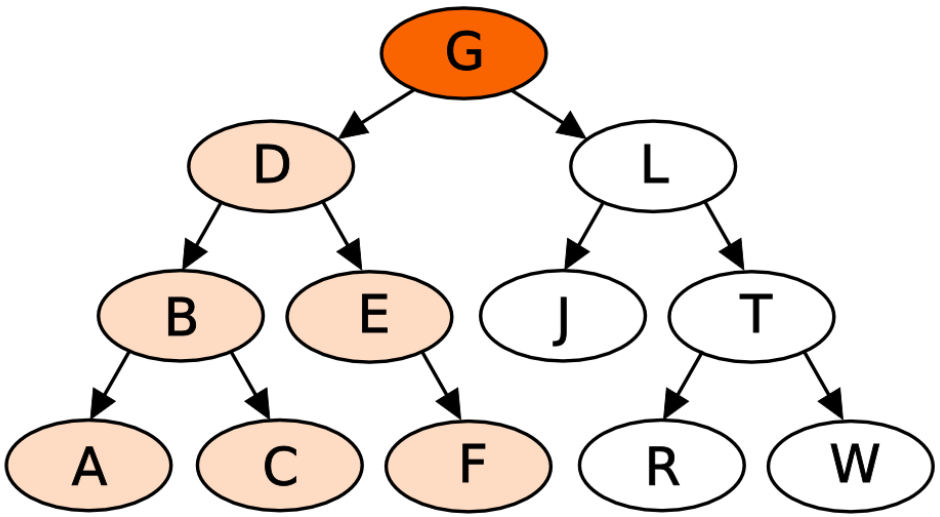


Binary Search Tree Traversals

Inorder Traversal –

- Visit the left subtree
- ***Visit the root/current node***
- Visit the right subtree

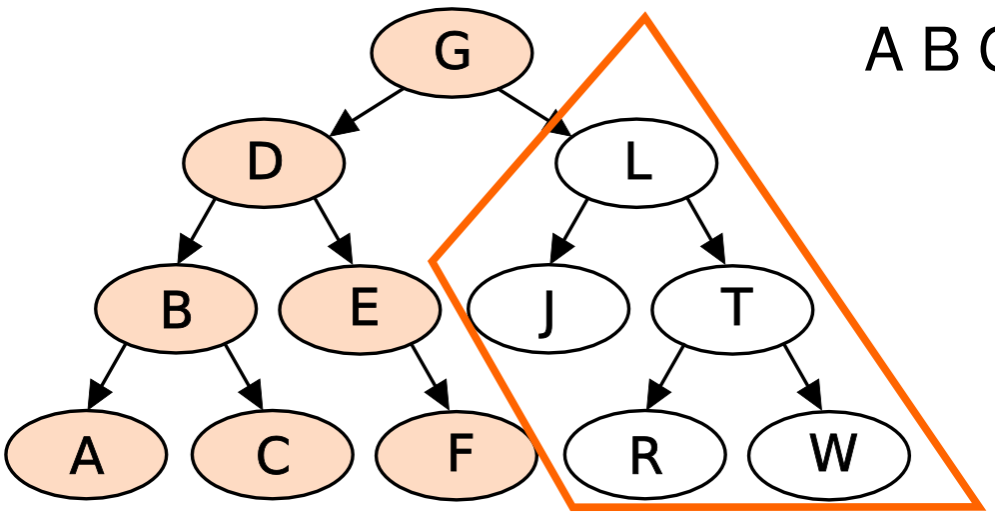
OUTPUT: A B C D E F G



Binary Search Tree Traversals

Inorder Traversal –

- Visit the left subtree
- Visit the root/current node
- ***Visit the right subtree***

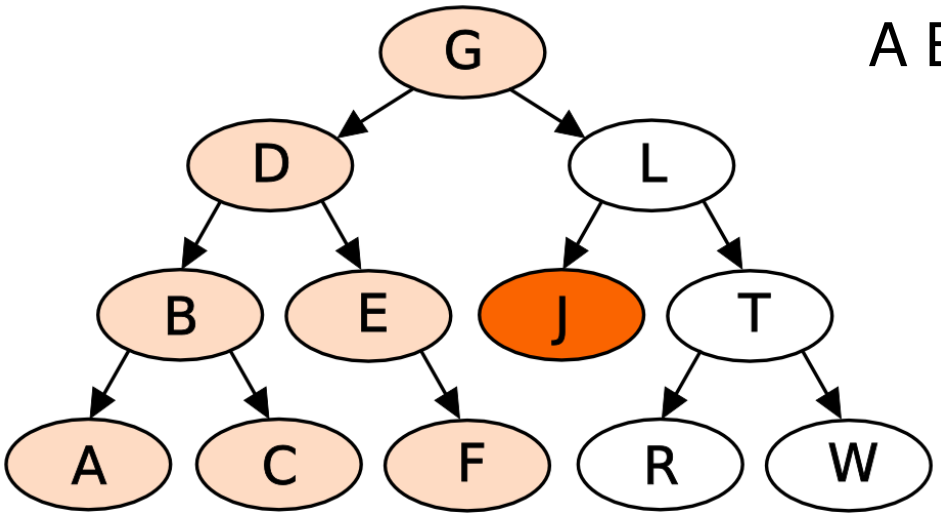


OUTPUT:
A B C D E F G

Binary Search Tree Traversals

Inorder Traversal –

- Visit the left subtree
- Visit the root/current node
- ***Visit the right subtree***
  - ***Visit the left subtree***
  - Visit the root/current node
  - Visit the right subtree

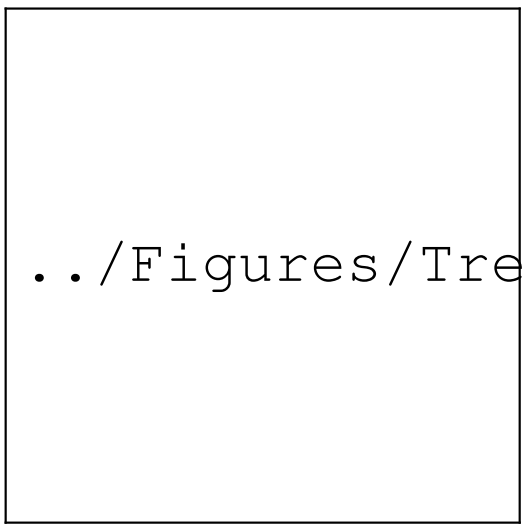


OUTPUT:
A B C D E F G J

Binary Search Tree Traversals

Inorder Traversal –

- Visit the left subtree
- Visit the root/current node
- ***Visit the right subtree***
  - Visit the left subtree
  - ***Visit the root/current node***
  - Visit the right subtree

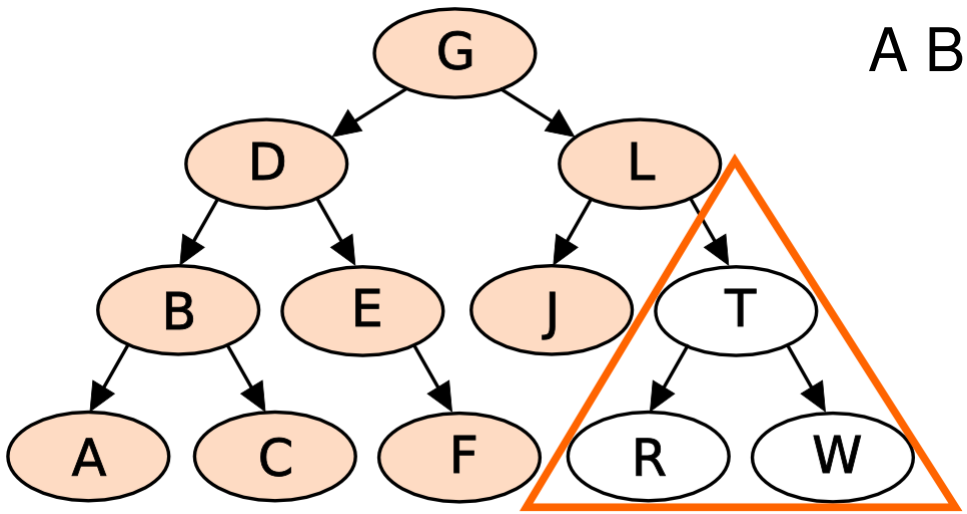


OUTPUT:
A B C D E F G J L

Binary Search Tree Traversals

Inorder Traversal –

- Visit the left subtree
- Visit the root/current node
- ***Visit the right subtree***
  - Visit the left subtree
  - Visit the root/current node
  - ***Visit the right subtree***



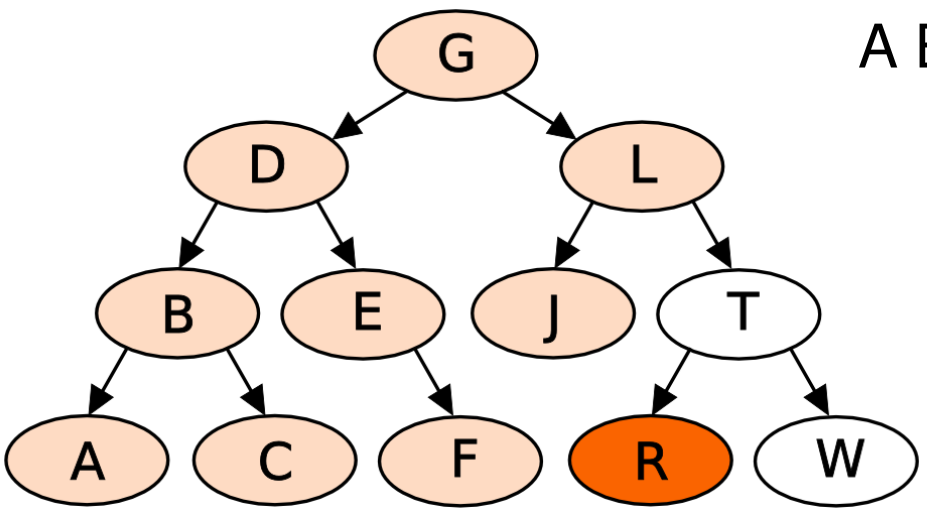
OUTPUT:

A B C D E F G J L

Binary Search Tree Traversals

Inorder Traversal –

- Visit the left subtree
- Visit the root/current node
- ***Visit the right subtree***
  - Visit the left subtree
  - Visit the root/current node
  - ***Visit the right subtree***
    - ...



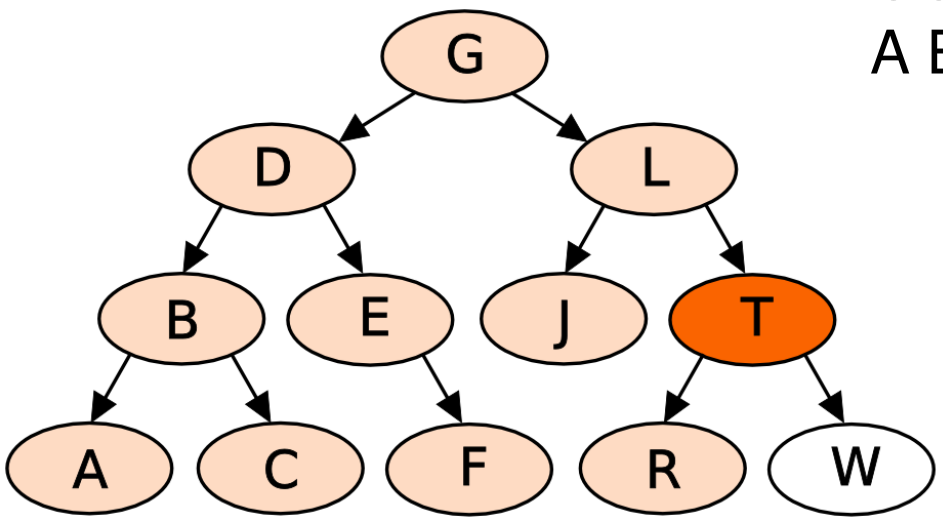
OUTPUT:

A B C D E F G J L R

Binary Search Tree Traversals

Inorder Traversal –

- Visit the left subtree
- Visit the root/current node
- ***Visit the right subtree***
  - Visit the left subtree
  - Visit the root/current node
  - ***Visit the right subtree***
    - ...



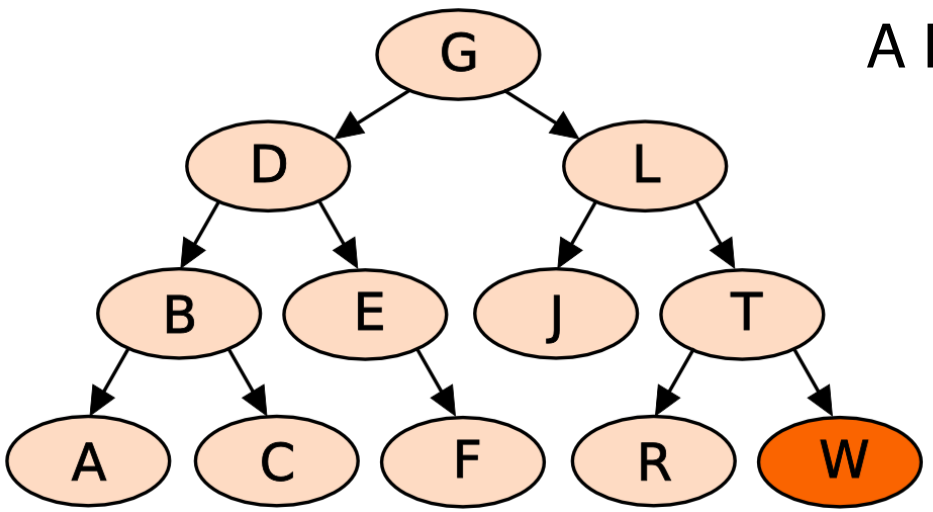
OUTPUT:

A B C D E F G J L R T

Binary Search Tree Traversals

Inorder Traversal –

- Visit the left subtree
- Visit the root/current node
- ***Visit the right subtree***
  - Visit the left subtree
  - Visit the root/current node
  - ***Visit the right subtree***
    - ...



OUTPUT:

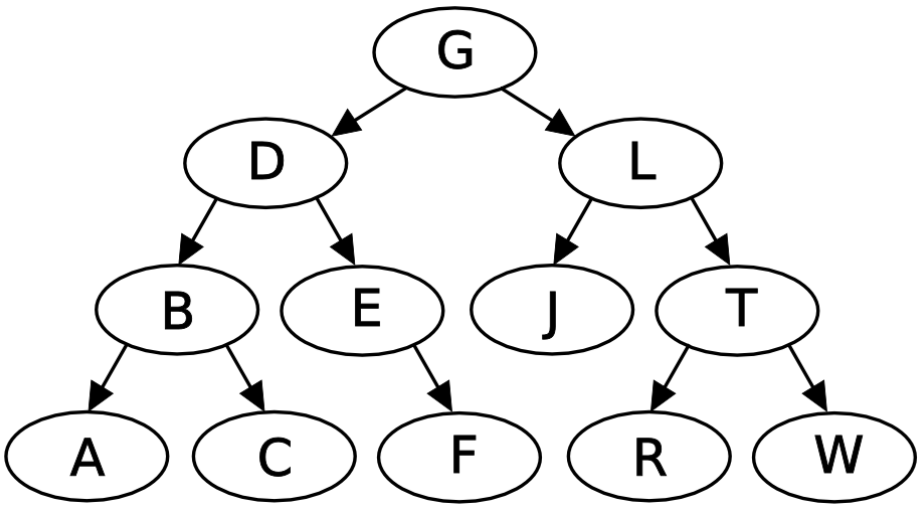
A B C D E F G J L R T W

Binary Search Tree Traversals

Postorder Traversal –

- Visit the left subtree
- Visit the right subtree
- Visit the root/current node

OUTPUT:

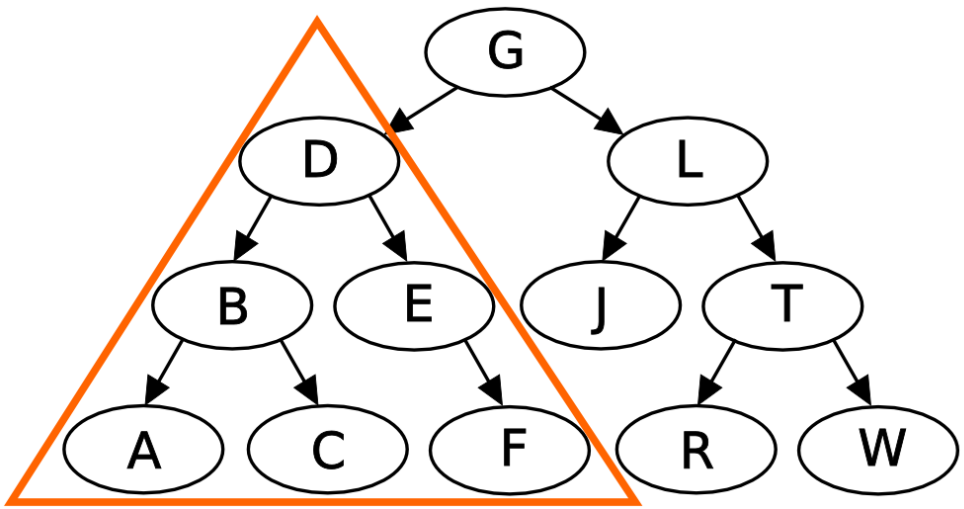


Binary Search Tree Traversals

Postorder Traversal –

- ***Visit the left subtree***
- Visit the right subtree
- Visit the root/current node

OUTPUT:

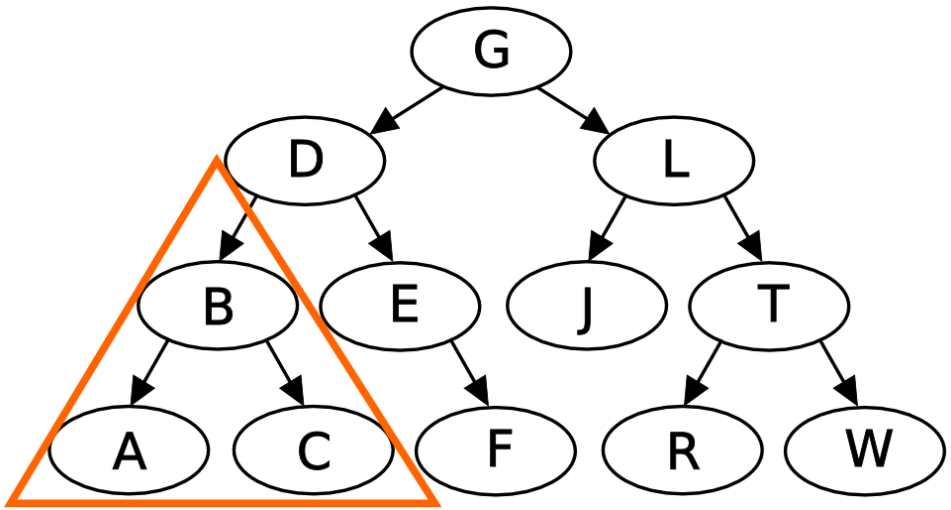


Binary Search Tree Traversals

Postorder Traversal –

- ***Visit the left subtree***
  - ***Visit the left subtree***
  - Visit the right subtree
  - Visit the root/current node
- Visit the right subtree
- Visit the root/current node

OUTPUT:

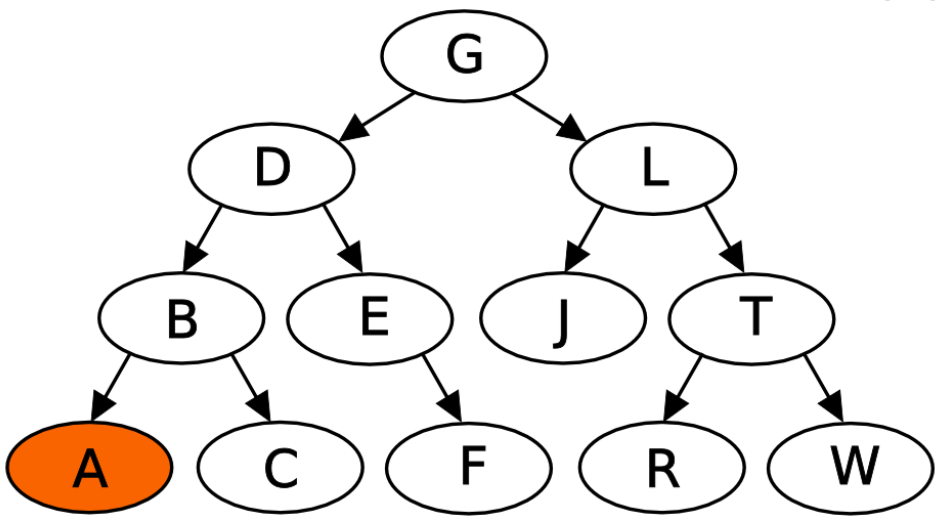


Binary Search Tree Traversals

Postorder Traversal –

- ***Visit the left subtree***
  - ***Visit the left subtree***
    - ...
  - Visit the right subtree
  - Visit the root/current node
- Visit the right subtree
- Visit the root/current node

OUTPUT: A

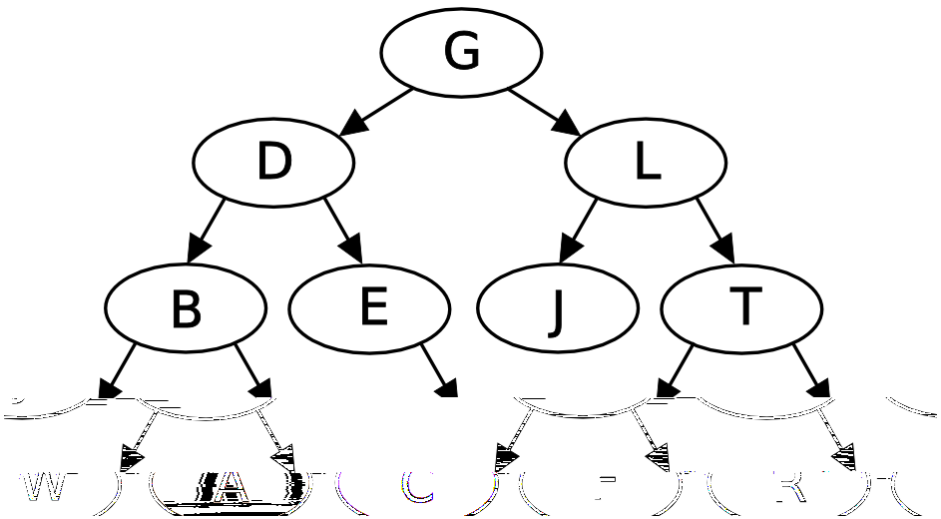


Binary Search Tree Traversals

Postorder Traversal –

- ***Visit the left subtree***
  - ***Visit the left subtree***
    - ...
  - Visit the right subtree
  - Visit the root/current node
- Visit the right subtree
- Visit the root/current node

OUTPUT: A C

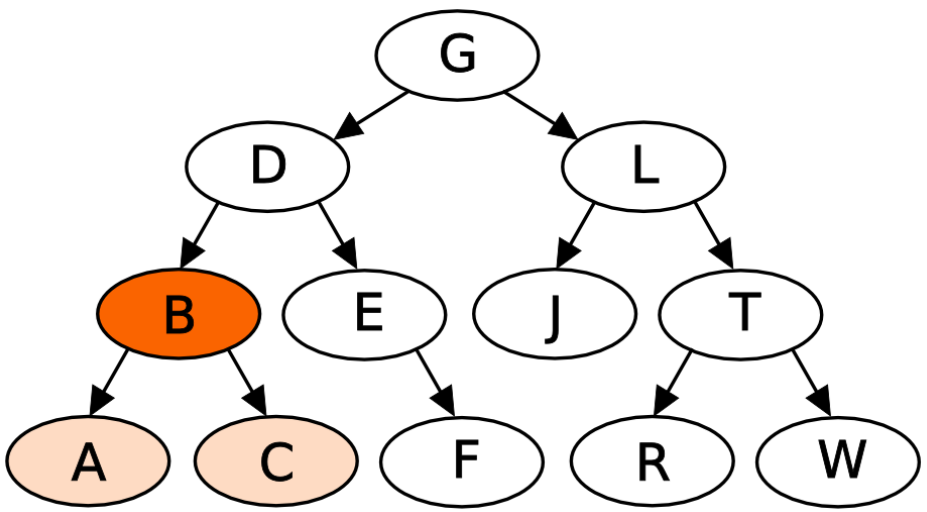


Binary Search Tree Traversals

Postorder Traversal –

- ***Visit the left subtree***
  - ***Visit the left subtree***
    - ...
  - Visit the right subtree
  - Visit the root/current node
- Visit the right subtree
- Visit the root/current node

OUTPUT: A C B

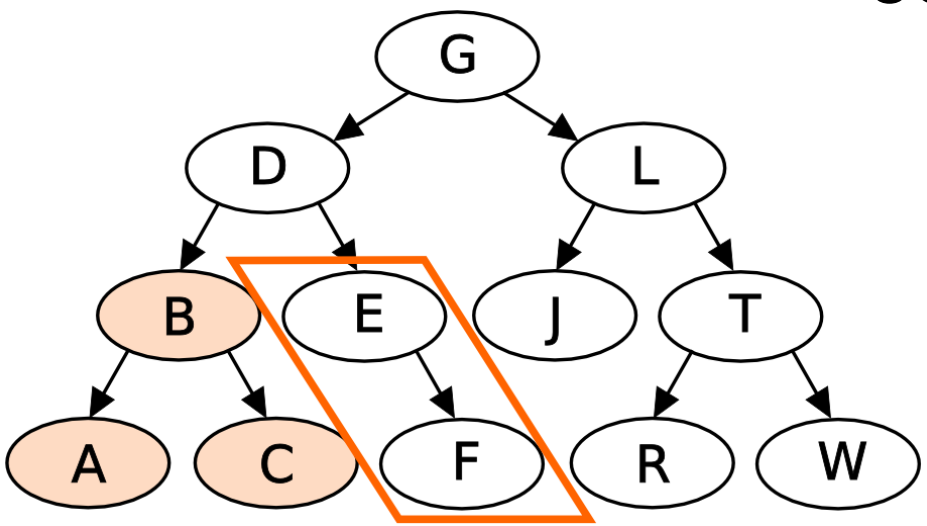


Binary Search Tree Traversals

Postorder Traversal –

- ***Visit the left subtree***
  - Visit the left subtree
  - ***Visit the right subtree***
  - Visit the root/current node
- Visit the right subtree
- Visit the root/current node

OUTPUT: A C B

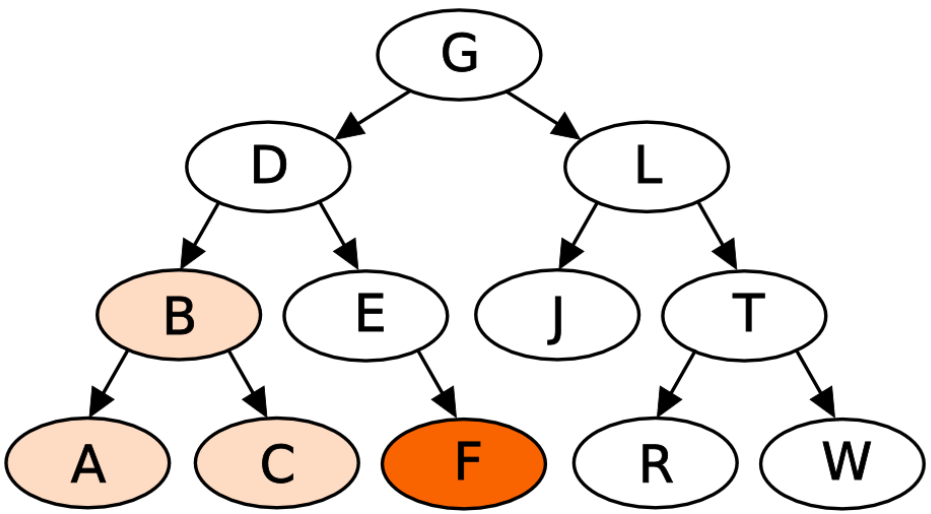


Binary Search Tree Traversals

Postorder Traversal –

- ***Visit the left subtree***
  - Visit the left subtree
  - ***Visit the right subtree***
    - ...
  - Visit the root/current node
- Visit the right subtree
- Visit the root/current node

OUTPUT: A C B F

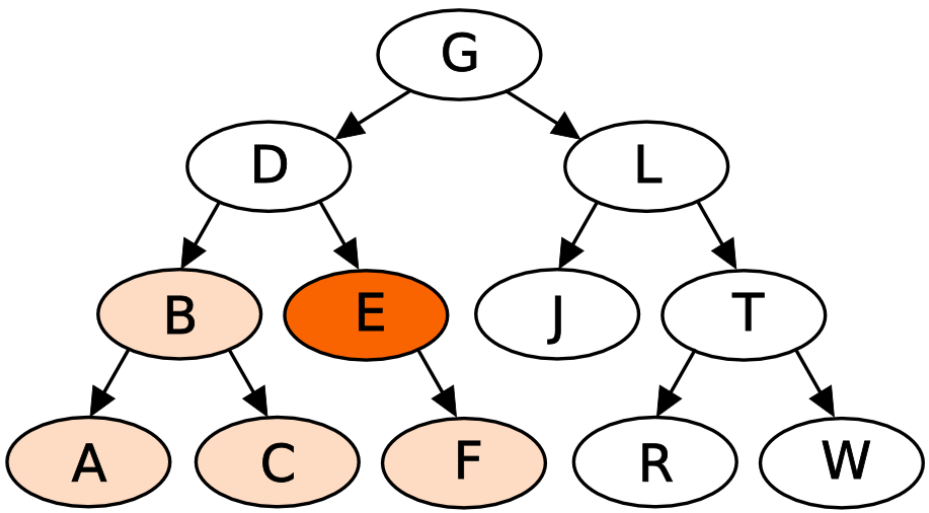


Binary Search Tree Traversals

Postorder Traversal –

- ***Visit the left subtree***
  - Visit the left subtree
  - ***Visit the right subtree***
    - ...
  - Visit the root/current node
- Visit the right subtree
- Visit the root/current node

OUTPUT: A C B F E

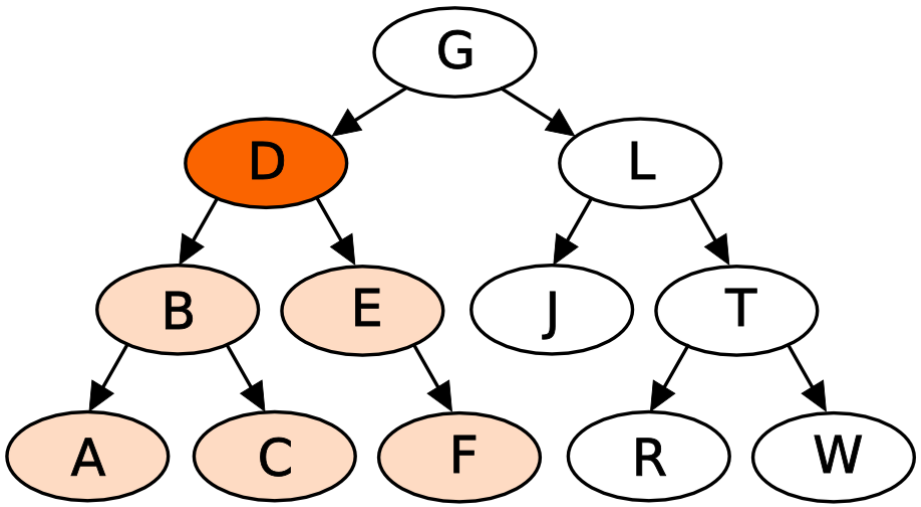


Binary Search Tree Traversals

Postorder Traversal –

- ***Visit the left subtree***
  - Visit the left subtree
  - Visit the right subtree
  - ***Visit the root/current node***
- Visit the right subtree
- Visit the root/current node

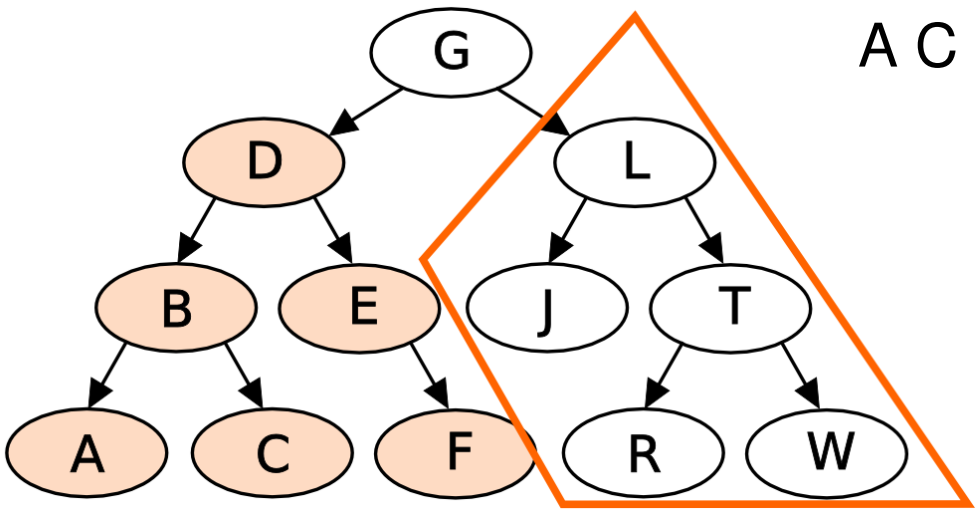
OUTPUT: A C B F E D



Binary Search Tree Traversals

Postorder Traversal –

- Visit the left subtree
- ***Visit the right subtree***
- Visit the root/current node

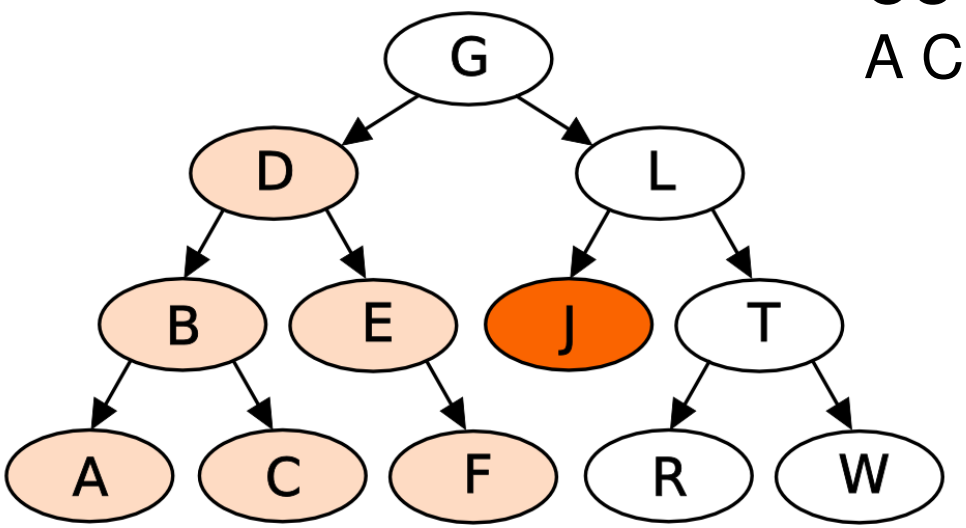


OUTPUT:
A C B F E D

Binary Search Tree Traversals

Postorder Traversal –

- Visit the left subtree
- ***Visit the right subtree***
  - ***Visit the left subtree***
  - Visit the right subtree
  - Visit the root/current node
- Visit the root/current node

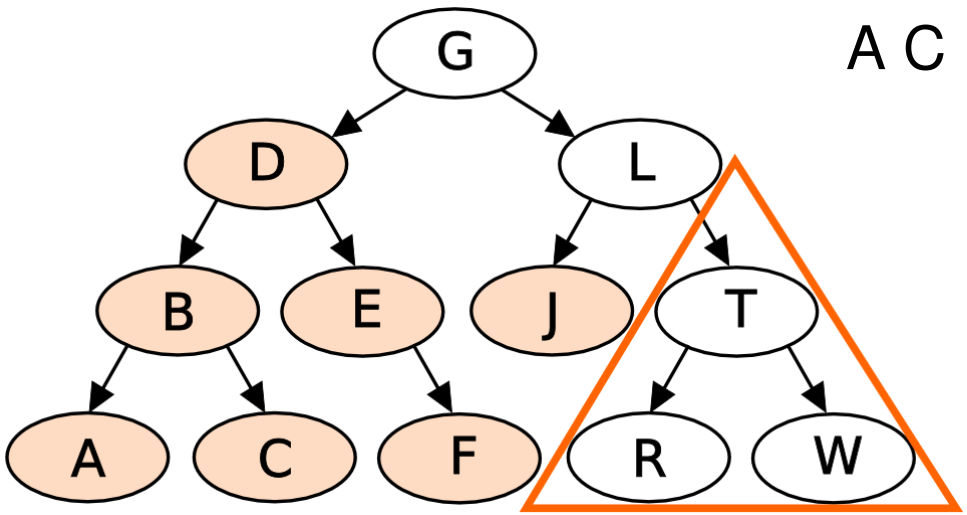


OUTPUT:
A C B F E D J

Binary Search Tree Traversals

Postorder Traversal –

- Visit the left subtree
- ***Visit the right subtree***
  - Visit the left subtree
  - ***Visit the right subtree***
  - Visit the root/current node
- Visit the root/current node

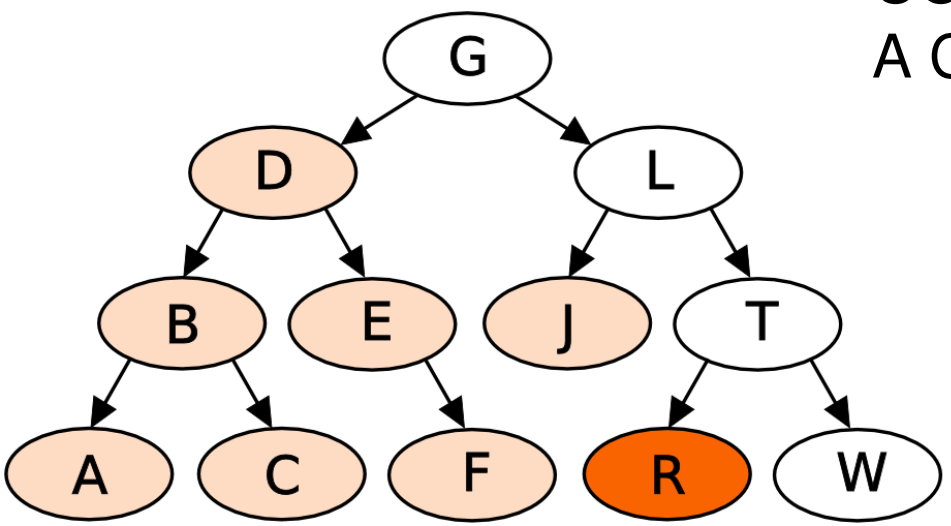


OUTPUT:
A C B F E D J

Binary Search Tree Traversals

Postorder Traversal –

- Visit the left subtree
- ***Visit the right subtree***
  - Visit the left subtree
  - ***Visit the right subtree***
    - ...
  - Visit the root/current node
- Visit the root/current node



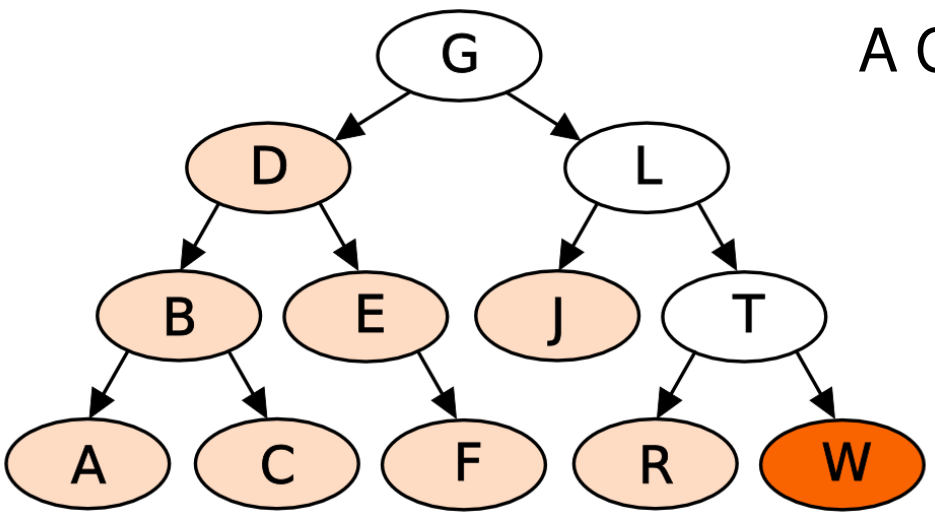
OUTPUT:

A C B F E D J R

Binary Search Tree Traversals

Postorder Traversal –

- Visit the left subtree
- ***Visit the right subtree***
  - Visit the left subtree
  - ***Visit the right subtree***
    - ...
  - Visit the root/current node
- Visit the root/current node



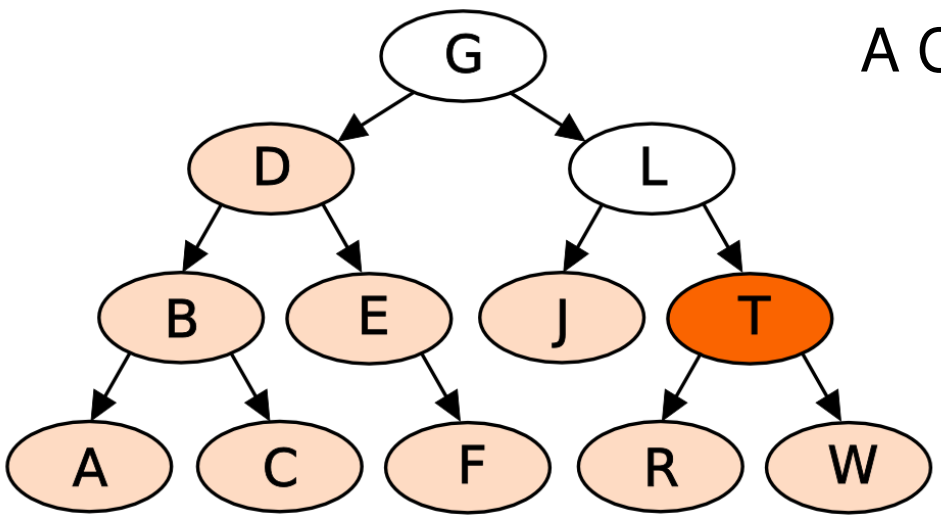
OUTPUT:

A C B F E D J R W

Binary Search Tree Traversals

Postorder Traversal –

- Visit the left subtree
- ***Visit the right subtree***
  - Visit the left subtree
  - ***Visit the right subtree***
    - ...
  - Visit the root/current node
- Visit the root/current node



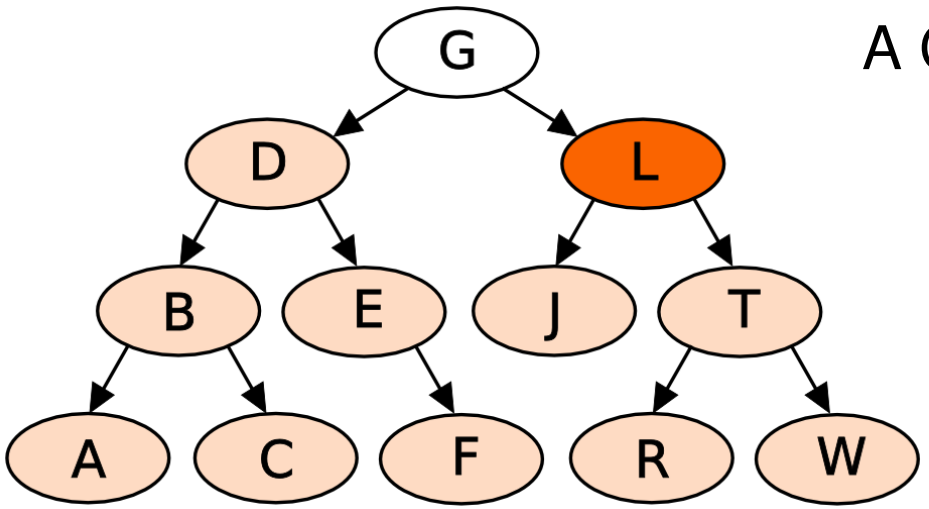
OUTPUT:

A C B F E D J R W T

Binary Search Tree Traversals

Postorder Traversal –

- Visit the left subtree
- ***Visit the right subtree***
  - Visit the left subtree
  - Visit the right subtree
  - ***Visit the root/current node***
- Visit the root/current node



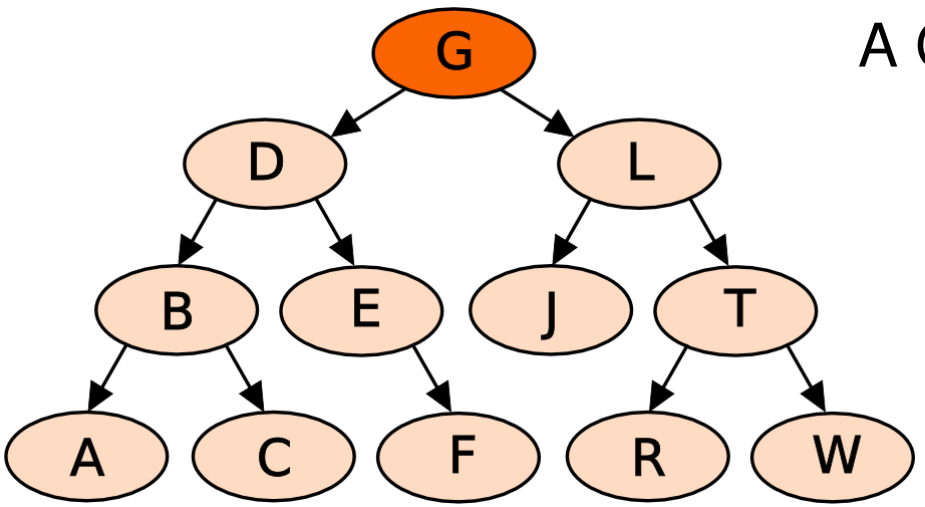
OUTPUT:

A C B F E D J R W T L

Binary Search Tree Traversals

Postorder Traversal –

- Visit the left subtree
- Visit the right subtree
- ***Visit the root/current node***

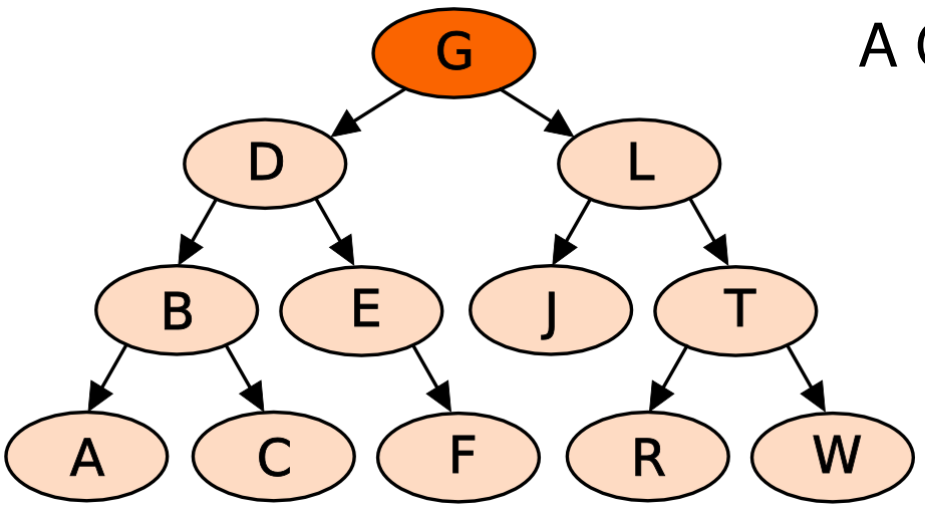


OUTPUT:
A C B F E D J R W T L G

Binary Search Tree Traversals

Postorder Traversal –

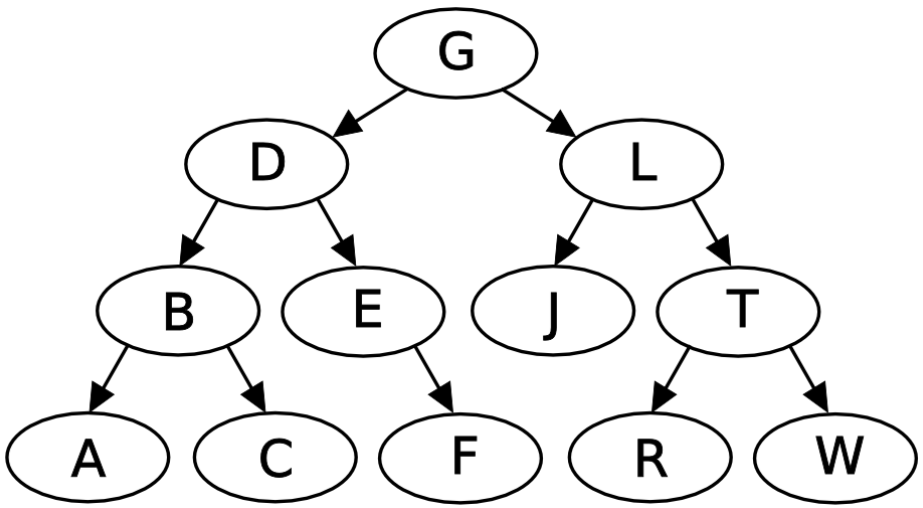
- Visit the left subtree
- Visit the right subtree
- ***Visit the root/current node***



OUTPUT:
A C B F E D J R W T L G

Binary Search Tree Traversals

Traversal Method	Output
Preorder	G D B A C E F L J T R W
Inorder	A B C D E F G J L R T W
Postorder	A C B F E D J R W T L G



BST Add Method

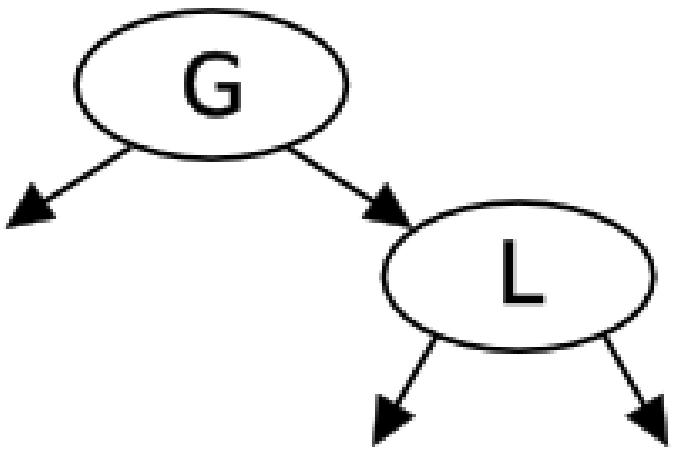
The **ADD** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to find a leaf location which satisfies the ordering for each element exploring the left or rght subtrees as necessary until a *null* is found:

ADD: **G** L J D E T R B C F W A

BST Add Method

The **ADD** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to find a leaf location which satisfies the ordering for each element exploring the left or rght subtrees as necessary until a *null* is found:

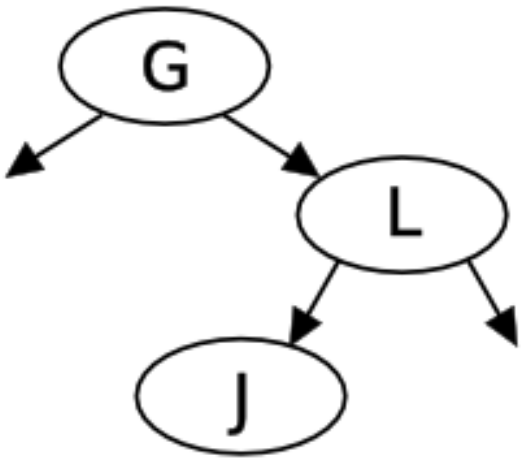
ADD: G **L** J D E T R B C F W A



BST Add Method

The **ADD** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to find a leaf location which satisfies the ordering for each element exploring the left or rght subtrees as necessary until a *null* is found:

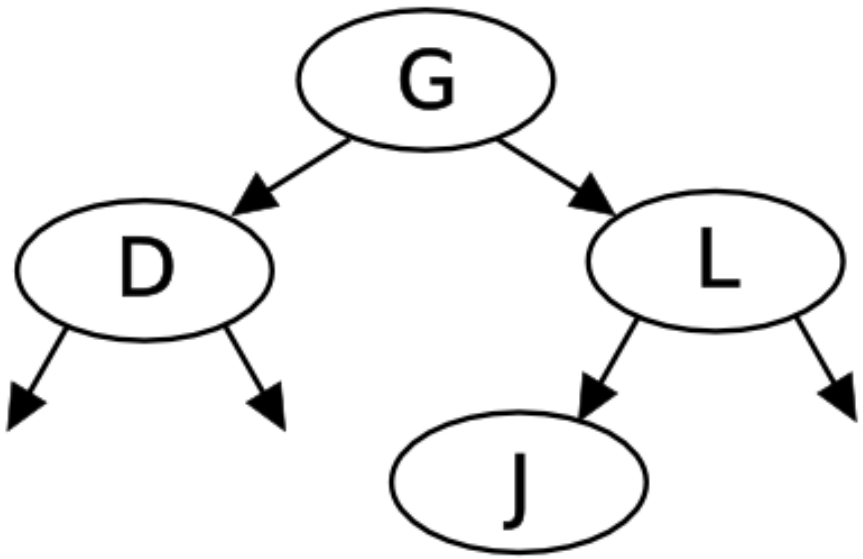
ADD: G L **J** D E T R B C F W A



BST Add Method

The **ADD** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to find a leaf location which satisfies the ordering for each element exploring the left or rght subtrees as necessary until a *null* is found:

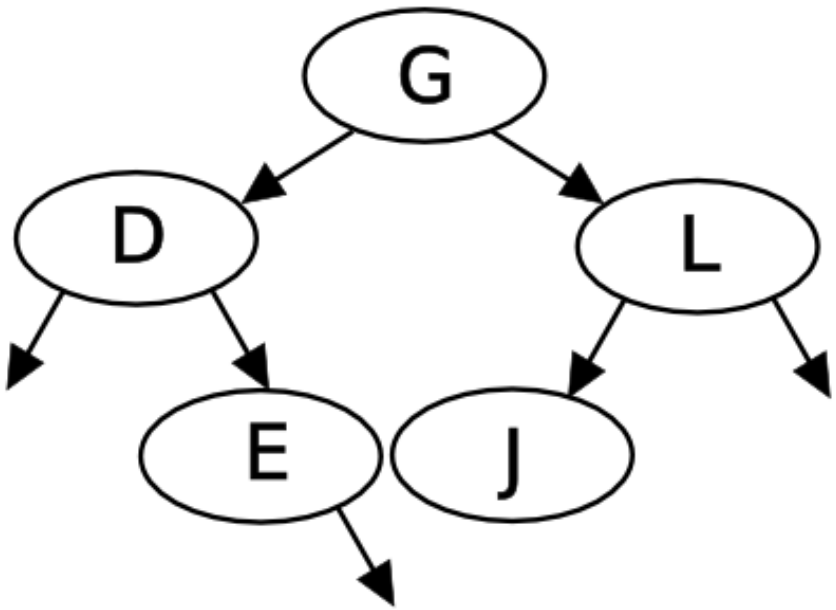
ADD: G L J **D** E T R B C F W A



BST Add Method

The **ADD** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to find a leaf location which satisfies the ordering for each element exploring the left or rght subtrees as necessary until a *null* is found:

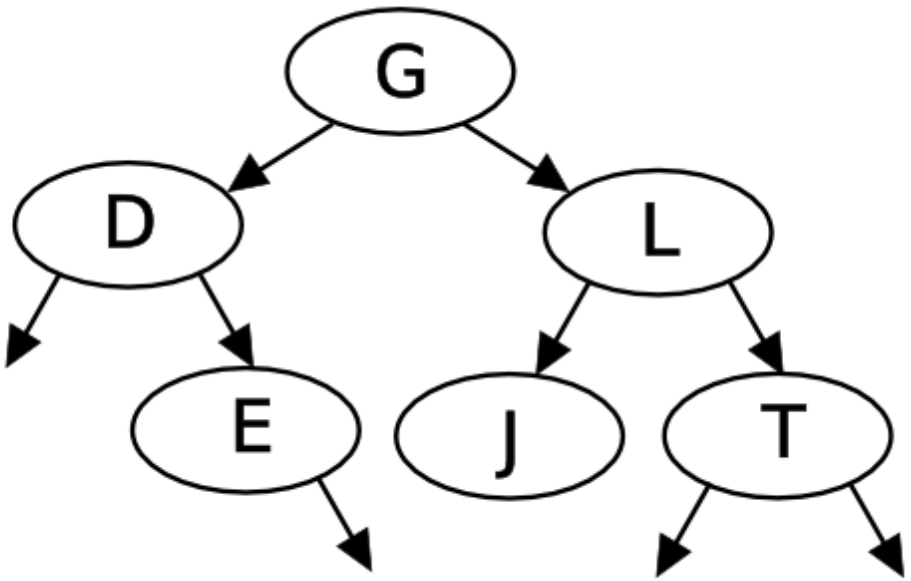
ADD: G L J D **E** T R B C F W A



BST Add Method

The **ADD** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to find a leaf location which satisfies the ordering for each element exploring the left or rght subtrees as necessary until a *null* is found:

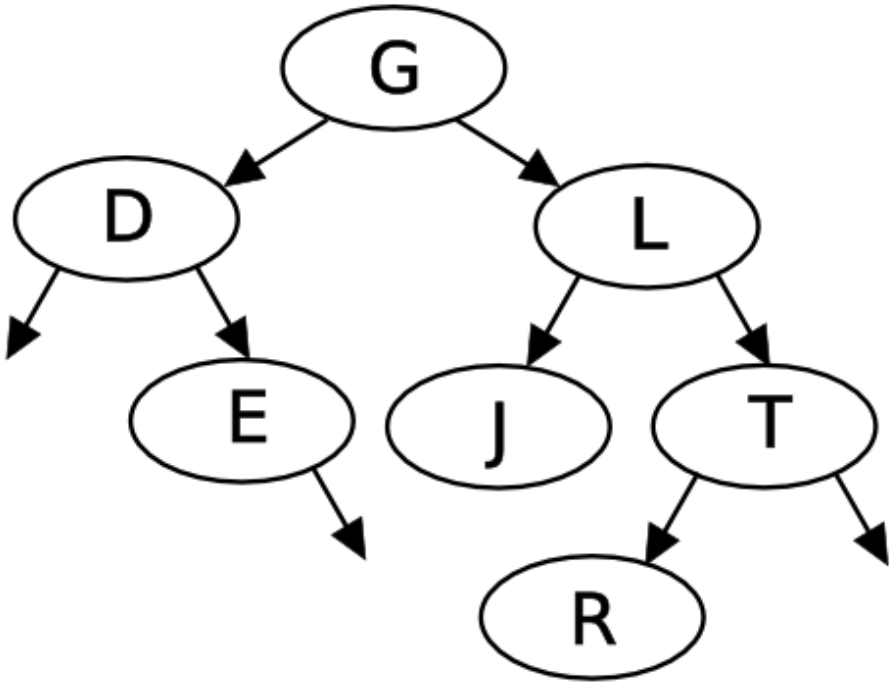
ADD: G L J D E **T** R B C F W A



BST Add Method

The **ADD** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to find a leaf location which satisfies the ordering for each element exploring the left or rght subtrees as necessary until a *null* is found:

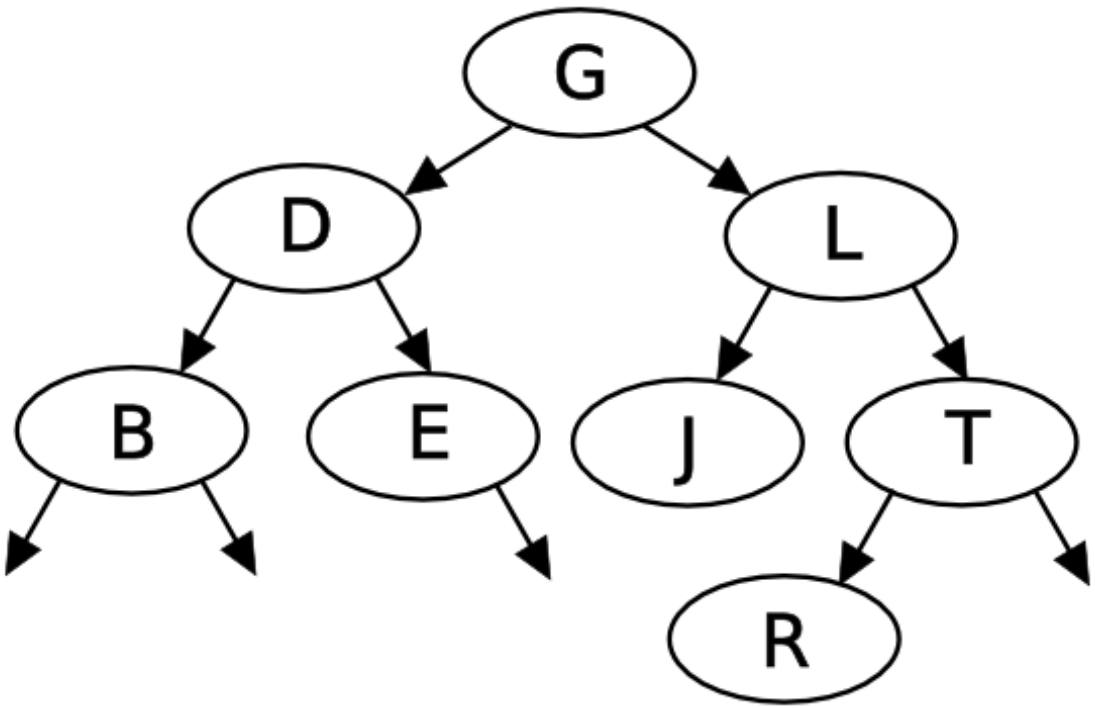
ADD: G L J D E T **R** B C F W A



BST Add Method

The **ADD** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to find a leaf location which satisfies the ordering for each element exploring the left or rght subtrees as necessary until a *null* is found:

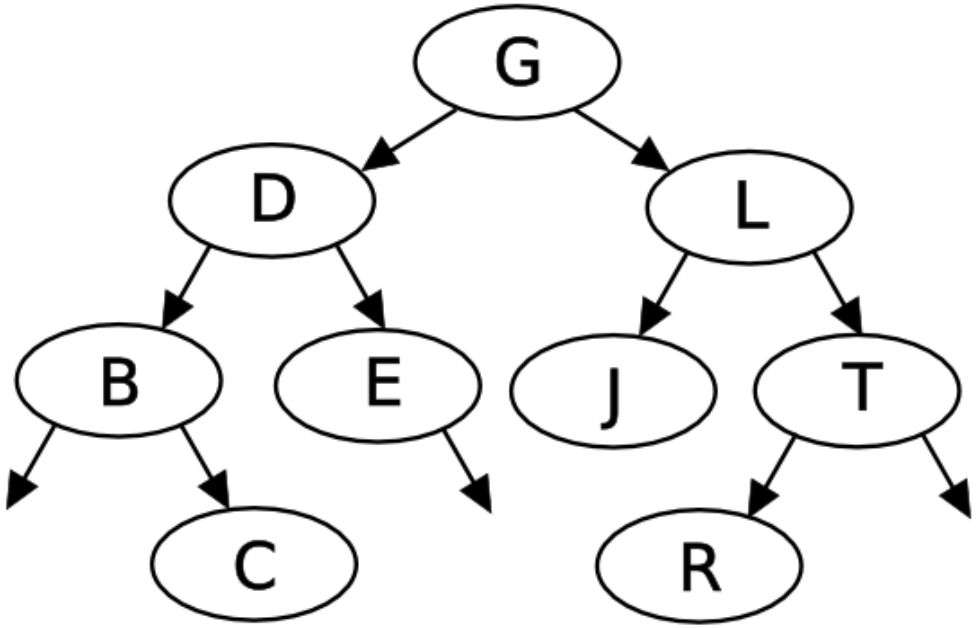
ADD: G L J D E T R **B** C F W A



BST Add Method

The **ADD** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to find a leaf location which satisfies the ordering for each element exploring the left or rght subtrees as necessary until a *null* is found:

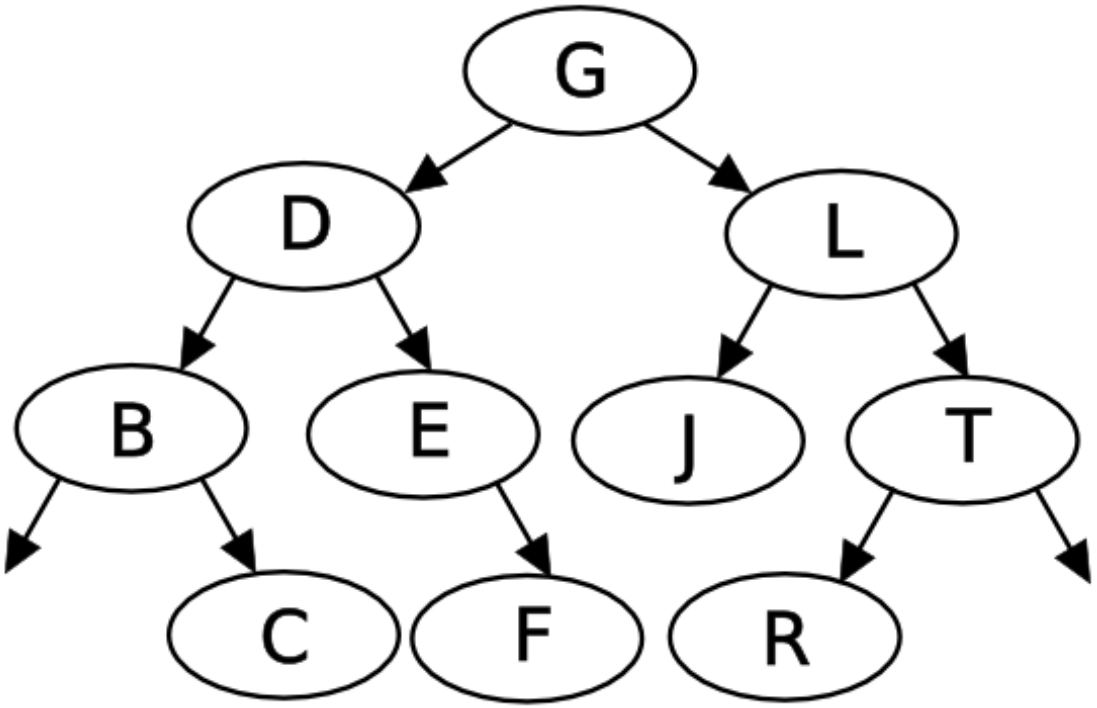
ADD: G L J D E T R B **C** F W A



BST Add Method

The **ADD** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to find a leaf location which satisfies the ordering for each element exploring the left or rght subtrees as necessary until a *null* is found:

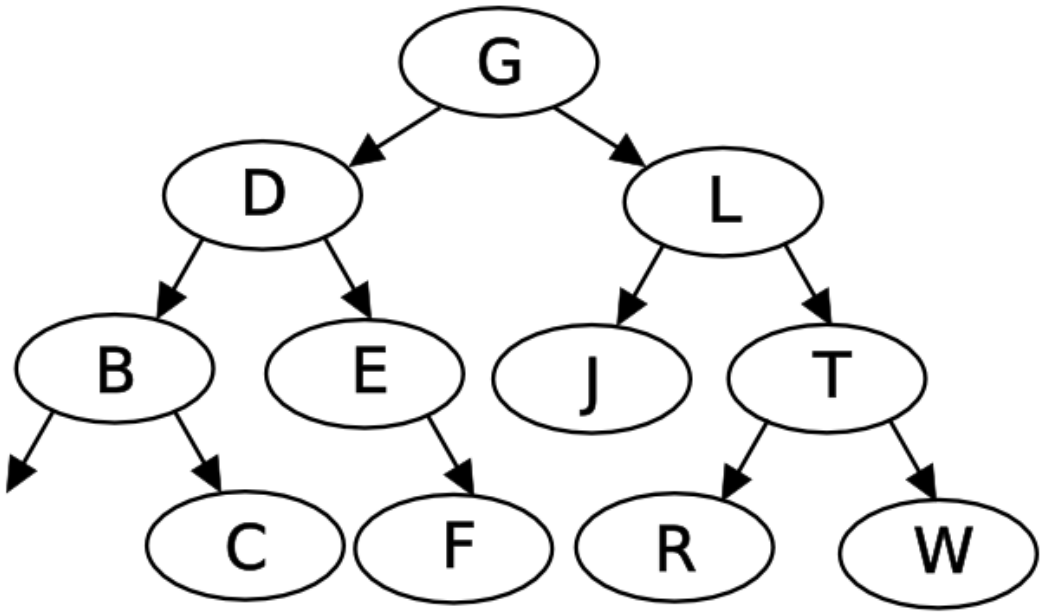
ADD: G L J D E T R B C **F** W A



BST Add Method

The **ADD** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to find a leaf location which satisfies the ordering for each element exploring the left or rght subtrees as necessary until a *null* is found:

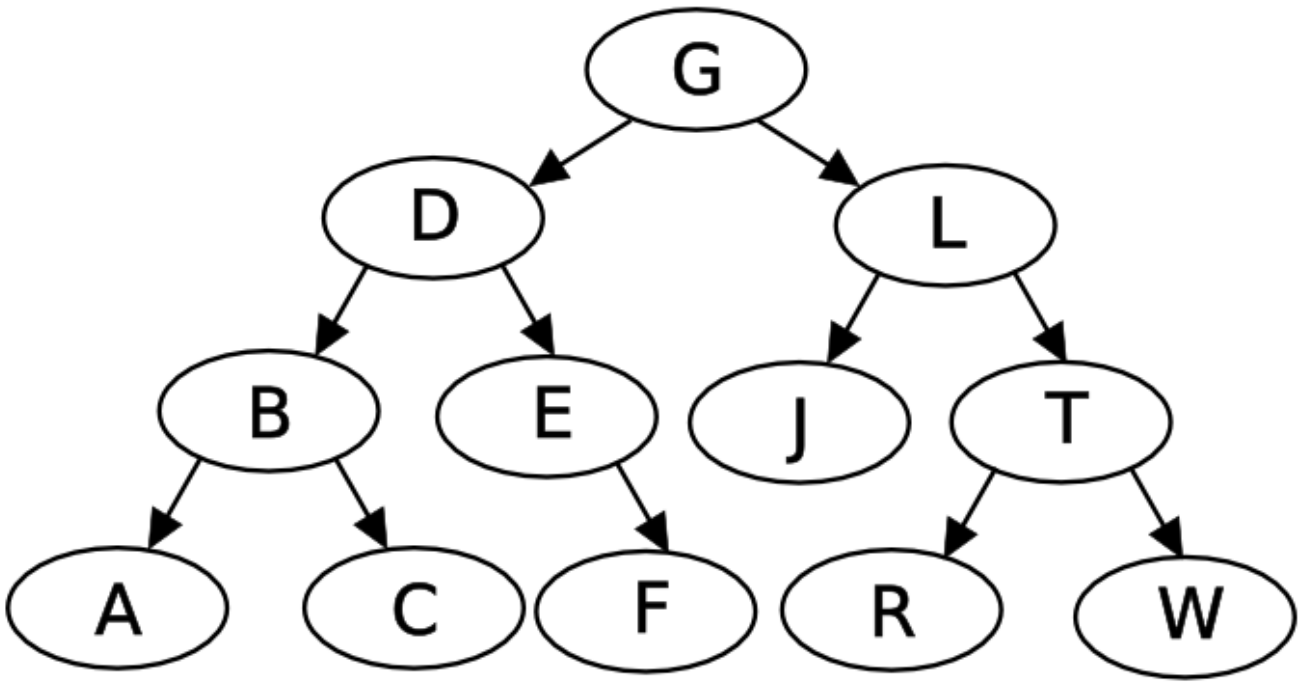
ADD: G L J D E T R B C F **W** A



BST Add Method

The **ADD** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to find a leaf location which satisfies the ordering for each element exploring the left or rght subtrees as necessary until a *null* is found:

ADD: G L J D E T R B C F W **A**



BST Add Method Order

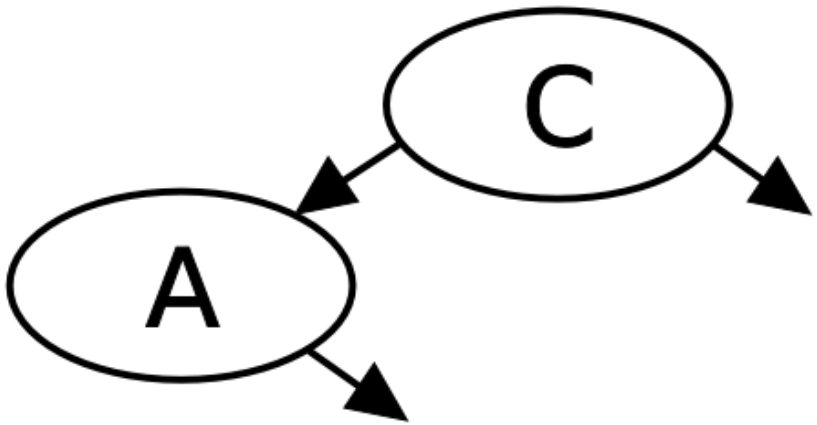
The **ADD** method results in a very different tree given a different order for the input:

ADD: **C** A D T E B F G R W L J

BST Add Method Order

The **ADD** method results in a very different tree given a different order for the input:

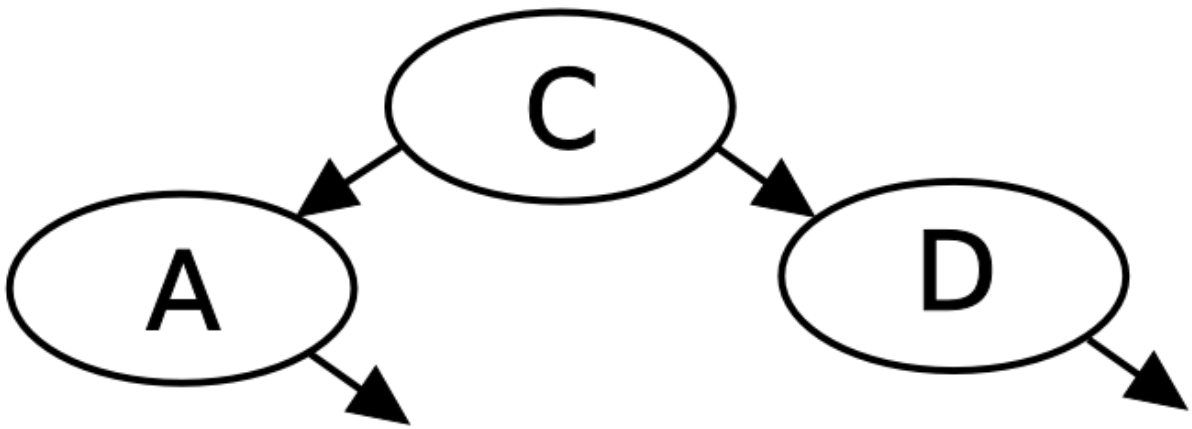
ADD: C **A** D T E B F G R W L J



BST Add Method Order

The **ADD** method results in a very different tree given a different order for the input:

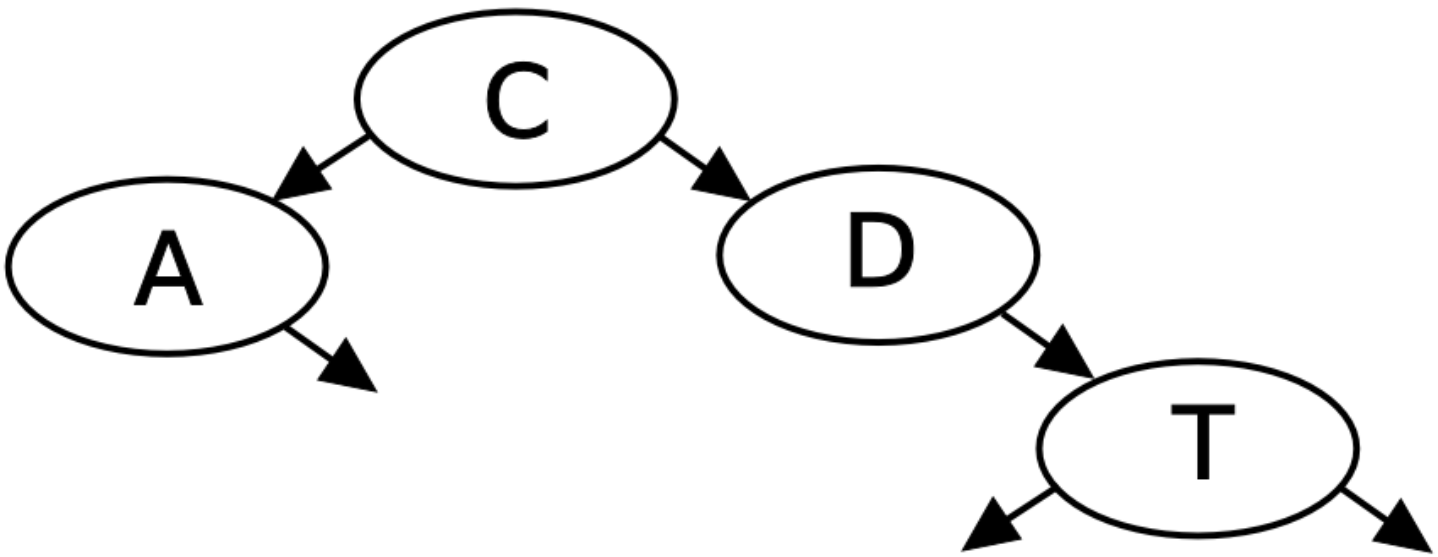
ADD: C A **D** T E B F G R W L J



BST Add Method Order

The **ADD** method results in a very different tree given a different order for the input:

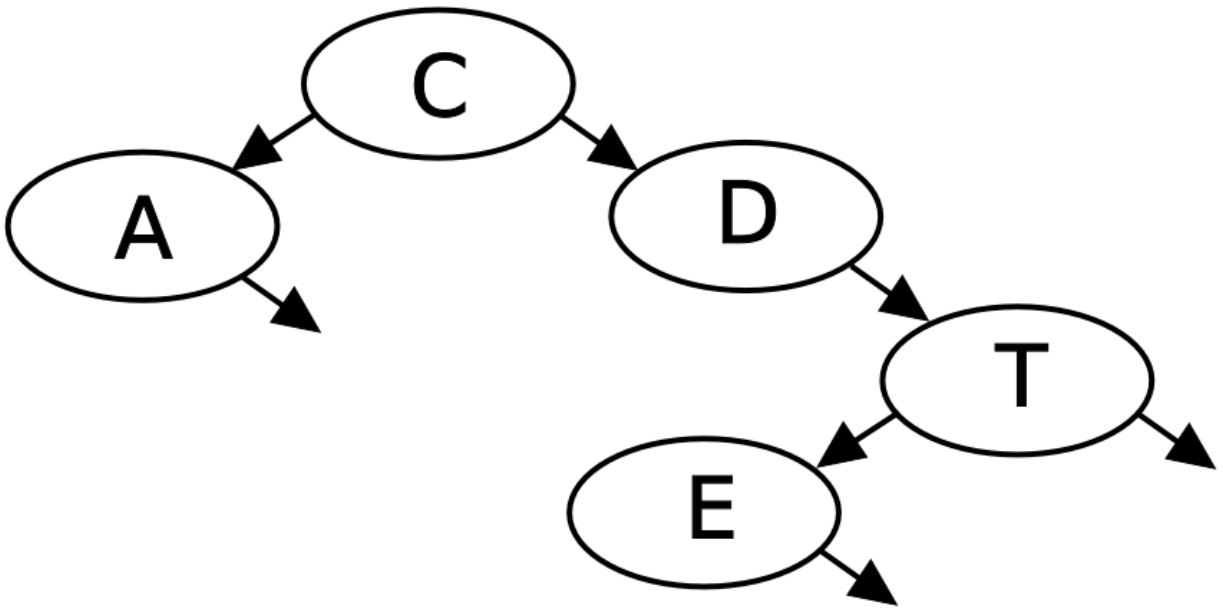
ADD: C A D **T** E B F G R W L J



BST Add Method Order

The **ADD** method results in a very different tree given a different order for the input:

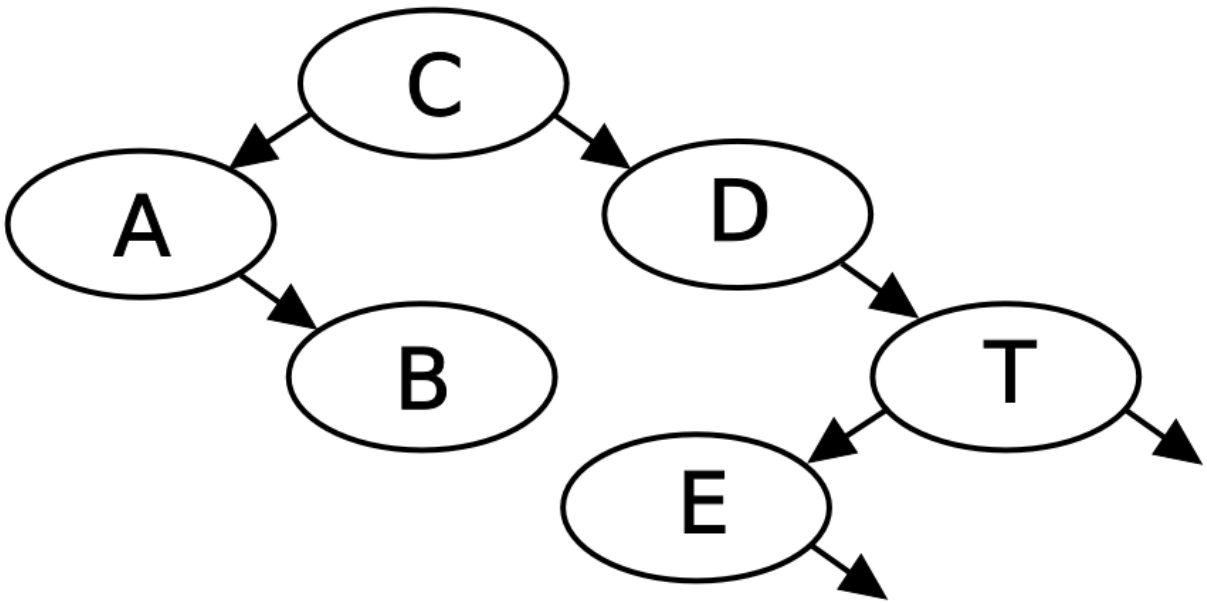
ADD: C A D T **E** B F G R W L J



BST Add Method Order

The **ADD** method results in a very different tree given a different order for the input:

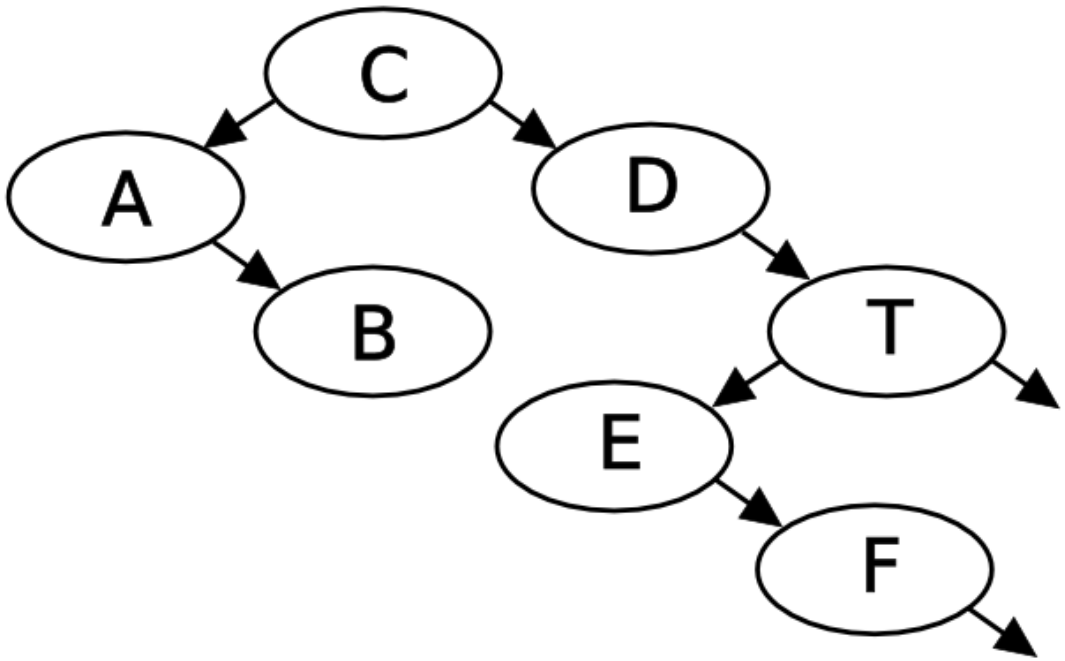
ADD: C A D T E **B** F G R W L J



BST Add Method Order

The **ADD** method results in a very different tree given a different order for the input:

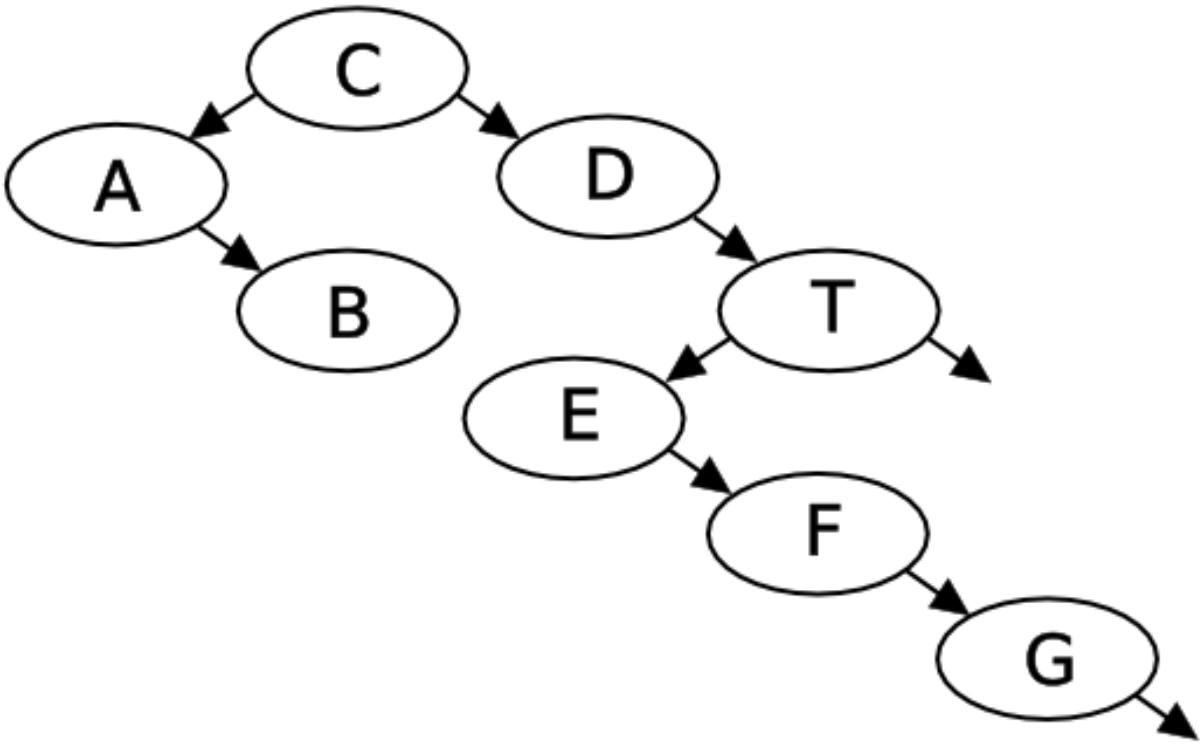
ADD: C A D T E B **F** G R W L J



BST Add Method Order

The **ADD** method results in a very different tree given a different order for the input:

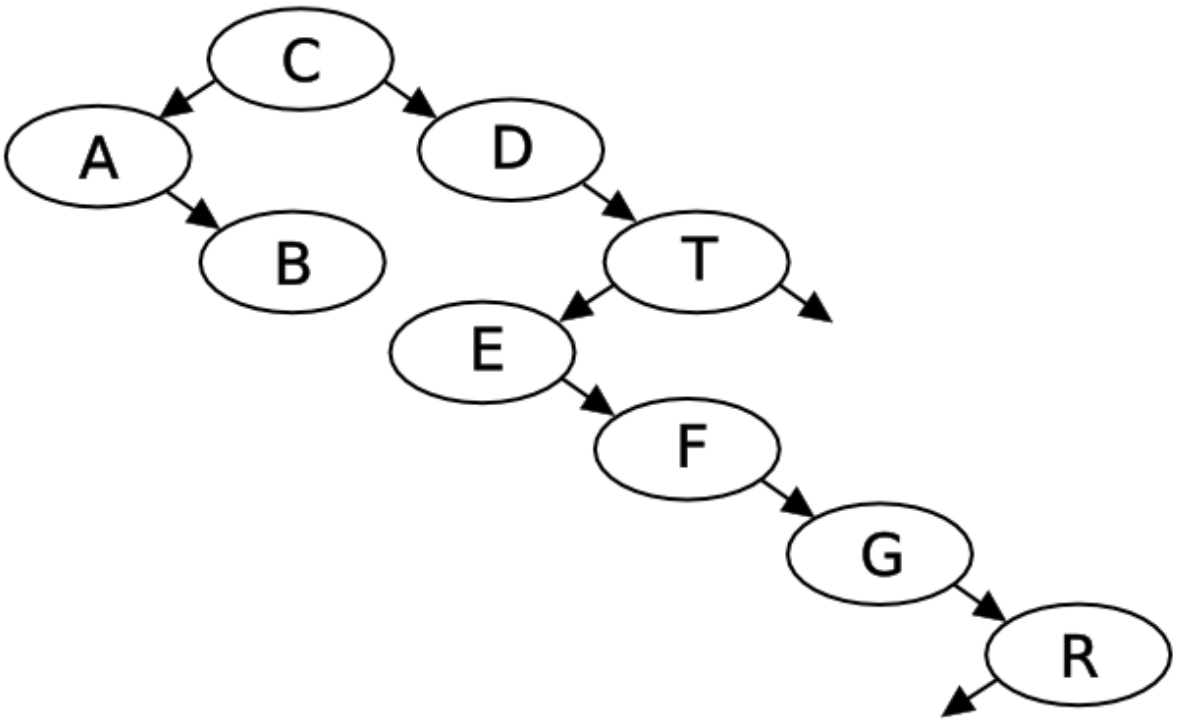
ADD: C A D T E B F **G** R W L J



BST Add Method Order

The **ADD** method results in a very different tree given a different order for the input:

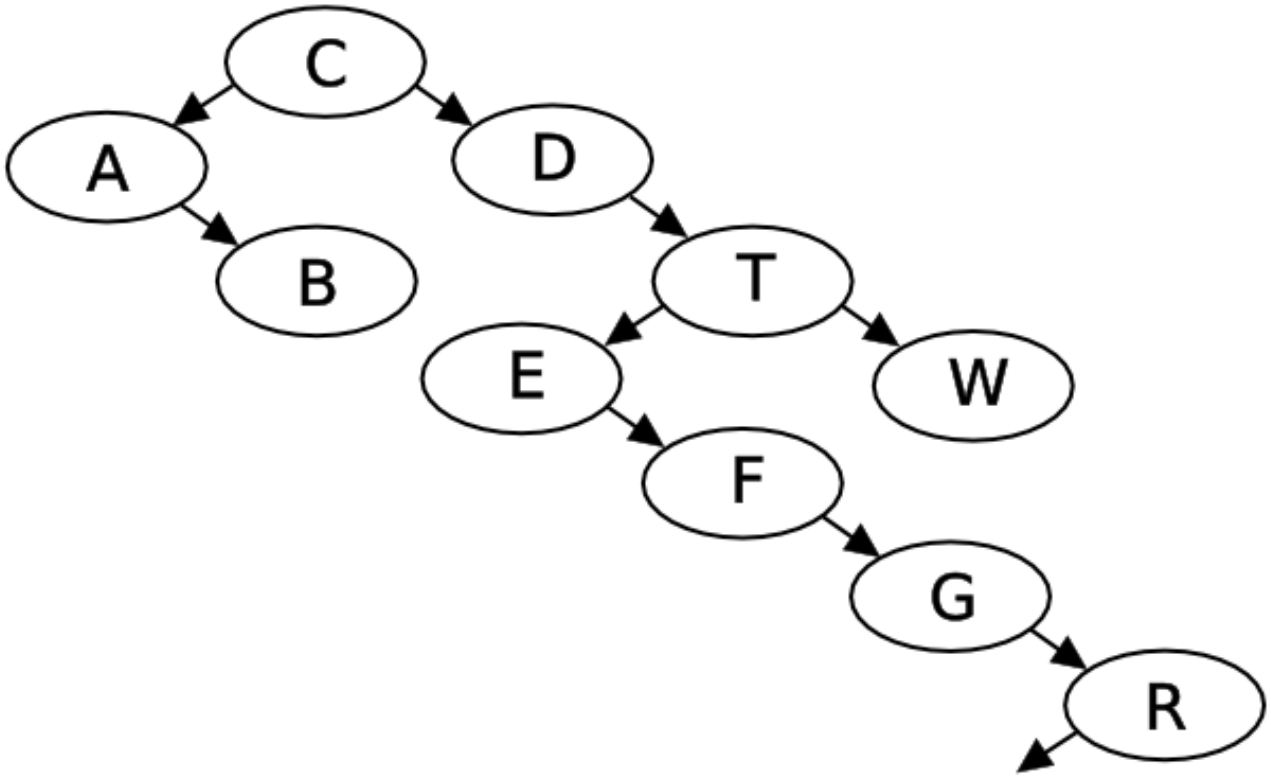
ADD: C A D T E B F G **R** W L J



BST Add Method Order

The **ADD** method results in a very different tree given a different order for the input:

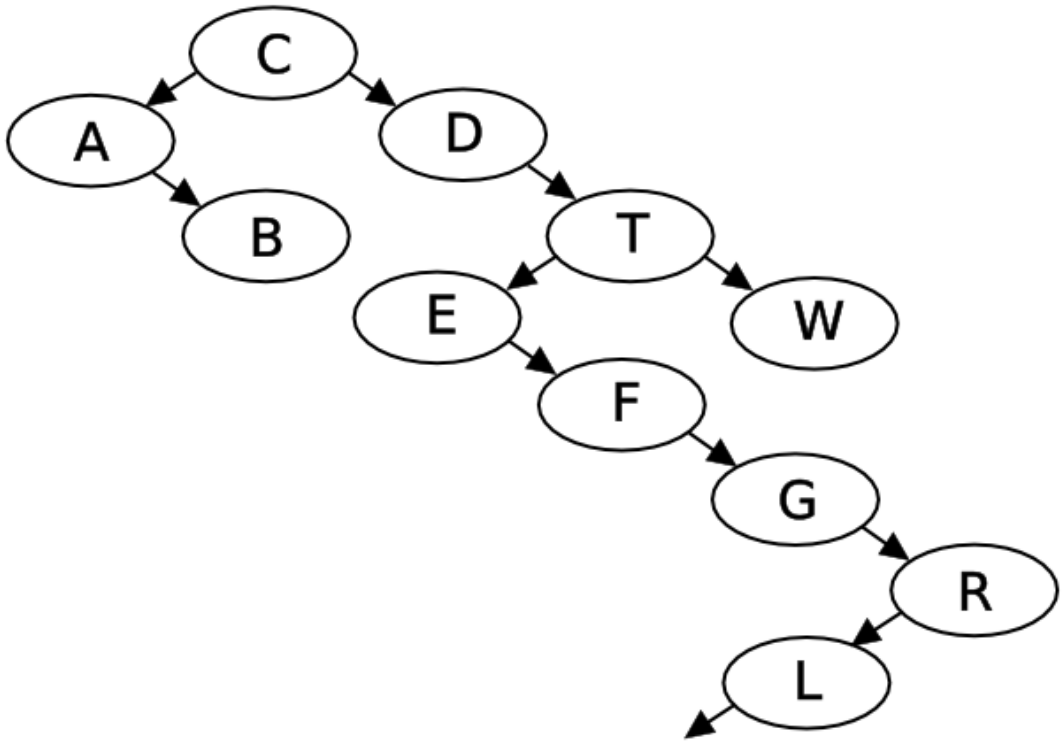
ADD: C A D T E B F G R **W** L J



BST Add Method Order

The **ADD** method results in a very different tree given a different order for the input:

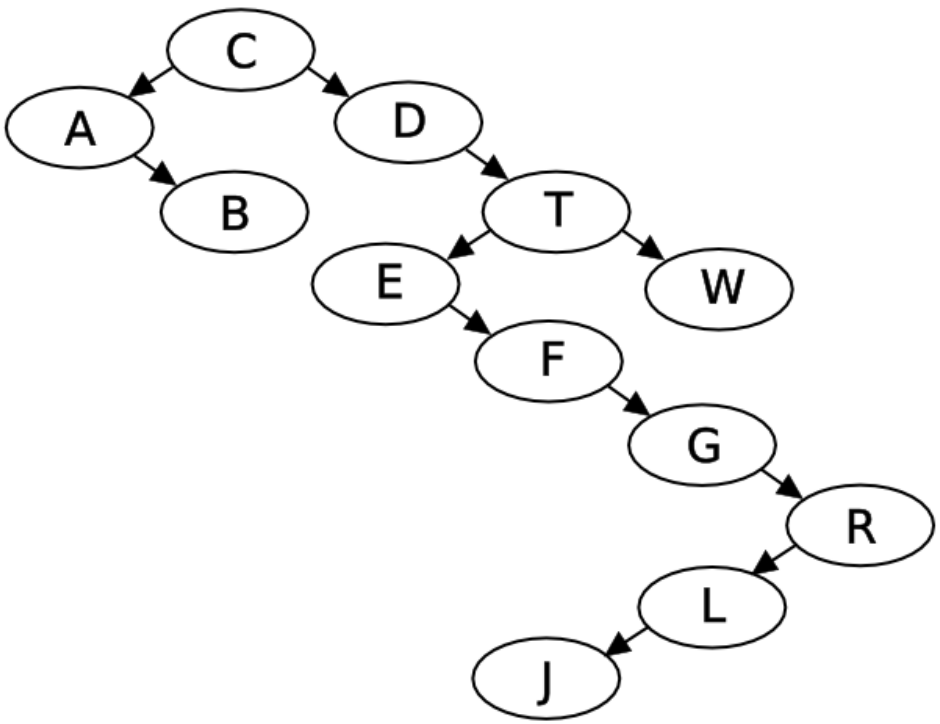
ADD: C A D T E B F G R W **L** J



BST Add Method Order

The **ADD** method results in a very different tree given a different order for the input:

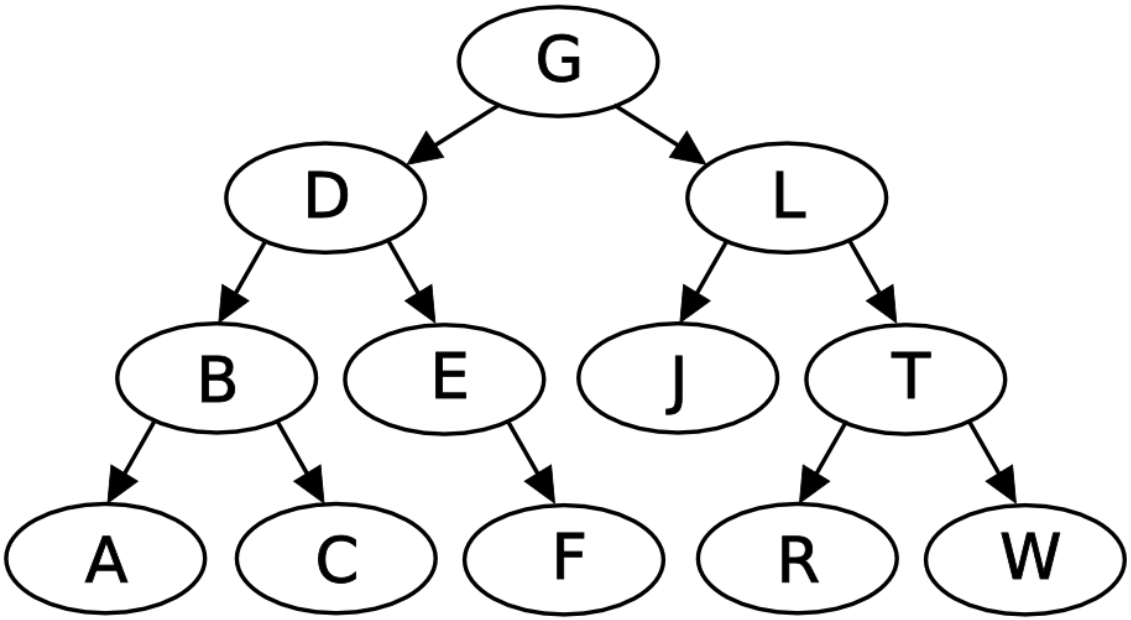
ADD: C A D T E B F G R W L **J**



BST Remove Method

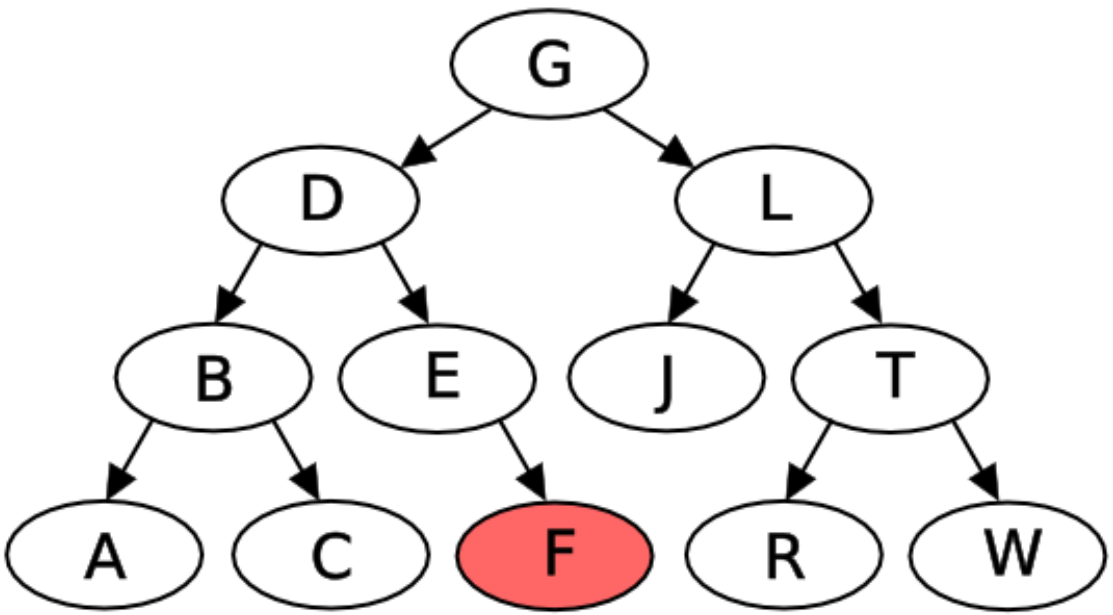
The **Remove** method must maintain order for the for it to satisfy the Binary Search Tree. So we will need to examine 3 cases in regards to the node to be removed:

- Leaf Node
- A Node with a single child
- A Node with two children.



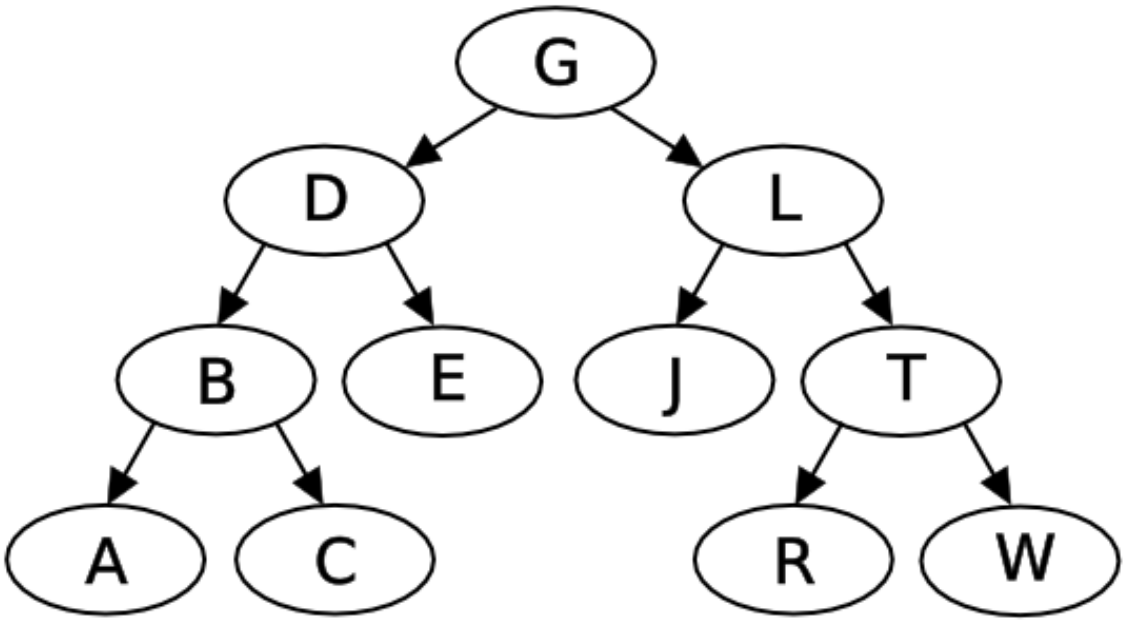
BST Remove Method

- **Remove a Leaf Node** - Just delete the node and update the child link of the parent
- A Node with a single child
- A Node with two children.



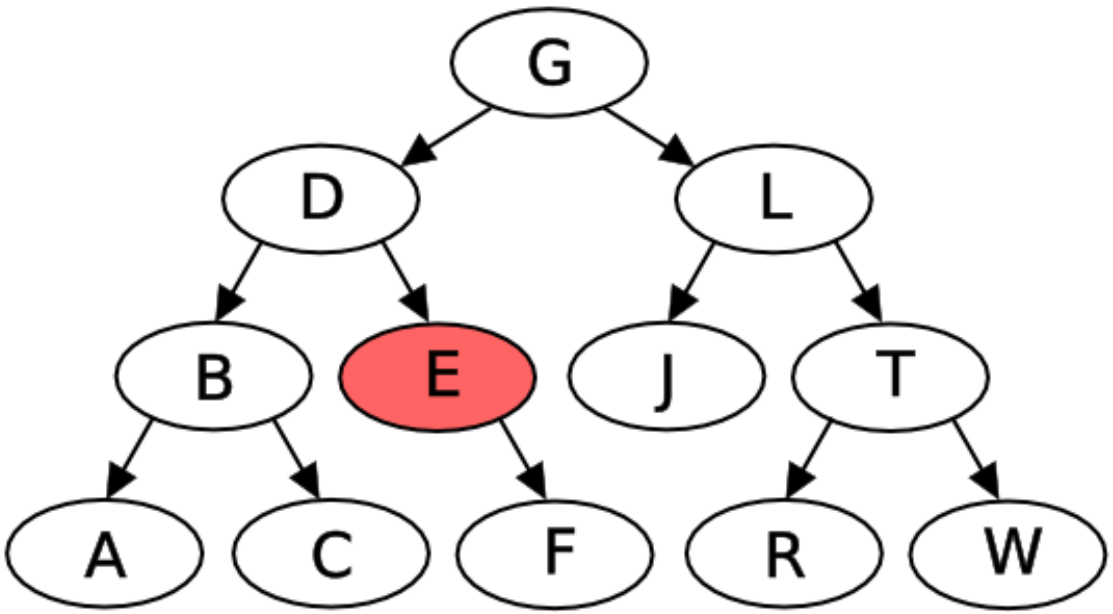
BST Remove Method

- **Remove a Leaf Node** - Just delete the node and update the child link of the parent
- A Node with a single child
- A Node with two children.



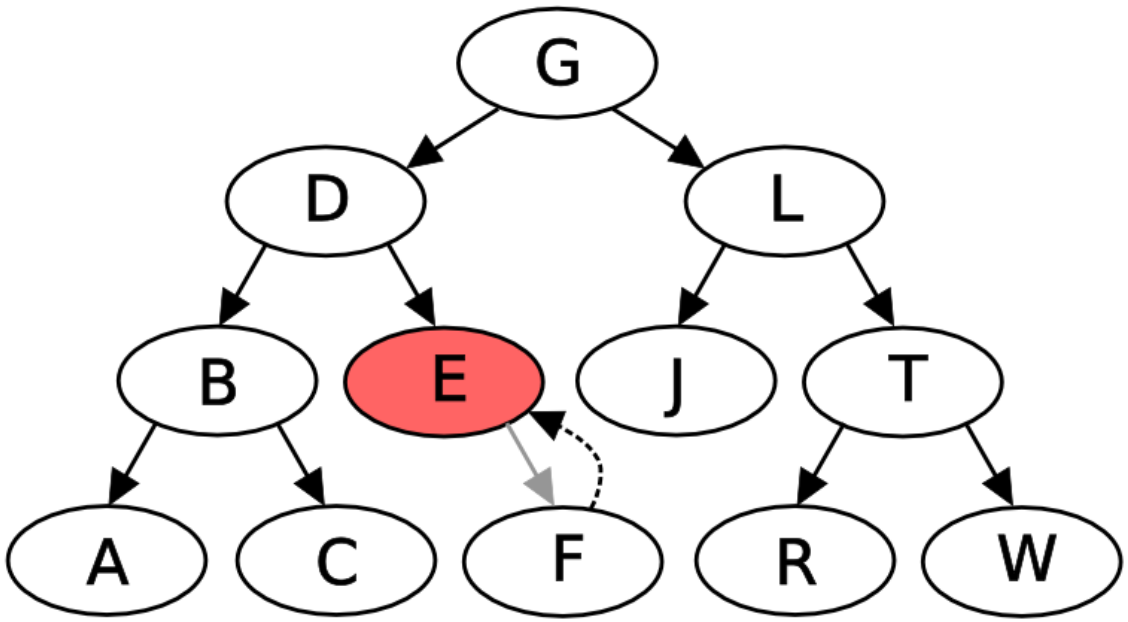
BST Remove Method

- Remove a Leaf Node
- **A Node with a single child** - Delete the node adjusting the parent's link to be the removed node's only child.
- A Node with two children.



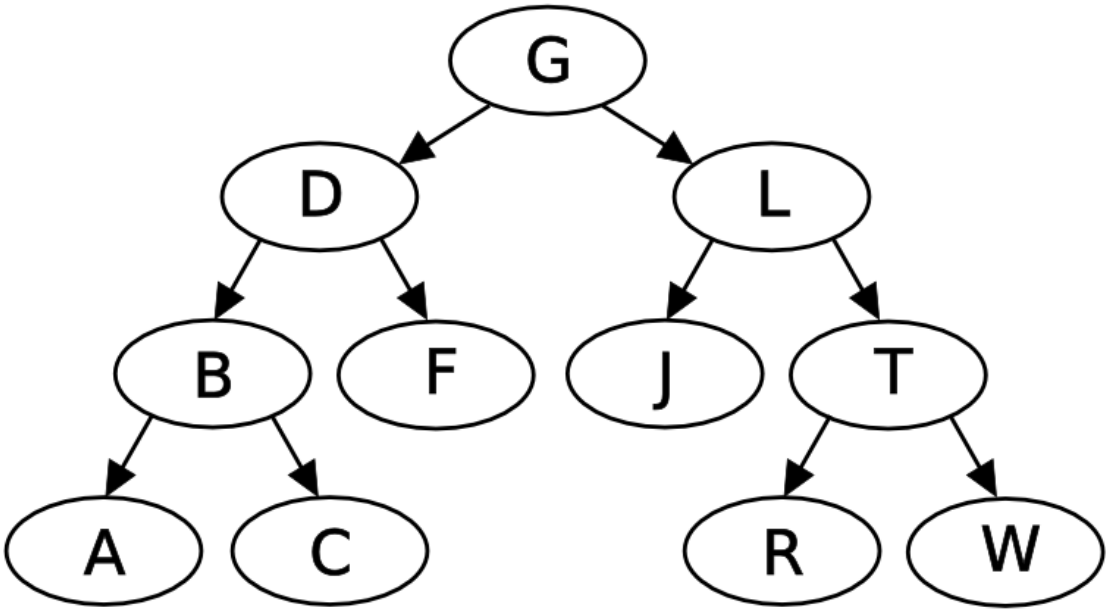
BST Remove Method

- Remove a Leaf Node
- **A Node with a single child** - Delete the node adjusting the parent's link to be the removed node's only child.
- A Node with two children.



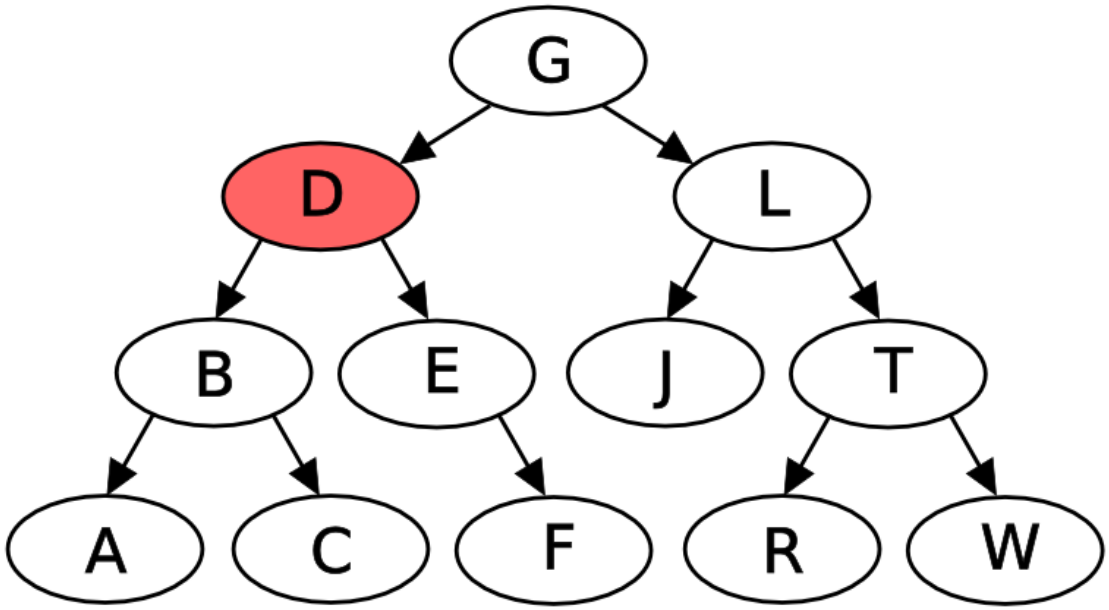
BST Remove Method

- Remove a Leaf Node
- **A Node with a single child** - Delete the node adjusting the parent's link to be the removed node's only child.
- A Node with two children.



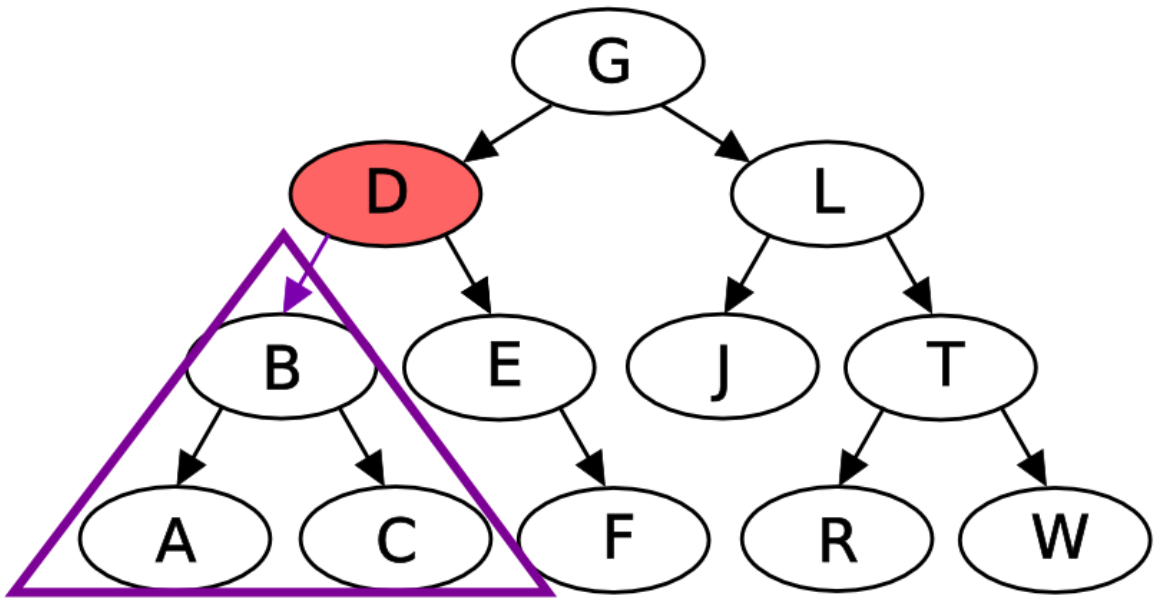
BST Remove Method

- Remove a Leaf Node
- A Node with a single child
- **A Node with two children** - adjust the links so that the largest node in the subtree of the left child takes the place of the removed node:
  - The largest node in the subtree is a leaf.
  - The largest node in the subtree has a single child



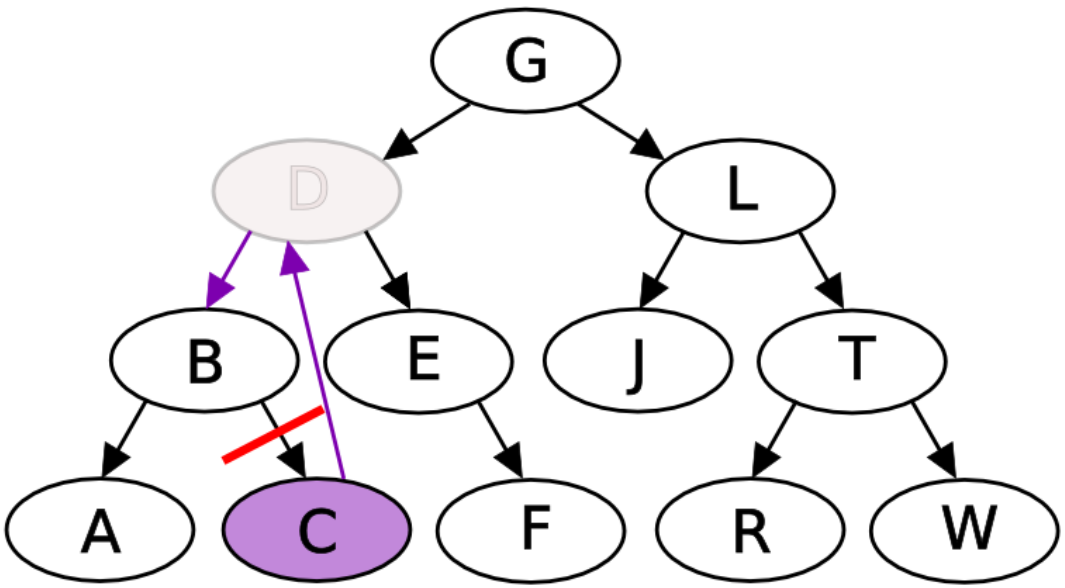
BST Remove Method

- Remove a Leaf Node
- A Node with a single child
- **A Node with two children** - adjust the links so that the largest node in the subtree of the left child takes the place of the removed node:
  - **The largest node in the subtree is a leaf.**
  - The largest node in the subtree has a single child



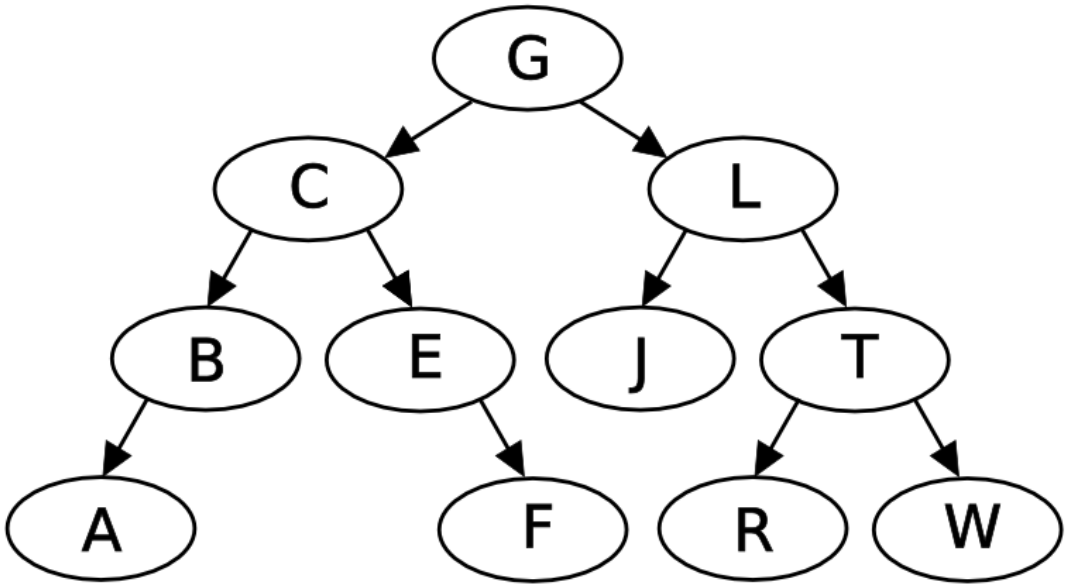
BST Remove Method

- Remove a Leaf Node
- A Node with a single child
- **A Node with two children** - adjust the links so that the largest node in the subtree of the left child takes the place of the removed node:
  - **The largest node in the subtree is a leaf.**
  - The largest node in the subtree has a single child



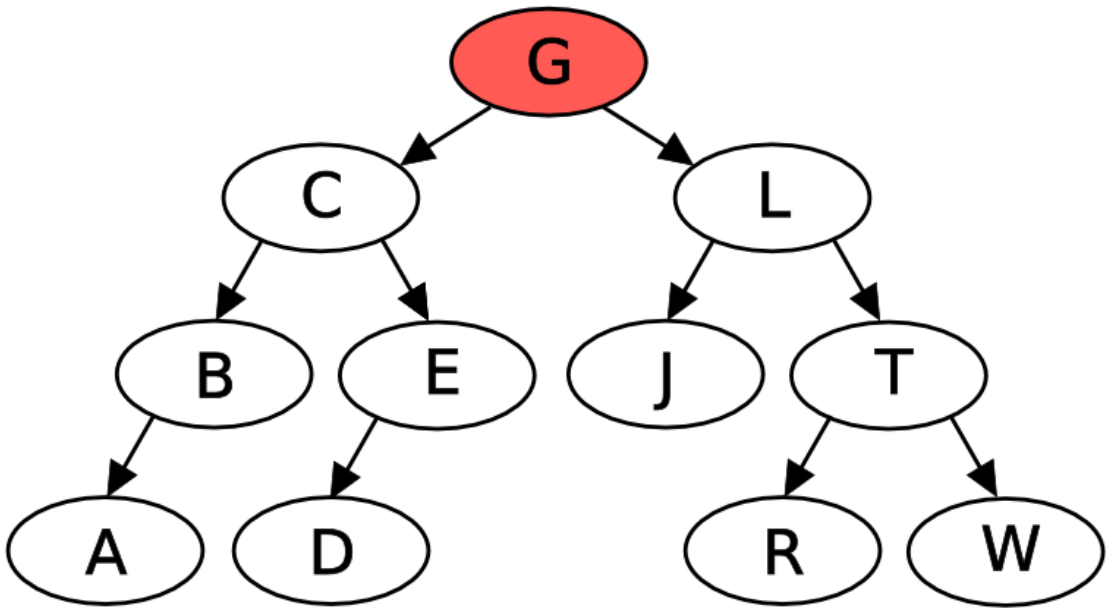
BST Remove Method

- Remove a Leaf Node
- A Node with a single child
- **A Node with two children** - adjust the links so that the largest node in the subtree of the left child takes the place of the removed node:
  - **The largest node in the subtree is a leaf.**
  - The largest node in the subtree has a single child



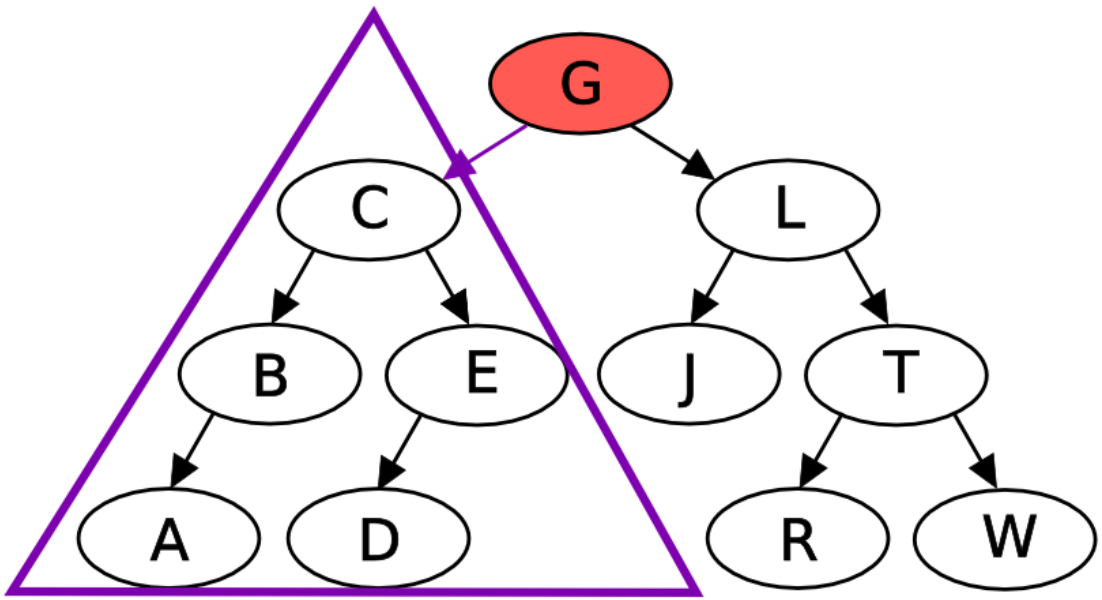
BST Remove Method

- Remove a Leaf Node
- A Node with a single child
- **A Node with two children** - adjust the links so that the largest node in the subtree of the left child takes the place of the removed node:
  - The largest node in the subtree is a leaf.
  - The largest node in the subtree has a single child



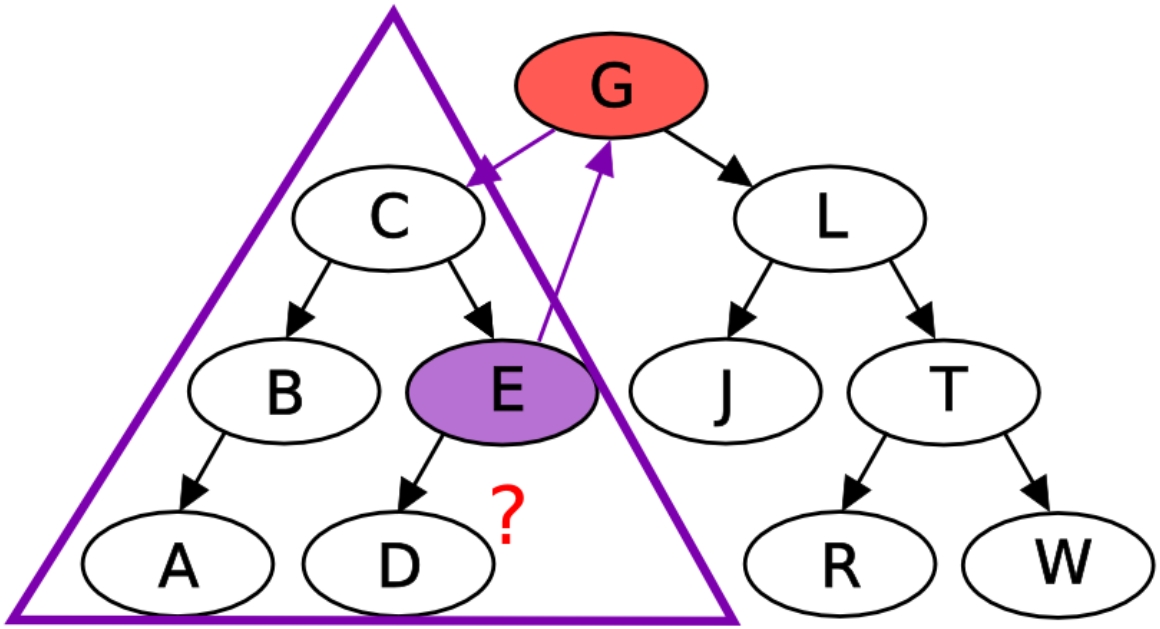
BST Remove Method

- Remove a Leaf Node
- A Node with a single child
- **A Node with two children** - adjust the links so that the largest node in the subtree of the left child takes the place of the removed node:
  - The largest node in the subtree is a leaf.
  - **The largest node in the subtree has a single child**



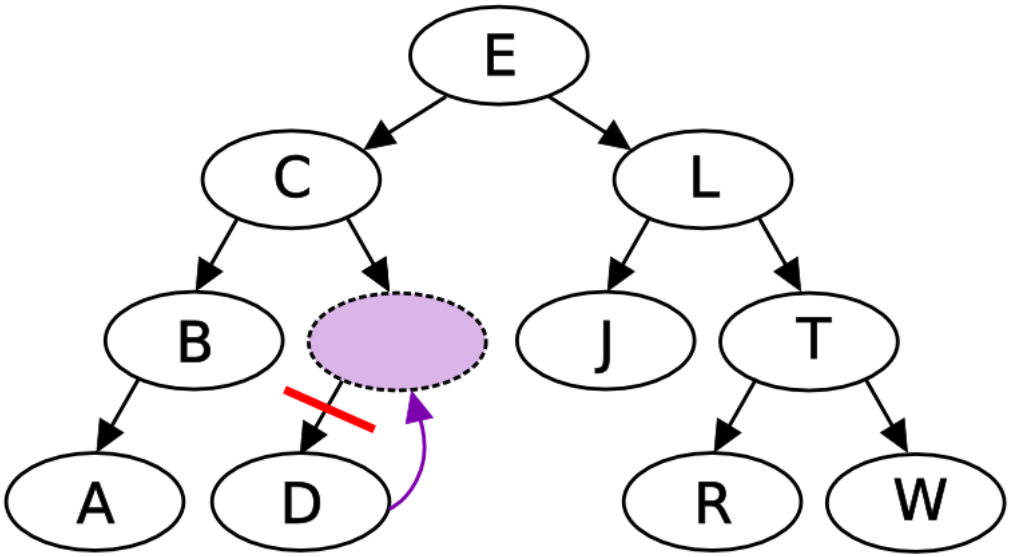
BST Remove Method

- Remove a Leaf Node
- A Node with a single child
- **A Node with two children** - adjust the links so that the largest node in the subtree of the left child takes the place of the removed node:
  - The largest node in the subtree is a leaf.
  - **The largest node in the subtree has a single child**



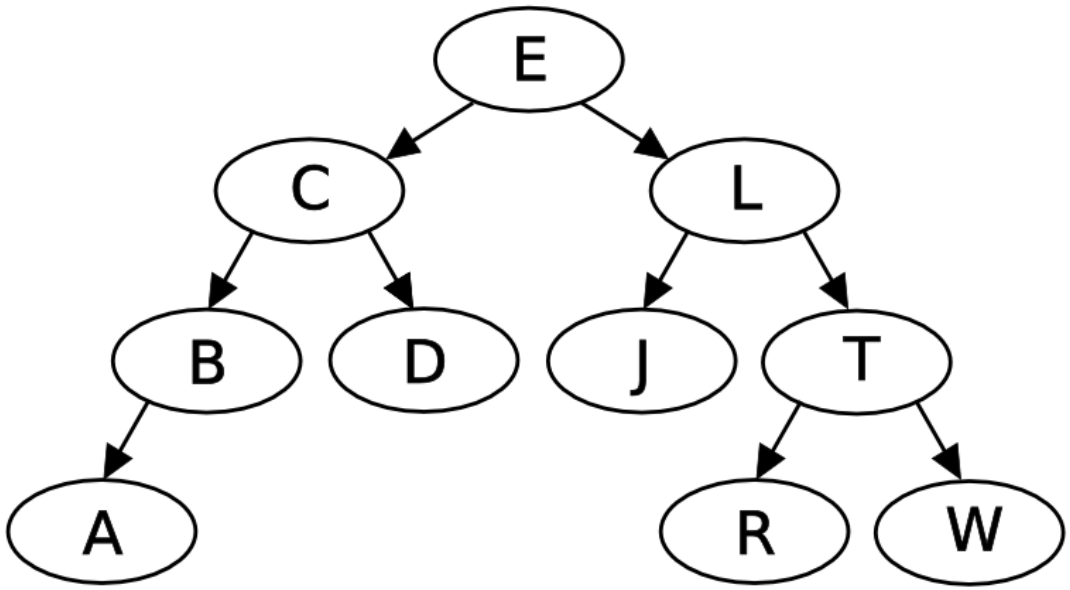
BST Remove Method

- Remove a Leaf Node
- A Node with a single child
- **A Node with two children** - adjust the links so that the largest node in the subtree of the left child takes the place of the removed node:
  - The largest node in the subtree is a leaf.
  - **The largest node in the subtree has a single child**



BST Remove Method

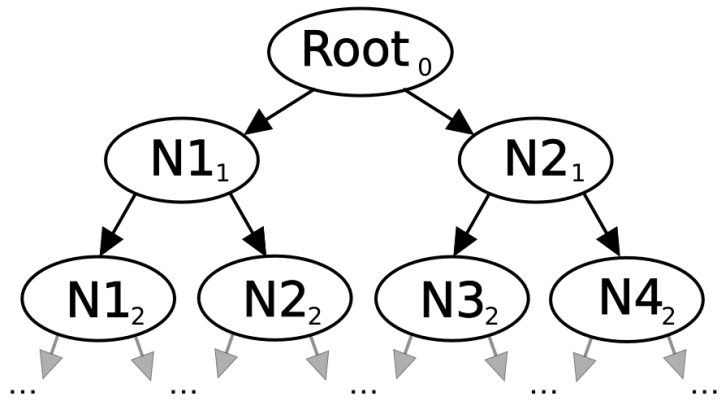
- Remove a Leaf Node
- A Node with a single child
- **A Node with two children** - adjust the links so that the largest node in the subtree of the left child takes the place of the removed node:
  - The largest node in the subtree is a leaf.
  - **The largest node in the subtree has a single child**



Array Representation

In order to store a binary tree into an array we know how many nodes are possible on each level:

- Level 0: 1
- Level 1: 2
- Level 2: 4
- Level 3: 8
- Level n: 2^n

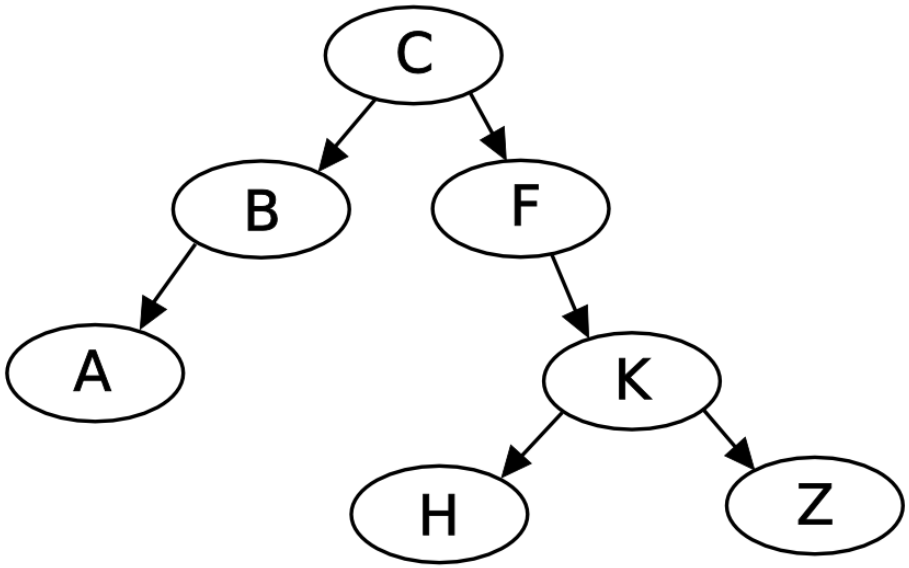


We can unwrap a tree such that every possible node is concatenated into an array:

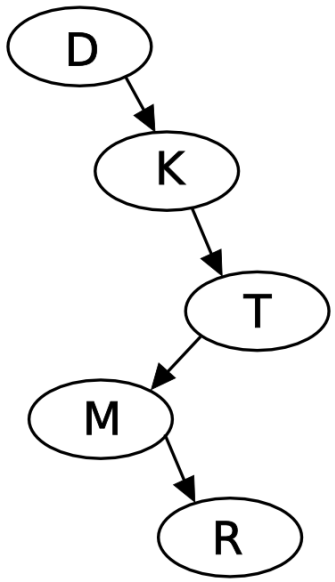
Lvl 0	Lvl 1		Lvl 2				Lvl 3...		
$Root_0$	$N1_1$	$N2_1$	$N1_2$	$N2_2$	$N3_2$	$N4_2$	$N1_3$	$N2_3$	...

Array Representation

BST 1:



BST 2:



BST 1:

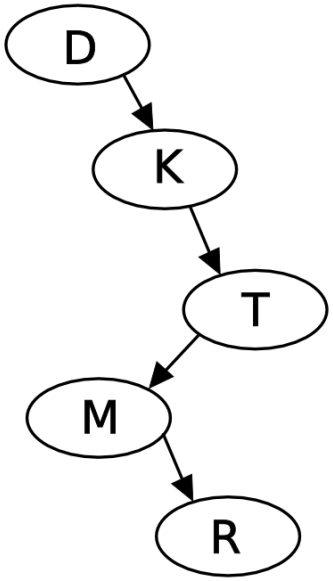
C	B	F	A	∅	∅	K	∅	∅	∅	∅	∅	∅	H	Z

BST 2:

D	∅	K	∅	∅	∅	T	∅	∅	∅	∅	∅	∅	M	∅	...

Balancing

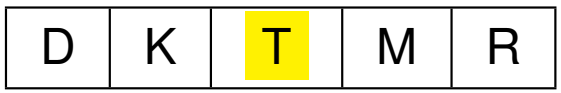
As can be seen it is possible for a binary search tree to be created in such a way that the run-time benefits are not applicable. In essence a tree constructed in one way is equvalent to a Single Linked List

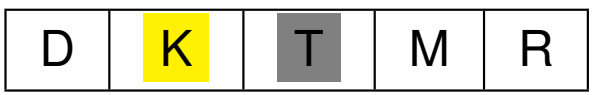


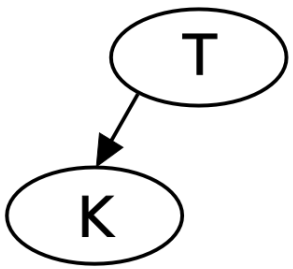
It is possible to reconstruct the tree such that the tree will be balanced...The best way is to split the ordered dat a in half and use that as a node.

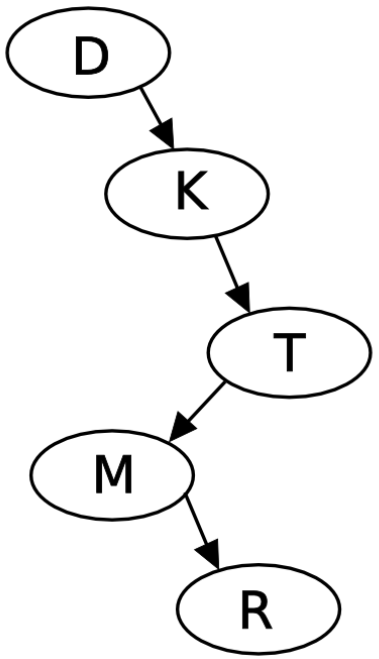
Balancing

Using in order traversing we can construct the array.



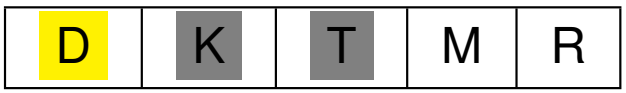


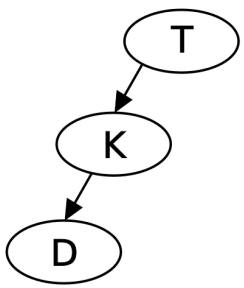


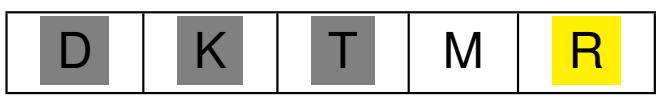


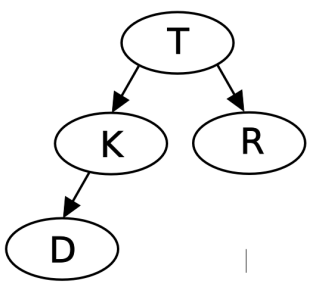
Balancing

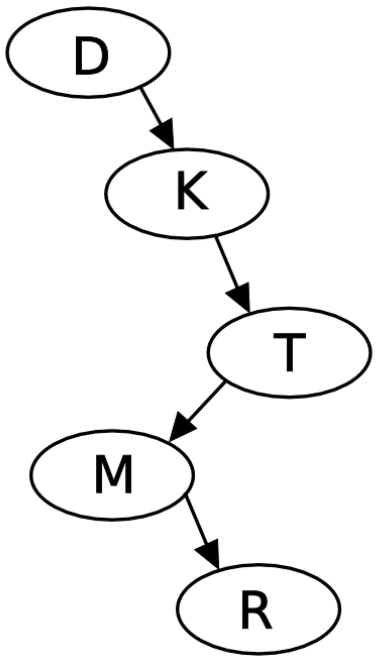
We can use a recusive call to subdivide the array adding the new center of the subdivided array.





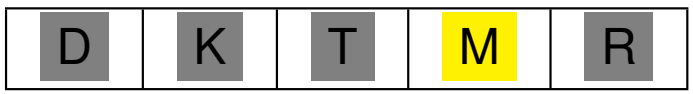


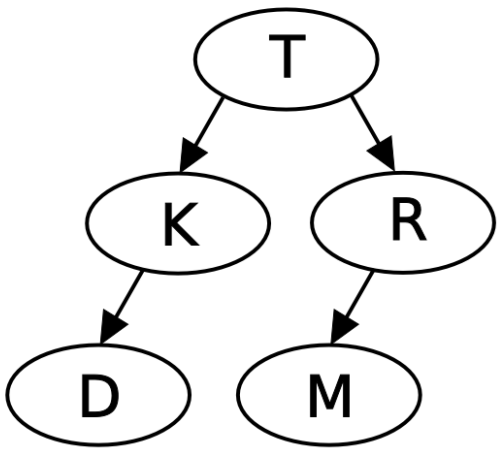


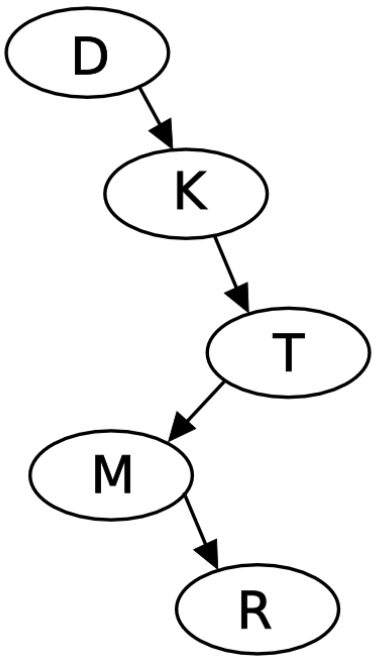


Balancing

We can use a recusive call to subdivide the array adding the new center of the subdivided array.







Big-Oh Comparisons

Methods	BST	Array LL**	Single LL
Constructor	O(1)	O(n)	O(1)
isEmpty	O(1)	O(1)	O(1)
contains	O($log_2(n)$)	O($log_2(n)$)	O(n)
get	O($log_2(n)$)	O($log_2(n)$)	O(n)
add			
–find	O($log_2(n)$)	O($log_2(n)$)	O(n)
–process	O(1)	O(n)	O(1)
–total	O($log_2(n)$)	O(n)	O(n)
remove			
–find	O($log_2(n)$)	O($log_2(n)$)	O(n)
–process	O(1)	O(n)	O(1)
–total	O($log_2(n)$)	O(n)	O(n)

***Sorted Array-Based Linked List*