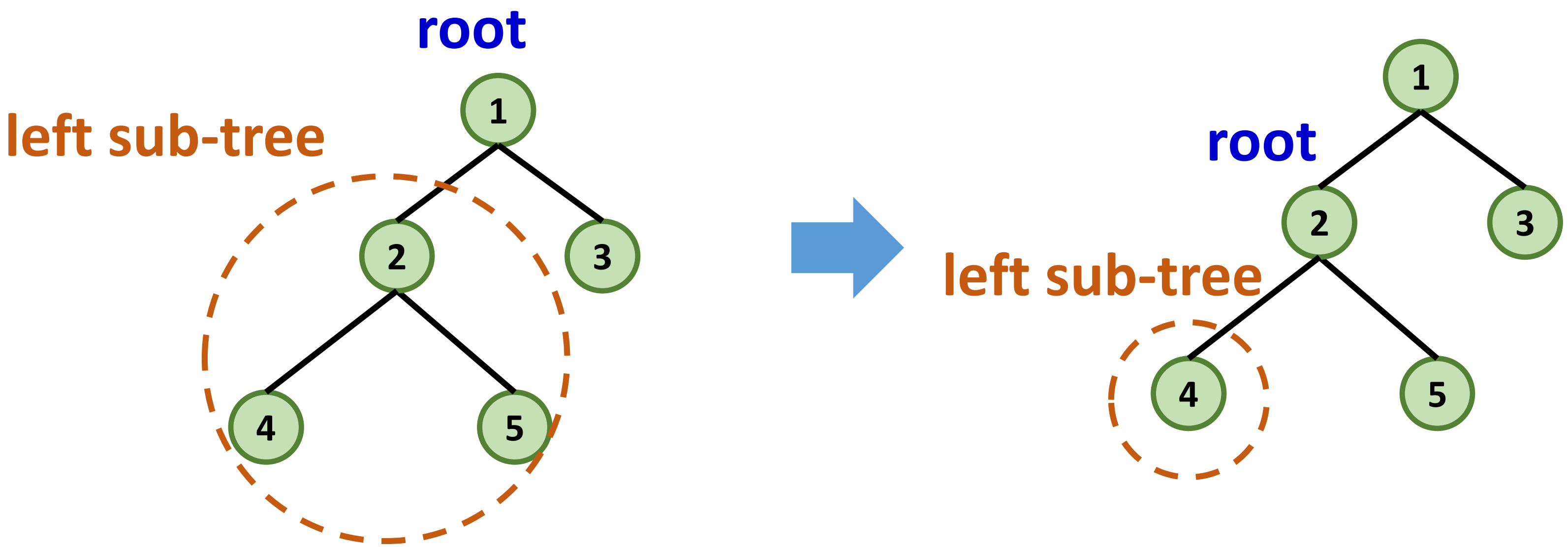
TREES

PROF. NAVRATI SAXENA

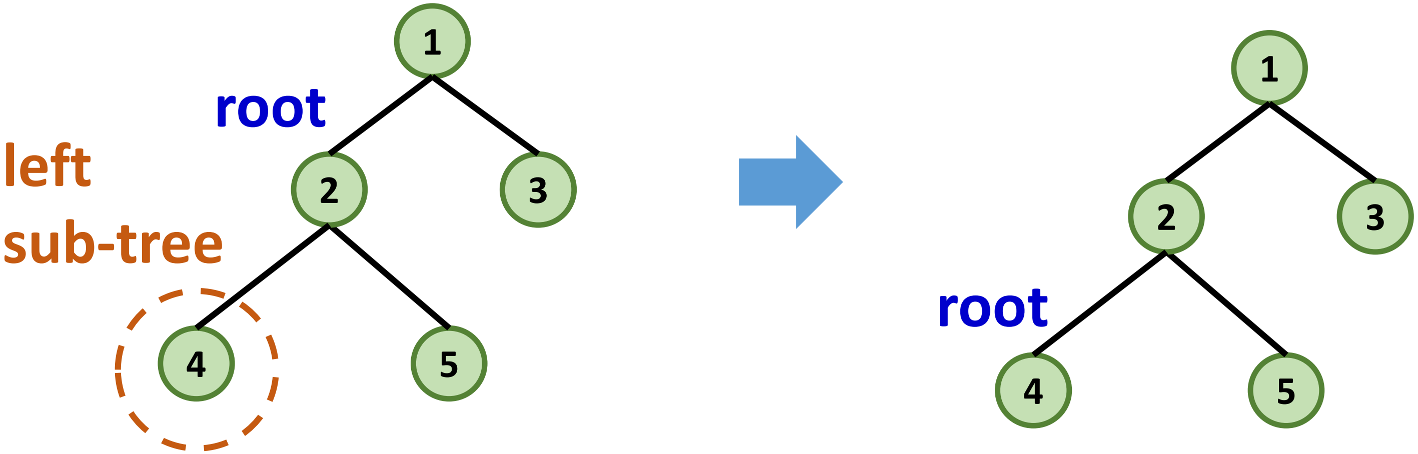
DFS: In-order Traversal (1/6)

- **left -> root -> right**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively



DFS: In-order Traversal (2/6)

- **left -> root -> right**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively

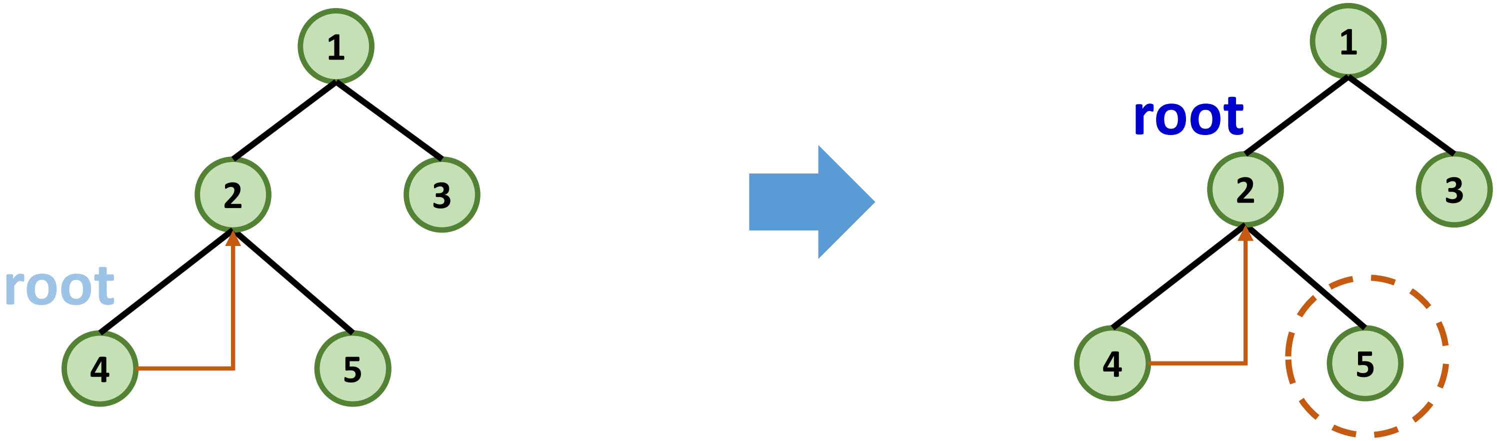


left sub-tree: Empty
=> Visit root = 4

4				

DFS: In-order Traversal (3/6)

- **left -> root -> right**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively



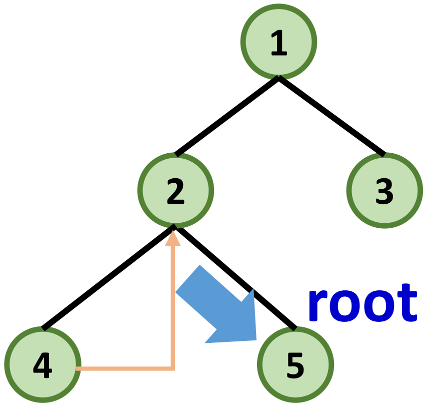
right sub-tree: Empty
=> Visit root(4) = 2

Visit right sub-tree

4	2			

DFS: In-order Traversal (4/6)

- **left -> root -> right**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively

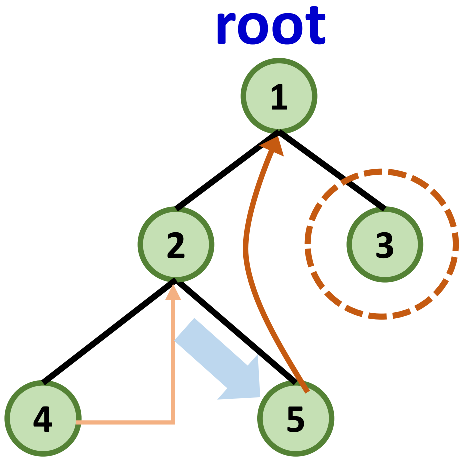


left sub-tree: Empty
right sub-tree: Empty
⇒ Root (5) = 2: Done
⇒ Root (2) = 1

4	2	5		

DFS: In-order Traversal (5/6)

- **left -> root -> right**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively

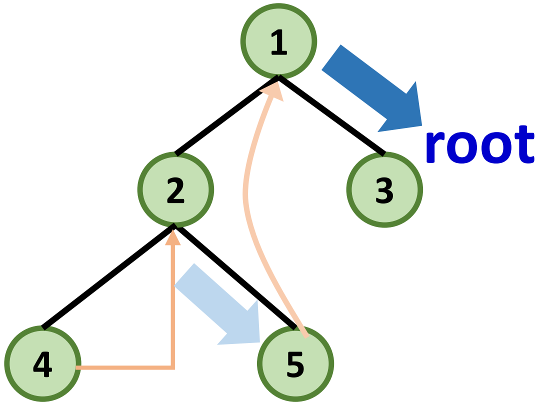


left sub-tree: Done
⇒ Visit right sub-tree

4	2	5	1	

DFS: In-order Traversal (6/6)

- **left -> root -> right**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively



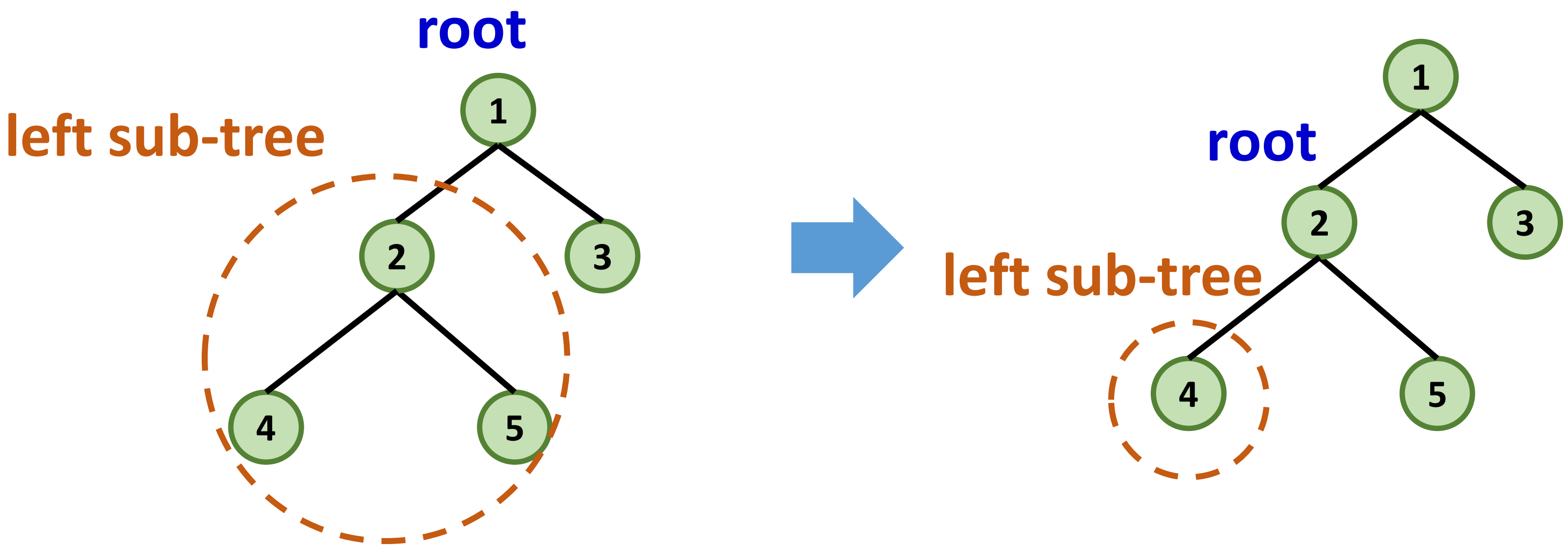
left sub-tree: Done
⇒ Visit right sub-tree

4	2	5	1	3

DONE

DFS: Post-order Traversal (1/7)

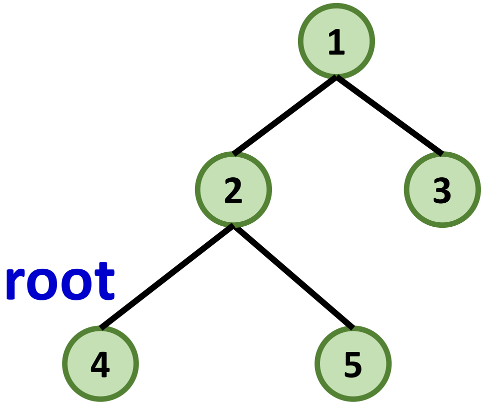
- **left -> right -> root**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively



Similar to In-order example

DFS: Post-order Traversal (2/7)

- **left -> right -> root**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively



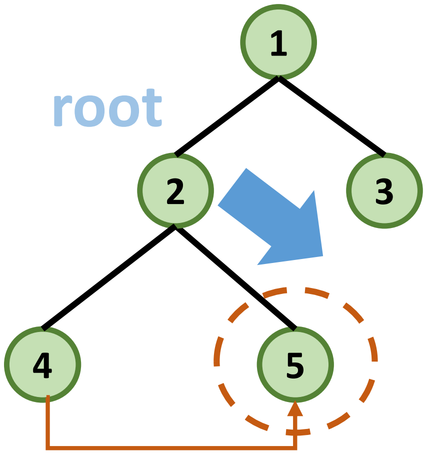
left sub-tree: Empty
right sub-tree: Empty
=> Visit root = 4

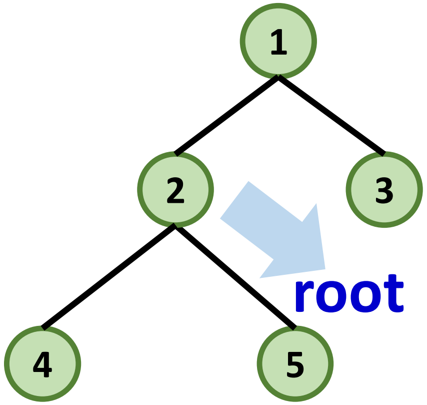
4				

Similar to In-order example

DFS: Post-order Traversal (3/7)

- **left -> right -> root**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively



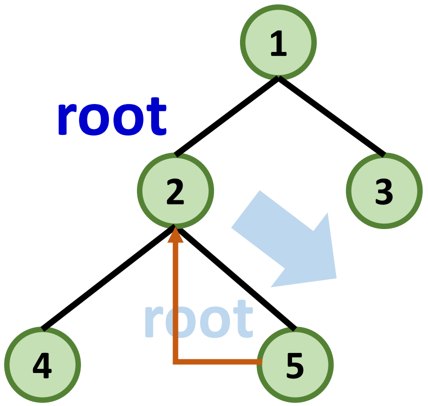


4	5			

root (4) = 2
go to right sub-tree of 2: 5
visit root = 5

DFS: Post-order Traversal (4/7)

- **left -> right -> root**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively

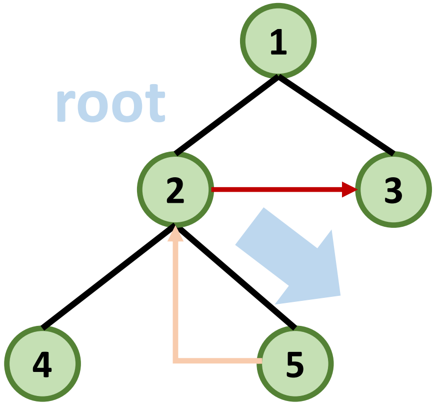


- **Left and right sub-trees of root = 2 are done**
- **Visit root = 2**

4	5	2		

DFS: Post-order Traversal (5/7)

- **left -> right -> root**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively

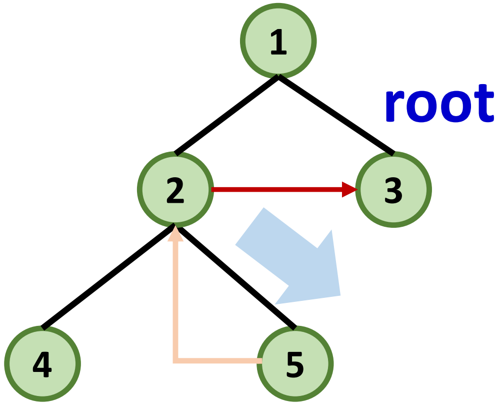


- **Left sub-tree of root = 1 is done**
- **Go to the right sub-tree of 1**

4	**5**	**2**		

DFS: Post-order Traversal (6/7)

- **left -> right -> root**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively

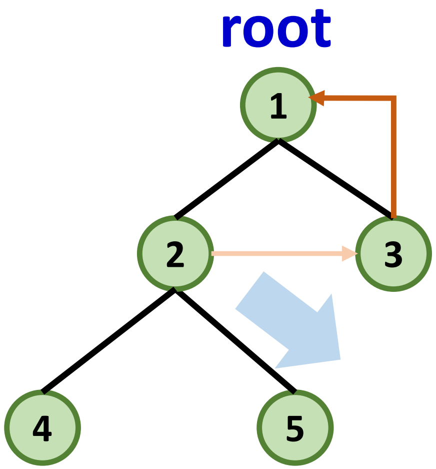


- Left and right sub-trees of root = 1: done
- Now, visit the root = 1

4	5	2	3	

DFS: Post-order Traversal (7/7)

- **left -> right -> root**
- **left** and **right: recursive calls** to **left** and **right sub-trees** respectively



DONE

4	5	2	3	1

BFS (Breadth First Search) Traversal

- **Idea**
  - Use of a **queue** data structure
  - A node is **traversed** when its **all successor nodes are generated, and queued**

- **Demo**: Use of a coloring scheme

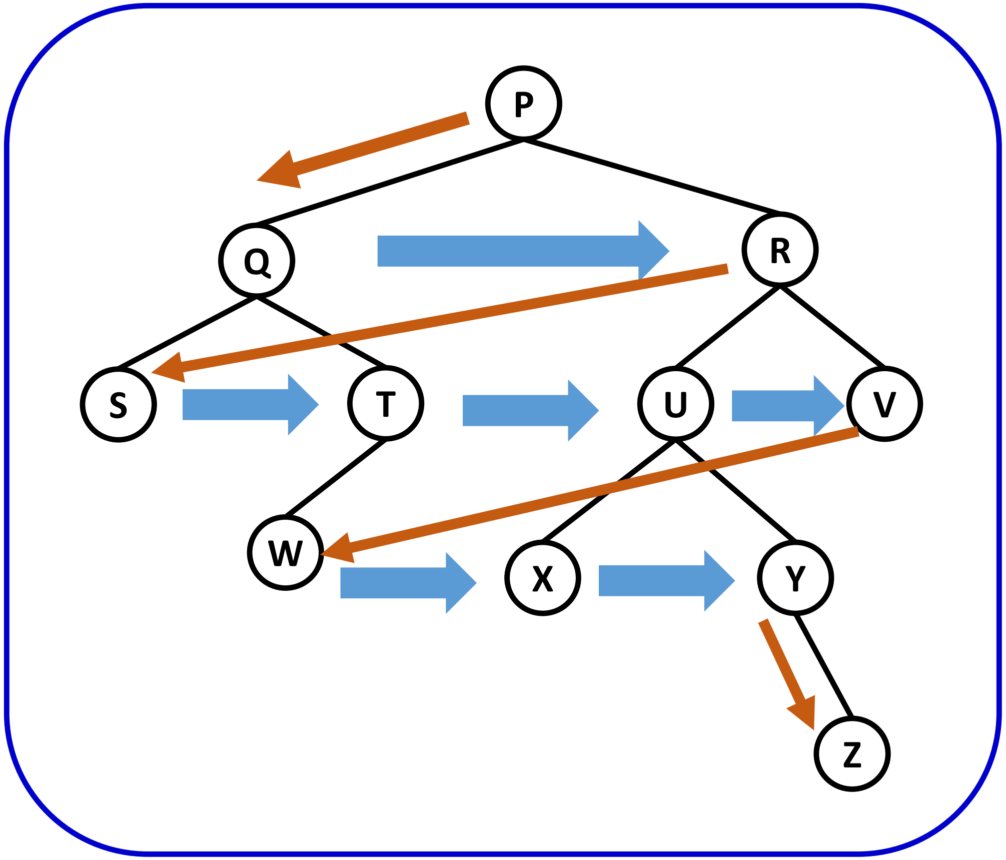
🟠 **Orange: Encountered but not traversed**

🟢 **Green: traversed**

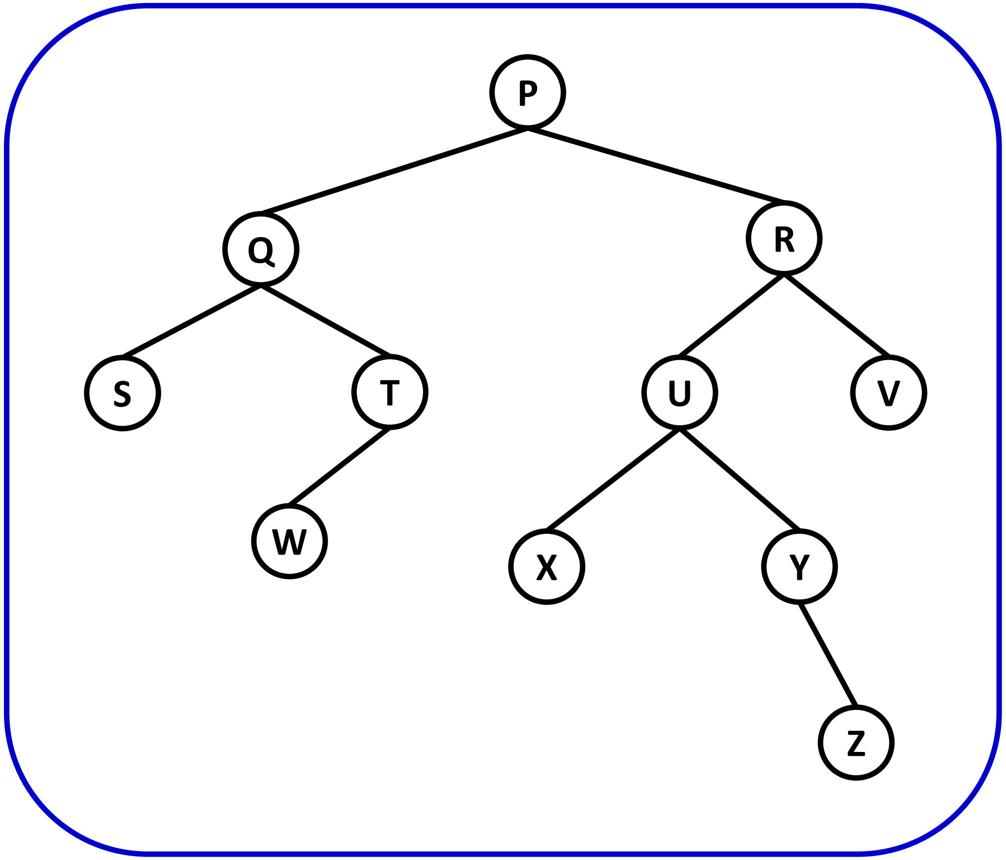
○ **White: not accessed/encountered/generated**

BFS Traversal

- **Traverse all nodes of same level before going to the next level**
- Note: A node is **traversed** when its **all successor nodes are generated, and queued**

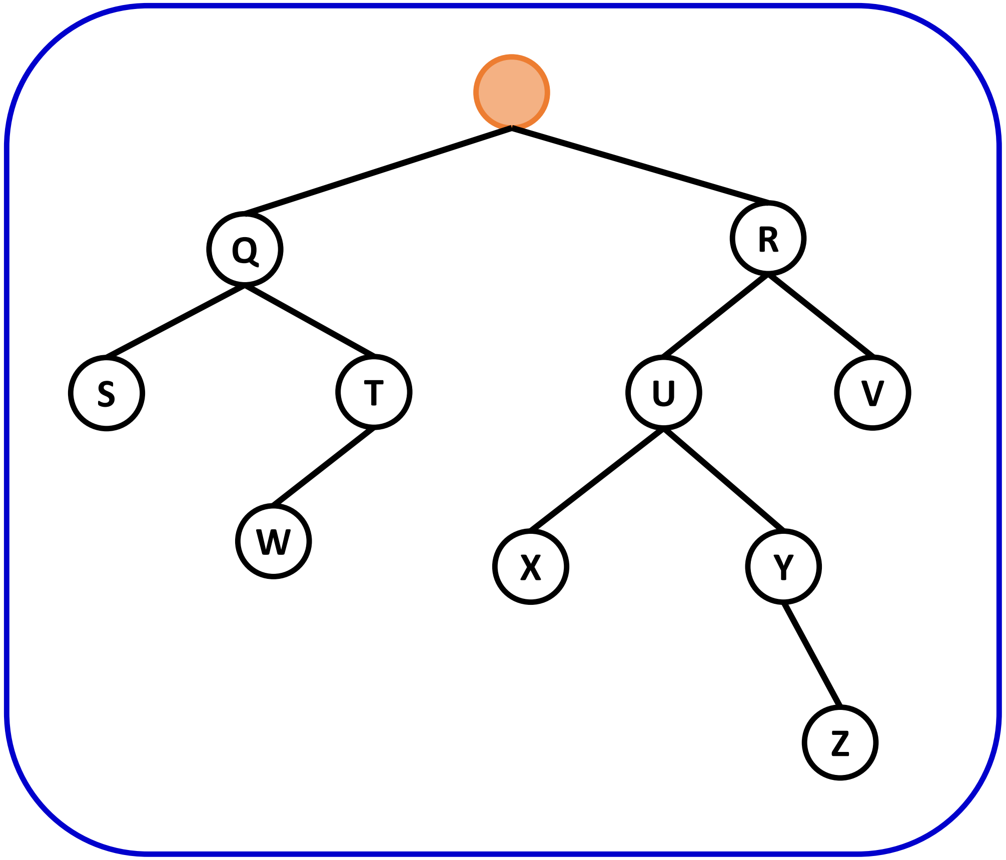


BFS Implementation: Step-1



P				

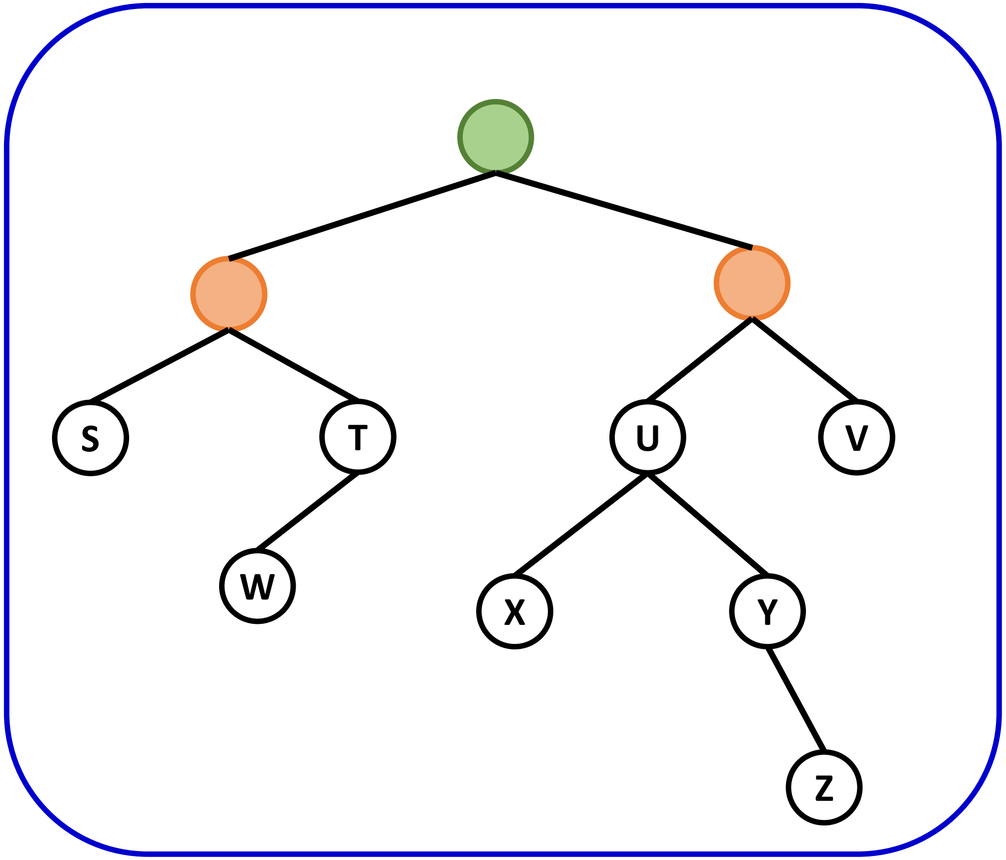
BFS Implementation: Step-2



Queue

Q	R			

BFS Implementation: Step-3

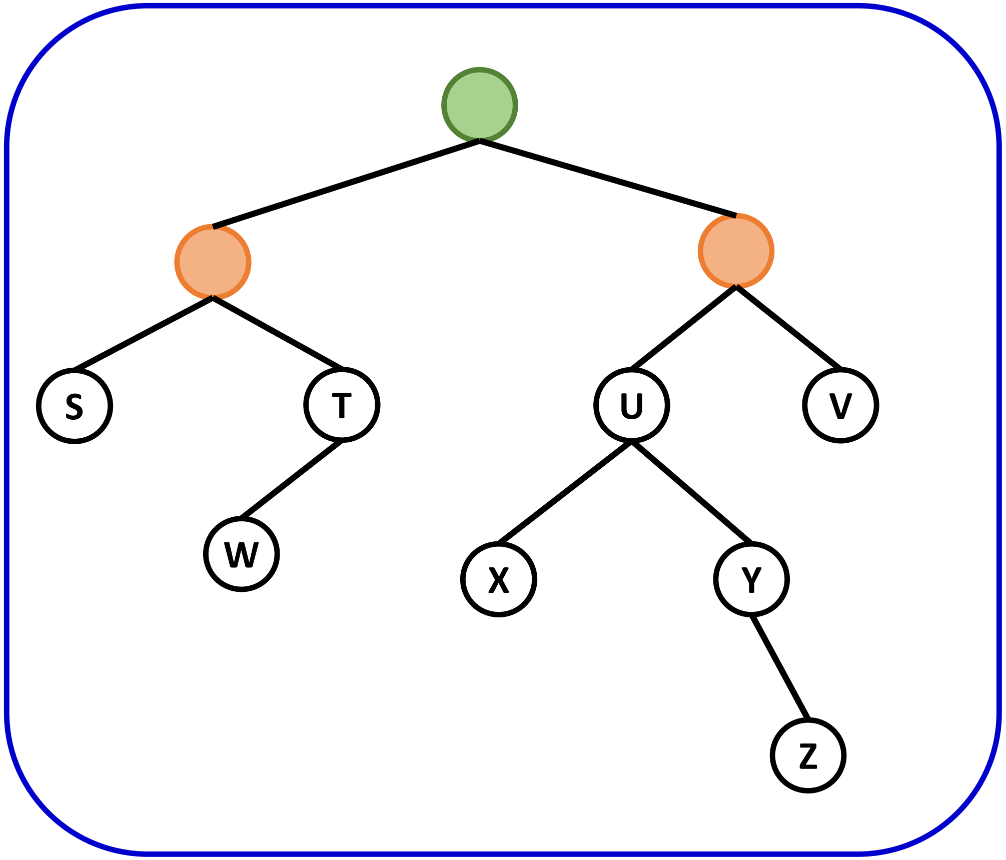


Queue

Q	R			

List of Nodes Traversed: P

BFS Implementation: Step-4

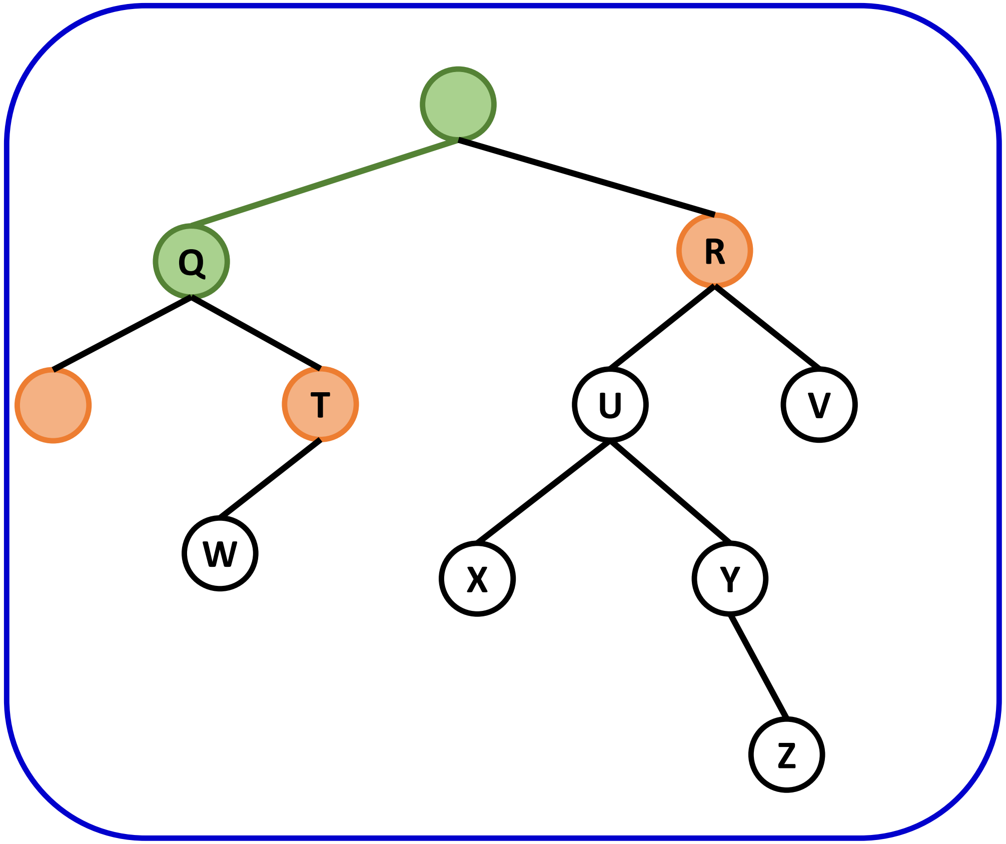


Queue

Q	R	S		

List of Nodes Traversed: P

BFS Implementation: Step-5

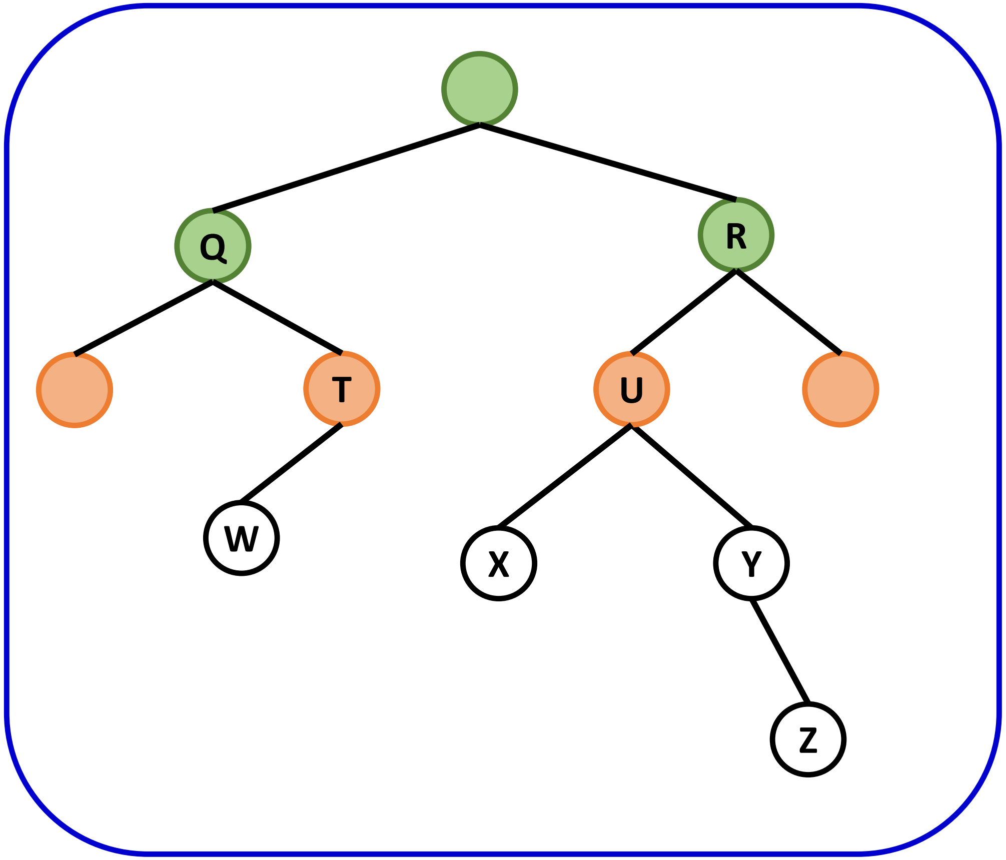


Queue

S	T	U	V	

List of Nodes Traversed: P, Q

BFS Implementation: Step-6

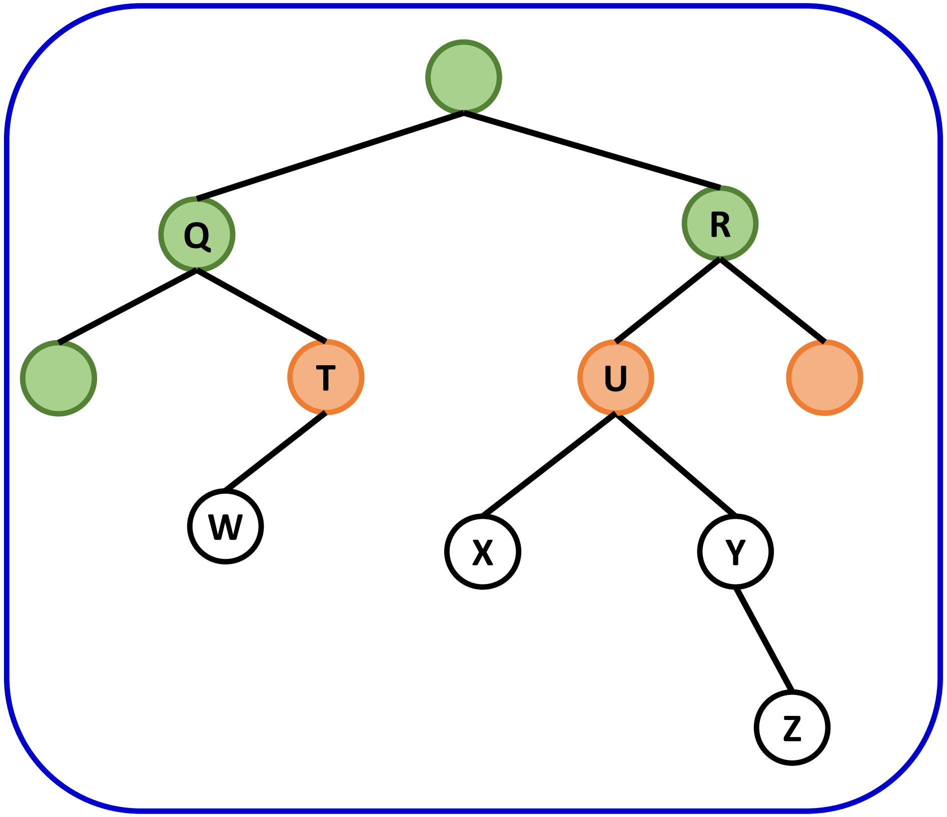


Queue

S	T	U	V	

List of Nodes Traversed: P, Q, R

BFS Implementation: Step-7

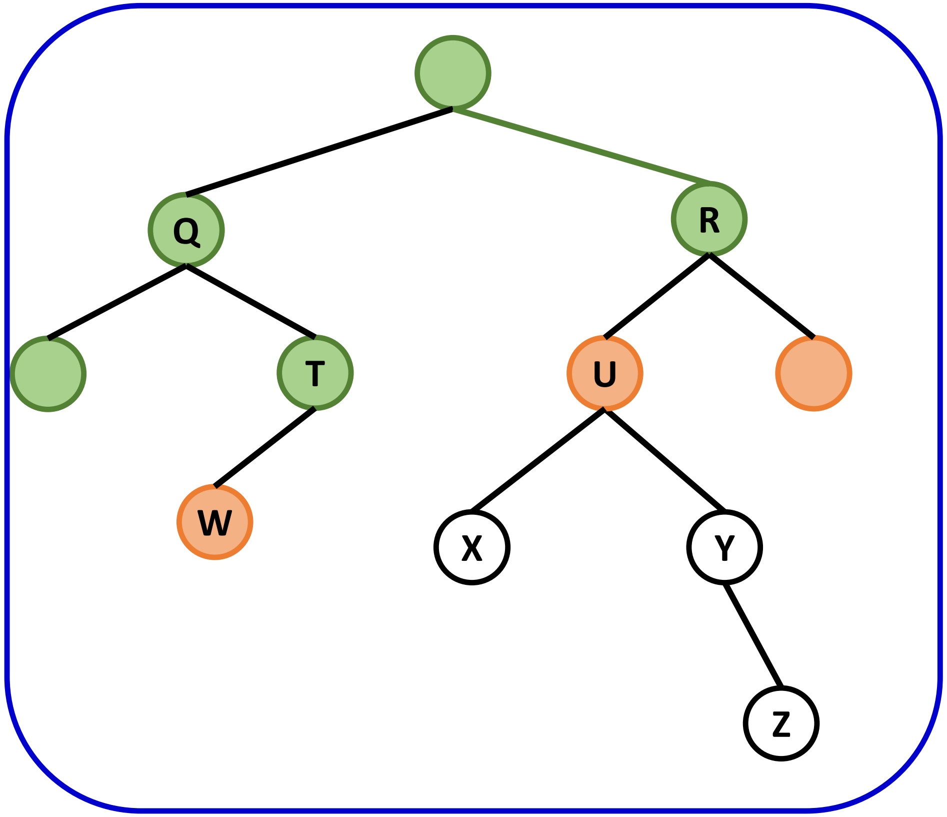


Queue

U	V	W		

List of Nodes Traversed: P, Q, R, S

BFS Implementation: Step-8

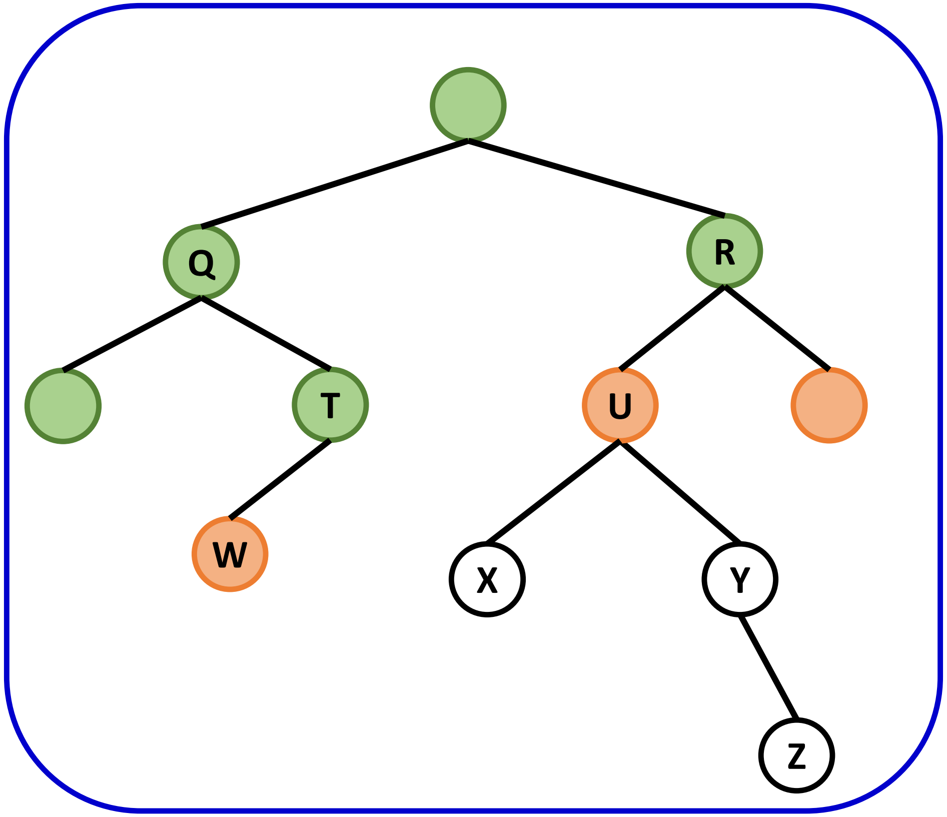


Queue

U	V	W		

List of Nodes Traversed: P, Q, R, S, T

BFS Implementation: Step-9

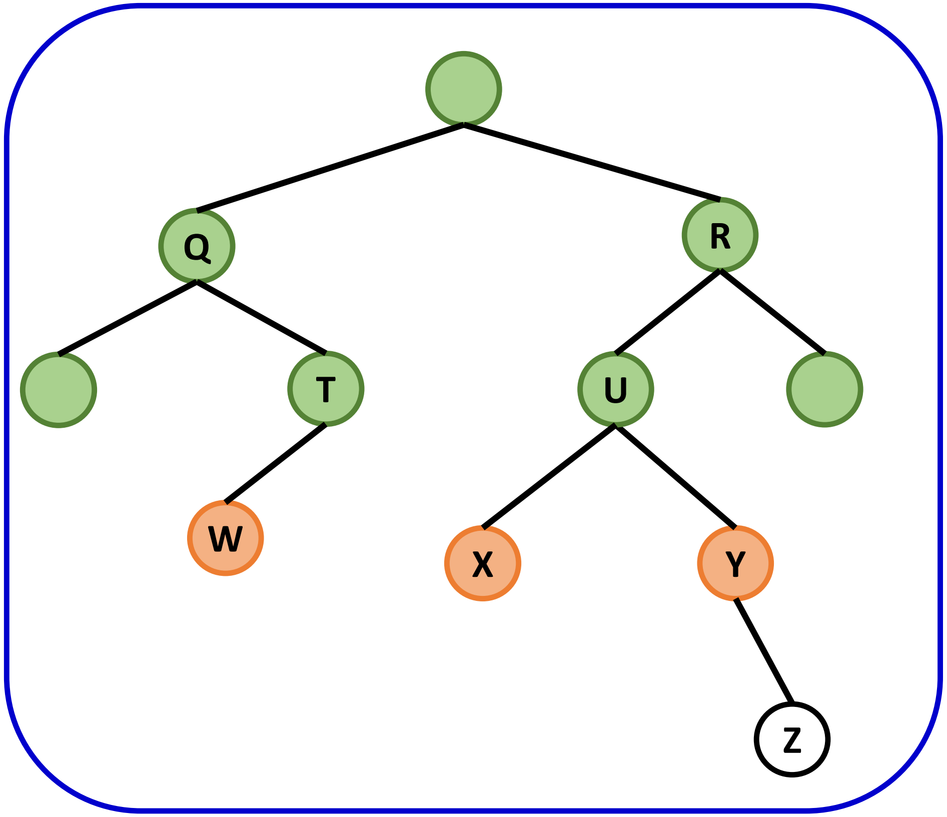


Queue

V	W	X	Y	

List of Nodes Traversed: P, Q, R, S, T

BFS Implementation: Step-10

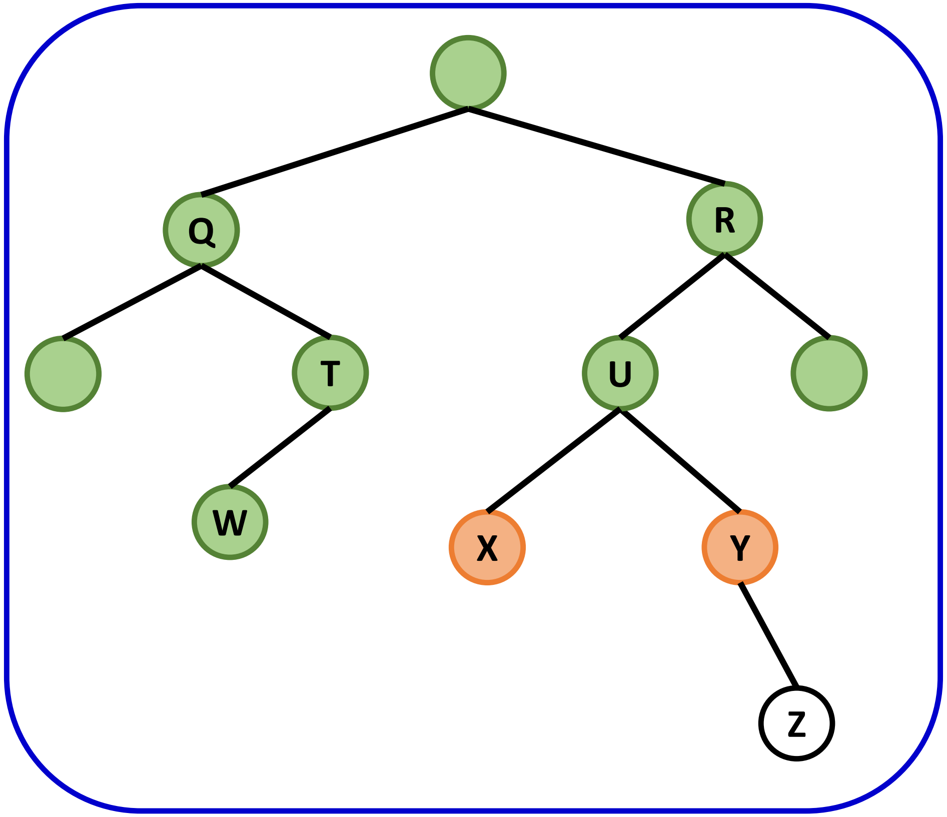


Queue

W	X	Y		

List of Nodes Traversed: P, Q, R, S, T, U, V

BFS Implementation: Step-11

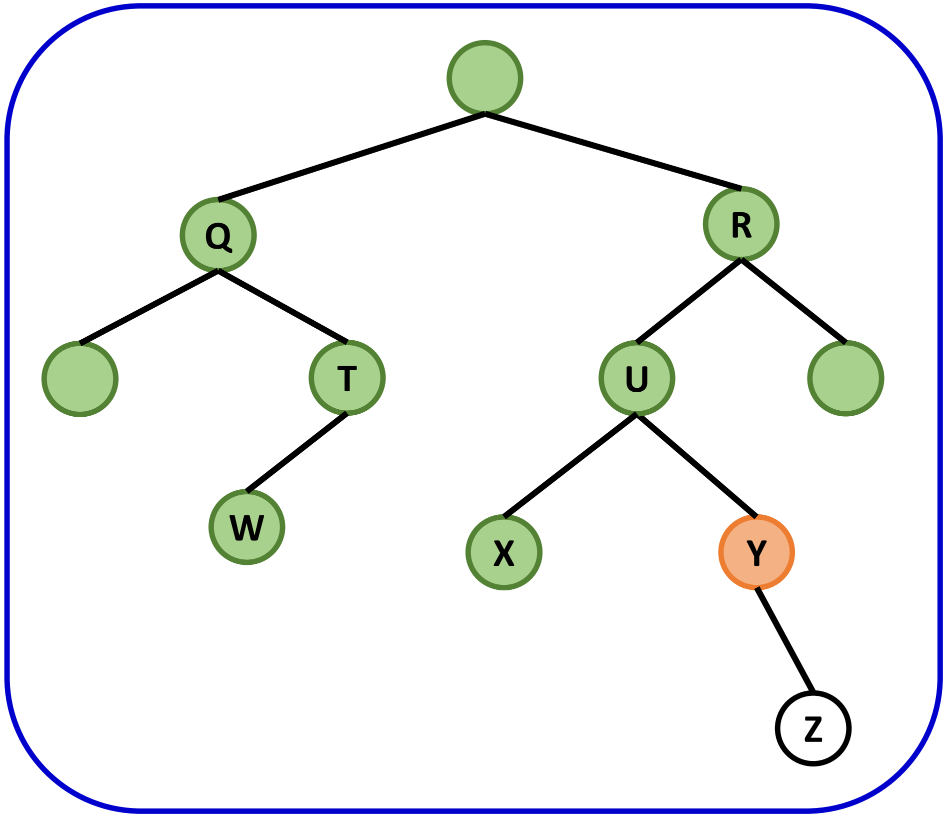


Queue

X	Y			

List of Nodes Traversed: P, Q, R, S, T, U, V, W

BFS Implementation: Step-12

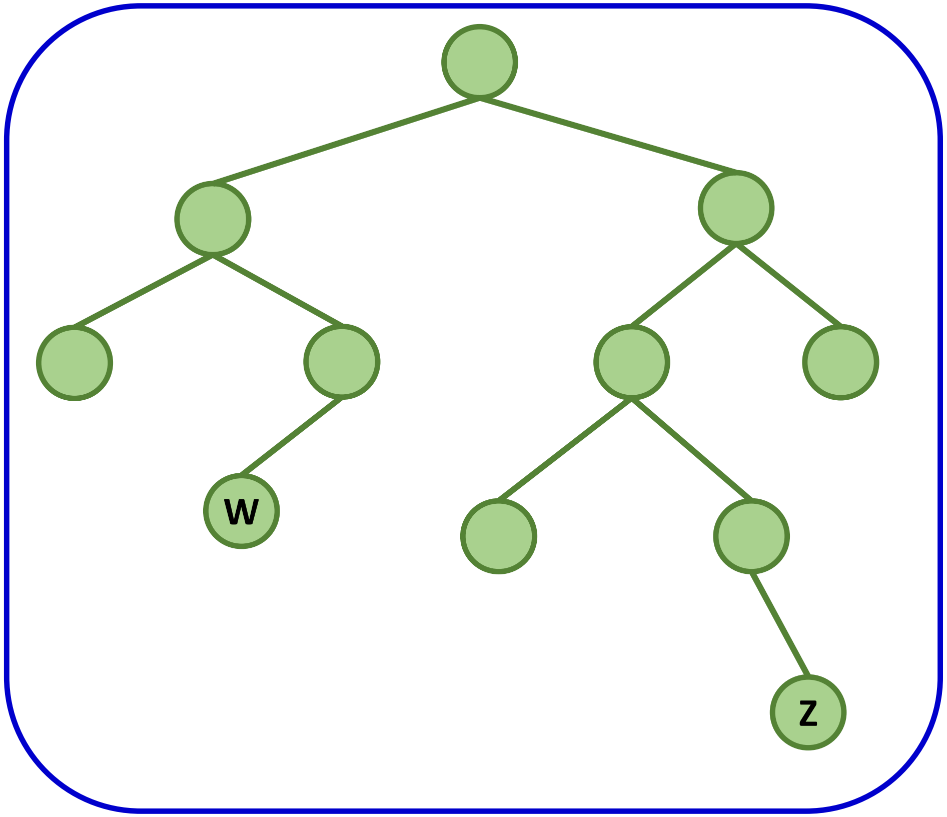


Queue

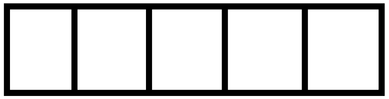
Y	Z			

List of Nodes Traversed: **P, Q, R, S, T, U, V, W, X,**

BFS Implementation: Step-13



Queue



DONE !

List of Nodes Traversed: **P, Q, R, S, T, U, V, W, X, Y, Z**

Binary Tree Traversals

In **preorder**, the **root** is visited **first**

```
public void preorder(BinaryTree bt)
{
   if (bt == null) return;
   printf(bt.value);
   preorder (bt.leftChild);
   preorder (bt.rightChild);
}
```

In **inorder**, the **root** is visited *in the **middle***

```
public void inorder (BinaryTree bt)
{
   if (bt == null) return;
   inorder(bt.leftChild);
   printf(bt.value);
   inorder(bt.rightChild);
}
```

In **postorder**, the **root** is visited **last**

```
public void postorder(BinaryTree bt)
{
     if (bt == null) return;
     postorder(bt.leftChild);
     postorder(bt.rightChild);
     printf(bt.value);
}
```

Other Traversals

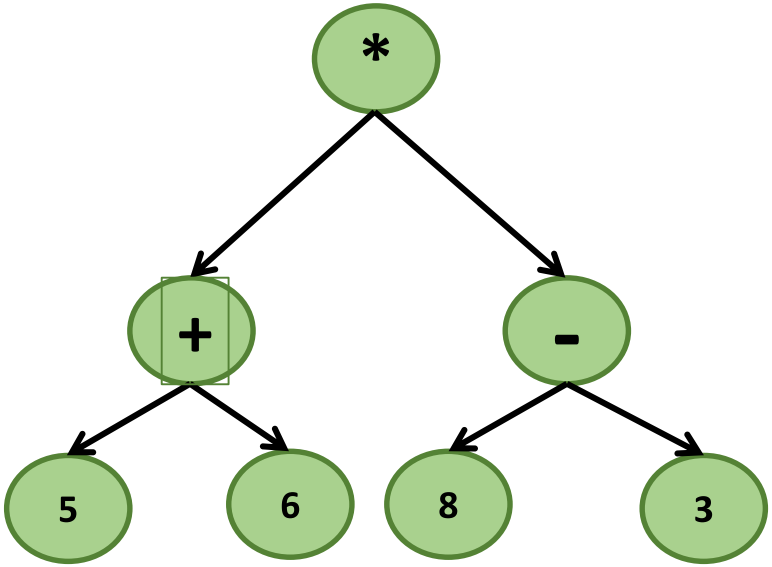
- Other traversals are the reverse of these three standard ones
  - The right subtree is traversed before the left subtree

- Reverse preorder: **root => right subtree => left subtree**

- Reverse in-order: **right subtree => root => left subtree**

- Reverse post-order: **right subtree => left subtree => root**

Application of Binary Tree



((5 + 6) * (8-3))

Summary

- A non-linear, hierarchical, and recursive data structures
- Form the basis of many useful and efficient data structures
- Traversals
  - Depth first
    - Preorder, Inorder, Postorder
  - Breadth First Traversal
- Applications of Binary Trees
  - Expression Trees
  - Huffman coding

Thank you!