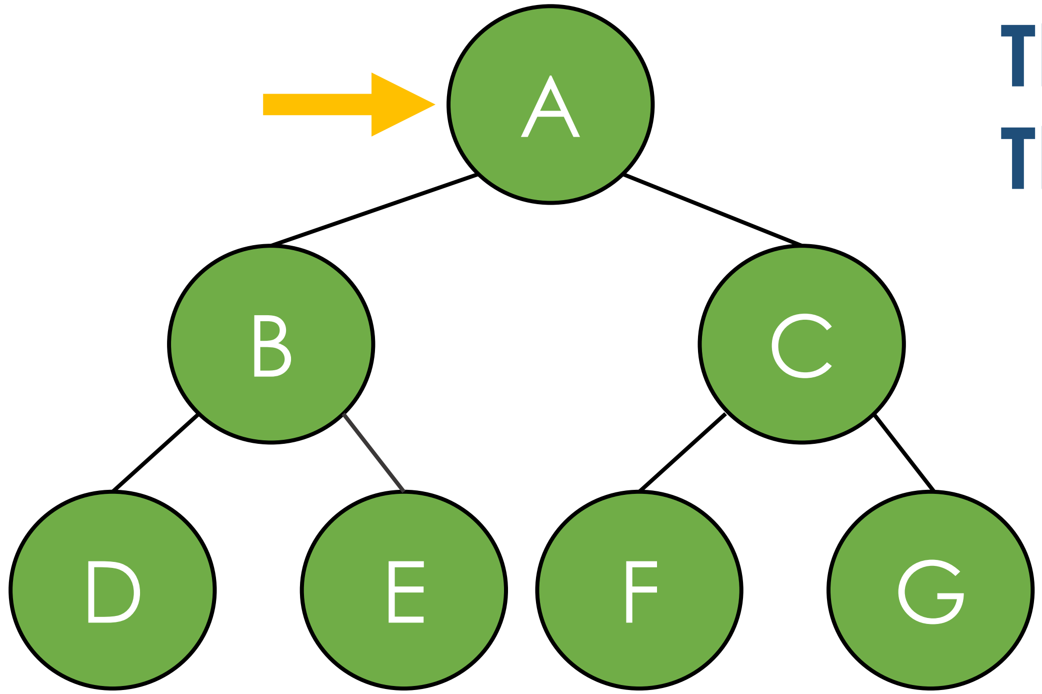
Tree Traversals Part 2

By the end of this video you will be able to…

- Perform in-order and post-order traversals

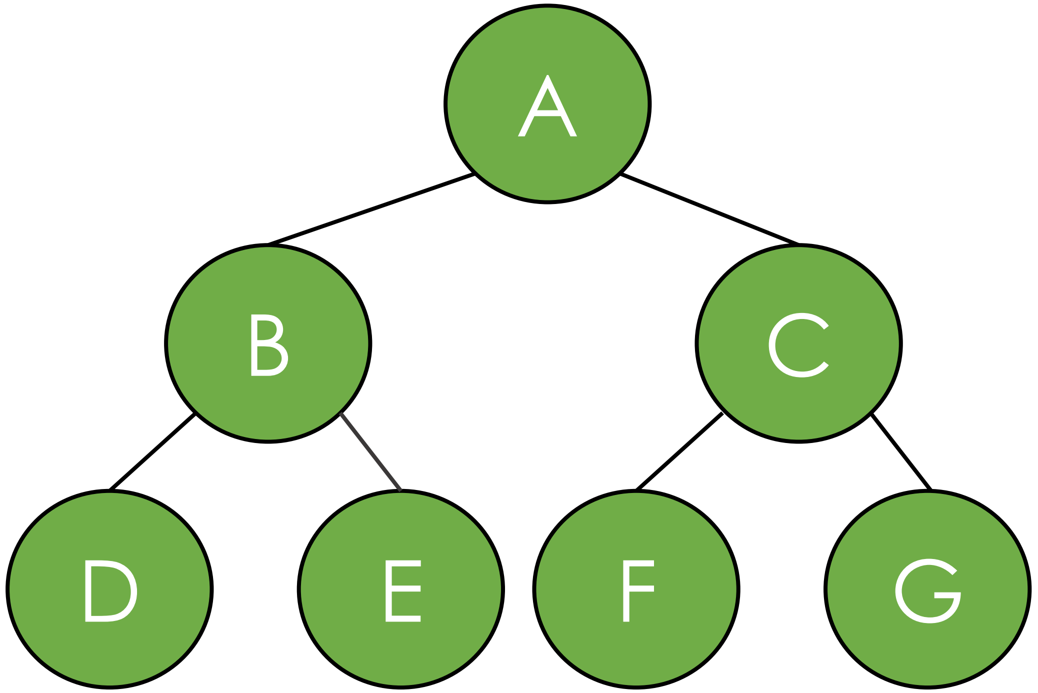
Preorder Traversal



Idea:
Visit yourself
Then visit all your left subtree
Then visit all your right subtree

Visited:
A B D E C F G

PostOrder Traversal



Visit:

D E B F G C A

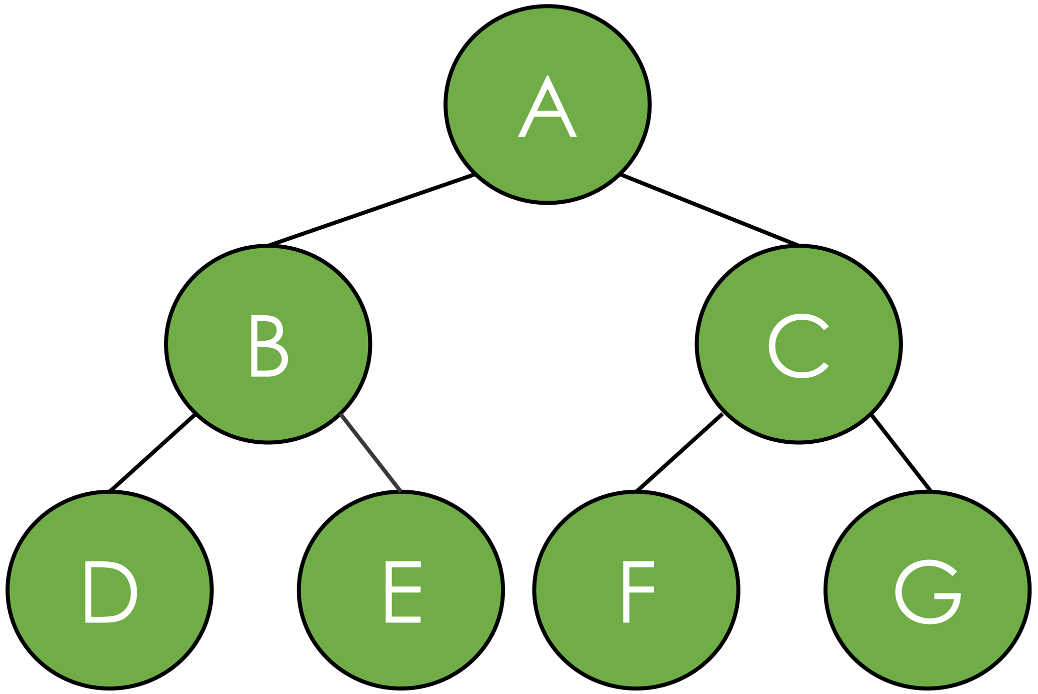
<u>REARRANGE</u>

Visit yourself

Visit all your left subtree

Visit all your right subtree

PostOrder Traversal



Visit:

D E B F G C A

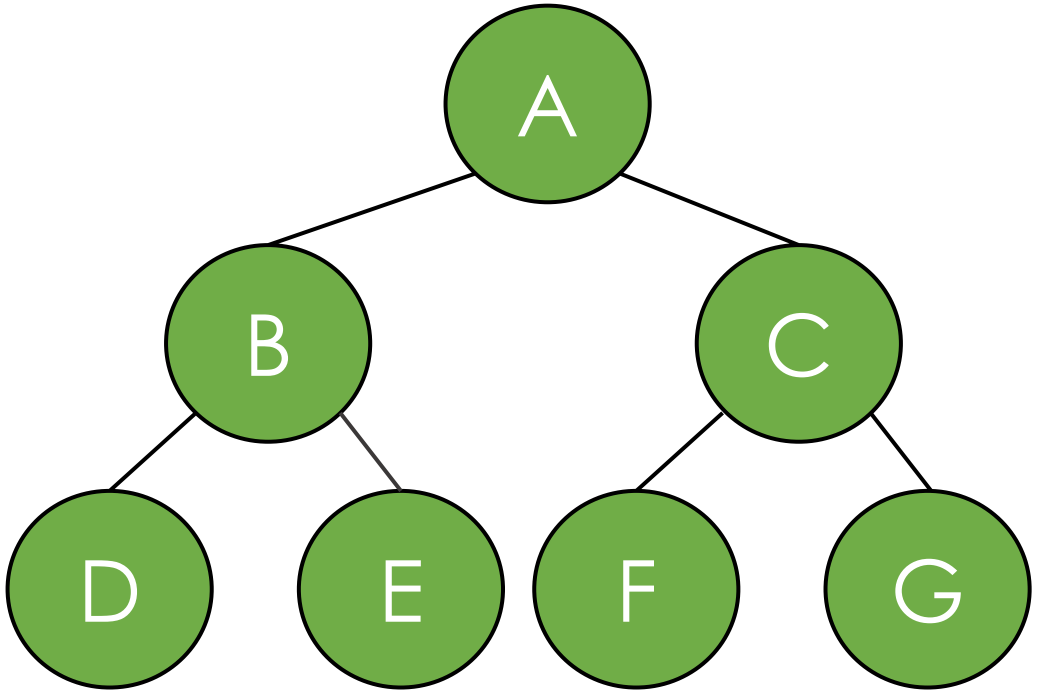
<u>**REARRANGE**</u>

Visit yourself

Visit all your left subtree

Visit all your right subtree

PostOrder Traversal



Visit:

D E B F G C A

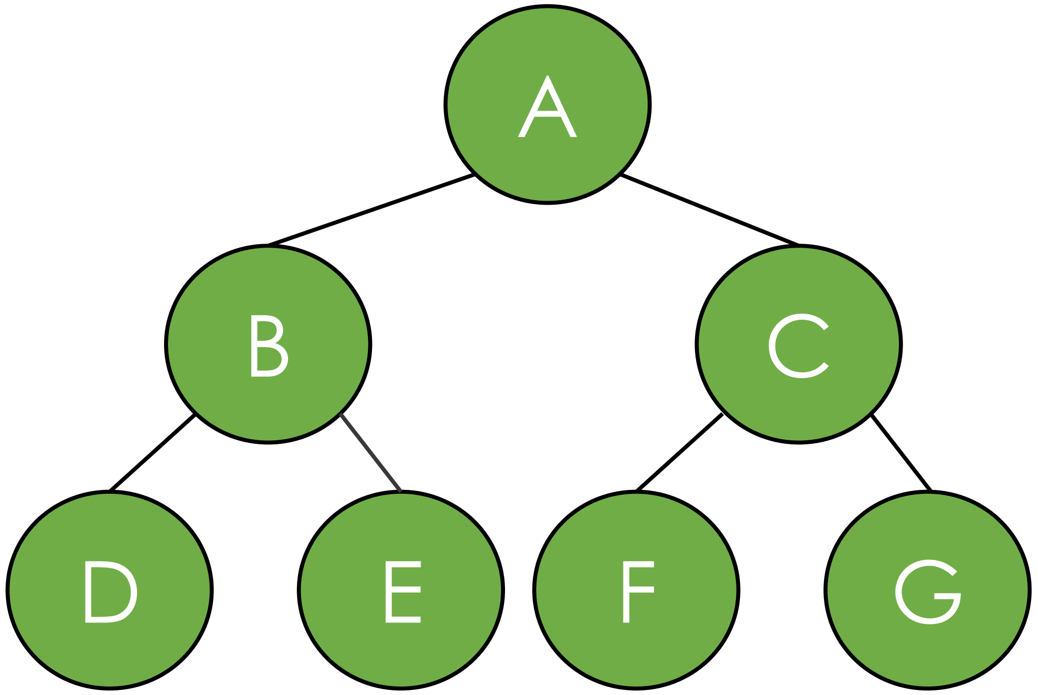
<u>**REARRANGE**</u>

Visit yourself

Visit all your left subtree

Visit all your right subtree

PostOrder Traversal



Visit:

D E B F G C A

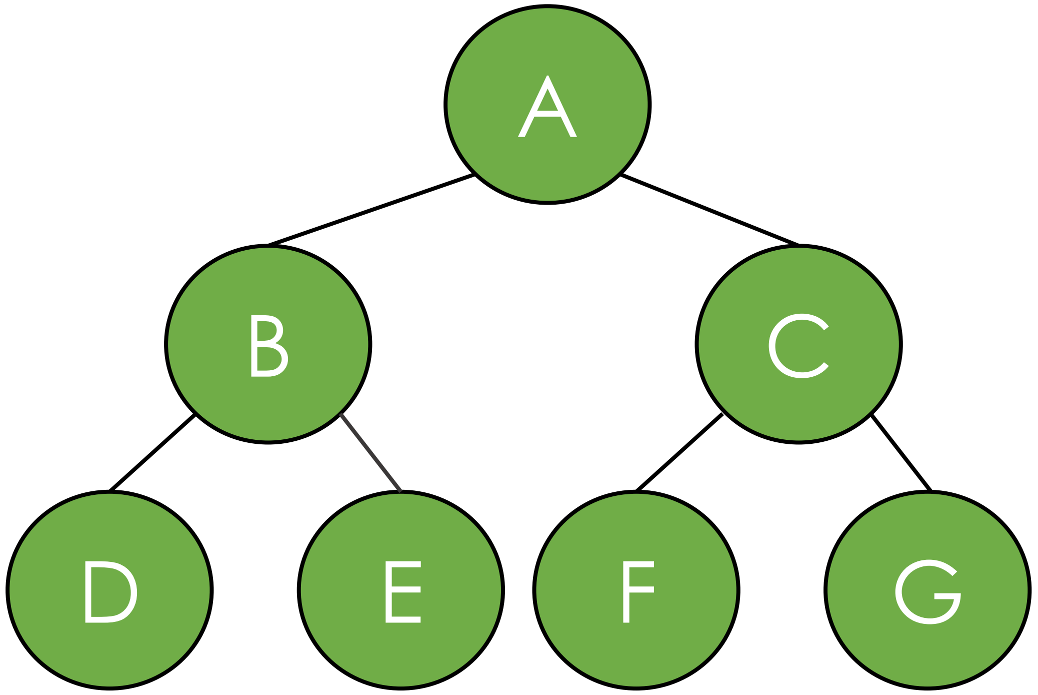
REARRANGE

Visit all your left subtree

Visit yourself

Visit all your right subtree

InOrder Traversal



Visit:

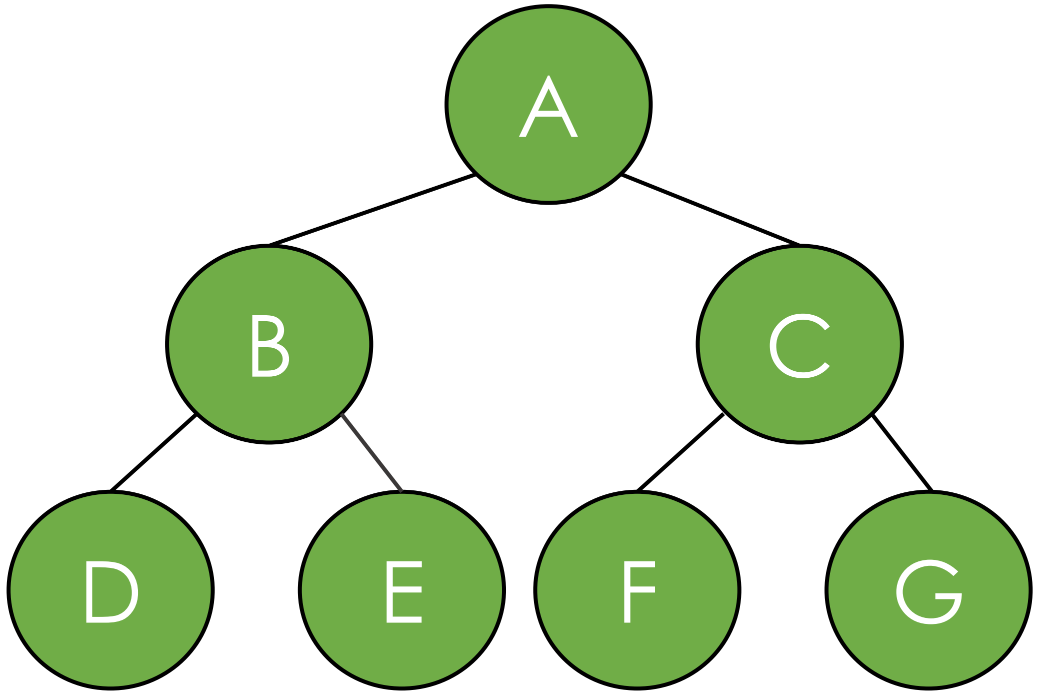
What does this do?

Visit all your left subtree

Visit yourself

Visit all your right subtree

InOrder Traversal



Visit:

What does this do?

Visit all your left subtree

Visit yourself

Visit all your right subtree

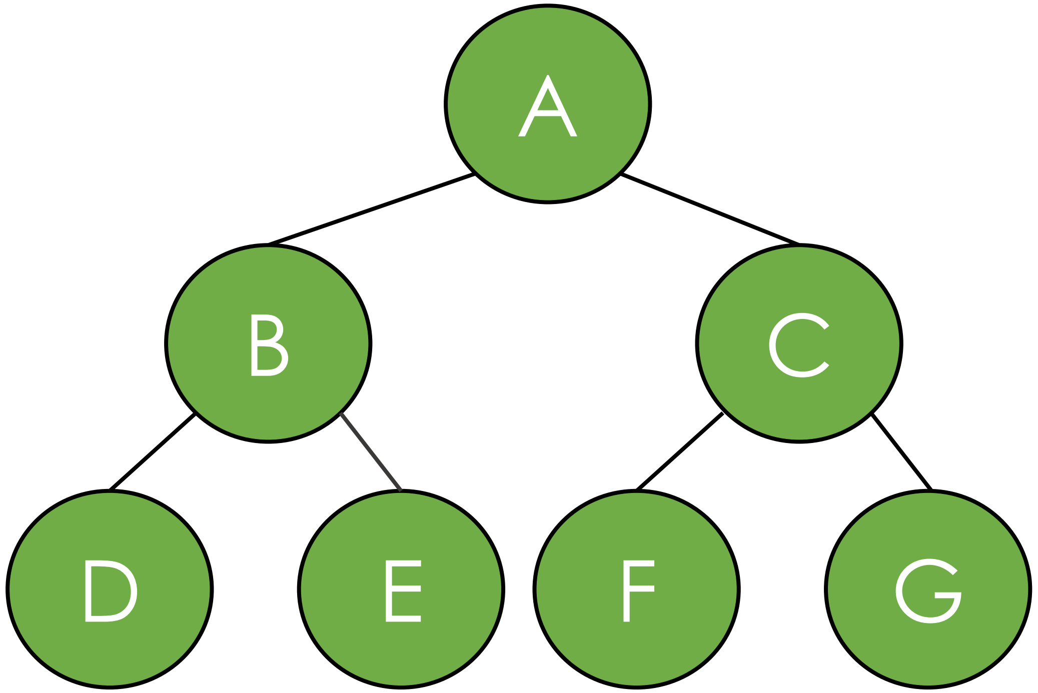
Fill in the Blank:

A. A B C D E F G

B. D B E A F C G

C. D B A E F C G

InOrder Traversal



Visit:

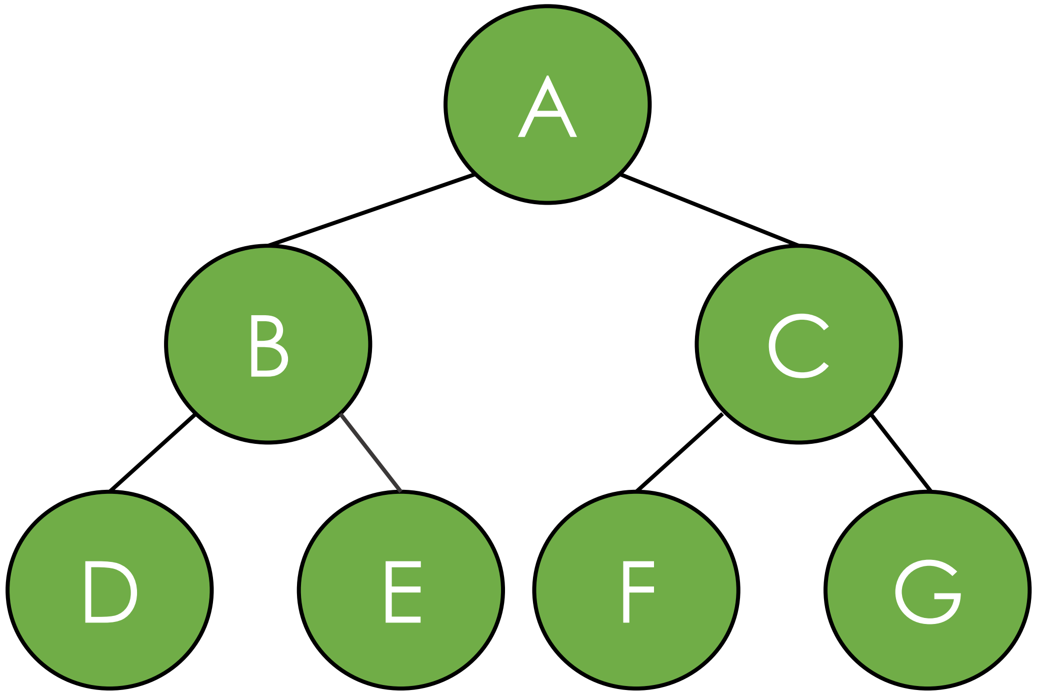
What does this do?

Visit all your left subtree

Visit yourself

Visit all your right subtree

InOrder Traversal



Visit:

D B E A F C G

What does this do?

Visit all your left subtree

Visit yourself

Visit all your right subtree

Next step

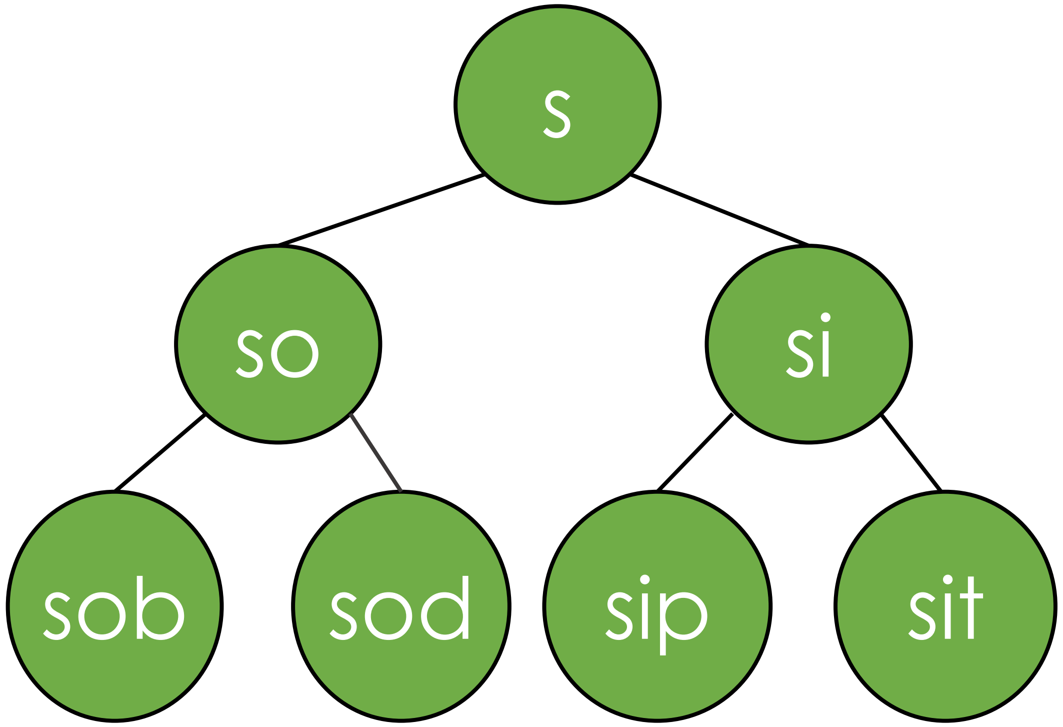
- Level Order Traversal (needed for the project)

Tree Traversals Part 3

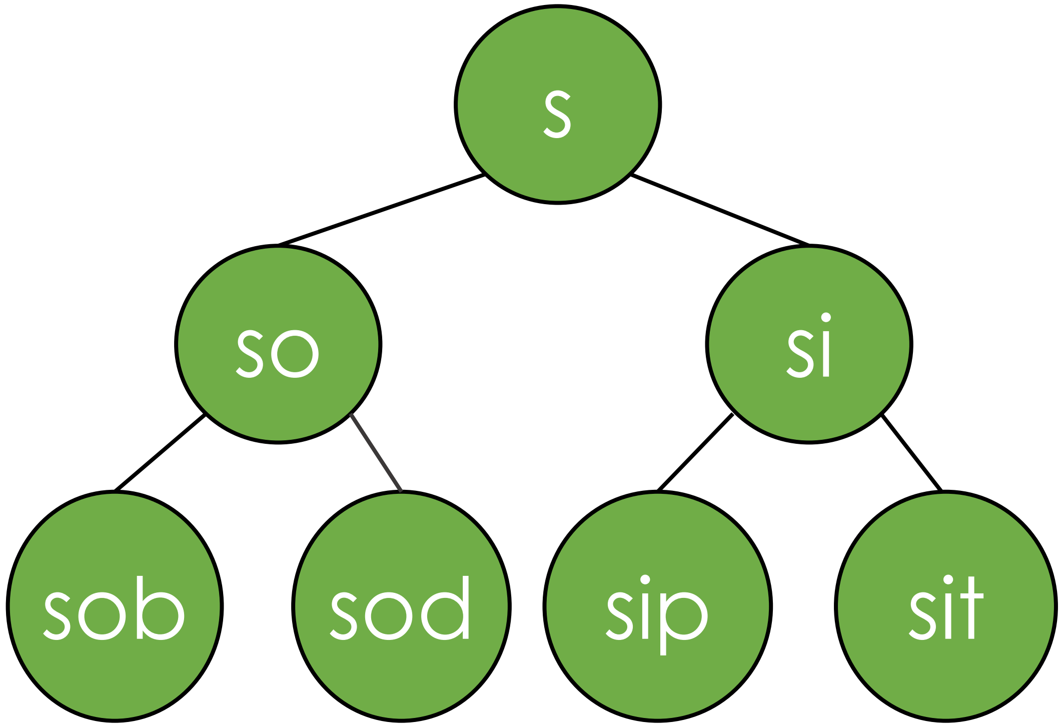
By the end of this video you will be able to…

- Perform a level-order traversal

Traversals Revisited - autocomplete

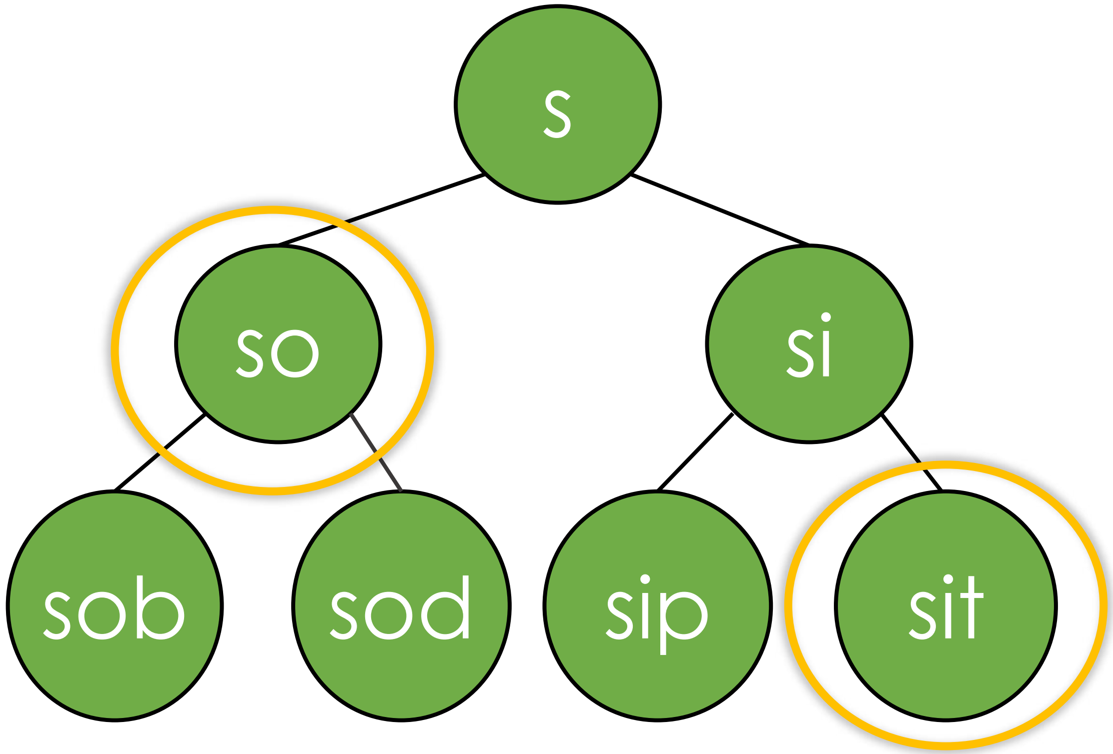


Traversals Revisited - autocomplete



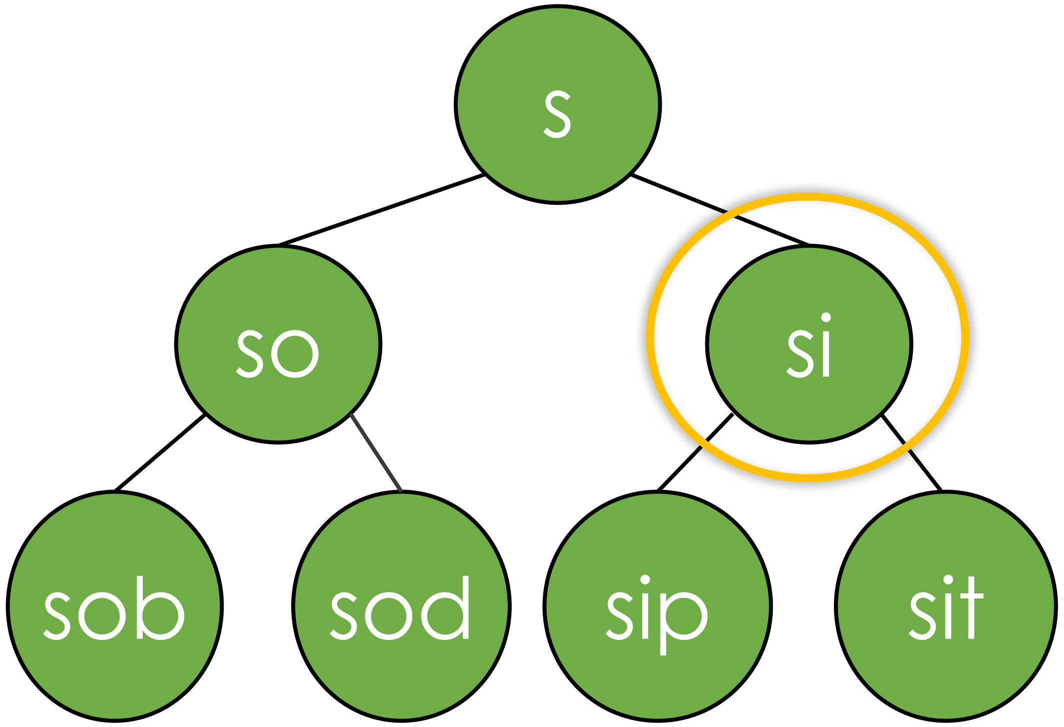
You've typed "s"
What words should we suggest?

Traversals Revisited - autocomplete



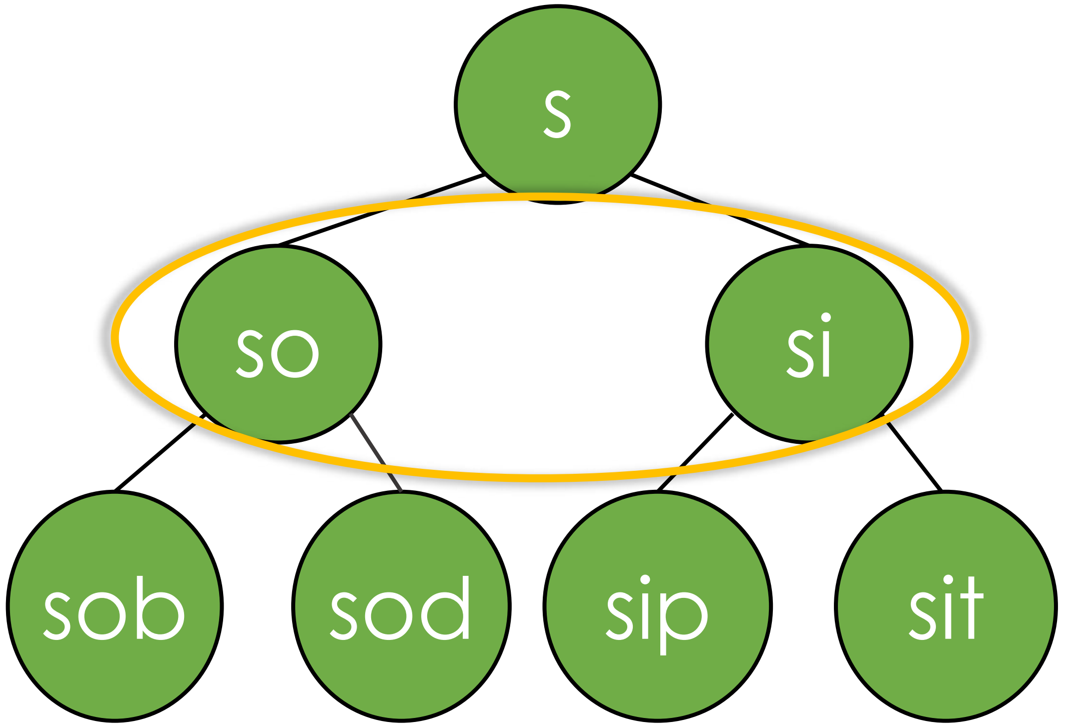
You've typed "s"
Most frequent?

Traversals Revisited - autocomplete



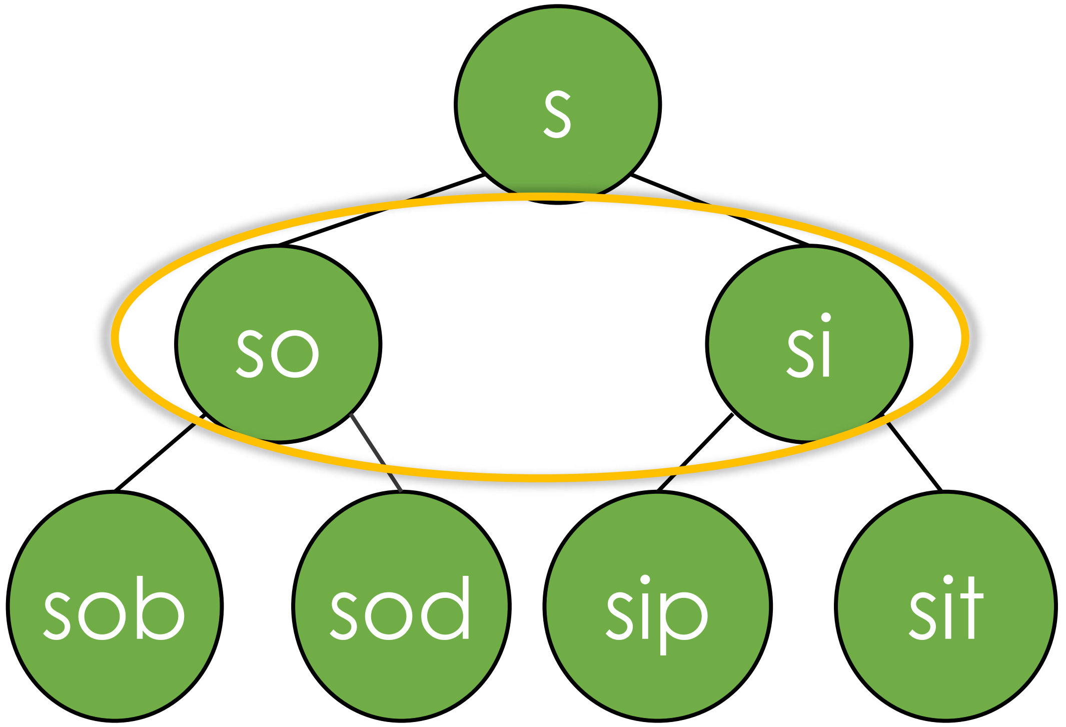
**You've typed "s"
Most frequent for whom?**

Traversals Revisited - autocomplete



You've typed "s"
How about
"closest"?

Traversals Revisited - autocomplete



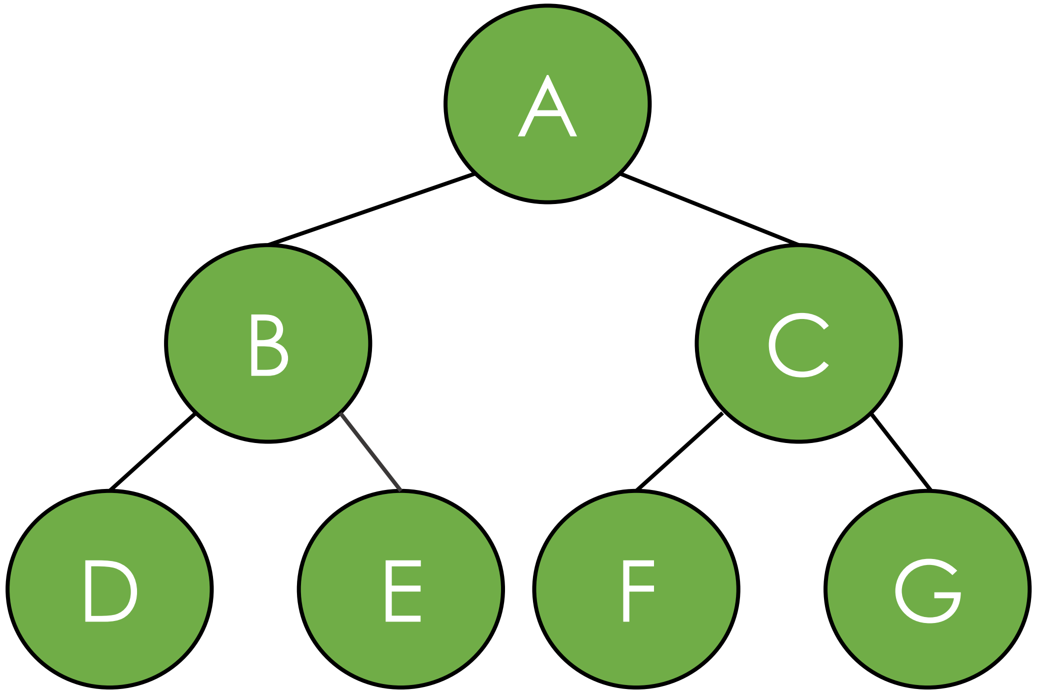
You've typed "s"
How about
"closest"?

"Breadth First Traversal"

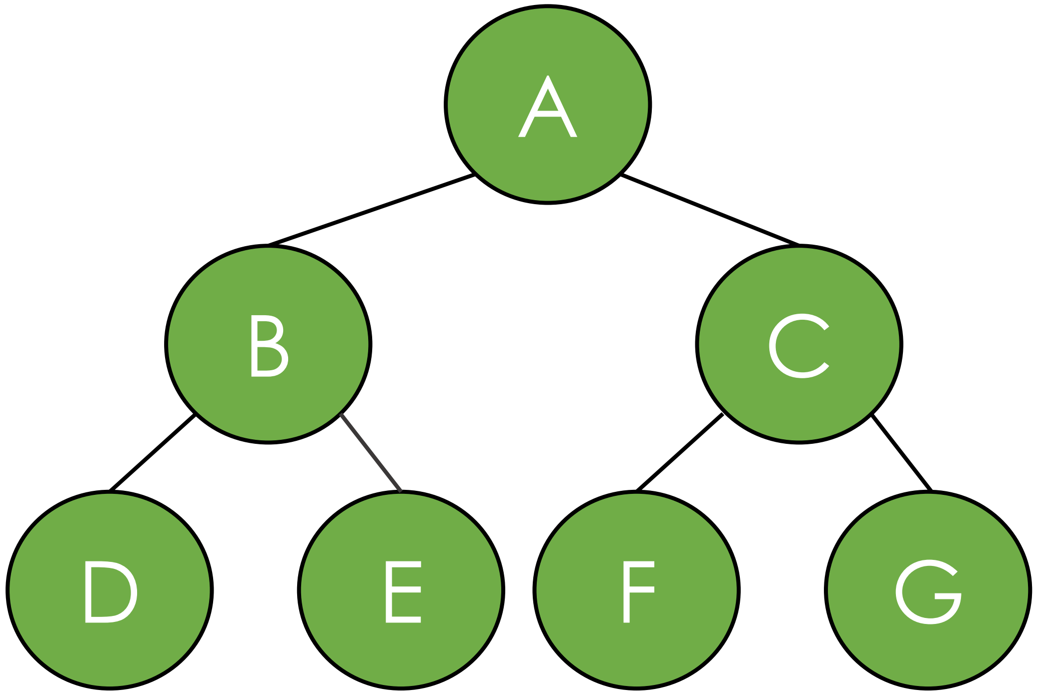
Level Order Traversal

Visit:

A B C D E F G



Level Order Traversal



Visit:

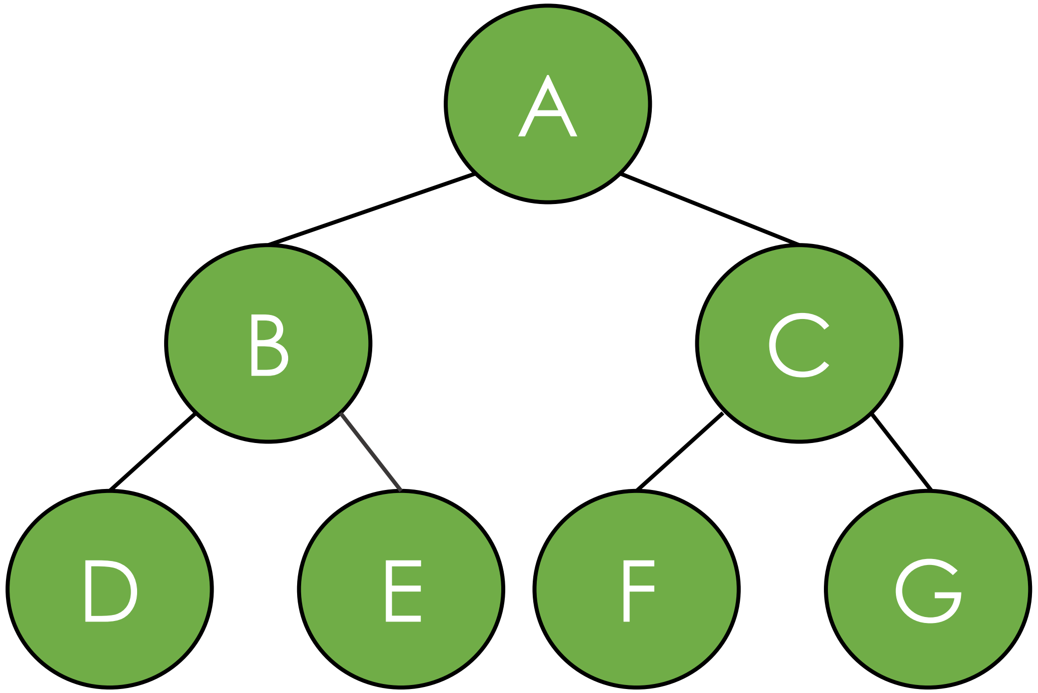
A B C D E F G

"Breadth First Traversal"

Post/Pre Order are:

"Depth First Traversals"

Level Order Traversal

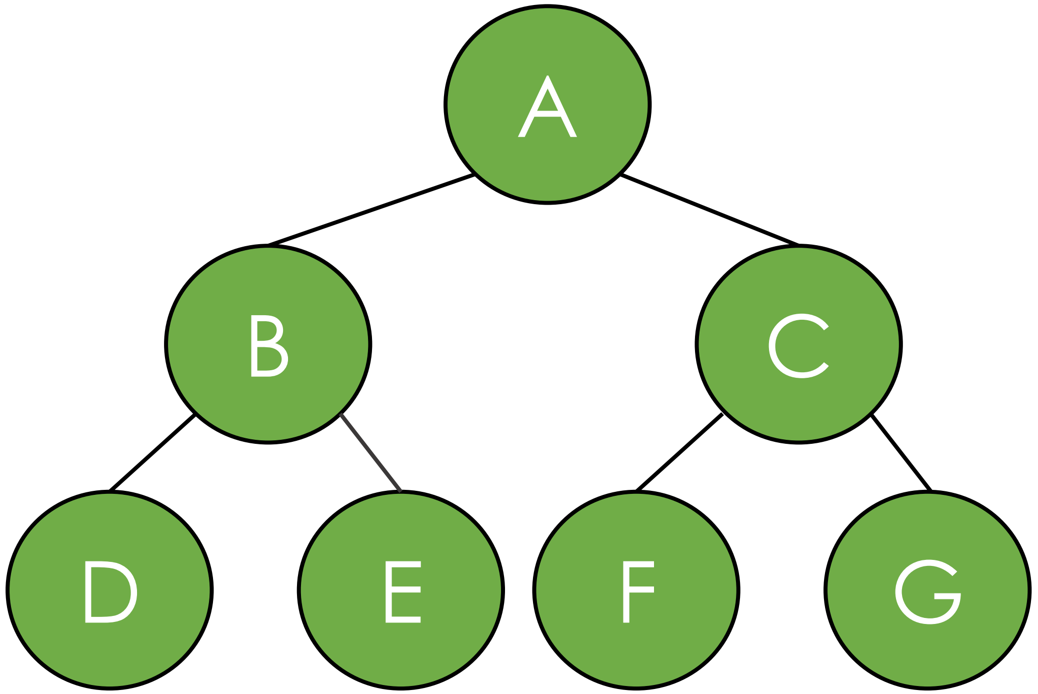


Visit:

A B C D E F G

Challenging:

Level Order Traversal

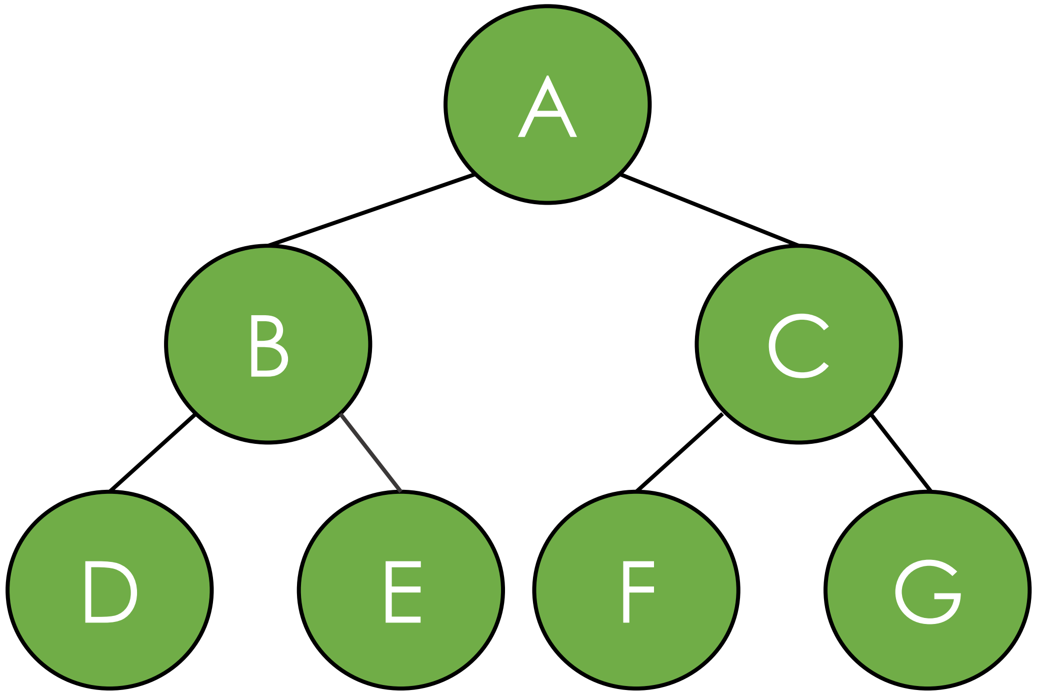


Visit:

A B C D E F G

Challenging: When we finish B, how do we know to go to C next?

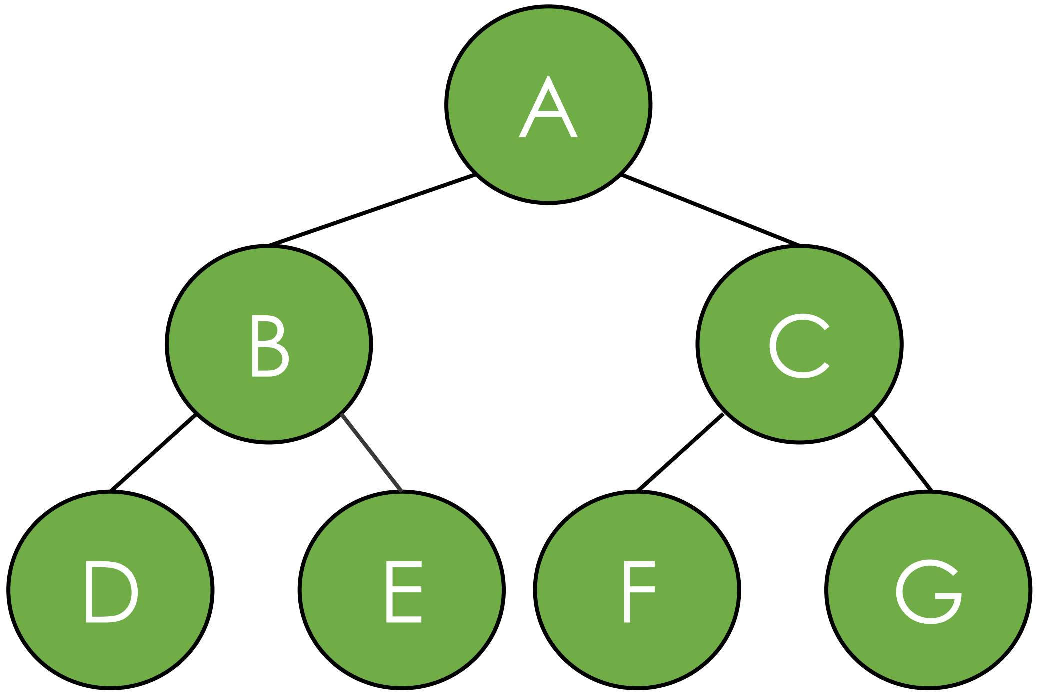
Level Order Traversal



Visit:
A B C D E F G

Idea: Keep a list and keep adding to it and removing from start.

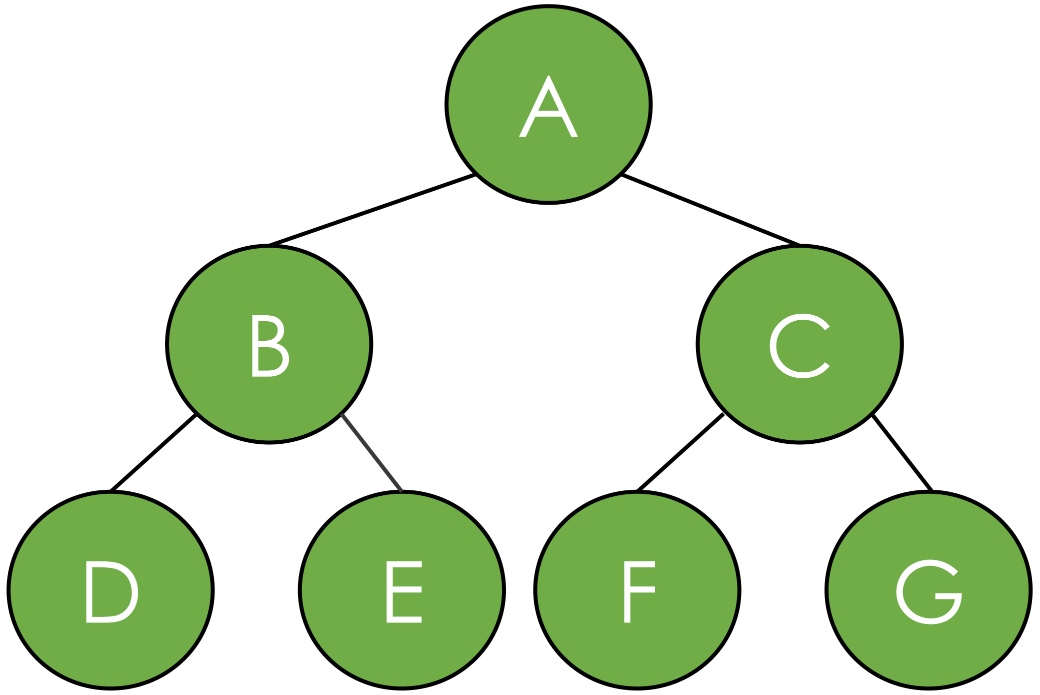
Level Order Traversal



Visited:

List: A

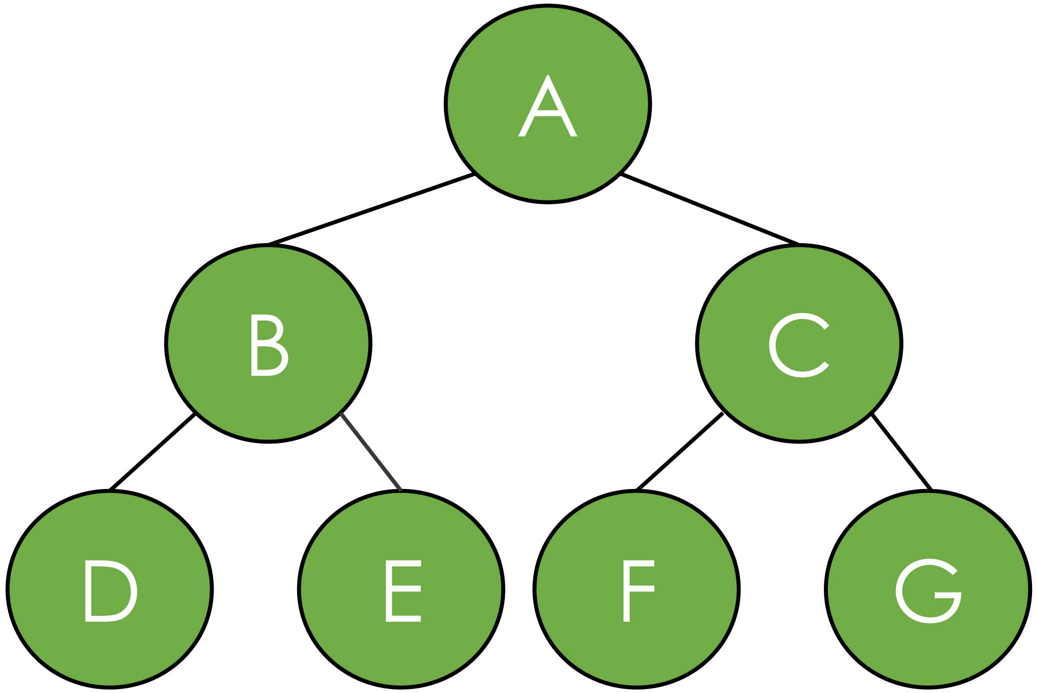
Level Order Traversal



Visited:
A

List: ~~**A**~~

Level Order Traversal

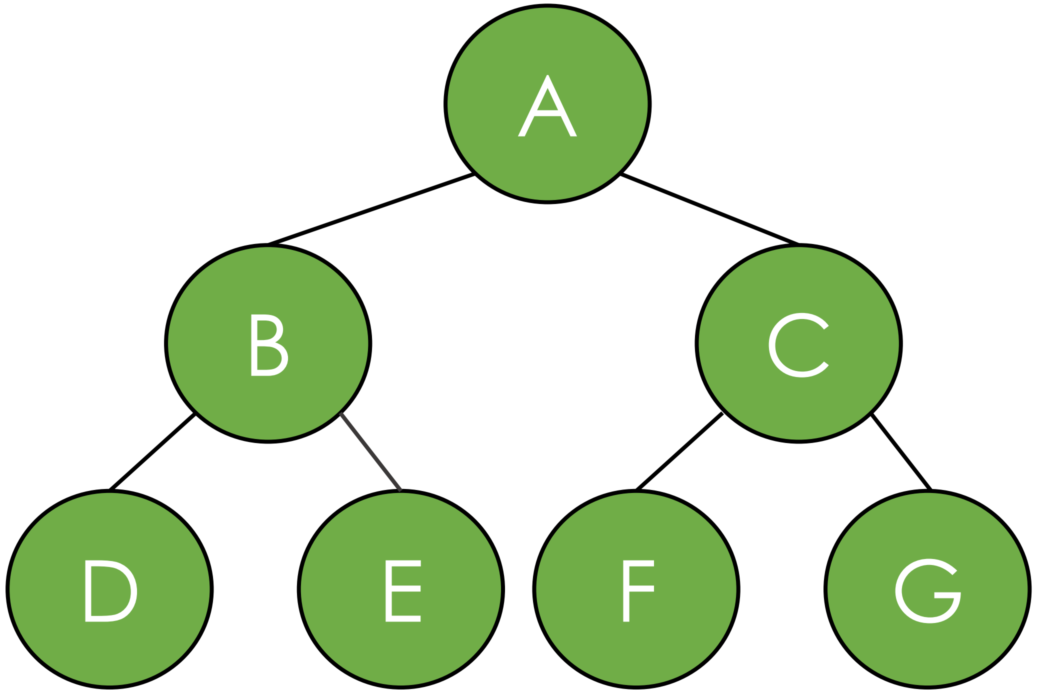


Visited:

A

List: ~~A~~ B C

Level Order Traversal

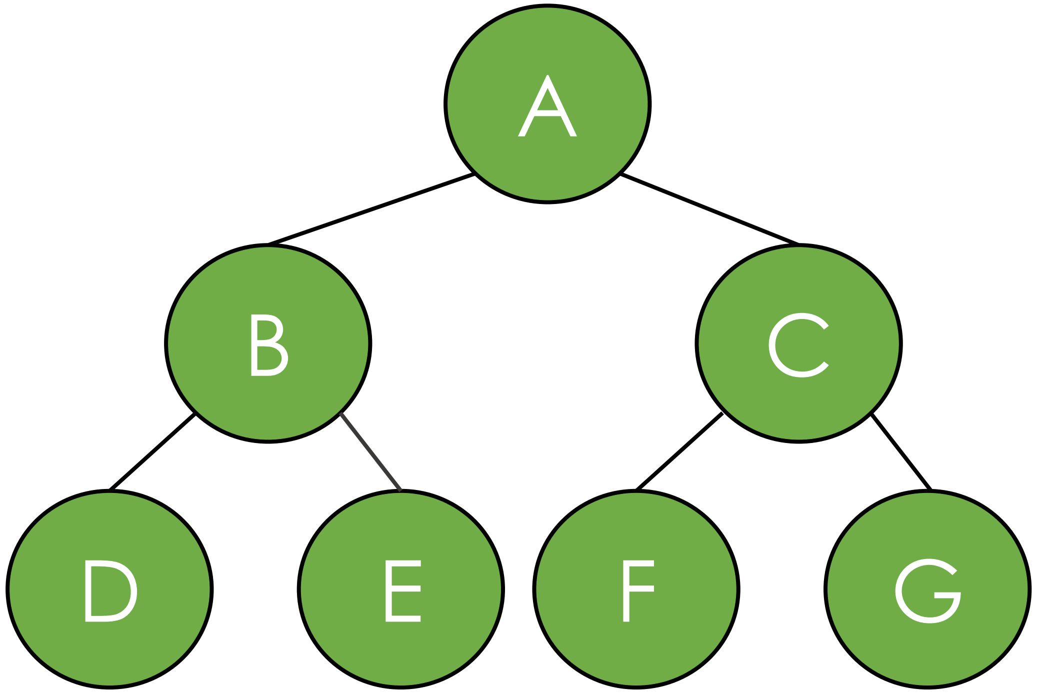


Visited:

A B

List: ~~A~~ ~~B~~ C

Level Order Traversal

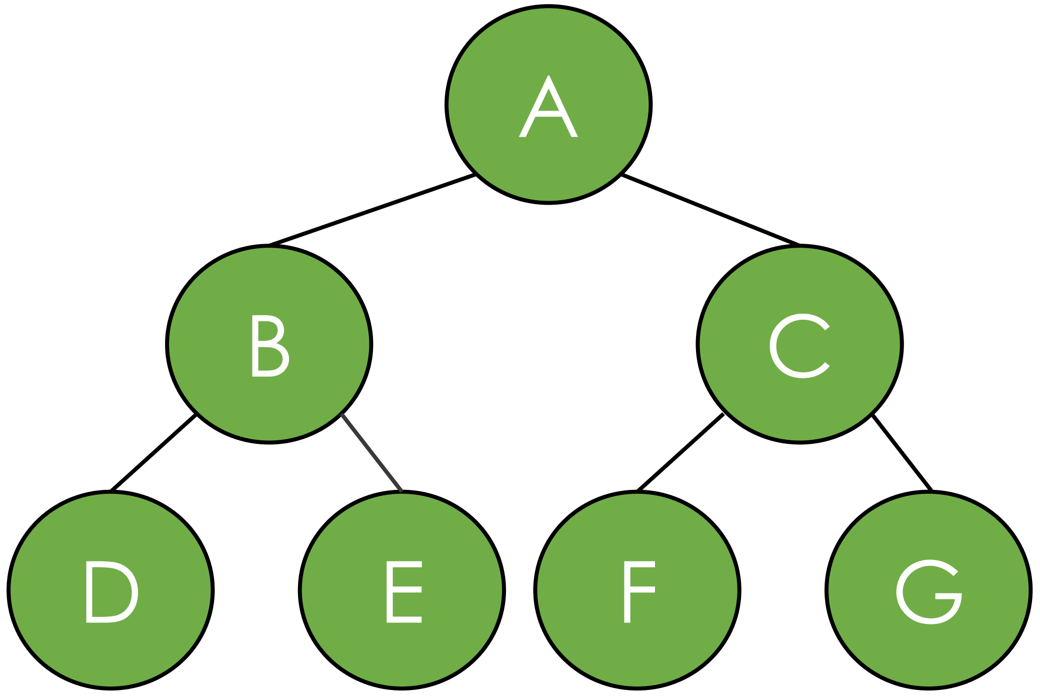


Visited:

A B

List: ~~A~~ ~~B~~ C D E

Level Order Traversal

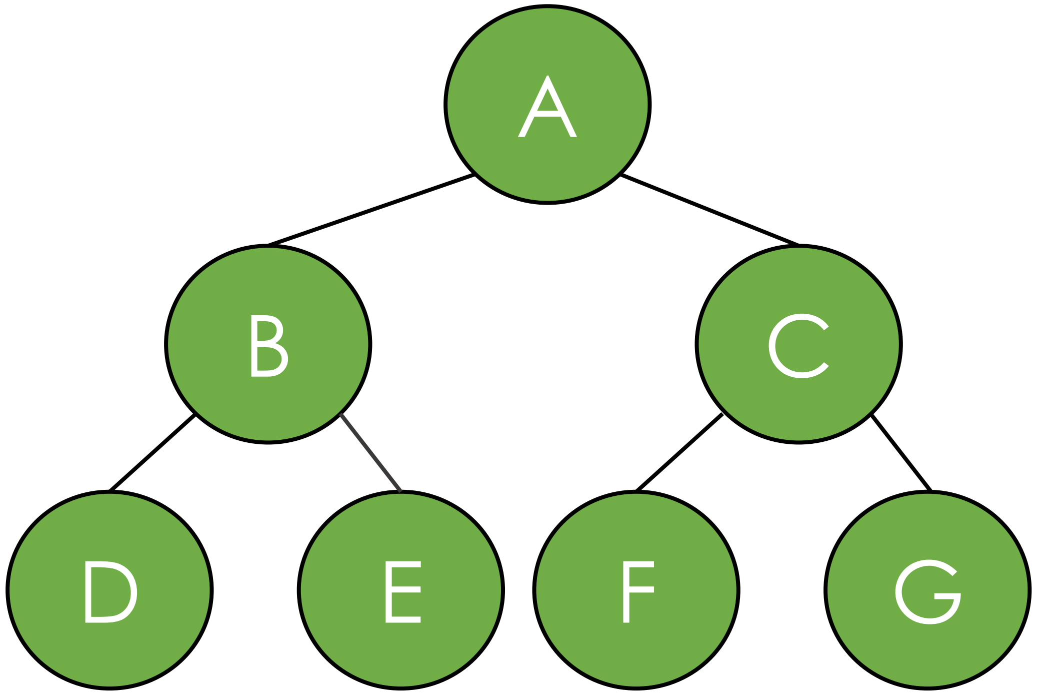


Visited:

A B C

List: ~~A~~ ~~B~~ ~~C~~ D E

Level Order Traversal

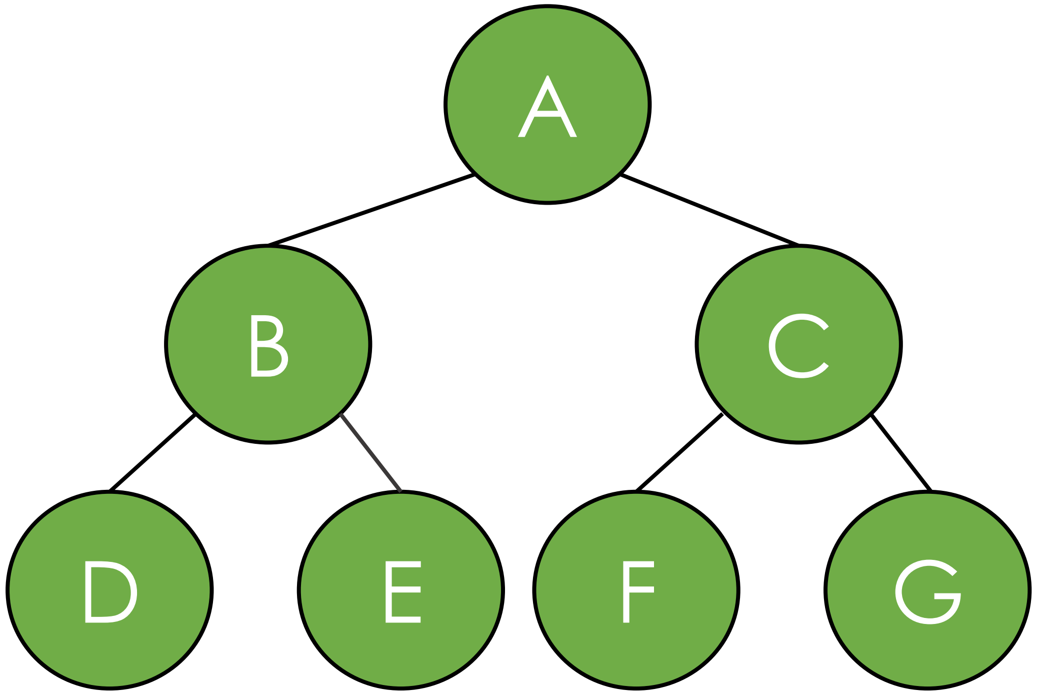


Visited:

A B C

List: ~~A~~ ~~B~~ ~~C~~ D E F G

Level Order Traversal

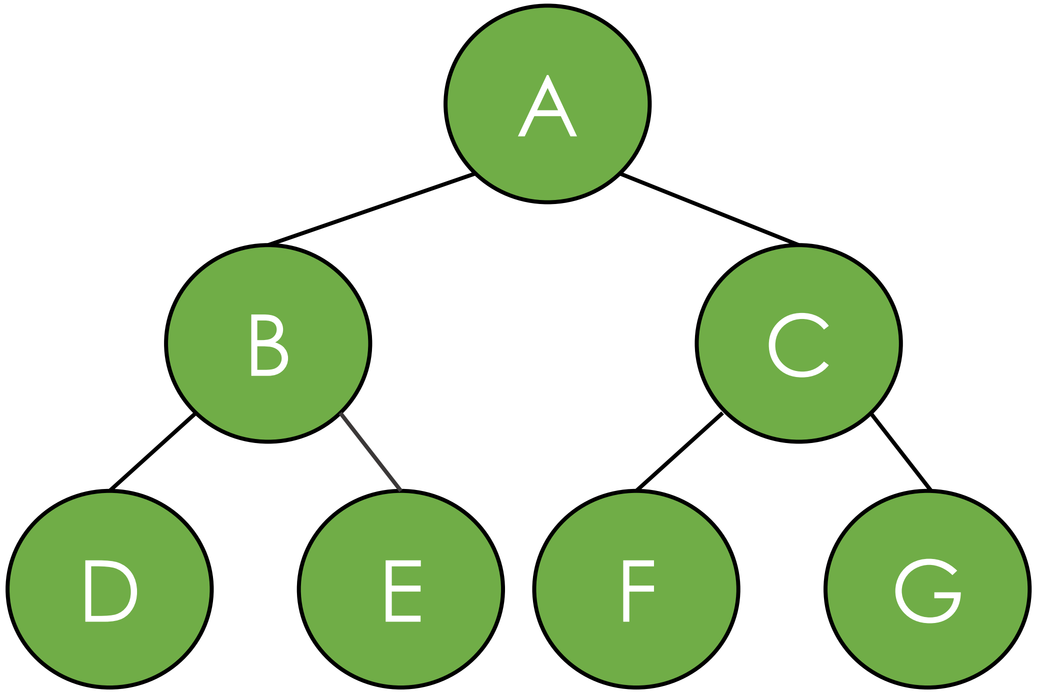


Visited:

A B C D

List: ~~A~~ ~~B~~ ~~C~~ ~~D~~ E F G

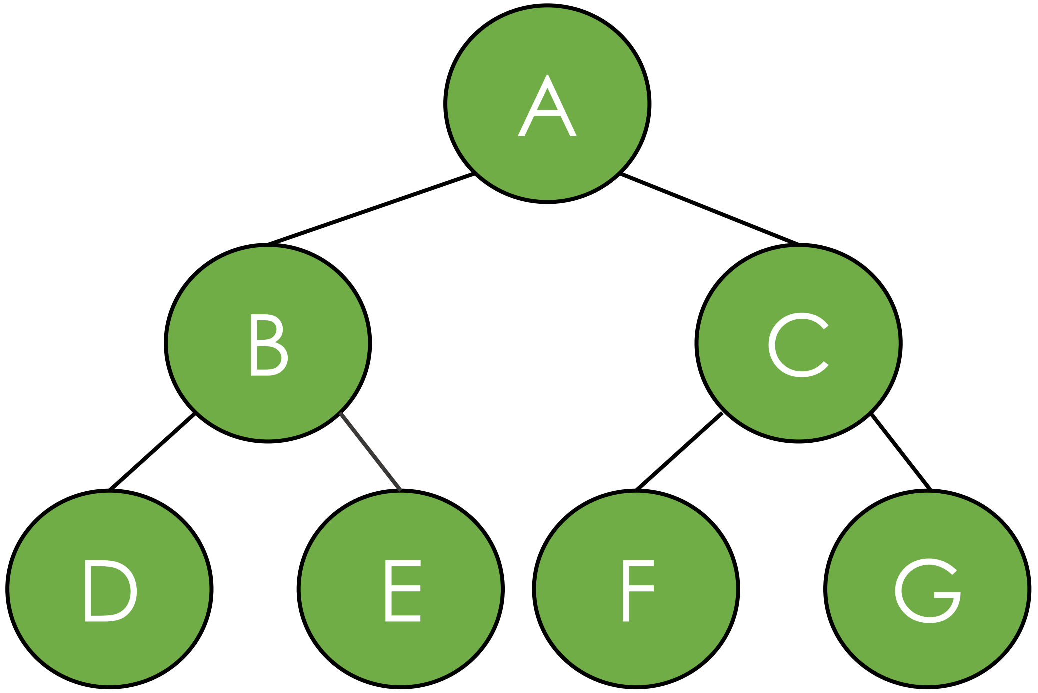
Level Order Traversal



Visited:
A B C D E

List: ~~A~~ ~~B~~ ~~C~~ ~~D~~ ~~E~~ F G

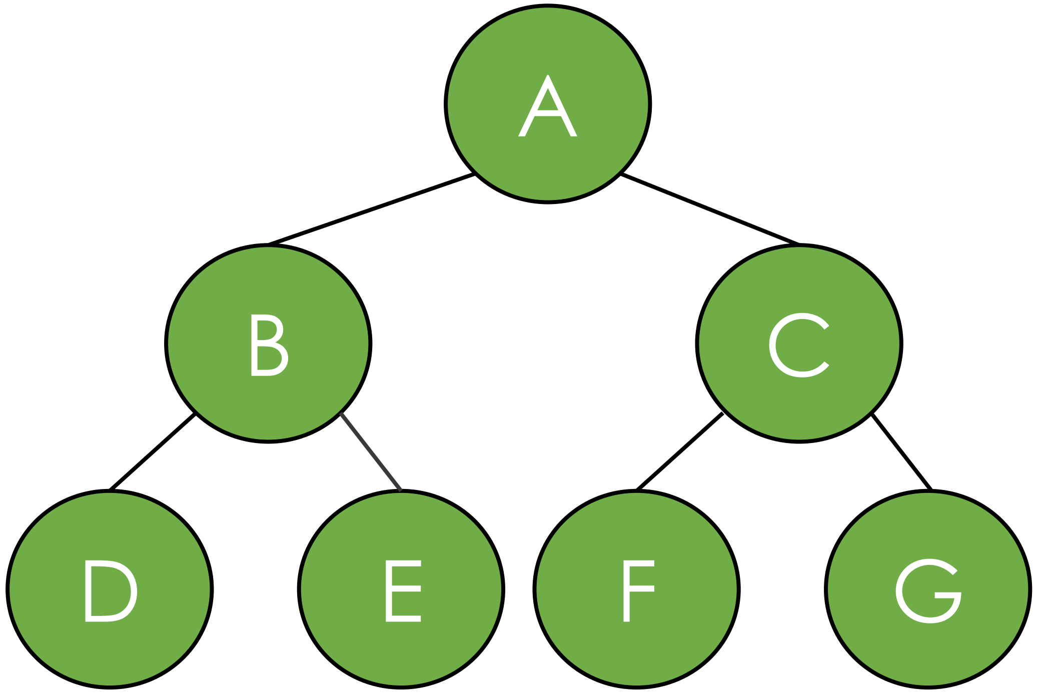
Level Order Traversal



Visited:
A B C D E F

List: ~~A~~ ~~B~~ ~~C~~ ~~D~~ ~~E~~ ~~F~~ G

Level Order Traversal

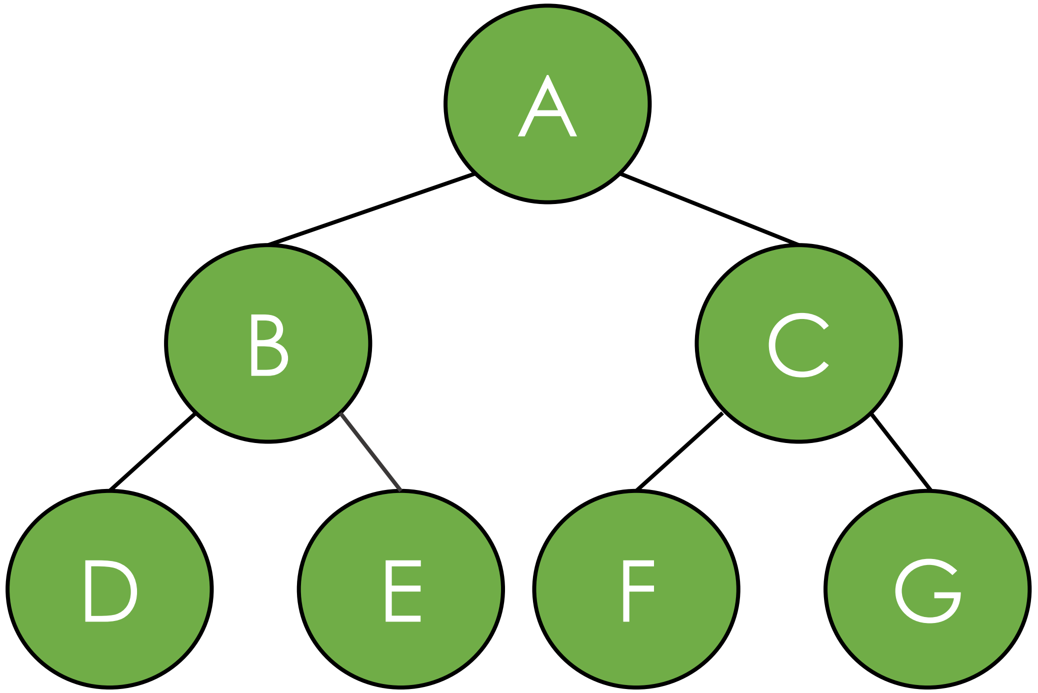


Visited:

A B C D E F G

List: ~~A~~ ~~B~~ ~~C~~ ~~D~~ ~~E~~ ~~F~~ ~~G~~

Level Order Traversal



Visited:

A B C D E F G

List: ~~A~~ ~~B~~ ~~C~~ ~~D~~ ~~E~~ ~~F~~ ~~G~~

We used this list like a "Queue"

Queues

Queues

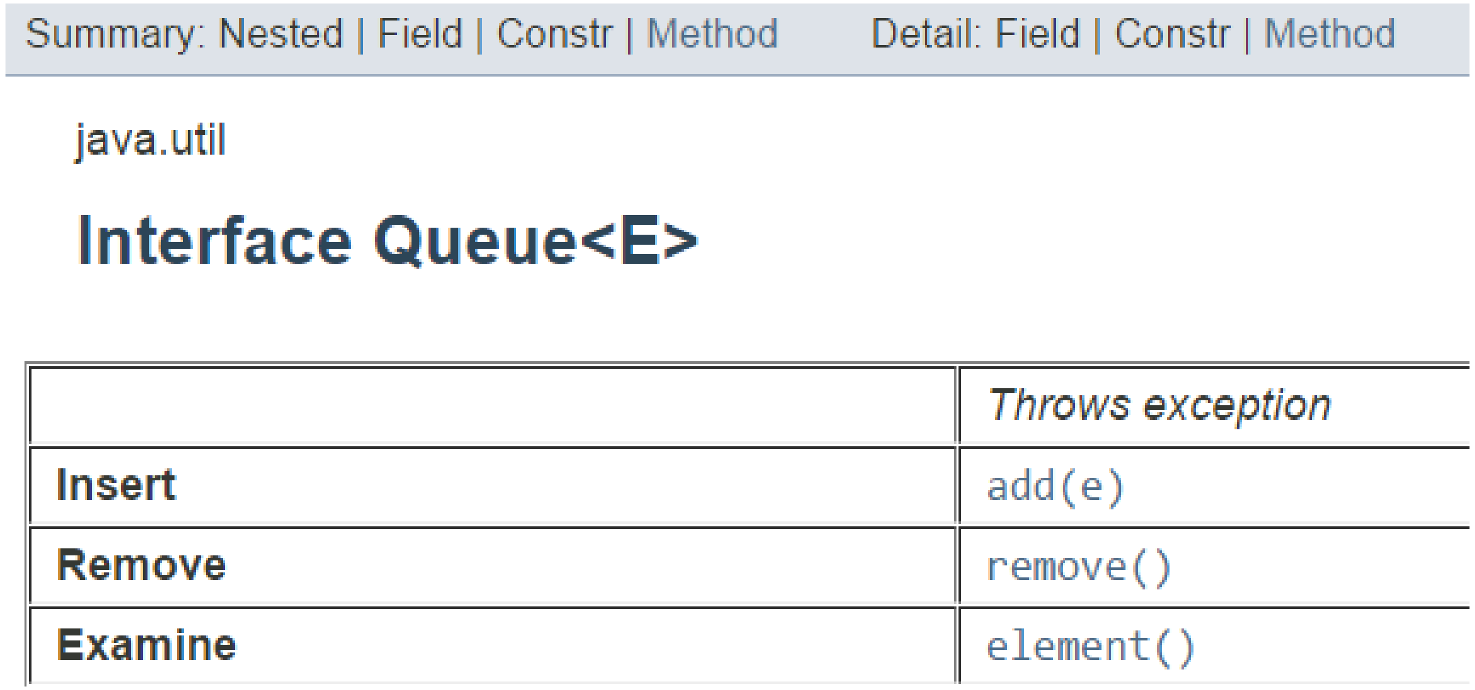
Add to the end

Queues

Remove from the front

Queues

First-In, First-Out (FIFO)



Summary: Nested | Field | Constr | Method   Detail: Field | Constr | Method

java.util

Interface Queue<E>

	Throws exception
Insert	`add(e)`
Remove	`remove()`
Examine	`element()`

http://docs.oracle.com/javase/7/docs/api/java/util/Queue.html

```
public class BinaryTree<E> {
  TreeNode<E> root;

  public void levelOrder() {
    Queue< TreeNode<E> > q = new LinkedList< TreeNode<E> >();







  }
}
```

```
public class BinaryTree<E> {
  TreeNode<E> root;

  public void levelOrder() {
    Queue< TreeNode<E> > q = new LinkedList< TreeNode<E> >();
    q.add(root);
    while(!q.isEmpty()) {
      TreeNode<E> curr = q.remove();
      if(curr != null) {
        curr.visit();
        q.add(curr.getLeftChild());
        q.add(curr.getRightChild());
      }
    }
  }
}
```

```
public class BinaryTree<E> {
  TreeNode<E> root;

  public void levelOrder() {
    Queue< TreeNode<E> > q = new LinkedList< TreeNode<E> >();
    q.add(root);
    while(!q.isEmpty()) {
      TreeNode<E> curr = q.remove();
      if(curr != null) {
        curr.visit();
        q.add(curr.getLeftChild());
        q.add(curr.getRightChild());
      }
    }
  }
}
```

```java
public class BinaryTree<E> {
   TreeNode<E> root;

   public void levelOrder() {
      Queue< TreeNode<E> > q = new LinkedList< TreeNode<E> >();
      q.add(root);
      while(!q.isEmpty()) {
         TreeNode<E> curr = q.remove();
         if(curr != null) {
            curr.visit();
            q.add(curr.getLeftChild());
            q.add(curr.getRightChild());
         }
      }
   }
}
```

```
public class BinaryTree<E> {
  TreeNode<E> root;

  public void levelOrder() {
    Queue< TreeNode<E> > q = new LinkedList< TreeNode<E> >();
    q.add(root);
    while(!q.isEmpty()) {
      TreeNode<E> curr = q.remove();
      if(curr != null) {
        curr.visit();
        q.add(curr.getLeftChild());
        q.add(curr.getRightChild());
      }
    }
  }
}
```

```
public class BinaryTree<E> {
  TreeNode<E> root;

  public void levelOrder() {
    Queue< TreeNode<E> > q = new LinkedList< TreeNode<E> >();
    q.add(root);
    while(!q.isEmpty()) {
      TreeNode<E> curr = q.remove();
      if(curr != null) {
        curr.visit();
        q.add(curr.getLeftChild());
        q.add(curr.getRightChild());
      }
    }
  }
}
```

```java
public class BinaryTree<E> {
  TreeNode<E> root;

  public void levelOrder() {
    Queue< TreeNode<E> > q = new LinkedList< TreeNode<E> >();
    q.add(root);
    while(!q.isEmpty()) {
      TreeNode<E> curr = q.remove();
      if(curr != null) {
        curr.visit();
        q.add(curr.getLeftChild());
        q.add(curr.getRightChild());
      }
    }
  }
}
```

```
public class BinaryTree<E> {
  TreeNode<E> root;

  public void levelOrder() {
    Queue< TreeNode<E> > q = new LinkedList< TreeNode<E> >();
    q.add(root);
    while(!q.isEmpty()) {
      TreeNode<E> curr = q.remove();
      if(curr != null) {
        curr.visit();
        q.add(curr.getLeftChild());
        q.add(curr.getRightChild());
      }
    }
  }
}
```

```
public class BinaryTree<E> {
  TreeNode<E> root;

  public void levelOrder() {
    Queue< TreeNode<E> > q = new LinkedList< TreeNode<E> >();
    q.add(root);
    while(!q.isEmpty()) {
      TreeNode<E> curr = q.remove();
      if(curr != null) {
        curr.visit();
        q.add(curr.getLeftChild());
        q.add(curr.getRightChild());
      }
    }
  }
}
```

Could also check for null children before adding

```
public class BinaryTree<E> {
  TreeNode<E> root;

  public void levelOrder() {
    Queue< TreeNode<E> > q = new LinkedList< TreeNode<E> >();
    q.add(root);
    while(!q.isEmpty()) {
      TreeNode<E> curr = q.remove();
      if(curr != null) {
        curr.visit();
        q.add(curr.getLeftChild());
        q.add(curr.getRightChild());
      }
    }
  }
}
```

```
public class BinaryTree<E> {
  TreeNode<E> root;

  public void levelOrder() {
    Queue< TreeNode<E> > q = new LinkedList< TreeNode<E> >();
    q.add(root);
    while(!q.isEmpty()) {
      TreeNode<E> curr = q.remove();
      if(curr != null) {
        curr.visit();
        q.add(curr.getLeftChild());
        q.add(curr.getRightChild());
      }
    }
  }
}
```

You'll use this idea in this week's project!

Next step

- Explore Binary Search Trees