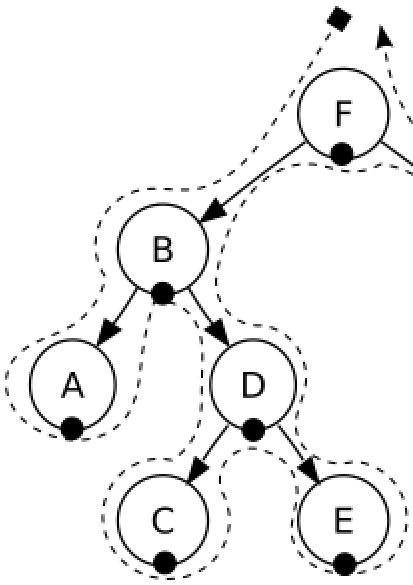
Binary Tree Traversals

In-order



Visual representation of In-order tree traversal. Start from above the root node and traverse around the binary tree counterclockwise until you reach the top of the root node again. While traversing, once you reach the bottom of a node, display the data. The fully displayed outcome of this traversal would be: **A B C D E F G J H I**

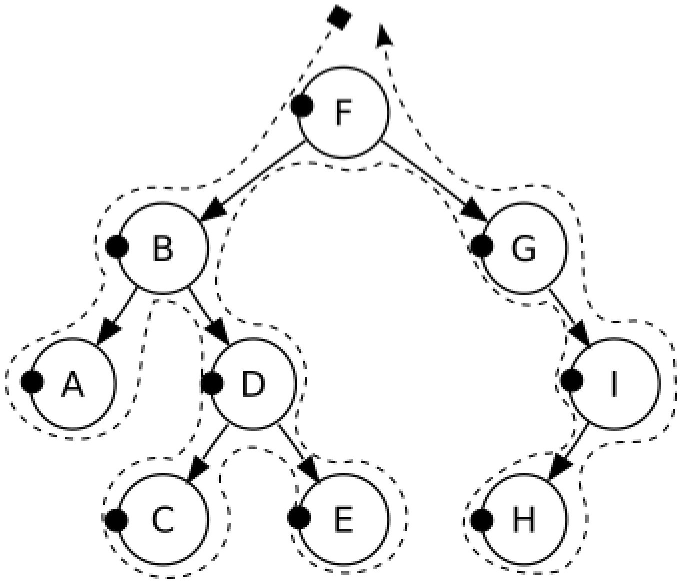
```
void displayInorder(Node* nodePtr)
{
      if(nodePtr) { // Step 1
         displayInorder(nodePtr->left); // Step 2
         cout << nodePtr->data << endl; // Step 3
         displayInorder(nodePtr->right); // Step 4
      }
}
```

Remember that in a binary tree the left child is always less than the parent and the right child is always greater than the parent. Therefore, to display the tree In-order (from least to greatest) the above algorithm is used in a recursive manner:

1. First, check to make sure the incoming node pointer is valid (does not point to **nullptr**). If the node exists...
2. Traverse the left subtree recursively
3. Print the node's data
4. Traverse the right subtree recursively

If you would like to print greatest to least, swap recursive calls such that you traverse the right subtree recursively, print the node's data, and then traverse the left subtree recursively.

Pre-order



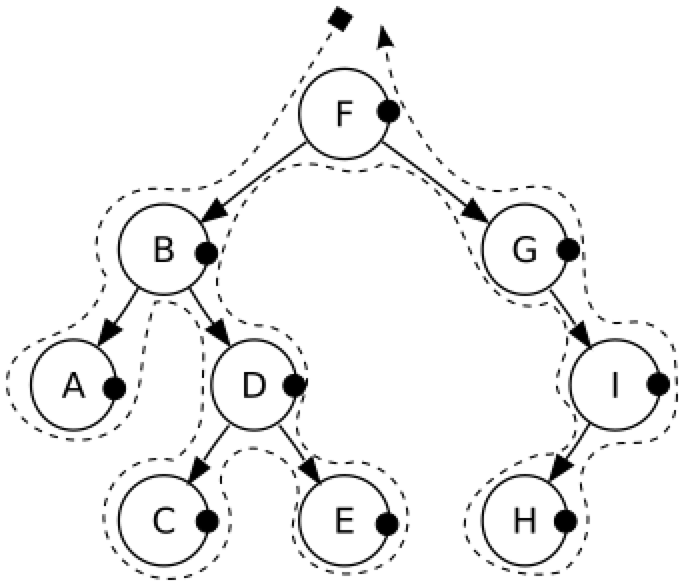
Visual representation of Pre-order tree traversal. Start from above the root node and traverse around the binary tree counterclockwise until you reach the top of the root node again. While traversing, once you reach the left side of a node, display the data. The fully displayed outcome of this traversal would be: **F B A D C E G I H**

```
void displayPreorder(Node* nodePtr)
{
      if(nodePtr) { // Step 1
         cout << nodePtr->data << endl; // Step 2
         displayPreorder(nodePtr->left); // Step 3
         displayPreorder(nodePtr->right); // Step 4
      }
}
```

Pre-order traversal is done by displaying the data of the node before making recursive calls to the left subtree and the right subtree respectively. Therefore, to display Pre-order the above algorithm is used in this recursive manner:

1. First, check to make sure the incoming node pointer is valid (does not point to **nullptr**). If the node exists...
2. Print the node's data
3. Traverse the left subtree recursively
4. Traverse the right subtree recursively

Post-order



Visual representation of Post-order tree traversal. Start from above the root node and traverse around the binary tree counterclockwise until you reach the top of the root node again. While traversing, once you reach the right side of a node, display the data. The fully displayed outcome of this traversal would be: **A C E D B H I G F**

```cpp
void displayPostorder(Node* nodePtr)
{
      if(nodePtr) { // Step 1
         displayPostorder(nodePtr->left); // Step 2
         displayPostorder(nodePtr->right); // Step 3
         cout << nodePtr->data << endl; // Step 4
      }
}
```

Post-order traversal is done by displaying the data of the node after making recursive calls to the left subtree and the right subtree respectively. Therefore, to display Post-order the above algorithm is used in this recursive manner:

1. First, check to make sure the incoming node pointer is valid (does not point to **nullptr**). If the node exists...
2. Traverse the left subtree recursively
3. Traverse the right subtree recursively
4. Print the node's data