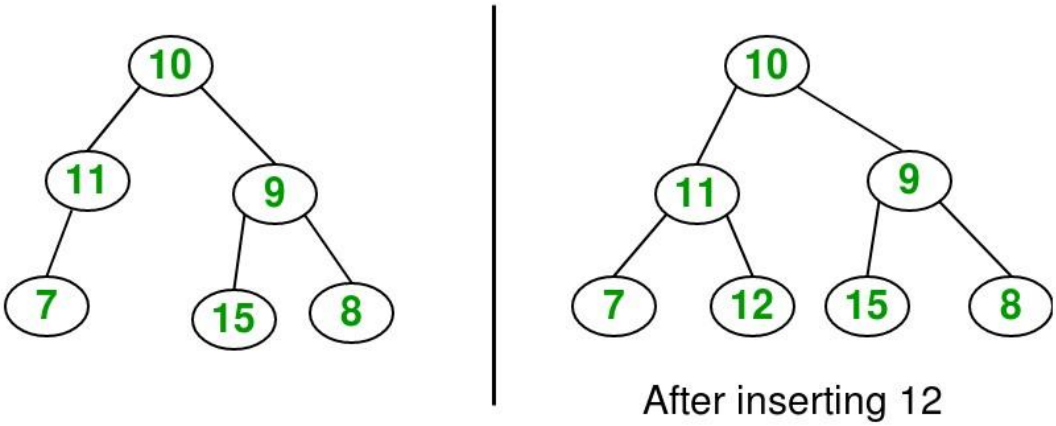
Insertion in a Binary Tree

Problem: Given a **Binary Tree** and a **Key**. The task is to insert the *key* into the binary tree at first position available in level order.



The idea is to do iterative level order traversal of the given tree using a queue. If we find a node whose left child is empty, we make new key as the left child of the node. Else if we find a node whose right child is empty, we make new key as the right child of that node. We keep traversing the tree until we find a node whose either left or right child is empty.

C++ Java

```cpp
// C++ program to insert element in binary tree
#include <iostream>
#include <queue>
using namespace std;
// A binary tree node
struct Node {
    int key;
    struct Node* left, *right;
};
// Utility function to create a new Node
struct Node* newNode(int key)
{
    struct Node* temp = new Node;
    temp->key = key;
    temp->left = temp->right = NULL;
    return temp;
};
// Function to print InOrder traversal of a Binary Tree
void inorder(struct Node* temp)
{
    if (!temp)
        return;

    inorder(temp->left);
    cout << temp->key << " ";
    inorder(temp->right);
}
```

```cpp
// Function to insert a new element in a Binary Tree
void insert(struct Node* temp, int key)
{
    queue<struct Node*> q;
    q.push(temp);

    // Do level order traversal until we find
    // an empty place.
    while (!q.empty()) {
        struct Node* temp = q.front();
        q.pop();
        if (!temp->left) {
            temp->left = newNode(key);
            break;
        } else
            q.push(temp->left);
        if (!temp->right) {
            temp->right = newNode(key);
            break;
        } else
            q.push(temp->right);
    }
}
// Driver code
int main()
{
    // Create the following Binary Tree
    //          10
    //         /  \
    //        11   9
    //       /      \
    //      7        8
    struct Node* root = newNode(10);
    root->left = newNode(11);
    root->left->left = newNode(7);
    root->right = newNode(9);
    root->right->left = newNode(15);
    root->right->right = newNode(8);

    cout << "Inorder traversal before insertion:";
    inorder(root);

    int key = 12;
    insert(root, key);

    cout << endl;
    cout << "Inorder traversal after insertion:";
    inorder(root);

    return 0;
}
```

Output:

```
Inorder traversal before insertion: 7 11 10 15 9 8
Inorder traversal after insertion: 7 11 12 10 15 9 8
```