Iterative Search for a key ‘x’ in Binary Tree

Given a Binary Tree and a key to be searched in it, write an iterative method that returns true if key is present in Binary Tree, else false.

For example, in the following tree, if the searched key is 6, then function should return true and if the searched key is 12, then function should return false.

We strongly recommend to minimize your browser and try this yourself first.

One thing is sure that we need to traverse complete tree to decide whether key is present or not. We can use any of the following traversals to iteratively search a key in a given binary tree.

1) Iterative Level Order Traversal.
2) Iterative Inorder Traversal
3) Iterative Preorder Traversal
4) Iterative Postorder Traversal

Below is iterative **Level Order Traversal** based solution to search an item x in binary tree.

```
// Iterative level order traversal based method to search in Binary Tree
#include <iostream>
#include <queue>
using namespace std;

/* A binary tree node has data, left child and right child */
struct node
{
    int data;
    struct node* left, *right;
};

/* Helper function that allocates a new node with the given data and
   NULL left and right  pointers.*/
struct node* newNode(int data)
{
    struct node* node = new struct node;
    node->data = data;
    node->left = node->right = NULL;
    return(node);
}

// An iterative process to search an element x in a given binary tree
bool iterativeSearch(node *root, int x)
{
    // Base Case
    if (root == NULL)
        return false;

    // Create an empty queue for level order traversal
    queue<node *> q;

    // Enqueue Root and initialize height
    q.push(root);

    // Queue based level order traversal
    while (q.empty() == false)
    {
        // See if current node is same as x
        node *node = q.front();
        if (node->data == x)
            return true;

        // Remove current node and enqueue its children
        q.pop();
        if (node->left != NULL)
            q.push(node->left);
        if (node->right != NULL)
            q.push(node->right);
    }

    return false;
}

// Driver program
int main(void)
{
    struct node*NewRoot=NULL;
    struct node *root = newNode(2);
    root->left        = newNode(7);
    root->right       = newNode(5);
    root->left->right = newNode(6);
    root->left->right->left=newNode(1);
    root->left->right->right=newNode(11);
    root->right->right=newNode(9);
    root->right->right->left=newNode(4);

    iterativeSearch(root, 6)? cout << "Found\n": cout << "Not Found\n";
    iterativeSearch(root, 12)? cout << "Found\n": cout << "Not Found\n";
    return 0;
}
```

Output:

```
Found
Not Found
```

Below implementation uses **Iterative Preorder Traversal** to find x in Binary Tree

```
// An iterative method to search an item in Binary Tree
#include <iostream>
#include <stack>
using namespace std;

/* A binary tree node has data, left child and right child */
struct node
{
    int data;
    struct node* left, *right;
};

/* Helper function that allocates a new node with the given data and
   NULL left and right  pointers.*/
struct node* newNode(int data)
{
    struct node* node = new struct node;
    node->data = data;
    node->left = node->right = NULL;
    return(node);
}

// iterative process to search an element x in a given binary tree
bool iterativeSearch(node *root, int x)
{
    // Base Case
    if (root == NULL)
        return false;

    // Create an empty stack and push root to it
    stack<node *> nodeStack;
    nodeStack.push(root);

    // Do iterative preorder traversal to search x
    while (nodeStack.empty() == false)
    {
        // See the top item from stack and check if it is same as x
        struct node *node = nodeStack.top();
        if (node->data == x)
            return true;
        nodeStack.pop();

        // Push right and left children of the popped node to stack
        if (node->right)
            nodeStack.push(node->right);
        if (node->left)
            nodeStack.push(node->left);
    }

    return false;
}

// Driver program
int main(void)
{
    struct node*NewRoot=NULL;
    struct node *root = newNode(2);
    root->left        = newNode(7);
    root->right       = newNode(5);
    root->left->right = newNode(6);
    root->left->right->left=newNode(1);
    root->left->right->right=newNode(11);
    root->right->right=newNode(9);
    root->right->right->left=newNode(4);

    iterativeSearch(root, 6)? cout << "Found\n": cout << "Not Found\n";
    iterativeSearch(root, 12)? cout << "Found\n": cout << "Not Found\n";
    return 0;
}
```

Output:

```
Found
Not Found
```

Similarly, Iterative Inorder and Iterative Postorder traversals can be used.