

Search and Traversing Operations in BST

Finding (Searching) a Node in a Binary Search Tree

Finding a node with a specific key is the simplest of the major tree operations.

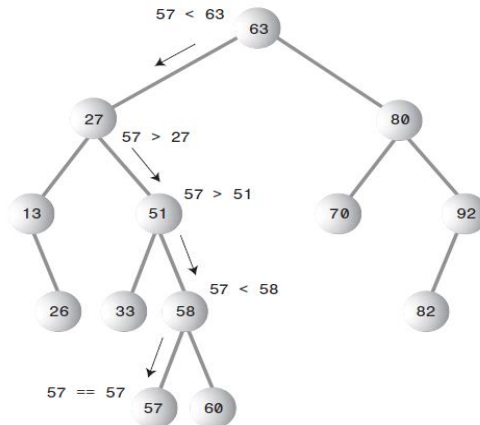


Figure 5: Finding node 57.

// find or search a node with a given key

```
Public void Search (int key)           // Iterative manner
{
    if (root == null)
        return ;
    else
    {
        node current = root;
        While (current . Data != key)
        {
            If (key < current . Data)      // go left
                current = current . leftChild ;
            else
                // or go right
                current = current . rightChild ;

            if (current == null)           // if no child
            {
                S.O.P ("not found") ;
                Return ;
            }
        }
        S.O.P ("found") ;
    }
}
```

or

```
if (root == null)
    return ;
else
{
    node current = root;
    while (current != null)
    {
        if (key == current . data)
        {
            S.O.P ("found") ;
            return ;
        }
        else
        {
            if (key < current . data)
                current = current . leftChild ;
            else
                current = current . rightChild ;
        }
    }
    // غير موجود
    S.O.P ("not found") ;
}
```

////////////////////////////////////

// Recursive manner

- Firstly point to root by current

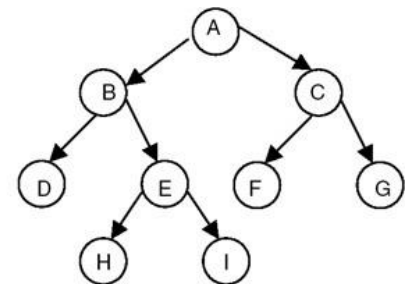
```
Public void Search (node current , int key)
{
    if (current == null)
    {
        S.O.P ("not found") ;
        Return ;
    }
    If (key == current . Data)           // current key is what you search for
        S.O.P ("found") ;
    Else
        If (key < current . Data)
            Search (current . leftChild , key) ;
        Else
            Search (current . rightChild , key) ;
}
```

Binary Tree Traversals

There are three traversals manners (preorder, inorder, postorder) to print the tree's nodes.

```
public void preorderPrint (node current)
{
    if (current == null)
        Return;
    S.O.P (current . Data) ;    // root of current subtree
    preorderPrint (current . leftChild);
    preorderPrint (current . rightChild);
}
```

A B D E H I C F G



```
public void inorderPrint (node current)
{
    if (current == null)
        return;
    inorderPrint (current . leftChild);
    S.O.P (current . Data) ;    // root of current subtree
    inorderPrint (current . rightChild);
}
```

D B H E I A F C G

```

public void postorderPrint (node current)
{
    if (current == null)
        return;
    postorderPrint (current . leftChild);
    postorderPrint (current . rightChild);
    S.O.P (current . Data) ;    // root of current subtree
}

```

D H I E B F G C A

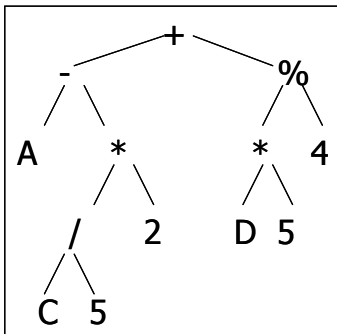
////////////////////////////////////

Parse Trees

Expressions, programs, etc. can be represented by tree structures

E.g. Arithmetic Expression Tree

$A - (C / 5 * 2) + (D * 5 \% 4)$



Tree Traversal

Goal: visit every node of a tree

inorder traversal, the output: $A - (C / 5 * 2) + (D * 5 \% 4)$