Brice
Page :
Date : / /

Traversal
(~~Left~~ Root left Right)

① preorder (Recursive & Iterative)

↳ (DFS)

*Recursive :⇒

Recursive case {
cout << root → data ;
preorder (root → left) ;
preorder (root → right) ;

Base case ← {root == null

**Iterative :⇒

(1) stack < Node* > stack ;
stack.push(root) ;

Node* temp = stack.top() ;
~~cout~~ stack.pop() ;
cout << root → data ;

(because stack is LIFO)
② Goto right
③ Goto left

} while(!stack.empty())

(Left Right Root)

Postorder (Recursive & Iterative)

→ DFS

* Recursive :⇒

Recursive { postorder(root → left)
postorder(root → right)
cout << root → data

Base { Root == NULL

** Iterative (Two stack needed)

① stack<Node*> S1;
push(root);
temp = S1.top();
S1.pop();
S2.push(data);
unless stack is not empty.

② Goto left

③ Goto Right

④ pop & print from S2

(Left Root Right)

Brice
Page :
Date : / /

Inorder Traversal :⇒ (Recursive & Iterative)

↳ DFS

Recursive :⇒

Recursive case
```
inorder(root->left)
cout<<root->data;
inorder(root->right);
```

Base case ← root == Null;

Iterative :⇒

```
stack<Node*> s;
Node*temp=root;
```

while(!stackempty() || temp!=NULL)

if (temp != NULL)

extreme left

```
stack push (temp)
temp = temp->left
```

points left subtree

else

```
{
temp=s.top();
s.pop()
print temp->data;
```

points Root

```
temp = temp->right
```

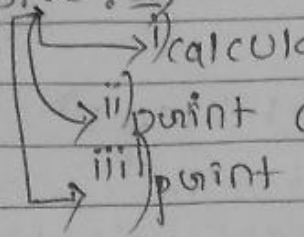
print Right

Brice
Page :
Date : 11 / 1

* Level order Traversal (Recursive)

→ BFS

Recursive :⇒



(i)

Recursive case
```
{ int lheight = height(root->left)
  int rheight = height(root->right)
  return max(rheight + lheight) + 1;
```

Base case
```
{ if(Node == NULL)
    return 0;
```

(2) print current level :⇒

```
void printCurrent(Node* root, k)
{
  if(root == NULL)
    return;
  if(k == 1)
    cout << root->data;
  printCurrent(root->left, k-1)
  printCurrent(root->right, k-1);
```

(3) printing whole tree :⇒

```
for(int i = 1; i < h; i++)
{
  printCurrent(root, i);
}
```