1.Binary tree traversal preorder, inorder and postorder

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>    /\* A binary tree node has data, pointer to left child     and a pointer to right child \*/  struct node  {       int data;       struct node\* left;       struct node\* 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 = (struct node\*)                                    malloc(sizeof(struct node));       node->data = data;       node->left = NULL;       node->right = NULL;         return(node);  }    /\* Given a binary tree, print its nodes according to the    "bottom-up" postorder traversal. \*/  void printPostorder(struct node\* node)  {       if (node == NULL)          return;         // first recur on left subtree       printPostorder(node->left);         // then recur on right subtree       printPostorder(node->right);         // now deal with the node       printf("%d ", node->data);  }    /\* Given a binary tree, print its nodes in inorder\*/  void printInorder(struct node\* node)  {       if (node == NULL)            return;         /\* first recur on left child \*/       printInorder(node->left);         /\* then print the data of node \*/       printf("%d ", node->data);         /\* now recur on right child \*/       printInorder(node->right);  }    /\* Given a binary tree, print its nodes in preorder\*/  void printPreorder(struct node\* node)  {       if (node == NULL)            return;         /\* first print data of node \*/       printf("%d ", node->data);         /\* then recur on left sutree \*/       printPreorder(node->left);         /\* now recur on right subtree \*/       printPreorder(node->right);  }    /\* Driver program to test above functions\*/  int main()  {       struct node \*root  = newNode(1);       root->left             = newNode(2);       root->right           = newNode(3);       root->left->left     = newNode(4);       root->left->right   = newNode(5);         printf("\nPreorder traversal of binary tree is \n");       printPreorder(root);         printf("\nInorder traversal of binary tree is \n");       printInorder(root);         printf("\nPostorder traversal of binary tree is \n");       printPostorder(root);         getchar();       return 0;  } |

**Output:**

Preorder traversal of binary tree is

1 2 4 5 3

Inorder traversal of binary tree is

4 2 5 1 3

Postorder traversal of binary tree is

4 5 2 3 1

2.Height of a Binary tree

#include<iostream>

using namespace std;

// get the max of two no.s

int max(int a, int b) {

return ((a > b) ? a : b);

}

typedef struct node {

int value;

struct node \*left, \*right;

}node;

// create a new node

node \*getNewNode(int value) {

node \*new\_node = new node;

new\_node->value = value;

new\_node->left = NULL;

new\_node->right = NULL;

return new\_node;

}

// compute height of the tree

int getHeight(node \*root) {

if (root == NULL)

return 0;

// find the height of each subtree

int lh = getHeight(root->left);

int rh = getHeight(root->right);

return 1 + max(lh,rh);

}

// create the tree

node \*createTree() {

node \*root = getNewNode(31);

root->left = getNewNode(16);

root->right = getNewNode(52);

root->left->left = getNewNode(7);

root->left->right = getNewNode(24);

root->left->right->left = getNewNode(19);

root->left->right->right = getNewNode(29);

return root;

}

int main() {

node \*root = createTree();

cout<<"\nHeight of the tree is "<<getHeight(root);

cout<<endl;

return 0;

}

3.Total nodes in a tree

Total number of nodes in a binary tree is determined by adding the number of nodes in the left subtree, the number of nodes in the right subtree and the root node. If the binary tree is empty then the number of nodes is zero.

Total\_node(struct node \*tree)

{

If(tree == NULL)

return 0;

else

return(total\_node(tree🡪lchild)+ total\_node(tree🡪rchild)+1);

}

4.Number of leaf nodes or external nodes

The number of external or leaf node in a binary tree is equal to the sum of the external nodes in the left subtree and the external nodes in the right subtree of a given node. Note that if a binary tree is empty then the number of external nodes is zero and if there is only one node in a binary tree the number of external node is equal to one.

external\_node(struct node \*tree)

{

If(tree == NULL)

return 0;

else

{

If(tree🡪lchild==NULL && tree🡪rchild==NULL)

return 1;

else

return(external\_node (tree🡪lchild)+ external\_node (tree🡪rchild));

}

}

5.Number of internal nodes in binary tree

Number of internal or non-leaf nodes in binary tree is equal to the number of non-leaf nodes in the left subtree plus the number of non-leaf nodes in the right subtree of a given node plus one. If the binary tree is empty or there is only one node then number of internal node is equal to zero.

internal\_node(struct node \*tree)

{

If(tree==NULL || (tree🡪lchild==NULL && tree🡪rchild==NULL))

return 0;

else

return(internal\_node (tree🡪lchild)+ internal\_node (tree🡪rchild)+1);

}

}

6.Mirror image of a binary tree

The mirror image of a binary tree is obtained by interchanging left and right child recursively.

Mirror\_image(struct node \*tree)

{

struct node \*Temp;

If(tree != NULL)

{

Mirror\_image(tree🡪lchild);

Mirror\_image(tree🡪rchild);

Temp= tree🡪lchild;

tree🡪lchild= tree🡪rchild;

tree🡪rchild=Temp;

}

}

7.Make copy of a given binary tree

Copy (struct node \*tree) /\* for a binary tree, copy returns a pointer to an exact copy of the original tree\*/

{

struct node \*temp;

If(tree!=null)

{

Temp=getnode();

Temp🡪lchild=copy(tree🡪lchild);

Temp🡪rchild=copy(tree🡪rchild);

Temp🡪data=tree🡪data;

Return temp;

}

Else

return NULL;

}

8. Determine two given binary trees are equal or not

Two binary trees are equivalent if they have same topology and the information in corresponding nodes are identical. By the same topology we mean that every branch in one tree corresponds to a branch in the second tree in the same order.

Equal(struct node \*tree1, struct node \*tree2) /\* this procedure returns false if the tree1 and tree2 are not equivalent otherwise it will return true\*/

{

Equals=false;

If(tree1==null && tree2==null)

Return(true);

Else if(tree1≠null && tree2≠null)

{

If(tree1🡪data==tree2🡪data)

{

If(equal(tree1🡪lchild,tree2🡪lchild))

Return(equal(tree1🡪rchild,tree2🡪rchild));

}

}

Return(equals);

}

Theorem1: Maximum number of nodes in a binary tree of depth k is (Root is at level 1).

PROOF:

The max no of nodes in a binary tree of depth k is

∑(maximum no. of nodes at level i)//i= 1 to k

=∑2^(i-1)

=2^0+2^1+…….+2^(k-1)// GP SERIES FORMULA, a(-1)/(r-1)

=// r=ratio=2nd term/ 1st term=2/1=2

=-1(proved)

**Theorem2:** The maximum number of nodes on level i of a binary is (level of root is 1).

Proof: The proof is induction on i.

**Induction base:** The root is only node on level i=1.

Hence the maximum number of nodes on level i=1 is 1.

**Induction hypothesis:** For all j, 1<=j<=i, the maximum number of nodes on level j is .

**Induction step:** The maximum number of nodes on level i-1 is , by the induction hypothesis. Since each node in a binary tree has maximum degree 2, the maximum number of nodes of level i is two times the maximum number of nodes of level i-1 i.e. 2. = .

**Theorem3:** Show that a binary tree with n nodes has exactly (n+1) null links.

Proof:

In a binary tree with n nodes, the total number of link is 2n. The reason is that for each of this n nodes there are two links.

Among this 2n links , only (n-1) links are used to point the different nodes. The reason is that, in a binary tree of n nodes there will be (n-1) branches, which will point (n-1) nodes, because no branch will point the root node. Therefore, the (n-1) non-null links are there.

Therefore Total number of null links=2n-(n-1)

=(n+1). (proved)

**Theorem4:** For every non empty binary tree if is the number of internal nodes and is the number of nodes of degree 2, then +1.

Proof:

Let, be the number of nodes of degree 1 and n the total number of nodes in a tree.

Since all nodes in a binary tree are of degree less than or equal to two. We have

n= + + ------------(1)

since each node propagate from a node with degree 1 or degree 2, except root.

Total no of branch(edge) B= n-1

And B=2+

So, 2+ =n-1

Or, n=2+ +1

Or, + + =2+ +1

Or, = +1 (proved)

**Theorem5:** If a complete binary tree with n nodes is represented sequentially then for any node with index i, 1<=i<=n.

1. Parent i is at └ i/2 ┘ if i≠1. When i=1, i is the root and has no parent.
2. lchild(i) is at 2\*i if 2\*i<=n. if 2\*i>n, then i has no left child.
3. rchild(i) is at 2\*i+1 if 2\*i+1<=n. if 2\*i+1>n, then i has no right child.

##ii) lchild(i) is at 2\*i if 2\*i<=n. if 2\*i>n, then i has no left child.

**Proof:** We proof (ii) by induction on i.

**Induction base:** for i=1, clearly the left child is at 2\*i=2\*1=2 unless 2>n in which case 1 has no left child.

**Induction hypothesis:** Now assume that for all j, 1<=j<=i, lchild(j) is at 2\*j.

**Induction step:** The two nodes immediately preceding lchild(i+1) in the representation are the right child of i and the left child of i. The left child of i is at 2\*i. Hence the left child of (i+1) is at2(i+1)= (2\*i +2) unless 2\*(i+1)>n in which case (i+1) has no left child.

****

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **1** | **2** |  | **4** | **5** |  |  | **8** |  | **10** |