Sub:- DSAL

PICT, PUNE

Assignment - 01

Name- Shrikrushna Zirape
Roll No:- 21286

Problem Statement -

Beginning with an empty B.T. Construct B.T. by inserting values in the order given. After Constructing a B.T. perform following operations on it.

* Perform inorder / Preorder / Postorder traversal
* change a tree so that the roles of the left and right pointers are swapped at every node.
* Find the height of tree.
* Copy the tree to another (= operator)
* Count the no of leaves, number of internal nodes
* Erase all the nodes in B.T.
  (impliment both recursive and non-recursive method)

Learning Objective -

1) To impliment non-linear data structure binary tree.
2) To impliment several operations on B.T. using the concept of OOP.

Learning Outcomes :-

1) We will able to construct binary tree data structure using oop concept.
2) We will get an idea of how to impliment operations on binary tree data structure.

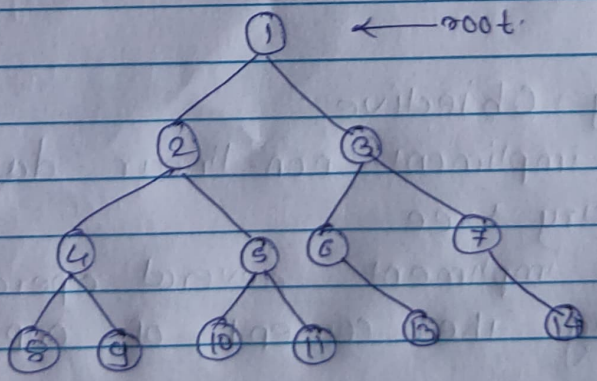
PICT, PUNE

Concepts related to binary tree:-

Binary Tree data structure-

- B.T is tree type non-linear data structure with maximum of two children for every parent.
- Every node in a binary tree has a left and right reference along with data element the nodes that hold other sub node are parent nodes. A parent node has two child nodes left child and right child.
- The top nodes in the B.T is reffer as 'root'.

eg:-



A refference (left or right which dose'nt point to any other node is reffered as "NULL".

* Pseudocode -

① Class Node // Contains basic structure of node in B.T.
{
int data; // stores info of node
Node *lchild; // ptr to left child
Node *rchild; // ptr to right child
}

② Class stack // storing nodes
Node *S.item[max] // stack using an array.
int top; // index of top elem.
void Push(Node*) // add node
Node* Pop() // deletes & return node
int isEmpty() // return 1 is stack is empty
}

③ Algorithm push (Node *ptr)
{ // add item of type Node* into the stack.
if (top == (max-1) then
print "stack overflow"
else
top = top + 1
S.item[top] = ptr
}

```
Class BinaryTree // Contains all operation on B.T.
{
  Node *root; // holds address of root node in BT.
  Node *CreateTreeRecursivly()
  Void inorderRecursivly(Node*)
  Void PreorderRecursivly(Node*)
  Void PostorderRecursivly(Node*)

  Node *Create_iteratively()
  Void inorder_iteratively()
  Void preorder_iteratively()
  Void Postorder_iteratively()

  Void heightRecursivly(Node*)
  Void heightIteratively(Node*)
  Void mirrorRecursivly(Node*)
  Void mirrorIteratively(Node*)

  int leafnodeRecursivly(Node*)
  int leafnodeIteratively()
  int internalRecursivly(Node*)
  int internalIteratively()

  Void deleteRecursivly(Node*)
  Void deleteIteratively()
}
```

```
Algorithm Create_tree_Recursivly ()
 // Create tree return root node by Recursion.
{
   int element;
   Node *ptr = New Node();
   input element;
   if(element == -1) then
          return NULL;
   ptr := data = element;
   input ptr → data  // left child data.
   ptr → lchild = call createtree Recursivly ();
   input ptr → data  //right child data;
   ptr → rchild = Call Create_tree Recursivly ();
   return ptr;
}

Algorithm inorder-traversal
 // display BT element in order seq via recursion
  {
    if (ptr == NULL) then
        return.
    else:
        Call inorder_traversal_Recursivly (ptr → rchild)
        print (ptr → data)
        Call inorder_traversal Recursivly (ptr → lchild)
  }
```

```
// Algorithm Preorder Recursivly (Node *ptr)
{
  // display BT element in Preorder seq via
     recursion.
{
   if (ptr == NULL) then
         return;
   else.
         Print ptr → data;
         Call Preorder Recursivly ( ptr → lchild )
         Call Preorder Recursivly (ptr → rchild)
}
```

```
Algorithm PostorderRecursivly (Node* ptr)
  {
   // display BT. elem in Postorder seq via
      recursion
{
   if (ptr == NULL) then
        return
   else.
     Call Postorder Recursivly (ptr → lchild)
     Call Postorder Recursivly (ptr → rchild)
     Print (ptr → data)
```

```
Algorithm CreateTree iteratively ()
  // Create tree using loop & returns node.
{
  int element;
  Queue q; // q is object of class Queue.
  if (root == NULL) then
     Print element;
     Node *ptr = newNode (element)
     root = ptr;
     q.enqueue (ptr)
  while ( ! q.isEmpty())
     Node *ptr = q.dequeu();
     input element
     if (element = -1)
        ptr -> lchild = NULL
     else
        ptr -> lchild = newNode (element)
        q.enqueue (lchild)
     input element
     if (element != -1) then
        ptr -> rchild = NULL
     else
        ptr -> rchild = newNode (element)
        q.enqueue (ptr -> rchild)
  return root;
}
```

```
Algorithm inorderIteratively()
{
  stack s.
  Node *curr = root.
  while( curr != NULL || !s.isEmpty() )
     while( curr != NULL )
        s.push(curr)
        curr = curr->lchild
     curr = s.pop()
     Print curr->data.
     curr := curr->rchild.
}
```

```
Algorithm PreorderIteratively(Node *ptr)
  // display BT element. Iteratively
  if (root == NULL) then
     return.
  Stack s.;
  s.push (ptr)
  While (!s.isEmpty())
     Node *node = s.pop();
     Print node -> data;
     if (node->lchild) then
        s.push(node->lchild);
     if (node->rchild) then
        s.push(node->rchild);
}
```

```
Algorithm heightRecursivly (Node * ptr)
{ if (ptr == NULL) return 0.
   right = heightRecursivly (ptr -> lchild)
   left = heightRecursivly (ptr -> rchild)
if (right > left) return (right + 1)
else return (left + 1);
}

Algorithm heightIteratively ()
{ if (root == NULL)
    return
  else.
  Queue q
   q.push (root)
   height = 0.
   while(true){
       int. n = q.size().
       if (n == 0)
         return height
       height++.
       while (n > 0)
       Node * node = q.pop();
       if (node -> lchild)
          q.push(node -> lchild)
       if (node -> rchild)
          q.push (node -> rchild)
```

PICT, PUNE

Algorithm

```
mirror Recursivly (Node *ptr)
{
 if (ptr -> lchild == NULL || ptr -> rchild == NULL) {
 return }
 Node *t = ptr -> lchild;
 ptr -> lchild = ptr -> rchild;
 ptr -> rchild = t;
 if (ptr -> lchild) {
    mirrorRecursivly (ptr -> lchild)
 if (ptr -> rchild)
    mirror Recursivly (ptr -> rchild)
```

Algorithm.

```
mirrortree Iteratively ()
 if (root == NULL) {
    return.
  }
 Queue. q;
  q.push (root)
 while. (!q.isEmpty()) {
   Node *node = q.pop();
   Swap (node -> lchild, node -> rchild)
   if (node -> lchild)
      q.push (node -> lchild);
   if (node -> rchild)
      q.push (node -> rchild);
```

PICT, PUNE

Algorithm

```
leafNodeRecursivly (Node *ptr)
  if (ptr == NULL)
      return 0.
  if (ptr -> lchild == NULL && ptr -> rchild == NULL)
      return 1
  else. return
      (leafnode Recursivly (ptr -> lchild) +
       leafnode Recursivly (ptr -> rchild)
```

Algorithm leafnode Iteratively ()

```
  if (root == NULL)
      return 0
  else.
  Queue q;
  q. push (root)
  while ( ! q. isEmpty())
    Struct. Node *temp = q.pop
  if temp -> lchild != NULL)
      q. push (temp -> lchild)
  if (temp -> rchild != NULL)
      q. push (temp -> rchild)
  return Count.
```

```
Algorithm deleteRecursivly (Node *ptr)
{ if(ptr == NULL)
    return;
  deleteRecursivly (ptr->lchild)
  deleteIterativly (ptr->rchild)
  delete ptr.
  ptr = NULL;
}
```

```
Algorithm deleteIterativly()
  if(root == NULL)
    return.
  Queue q.
  q.push(root);
  while(!q.isEmpty()){
    Node *front;
    front = q.pop();
    if(front->lchild)
      q.push(front->lchild)
    if(front->rchild)
      q.push(front->rchild)
    delete front.
  root = NULL
}
```

PICT, PUNE

Test case.	Description.	Actual O/P	exp. O/P	Result.
①	Root data: 10 lchild of 10: 20 rchild of 10: 30			
	inorder.	20→10→30	20→10→30	pass.
	Preorder	10→20→30	10→20→30	pass.
	Postorder.	20→30→10	20→30→10	Pass
	height	2	2	pass.
	leaf node	2	2	Pass
	internal node	1	1	Pass

* Time Complexities :-

CreateRecursivly()	O(n)
CreateIteratively()	O(n)
inorderRecursivly(Node*)	O(n)
inorderIteratively()	$O(n^2)$
preorderRecursivly(Node*)	O(n)
PreorderIteratively()	O(n)
PostorderRecursivly(Node*)	O(n)
postorderiteratively	O(n)
heightRecursivly(Node*	O(n)
heightiteratively()	O(n)
mirrorRecursivly(Node*)	O(n)
mirroriteratively()	O(n)
NodecountRecursivly(Node*)	O(n)
Nodecountiteratively()	O(n)
deleteRecursivly(Node*)	O(n)
deleteIteratively()	O(n)

PICT, PUNE

Realtime Applications -

① Store hierarchical data like folder structure xml/ HTML parser.

② BST is a tree that allows fast search.

Conclusion -

Successfully implimented B.T. data structure & performed basic operations on it.