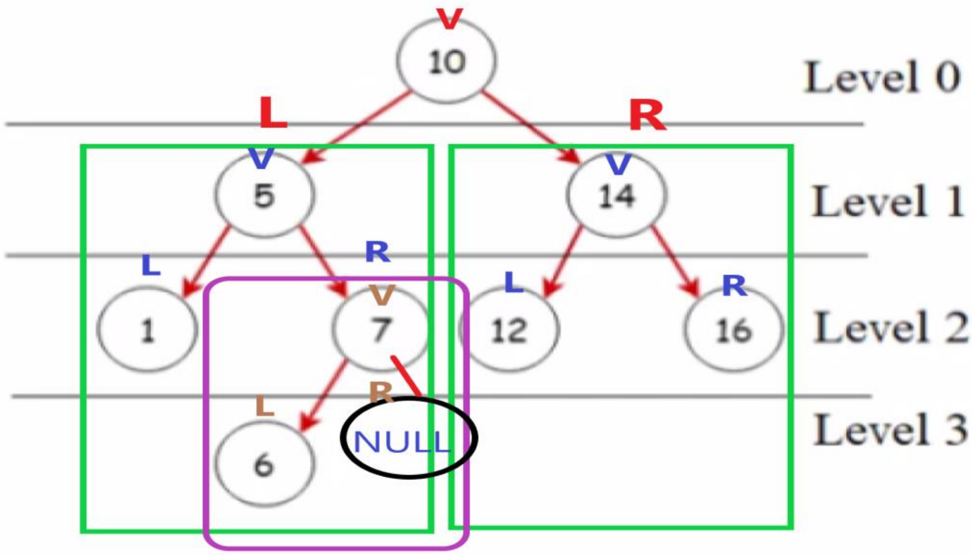
REVISION A BOUT (preorder, inorder and postorder traversal).



-Preorder : [root][left][right]

-Inorder : [left][root][right]

-Postorder : [left][right][root]

Solution

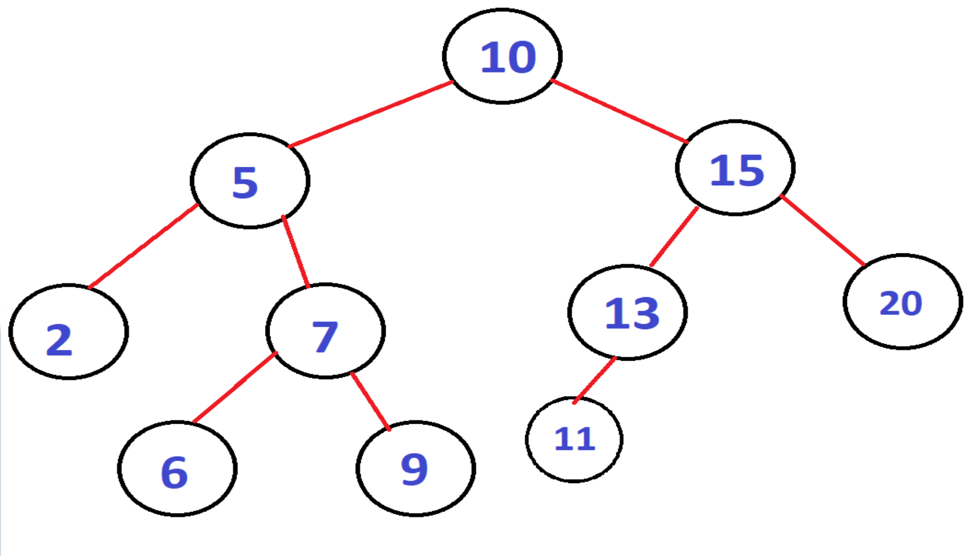
Preorder: 10, 5, 1, 7, 6, 14, 12, 16

Inorder: 1, 5, 6, 7, 10, 12, 14, 16

postorder : 1, 6, 7, 5, 12, 16, 14, 10

1. **Draw a binary search tree (BST) after inserting the following values in this order: 10, 5, 15, 13, 7, 2, 9, 6, 11, and 20.**

Solution



2. **Traverse the previous tree in the three different traversal approaches covered in the lecture (preorder, inorder and postorder traversal).**

Solution

Inorder : 2, 5, 6, 7, 9, 10, 11, 13, 15, 20

preorder: 10, 5, 2, 7, 6, 9, 15, 13, 11, 20

postorder: 2, 6, 9, 7, 5, 11, 13, 20, 15, 10

Tree implementation

TREE .H

```c
#ifndef TREE_H_INCLUDED
#define TREE_H_INCLUDED

void create_tree(tree *t);
int is_tree_empty(tree t);
int is_tree_full(tree t);
void inorder_traversal(tree t, void(*pvisit)(entry_type));
void postorder_traversal(tree t,void(*pvisit)(entry_type));
void preorder_traversal(tree t,void(*pvisit)(entry_type));
int tree_size(tree t);
int tree_depth_orHeight(tree t);
void clear_tree(tree *t);



#endif // TREE_H_INCLUDED
```

TREE.C

```c
void create_tree(tree *t){
        *t = NULL;
}
int is_tree_empty(tree t){    return (!t);     }
int is_tree_full(tree t){     return 0;}
void inorder_traversal(tree t, void(*pvisit)(entry_type)){
    if(t){
            inorder_traversal(t->left, pvisit);
            (*pvisit)(t->info);
            inorder_traversal(t->right, pvisit);
        }
}
void postorder_traversal(tree t,void(*pvisit)(entry_type)){
  if(t){
            postorder_traversal(t->left, pvisit);
            postorder_traversal(t->right, pvisit);
            (*pvisit)(t->info);
  }
}
void preorder_traversal(tree t,void(*pvisit)(entry_type)){
      if(t){
                (*pvisit)(t->info);
                preorder_traversal(t->left, pvisit);
                preorder_traversal(t->right, pvisit);
           }
}
int tree_size(tree t){
      if (!t)
         return 0;
      return(1 + tree_size (t->left) + tree_size (t->right));
}
int tree_depth_orHeight(tree t){
      if (!t)
          return 0;
      int a= tree_depth_orHeight(t->left);
      int b= tree_depth_orHeight(t->right);
      return (a>b)? 1+a : 1+b;
}
void clear_tree(tree *t){
       if (*t){
           clear_tree(&(*t)->left);
           clear_tree(&(*t)->right);
           free(*t);
           *t=NULL;
      }        }
```

Binary search Tree implementation

Binary search Tree .H

```
#ifndef TREE_H_INCLUDED
#define TREE_H_INCLUDED

void create_tree(tree *t);
int is_tree_empty(tree t);
int is_tree_full(tree t);
void inorder_traversal(tree t, void(*pvisit)(entry_type));
void postorder_traversal(tree t,void(*pvisit)(entry_type));
void preorder_traversal(tree t,void(*pvisit)(entry_type));
int tree_size(tree t);
int tree_depth_orHeight(tree t);
void clear_tree(tree *t);
///Binary Search Tree
void insert_node(tree *t, entry_type item);
int search_To_delete(tree *t, tree_entry k);
void delete_node(tree *pt);

#endif // TREE_H_INCLUDED
```

Binary search Tree .C

```c
void create_tree(tree *t){
     *t = NULL;
}
int is_tree_empty(tree t){    return (!t);     }
int is_tree_full(tree t){     return 0;}
void inorder_traversal(tree t, void(*pvisit)(entry_type)){
  if(t){
        inorder_traversal(t->left, pvisit);
        (*pvisit)(t->info);
        inorder_traversal(t->right, pvisit);
     }
}
void postorder_traversal(tree t,void(*pvisit)(entry_type)){
 if(t){
        postorder_traversal(t->left, pvisit);
        postorder_traversal(t->right, pvisit);
        (*pvisit)(t->info);
 }
}
void preorder_traversal(tree t,void(*pvisit)(entry_type)){
    if(t){
            (*pvisit)(t->info);
            preorder_traversal(t->left, pvisit);
            preorder_traversal(t->right, pvisit);
        }
}
int tree_size(tree t){
   if (!t)
      return 0;
   return(1 + tree_size (t->left) + tree_size (t->right));
}
int tree_depth_orHeight(tree t){
    if (!t)
        return 0;
    int a= tree_depth_orHeight(t->left);
    int b= tree_depth_orHeight(t->right);
    return (a>b)? 1+a : 1+b;
}
void clear_tree(tree *t){
     if (*t){
        clear_tree(&(*t)->left);
        clear_tree(&(*t)->right);
        free(*t);
        *t=NULL;
     }       }
```

```c
void insert_node(tree *t, entry_type item){
 tree_node p =(tree_node)malloc(sizeof(tree_node));
    p->info = item;
    p->left = NULL;
    p->right = NULL;
    if (!(*t))
    *t= p;
else {
    tree_node *pre,*cur;
       cur = *t;
while(cur){
         pre = cur;
         if(item < cur->info)
             cur = cur->left;
         else   cur = cur->right;
  }
      if(item < pre->info)  pre->left = p;
        else  pre->right = p;
  } }
int search_To_delete(tree *t, tree_entry k){
   int found = 0; tree_node *q=*t, *r = NULL;
   while(q && !(found=(k==q->info))){
      r = q;
      if(k < q->info)     q = q->left;
      else            q = q->right;
   }
   if (found){
      if(!r) //Case of deleting the root
         delete_node(t);
      else if((k < r->info)) delete_node(&r->left);
      else  delete_node(&r->right);
   }
   return found;
}
void delete_node(tree *pt){
   tree_node *q = *pt;
   tree_node *r = NULL;

   if(!(*pt)->left)  *pt = (*pt)->right;
   else if(!(*pt)->right)  *pt = (*pt)->left;
    else { //third case
     q = (*pt)->left;
    while(q->right){
        r = q;
        q = q->right;
     }(*pt)->info = q->info;
      if(!r)            (*pt)->left = q->left;
      else             r->right = q->left
}free(q); }
```

3. **Write a C function that increment all the values of a given binary tree by one**

Solution

```
void Increament(tree *pt){
if(!*pt){
    (*pt)->info++;

       Increament((*pt)->left);
       Increament((*pt)->right);

 }

 }
```

5. **Write a C function to search for a specific value in a BST and return 1 if found and 0 otherwise.**

Solution

```
int search(tree *t, tree_entry k){
    int found = 0; tree_node *q=*t;
    while(q && !(found=(k==q->info))){

        if(k < q->info)
              q = q->left;
        else
           q = q->right;
    }

    return found;
}
```

6. Write the definition of the C function, leaves_count that takes a pointer to the root node of a binary tree as input and returns the number of leaves in a binary tree.

Solution

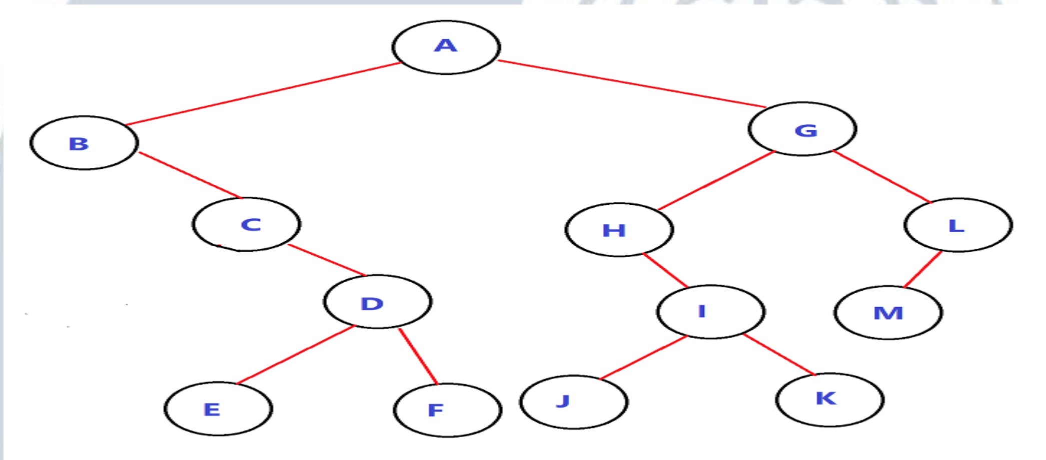
```c
int countleaves(tree *pt ){

if(!(*pt))
    return 0;

if (!(*pt)->left&&!(*pt)->right)
    return 1;

    return (countleaves(&(*pt)->left))+countleaves(&(*pt)->right))

}
```

9. **Given the preorder and inorder traversal sequences of a binary tree as follows:**

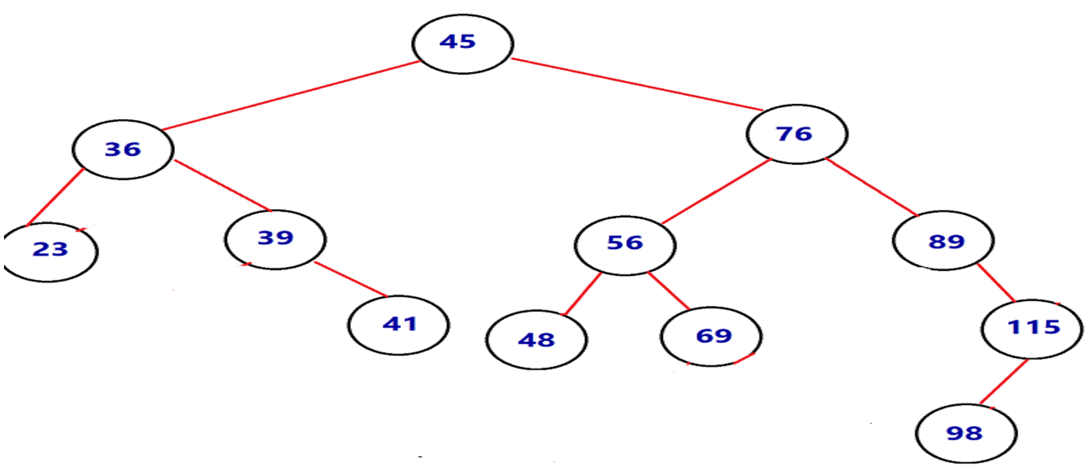
preorder: ABCDEFGHIJKLM **inorder: CEDFBAHJIKGML**

Draw the binary tree.

Solution

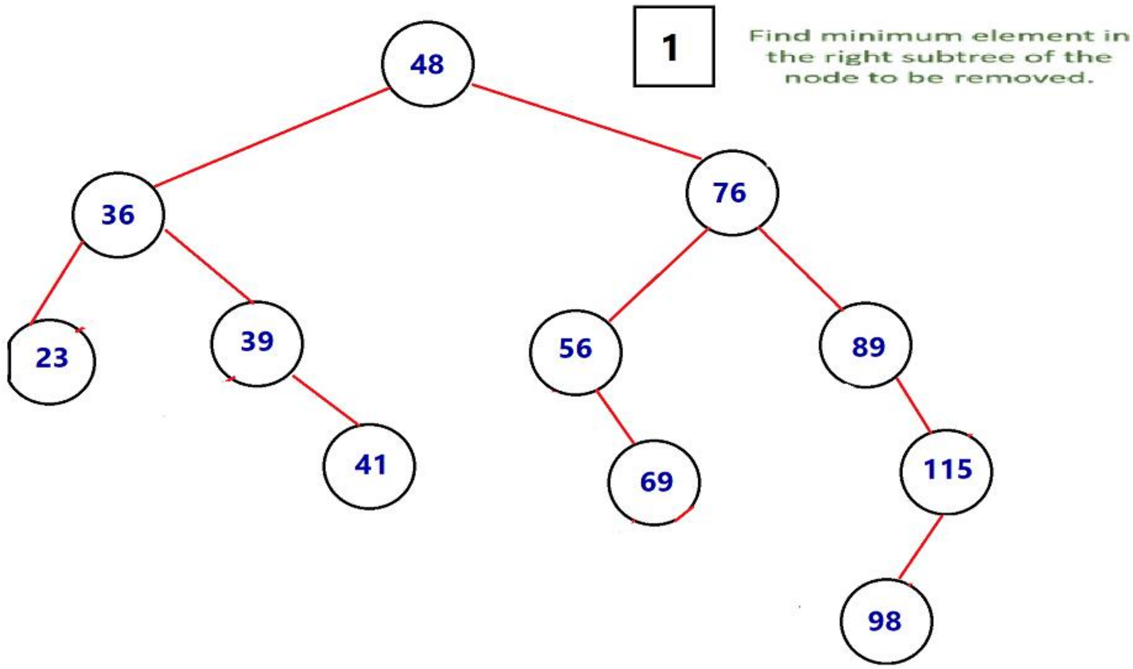


10. **From the following tree, show by drawing how to:**

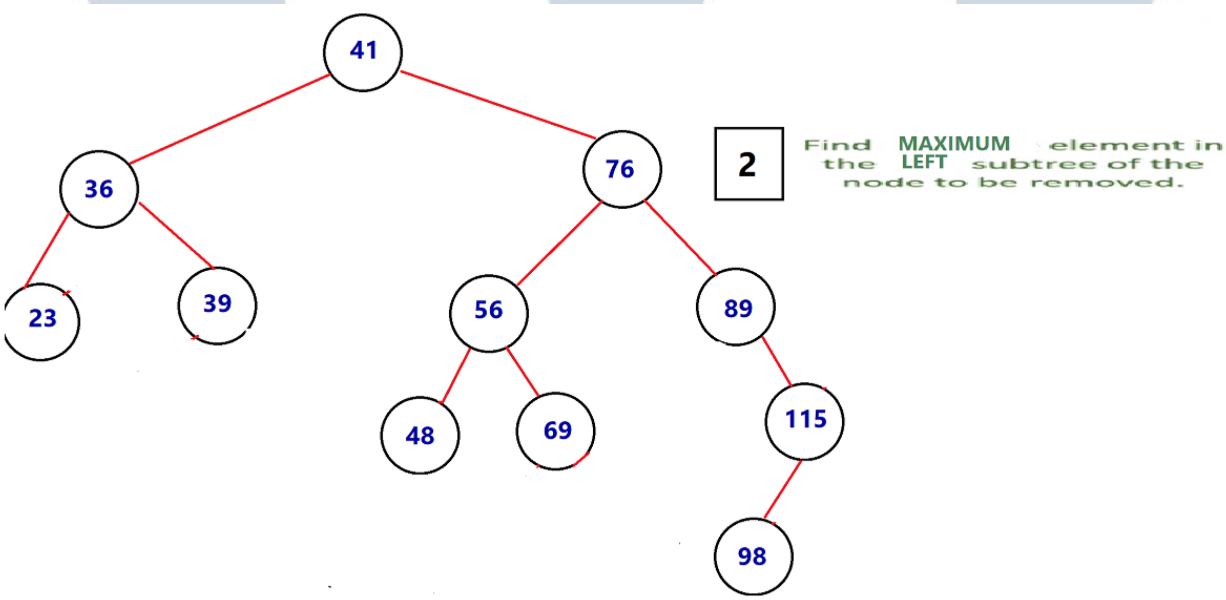


a) Delete the value 45 in two different ways.

Solution 1

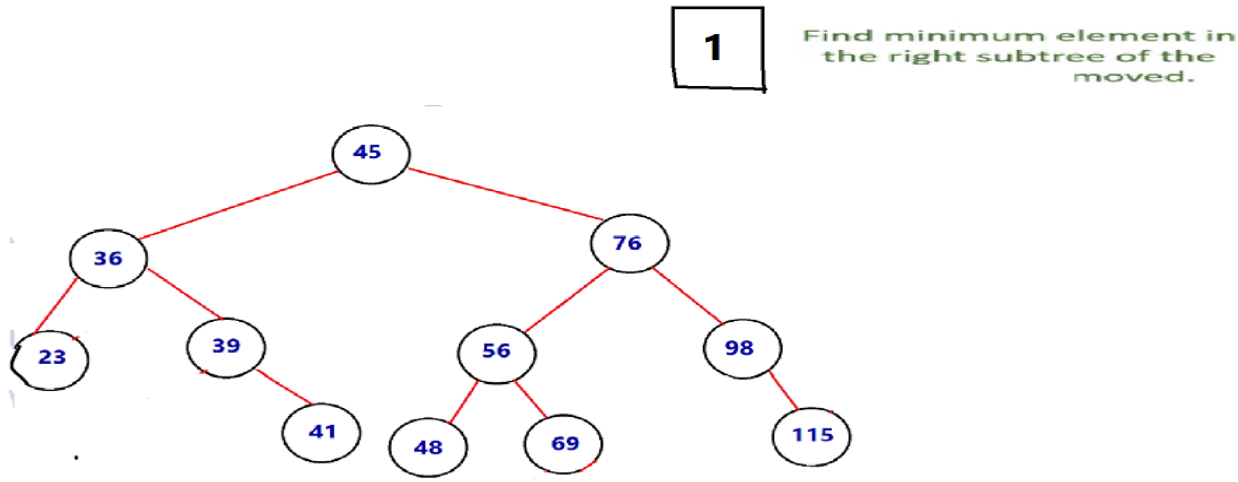


Solution 2



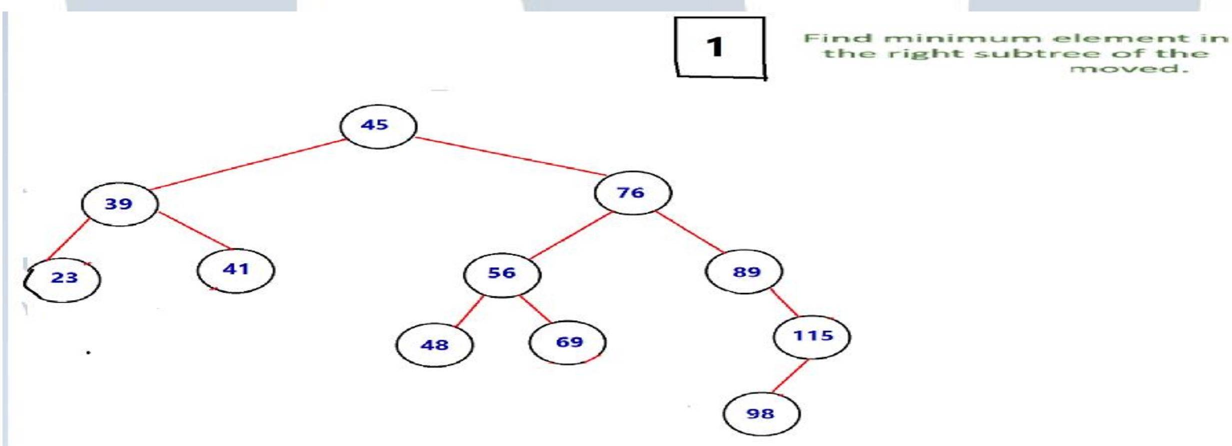
b) Delete the value 89.

Solution



c) Delete the value 36.

Solution



Minders

Don’t Let Your Be Dreams