LAB 11

M. HUZAIFA MUSTAFA

SECTION AM

SP22-BSCS-0046

Breadth First Code:

SOURCE CODE:

```
#include<iostream>

using namespace std;

class Node{

   public:

      int data;

      Node *left;

      Node *right;

      Node(int data){

         this->data = data;

         this->left = NULL;

         this->right = NULL;

      }

};

class tree{

   public:

      Node *head;

      tree(){

         this->head = NULL;

      }
```

```
void insertNode(Node *node, int value){
 if(value < node->data){
  if(node->left==NULL){
      Node *newNode = new Node(value);
      node->left = newNode;
              }
              else{
                  insertNode(node->left,value);
              }
          }
          else{
              if(node->right==NULL){
                  Node *newNode = new Node(value);
                  node->right = newNode;
              }
              else{
                  insertNode(node->right,value);
              }
          }
      }
      void inorderTraversal(Node *temp){
          if(temp!=NULL){
              inorderTraversal(temp->left);
              cout << temp->data << " ";
              inorderTraversal(temp->right);
```

```
                }

        }

        void preorderTraversal(Node *temp){

                if(temp != NULL){

                        cout<<temp->data<<" ";

                        preorderTraversal(temp->left);

                        preorderTraversal(temp->right);

                }

        }

        void postorderTraversal(Node *temp){

                if(temp!=NULL){

                postorderTraversal(temp->left);

                postorderTraversal(temp->right);

                cout << temp->data << " ";

        }

        }

        void bfsTraversal(Node *root){

Node **queue = new Node*[1000];

int front = 0, rear = 0;

queue[rear++] = root;

while (front != rear) {

  Node *curr = queue[front++];

  cout << curr->data << " ";

  if (curr->left != NULL) {

    queue[rear++] = curr->left;
```

```
        }
        if (curr->right != NULL) {
            queue[rear++] = curr->right;
        }
    }
    delete[] queue;
}
                    void print2DUtil(Node* root, int space)
{
      int COUNT = 5;
    // Base case
    if (root == NULL)
        return;

    // Increase distance between levels
    space += COUNT;

    // Process right child first
    print2DUtil(root->right, space);

    // Print current node after space
    // count
    cout << endl;
    for (int i = COUNT; i < space; i++)
        cout << " ";
```

```
  cout << root->data << "\n";

  // Process left child
  print2DUtil(root->left, space);
}

// Wrapper over print2DUtil()
void print2D(Node* root)
{
  // Pass initial space count as 0
  print2DUtil(root, 0);
}
bool isFullBinaryTree(struct Node *root) {

    // Checking for emptiness
    if (root == NULL)
     return true;

    // Checking for the presence of children
    if (root->left == NULL && root->right == NULL)
     return true;

    if ((root->left) && (root->right))
     return (isFullBinaryTree(root->left) && isFullBinaryTree(root->right));
  return false;
}
        int depth(Node *node) {
  int d = 0;
  while (node != NULL) {
    d++;
    node = node->left;
  }
  return d;
}

bool isPerfectR(struct Node *root, int d, int level = 0) {
  if (root == NULL)
    return true;

  if (root->left == NULL && root->right == NULL)
    return (d == level + 1);

  if (root->left == NULL || root->right == NULL)
    return false;

  return isPerfectR(root->left, d, level + 1) &&
       isPerfectR(root->right, d, level + 1);
}
```

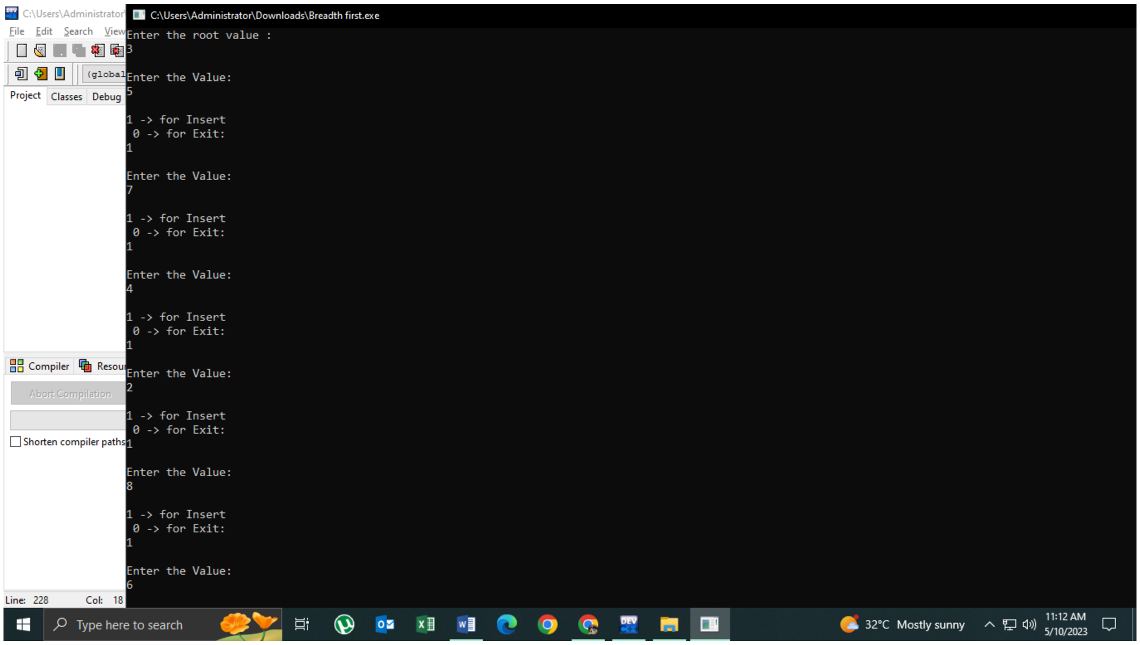
```
bool isPerfect(Node *root) {
  int d = depth(root);
  return isPerfectR(root, d);
}
int countNumNodes(struct Node *root) {
  if (root == NULL)
    return (0);
  return (1 + countNumNodes(root->left) + countNumNodes(root->right));
}

// Check if the tree is a complete binary tree
bool checkComplete(struct Node *root, int index, int numberNodes) {

  // Check if the tree is empty
  if (root == NULL)
    return true;

  if (index >= numberNodes)
    return false;

  return (checkComplete(root->left, 2 * index + 1, numberNodes) && checkComplete(root->right, 2 * index + 2, numberNodes));
}
int absDiff(int a, int b) {
    if (a > b) {
        return a - b;
    } else {
        return b - a;
    }
}

// Check height balance
bool checkHeightBalance(Node *root, int *height) {
    // Check for emptiness
    int leftHeight = 0, rightHeight = 0;

    int l = 0, r = 0;

    if (root == NULL) {
        *height = 0;
        return 1;
    }

    l = checkHeightBalance(root->left, &leftHeight);
    r = checkHeightBalance(root->right, &rightHeight);

    *height = (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;

    if (absDiff(leftHeight, rightHeight) >= 2)
        return 0;
```

```
     else
      return l && r;
    }
};


int main(){
    tree newtree;
    int rootvalue;
    cout << "Enter the root value" << endl;
    cin >> rootvalue;
    Node*root = new Node(rootvalue);
    int value;
    int i = 1;
    while(i == 1){
        cout<<"\n";
        cout<<" Enter the Value: ";
        cin>>value;
        newtree.insertNode(root, value);
        cout<<"\n";
        cout<<" 1 -> for Insert\n 0 -> for Exit: ";
        cin>>i;
    }
    cout << "inorder traversal: ";
    newtree.inorderTraversal(root);
```

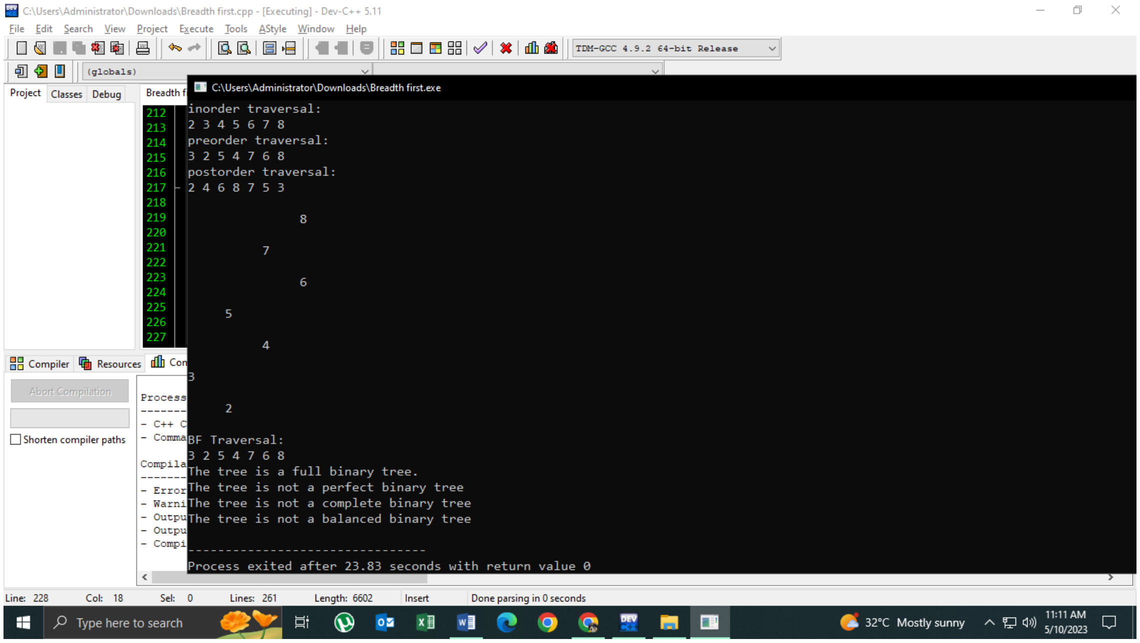
```
          cout << endl;

          cout << "preorder traversal: ";

          newtree.preorderTraversal(root);

          cout << endl;

          cout << "postorder traversal: ";

          newtree.postorderTraversal(root);

          cout << endl;

          newtree.print2D(root);

          cout << "BF Traversal: " << endl;

          newtree.bfsTraversal(root);

          if(newtree.isFullBinaryTree(root)){

   cout << "The tree is a full binary tree." << endl;

}

else{

   cout << "The tree is not a full binary tree." << endl;

}

if(newtree.isPerfect(root)){

          cout << "The tree is a perfect binary tree" << endl;

          }

          else{

                    cout << "The tree is not a perfect binary tree" << endl;

          }

          int node_count = newtree.countNumNodes(root);

          int index = 0;

          if(newtree.checkComplete(root, index, node_count)){
```

```
        cout << "The tree is a complete binary tree" << endl;

    }

    else{

        cout << "The tree is not a complete binary tree" << endl;

    }

    int height = 0;

    if(newtree.checkHeightBalance(root, &height)){

        cout << "The tree is a balanced binary tree" << endl;

    }

    else{

        cout << "The tree is not a balanced binary tree" << endl;

    }

    return 0;

}
```

PICTURE:



```
Enter the root value :
3

Enter the Value:
5

1 -> for Insert
 0 -> for Exit:
1

Enter the Value:
7

1 -> for Insert
 0 -> for Exit:
1

Enter the Value:
4

1 -> for Insert
 0 -> for Exit:
1

Enter the Value:
2

1 -> for Insert
 0 -> for Exit:
1

Enter the Value:
8

1 -> for Insert
 0 -> for Exit:
1

Enter the Value:
6
```



C:\Users\Administrator\Downloads\Breadth first.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Compiler Resources
Abort Compilation
Shorten compiler paths
C:\Users\Administrator\Downloads\Breadth first.exe
Line: 228 Col: 18 Sel: 0 Lines: 261 Length: 6602 Insert Done parsing in 0 seconds
Type here to search
32°C Mostly sunny
11:11 AM
5/10/2023

```
inorder traversal:
2 3 4 5 6 7 8
preorder traversal:
3 2 5 4 7 6 8
postorder traversal:
2 4 6 8 7 5 3

               8

          7

               6

     5

          4

3

     2

BF Traversal:
3 2 5 4 7 6 8
The tree is a full binary tree.
The tree is not a perfect binary tree
The tree is not a complete binary tree
The tree is not a balanced binary tree

--------------------------------
Process exited after 23.83 seconds with return value 0
```

SOURCE CODE CPP FILE:

Breadth first.cpp

TIME COMPLEXITY OF THE CODE:

After adding the counter to the code to find the time complexity and Divided the counter by processor GHz.

SOURCE CODE:

```
#include<iostream>

using namespace std;

class Node{

    public:

        int data;

        Node *left;

        Node *right;

        Node(int data){

            this->data = data;

            this->left = NULL;
```

```
      this->right = NULL;
    }
};
class tree{
  public:
    Node *head;
       float counter;
    tree(){
      this->head = NULL;
       this->counter = 0;
    }
    void insertNode(Node *node, int value){
       counter++;
      if(value < node->data){
       if(node->left==NULL){
              Node *newNode = new Node(value);
              node->left = newNode;
              counter++;
                            }
                            else{
                                   insertNode(node->left,value);
                                   counter++;

                            }
                     }
    else{
        if(node->right==NULL){
            Node *newNode = new Node(value);
            node->right = newNode;
            counter++;

        }
        else{
            insertNode(node->right,value);
            counter++;
        }
    }
}
void inorderTraversal(Node *temp){
    counter++;
    if(temp!=NULL){
        inorderTraversal(temp->left);
        cout << temp->data << " ";
        inorderTraversal(temp->right);
    }
}
void preorderTraversal(Node *temp){
    counter++;
    if(temp != NULL){
        cout<<temp->data<<" ";
```

```
                                                preorderTraversal(temp->left);

                                                preorderTraversal(temp->right);

                                }

                }

                void postorderTraversal(Node *temp){

                                counter++;

                                if(temp!=NULL){

                                postorderTraversal(temp->left);

                                postorderTraversal(temp->right);

                                cout << temp->data << " ";

                                }

                }

                void bfsTraversal(Node *root){

Node **queue = new Node*[1000];

int front = 0, rear = 0;

queue[rear++] = root;

while (front != rear) {

    Node *curr = queue[front++];

    cout << curr->data << " ";

    if (curr->left != NULL) {

        queue[rear++] = curr->left;

        counter++;

    }

    if (curr->right != NULL) {

        queue[rear++] = curr->right;
```

```
            counter++;
          }
      }
      delete[] queue;
  }
                    void print2DUtil(Node* root, int space)
  {
        int COUNT = 5;
      // Base case
      if (root == NULL)
          return;

      // Increase distance between levels
      space += COUNT;

      // Process right child first
      print2DUtil(root->right, space);

      // Print current node after space
      // count
      cout << endl;
      for (int i = COUNT; i < space; i++)
          cout << " ";
      cout << root->data << "\n";
  // Process left child

  print2DUtil(root->left, space);

}


// Wrapper over print2DUtil()

void print2D(Node* root)

{

  // Pass initial space count as 0

  print2DUtil(root, 0);

}

bool isFullBinaryTree(struct Node *root) {

          counter++;

    // Checking for emptiness

    if (root == NULL)

     return true;


    // Checking for the presence of children

    if (root->left == NULL && root->right == NULL)

     return true;


    if ((root->left) && (root->right))

     return (isFullBinaryTree(root->left) && isFullBinaryTree(root->right));


    return false;

   }
```

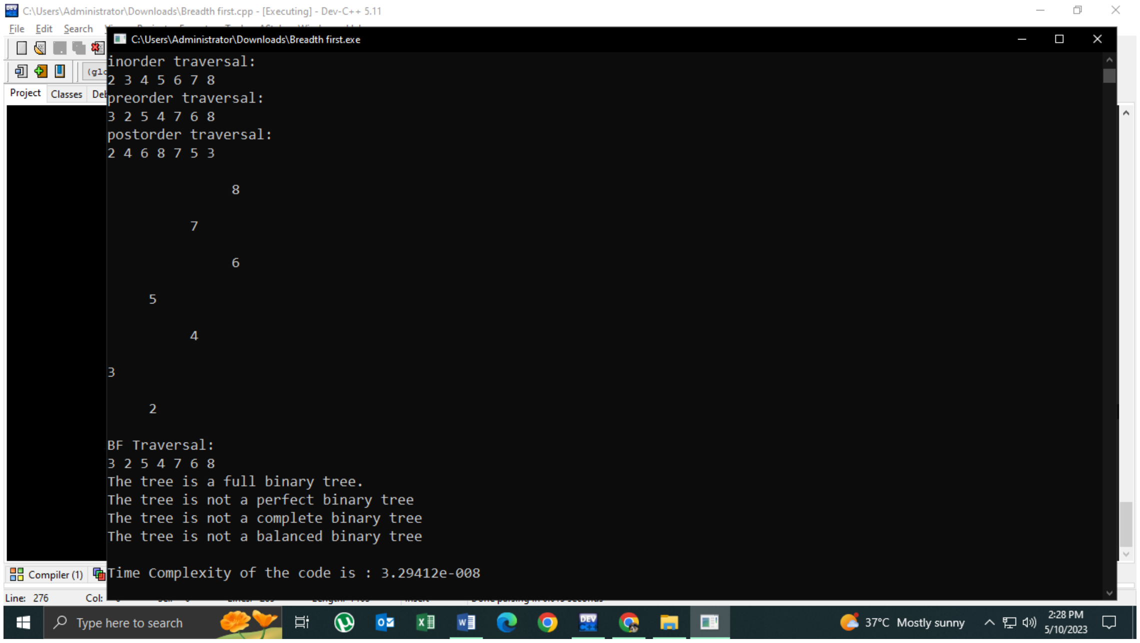
```
    int depth(Node *node) {
              int d = 0;
              while (node != NULL) {
              d++;
              node = node->left;
              }
              counter++;
  return d;
}

bool isPerfectR(struct Node *root, int d, int level = 0) {
              counter++;
  if (root == NULL)
    return true;

  if (root->left == NULL && root->right == NULL)
    return (d == level + 1);

  if (root->left == NULL || root->right == NULL)
    return false;

  return isPerfectR(root->left, d, level + 1) &&
       isPerfectR(root->right, d, level + 1);
}
```

```
bool isPerfect(Node *root) {
  int d = depth(root);
  return isPerfectR(root, d);
}
int countNumNodes(struct Node *root) {
  if (root == NULL)
    return (0);
  return (1 + countNumNodes(root->left) + countNumNodes(root->right));
}

// Check if the tree is a complete binary tree
bool checkComplete(struct Node *root, int index, int numberNodes) {
              counter++;
  // Check if the tree is empty
  if (root == NULL)
    return true;

  if (index >= numberNodes)
    return false;

  return (checkComplete(root->left, 2 * index + 1, numberNodes) && checkComplete(root->right, 2 * index + 2, numberNodes));
}
int absDiff(int a, int b) {
    if (a > b) {
        return a - b;
    } else {
        return b - a;
    }
}

// Check height balance
bool checkHeightBalance(Node *root, int *height) {
              counter++;
    // Check for emptiness
    int leftHeight = 0, rightHeight = 0;

    int l = 0, r = 0;

    if (root == NULL) {
        *height = 0;
        return 1;
    }

    l = checkHeightBalance(root->left, &leftHeight);
    r = checkHeightBalance(root->right, &rightHeight);

    *height = (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;

    if (absDiff(leftHeight, rightHeight) >= 2)
        return 0;
```

```
           else
             return l && r;
        }
        float Counter(){
     return counter;
  }

};



int main(){
        tree newtree;
        int rootvalue;
        cout << "Enter the root value : " << endl;
        cin >> rootvalue;
        Node*root = new Node(rootvalue);
        int value;
        int i = 1;
        while(i == 1){
                cout<<"\n";
                cout<<"Enter the Value: "<<endl;
                cin>>value;
                newtree.insertNode(root, value);
                cout<<endl;
                cout<<"1 -> for Insert\n0 -> for Exit: "<<endl;

                cin>>i;

        }

        cout << "inorder traversal: "<<endl;

        newtree.inorderTraversal(root);

        cout << endl;

        cout << "preorder traversal: "<<endl;

        newtree.preorderTraversal(root);

        cout << endl;

        cout << "postorder traversal: "<<endl;

        newtree.postorderTraversal(root);

        cout << endl;

        newtree.print2D(root);

        cout << endl;

        cout << "BF Traversal: " << endl;

        newtree.bfsTraversal(root);

        cout<<endl;

        if(newtree.isFullBinaryTree(root)){

   cout << "The tree is a full binary tree." << endl;

}

else{

   cout << "The tree is not a full binary tree." << endl;

}

if(newtree.isPerfect(root)){

        cout << "The tree is a perfect binary tree" << endl;
```

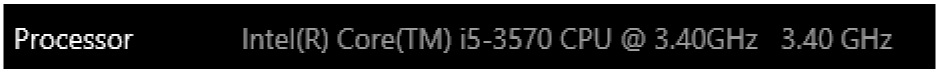
```
    }
    else{
        cout << "The tree is not a perfect binary tree" << endl;
    }
    int node_count = newtree.countNumNodes(root);
    int index = 0;
    if(newtree.checkComplete(root, index, node_count)){
        cout << "The tree is a complete binary tree" << endl;
    }
    else{
        cout << "The tree is not a complete binary tree" << endl;
    }
    int height = 0;
    if(newtree.checkHeightBalance(root, &height)){
        cout << "The tree is a balanced binary tree" << endl;
    }
    else{
        cout << "The tree is not a balanced binary tree" << endl;
    }
    cout<<endl;
    cout << "Time Complexity of the code is : " << newtree.Counter()/3400000000 <<endl;

    return 0;
}
```

PICTURE:


C:\Users\Administrator\Downloads\Breadth first.cpp - [Executing] - Dev-C++ 5.11
File
Edit
Search
Project
Classes
C:\Users\Administrator\Downloads\Breadth first.exe
Compiler (1)
Line: 276
Col:
Type here to search
37°C Mostly sunny
2:28 PM
5/10/2023

```
inorder traversal:
2 3 4 5 6 7 8
preorder traversal:
3 2 5 4 7 6 8
postorder traversal:
2 4 6 8 7 5 3

               8

          7

               6

     5

          4

3

     2

BF Traversal:
3 2 5 4 7 6 8
The tree is a full binary tree.
The tree is not a perfect binary tree
The tree is not a complete binary tree
The tree is not a balanced binary tree

Time Complexity of the code is : 3.29412e-008
```

G.H.Z OF THE CPU:



Processor Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz 3.40 GHz