

data Structure

PROJECT



Code:

#include <iostream>

#include"graphics.h"

using namespace std;

void menu()

{

setcolor(GREEN);

int left, top, right, bottom;

left = 80;

top = 9;

right = 220;

bottom = 41;

rectangle(left, top, right, bottom);

moveto(89, 15);

outtext("Insert:");

rectangle(900, 5, 940, 40);

outtextxy(905, 15, "EXIT");

rectangle(700, 5, 850, 30);

outtextxy(705, 10, "Search:");

}

class tree\_node\_BST

{

public:

int data;

tree\_node\_BST\* right;

tree\_node\_BST\* left;

tree\_node\_BST(int val)

{

data = val;

right = NULL;

left = NULL;

}

};

class binary\_tree

{

private:

tree\_node\_BST\* root;

public:

binary\_tree()

{

root = NULL;

}

bool insert\_node(int n)

{

tree\_node\_BST\* new\_node = new tree\_node\_BST(n);

if (root == NULL)

{

root = new\_node;

}

else

{

tree\_node\_BST\* temp = root;

while (true)

{

if (n < temp->data)

{

if (temp->left != NULL)

{

temp = temp->left;

}

else

{

temp->left = new\_node;

return true;

}

}

else if (n > temp->data)

{

if (temp->right != NULL)

{

temp = temp->right;

}

else

{

temp->right = new\_node;

return true;

}

}

else if (n == temp->data)

{

return true;

}

}

}

return false;

}

void printTree\_BST(int x, int y, tree\_node\_BST\* root, int index)

{

tree\_node\_BST\* left;

tree\_node\_BST\* right;

int lx, ry;

char str[10];

//Base case of our recursive function

if (!root)

return;

// Convert int value into string

itoa(root->data, str, 10);

circle(x, y, 15);

outtextxy(x - 5, y - 7, str);

left = root->left;

right = root->right;

lx = 2 \* index + 1;

ry = 2 \* index + 1;

printTree\_BST(x - y / (index + 1), y + 60, left, lx);

printTree\_BST(x + y / (index + 1), y + 60, right, ry);

// Draw the line (or link) when the node is not the leaf node

if (left != NULL)

{

delay(200);

line(x, y + 14, x - y / (index + 1), y + 60 - 14);

}

if (right != NULL)

{

delay(200);

line(x, y + 14, x + y / (index + 1), y + 60 - 14);

}

}

void draw\_tree\_BST()

{

outtextxy(500, 500, "----BST----");

int x = 600;

int y = 200;

printTree\_BST(x, y, root, 0);

delay(900);

}

void to\_find\_BST(int x, int y, tree\_node\_BST\* root, int index, int val, bool& check)

{

tree\_node\_BST\* left;

tree\_node\_BST\* right;

int lx, ry;

char str[10];

//Base case of our recursive function

if (!root)

{

return;

}

int temp = root->data;

// Convert int value into string

itoa(root->data, str, 10);

setcolor(GREEN);

circle(x, y, 15);

delay(200);

outtextxy(x - 5, y - 7, str);

delay(100);

if (val == root->data)

{

outtextxy(x + 30, y - 10, " <- VALUE FOUNDED");

check = true;

return;

}

lx = 2 \* index + 1;

ry = 2 \* index + 1;

to\_find\_BST(x - y / (index + 1), y + 60, root->left, lx, val, check);

to\_find\_BST(x + y / (index + 1), y + 60, root->right, ry, val, check);

// Draw the line (or link) when the node is not the leaf node

if (root->left)

{

//delay(200);

line(x, y + 14, x - y / (index + 1), y + 60 - 14);

}

if (root->right)

{

//delay(200);

line(x, y + 14, x + y / (index + 1), y + 60 - 14);

}

//return;

}

void find\_in\_BST(int val)

{

cleardevice();

menu();

int x = 600;

int y = 200;

bool check = false;

to\_find\_BST(x, y, root, 0, val, check);

if (!check)

{

outtextxy(500, 500, "--Value Not Founded--");

}

delay(2000);

cleardevice();

}

~binary\_tree()

{

}

};

class tree\_node

{

public:

int data;

tree\_node\* right;

tree\_node\* left;

binary\_tree obj;

tree\_node(int val)

{

data = val;

right = NULL;

left = NULL;

}

};

class AVL\_Tree

{

private:

tree\_node\* root;

public:

int temp, temp\_01;

AVL\_Tree()

{

temp = 0;

temp\_01 = 0;

root = NULL;

}

bool is\_find(int num)

{

tree\_node\* temp = root;

while (temp)

{

if (temp->data == num)

{

cout << num << " :is present in tree" << endl;

return true;

}

else if (num < temp->data)

{

temp = temp->left;

}

else

{

temp = temp->right;

}

}

cout << "Value isn't present in tree" << endl;

return false;

}

int count\_height(tree\_node\* root)

{

//if there's no root exist such that it has -1 height

//so returning -1

if (root == NULL)

{

return -1;

}

//if there exist some elements

//Root !=NULL so therefore

else

{

//Recursive call of count\_height

//such that the tree will be get traversed

//to the left subtree recursively

//this chunk of Code will recursively

//iterate the tree such like (Post\_Order Traversal)

//thus height of left sub-tree and right subtree

//will thus get measured!

int left\_height = count\_height(root->left);

int right\_height = count\_height(root->right);

//Adding one in both left\_height and right\_hieght

//because height is start from the root to the

//most depth node and 1 is thus added which is the

//addition of one extra because tree height start from 1-2-3..

left\_height += 1;

right\_height += 1;

//this if-else will return greater height of the tree

//if left\_height is greater then right one

//then height of left\_height will return

//and vice versa

if (left\_height > right\_height)

{

return (left\_height);

}

else

{

return (right\_height);

}

}

}

tree\_node\* left\_rotation(tree\_node\* node)

{

tree\_node\* temp\_01 = node->right;

tree\_node\* temp\_02 = temp\_01->left;

{

temp\_01->left = node;

node->right = temp\_02;

}

return temp\_01;

}

tree\_node\* right\_rotation(tree\_node\* node)

{

tree\_node\* temp\_01 = node->left;

tree\_node\* temp\_02 = temp\_01->right;

{

temp\_01->right = node;

node->left = temp\_02;

}

return temp\_01;

}

int for\_get\_balance(tree\_node\* root)

{

if (!root)

{

return 0;

}

else

{

return count\_height(root->left) - count\_height(root->right);

}

}

tree\_node\* insert(tree\_node\*& root, int data, int val\_to\_BST)

{

tree\_node\* new\_node = new tree\_node(data);

if (!root)

{

cout << "Inserted" << endl;

root = new\_node;

root->obj.insert\_node(val\_to\_BST);

cleardevice();

menu();

outtextxy(500, 500, "---BST---");

root->obj.draw\_tree\_BST();

cleardevice();

menu();

return root;

}

if (root->data > data)

{

root->left = insert(root->left, data, val\_to\_BST);

}

else if (root->data < data)

{

root->right = insert(root->right, data, val\_to\_BST);

}

else

{

//Duplication....

root->obj.insert\_node(val\_to\_BST);

cleardevice();

menu();

outtextxy(500, 500, "---BST---");

root->obj.draw\_tree\_BST();

cleardevice();

menu();

return root;

}

int check\_balace\_fac = for\_get\_balance(root);

if (check\_balace\_fac > 1 && new\_node->data < root->left->data)

{

return right\_rotation(root);

}

if (check\_balace\_fac < -1 && new\_node->data > root->right->data)

{

return left\_rotation(root);

}

if (check\_balace\_fac > 1 && new\_node->data > root->left->data)

{

root->left = left\_rotation(root->left);

return right\_rotation(root);

}

if (check\_balace\_fac < -1 && new\_node->data < root->right->data)

{

root->right = right\_rotation(root->right);

return left\_rotation(root);

}

return root;

}

void insert\_data(int val)

{

int temp = (val % 10);

root = insert(root, temp, val);

return;

}

void printTree(int x, int y, tree\_node\* root, int dis\_var)

{

tree\_node\* left;

tree\_node\* right;

int left\_coord, right\_cord;

char str[10];

//Base case of our recursive function

if (!root)

{

return;

}

// Convert int value into string

itoa(root->data, str, 10);

circle(x, y, 15);

outtextxy(x - 5, y - 7, str);

left = root->left;

right = root->right;

left\_coord = 2 \* dis\_var + 1;

right\_cord = 2 \* dis\_var + 1;

printTree(x - y / (dis\_var + 1), y + 60, left, left\_coord);

printTree(x + y / (dis\_var + 1), y + 60, right, right\_cord);

// Draw the line (or link) when the node is not the leaf node

if (left != NULL)

{

delay(200);

line(x, y + 14, x - y / (dis\_var + 1), y + 46);

}

if (right != NULL)

{

delay(200);

line(x, y + 14, x + y / (dis\_var + 1), y + 46);

}

}

void draw\_tree()

{

outtextxy(500, 500, "---AVL---");

int x = 600;

int y = 200;

printTree(x, y, root, 0);

return;

}

void to\_print()

{

root->obj.draw\_tree\_BST();

return;

}

void to\_find(int x, int y, tree\_node\* root, int dis\_var, int temp, int to\_BST)

{

tree\_node\* left;

tree\_node\* right;

int left\_cor, right\_cor;

char str[10];

//Base case of our recursive function

if (!root)

{

return;

}

// Convert int value into string

itoa(root->data, str, 10);

setcolor(GREEN);

circle(x, y, 15);

delay(200);

outtextxy(x - 5, y - 7, str);

delay(100);

if (temp == root->data)

{

outtextxy(x + 15, y - 10, " <- VALUE FOUNDED");

delay(300);

root->obj.find\_in\_BST(to\_BST);

return;

}

left = root->left;

right = root->right;

left\_cor = 2 \* dis\_var + 1;

right\_cor = 2 \* dis\_var + 1;

to\_find(x - y / (dis\_var + 1), y + 60, left, left\_cor, temp, to\_BST);

to\_find(x + y / (dis\_var + 1), y + 60, right, right\_cor, temp, to\_BST);

// Draw the line (or link) when the node is not the leaf node

if (left != NULL)

{

line(x, y + 14, x - y / (dis\_var + 1), y + 46);

}

if (right != NULL)

{

line(x, y + 14, x + y / (dis\_var + 1), y + 46);

}

}

void find\_in(int val)

{

int temp = (val % 10);

int x = 600;

int y = 200;

to\_find(x, y, root, 0, temp, val);

cleardevice();

}

};

int input()

{

int x = 140;

int y = 15;

int i = -1;

int cursor\_x = 140;

int cursor\_y = 15;

int temp\_cursor\_x = 140;

char arr[8];

char arr\_02[2];

arr\_02[1] = '\0';

arr[7] = '\0';

bool to\_termi = true;

outtextxy(cursor\_x, cursor\_y, "|");

while (to\_termi)

{

char c = getch();

//index reached max

if (i == 7)

{

to\_termi = false;

break;

}

if (c == 32)

{

continue;

}

if (c != 14 && c != 8)

{

i++;

arr[i] = c;

arr\_02[0] = c;

outtextxy(x, y, arr\_02);

x += 10;

cursor\_x += 10;

outtextxy(cursor\_x, cursor\_y, "|");

}

if (c == 13)

{

to\_termi = false;

for (i; i <= 7; i++)

{

arr[i] = '\0';

}

break;

}

if (c == 8)

{

if (i == -1)

{

continue;

}

arr[i] = '\0';

x -= 10;

cursor\_x -= 10;

outtextxy(x, y, "| ");

i = i - 1;

}

}

for (int i = 0; i < 8; i++)

{

if (arr[i] == '\0')

{

continue;

}

if (arr[i] < 48 || arr[i]>57)

{

outtextxy(500, 500, "invalid input try Again");

input();

}

}

outtextxy(500, 500, "Input Success");

return stoi(arr);

}

int search\_input()

{

int x = 750;

int y = 10;

int i = -1;

int cursor\_x = 750;

int cursor\_y = 10;

int temp\_cursor\_x = 750;

char arr[8];

char arr\_02[2];

arr\_02[1] = '\0';

arr[7] = '\0';

bool to\_termi = true;

outtextxy(cursor\_x, cursor\_y, "|");

while (to\_termi)

{

char c = getch();

if (i == 7)

{

to\_termi = false;

outtextxy(600, 600, "Enter pressed");

break;

}

if (c != 14 && c != 8)

{

i++;

arr[i] = c;

arr\_02[0] = c;

outtextxy(x, y, arr\_02);

x += 10;

cursor\_x += 10;

outtextxy(cursor\_x, cursor\_y, "|");

}

if (c == 13)

{

to\_termi = false;

for (i; i <= 7; i++)

arr[i] = '\0';

break;

}

if (c == 8)

{

if (i == -1)

{

continue;

}

arr[i] = '\0';

x -= 10;

cursor\_x -= 10;

outtextxy(x, y, "| ");

i = i - 1;

}

}

for (int i = 0; i < 8; i++)

{

if (arr[i] == '\0')

{

continue;

}

if (arr[i] < 48 || arr[i]>57)

{

outtextxy(500, 500, "invalid input try Again");

input();

}

}

outtextxy(500, 500, "Input Success");

for (int i = 0; i < 7; i++)

{

cout << arr[i] << " ";

}

return stoi(arr);

}

int main()

{

int left, top, right, bottom;

left = 80;

top = 9;

right = 220;

bottom = 41;

int x, y, value;

initwindow(1200, 600);

cleardevice();

readimagefile("final.jpg",0,0,1200,600);

getch();

readimagefile("loadd.jpg", 0, 0, 1200, 600);

delay(2000);

initwindow(1200, 600);

setcolor(GREEN);

menu();

AVL\_Tree obj;

bool check = true;

while (check)

{

while (!ismouseclick(WM\_LBUTTONDOWN))

{

if (ismouseclick(WM\_LBUTTONDOWN))

{

getmouseclick(WM\_LBUTTONDOWN, x, y);

if (x >= left && x <= right && y >= top && y <= bottom)

{

value = input();

obj.insert\_data(value);

cleardevice();

menu();

obj.draw\_tree();

check = true;

break;

}

if (x >= 900 && x <= 940 && y >= 5 && y <= 40)

{

readimagefile("exit\_01.jpg", 0, 0, 1200, 600);

delay(2000);

return 0;

}

if (x >= 700 && x <= 850 && y >= 5 && y <= 30)

{

value = search\_input();

int temp = value % 10;

bool check = (obj.is\_find(temp));

if (check)

{

obj.find\_in(value);

delay(500);

cleardevice();

menu();

obj.draw\_tree();

}

else

{

outtextxy(500, 500, "Value is not Present in TREE");

delay(1000);

cleardevice();

menu();

obj.draw\_tree();

}

}

}

}

}

system("pause");

return 0;

}

Calendar

Description automatically generated with medium confidence

A screenshot of a video game

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing text

Description automatically generated