**15B17CI371 – Data Structures Lab**

**ODD 2024**

**Week 9-LAB A**

**Practice Lab**

**Q1. Write a C/C++ program to insert elements into a Red-Black Tree and ensure the**

**tree maintains its balancing properties.**

**Sample Input: Insert the following elements in sequence: 10,20,30,15,25,5.**

**Output: Display the Red-Black Tree after each insertion.**

#include <iostream>

using namespace std;

enum Color { RED,BLACK };

struct Node

{

int data;

Node \*left,\*right,\*parent;

Color color;

Node(int data)

{

this->data=data;

left=right=parent=nullptr;

this->color=RED;

}

};

class RedBlackTree

{

Node \*root;

void rotateLeft(Node \*&node)

{

Node \*rightChild=node->right;

node->right=rightChild->left;

if(node->right != nullptr)

node->right->parent=node;

rightChild->parent=node->parent;

if(node->parent==nullptr)

root=rightChild;

else if(node==node->parent->left)

node->parent->left=rightChild;

else

node->parent->right=rightChild;

rightChild->left=node;

node->parent=rightChild;

}

void rotateRight(Node \*&node)

{

Node \*leftChild=node->left;

node->left=leftChild->right;

if(node->left != nullptr)

node->left->parent=node;

leftChild->parent=node->parent;

if(node->parent==nullptr)

root=leftChild;

else if(node==node->parent->left)

node->parent->left=leftChild;

else

node->parent->right=leftChild;

leftChild->right=node;

node->parent=leftChild;

}

void fixViolation(Node \*&node)

{

Node \*parent=nullptr;

Node \*grandparent=nullptr;

while(node != root&&node->color != BLACK&&node->parent->color==RED)

{

parent=node->parent;

grandparent=node->parent->parent;

if(parent==grandparent->left)

{

Node \*uncle=grandparent->right;

if(uncle != nullptr&&uncle->color==RED)

{

grandparent->color=RED;

parent->color=BLACK;

uncle->color=BLACK;

node=grandparent;

}

else

{

if(node==parent->right)

{

rotateLeft(parent);

node=parent;

parent=node->parent;

}

rotateRight(grandparent);

swap(parent->color,grandparent->color);

node=parent;

}

}

else

{

Node \*uncle=grandparent->left;

if(uncle != nullptr&&uncle->color==RED)

{

grandparent->color=RED;

parent->color=BLACK;

uncle->color=BLACK;

node=grandparent;

}

else

{

if(node==parent->left)

{

rotateRight(parent);

node=parent;

parent=node->parent;

}

rotateLeft(grandparent);

swap(parent->color,grandparent->color);

node=parent;

}

}

}

root->color=BLACK;

}

void inorderHelper(Node \*node)

{

if(node==nullptr)

return;

inorderHelper(node->left);

cout<<node->data <<(node->color==RED?"R " : "B ");

inorderHelper(node->right);

}

public:

RedBlackTree() { root=nullptr;}

void insert(const int &data)

{

Node \*node=new Node(data);

Node \*parent=nullptr;

Node \*current=root;

while(current != nullptr)

{

parent=current;

if(node->data<current->data)

current=current->left;

else

current=current->right;

}

node->parent=parent;

if(parent==nullptr)

root=node;

else if(node->data<parent->data)

parent->left=node;

else

parent->right=node;

if(node->parent==nullptr)

{

node->color=BLACK;

return;

}

if(node->parent->parent==nullptr)

return;

fixViolation(node);

}

void display()

{

inorderHelper(root);

cout<<endl;

}

};

int main()

{

RedBlackTree tree;

int n;

cout<<"Input the number of elements : ";

cin>>n;

int elements[n];

cout<<"Input the elements : ";

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

{

cin>>elements[i];

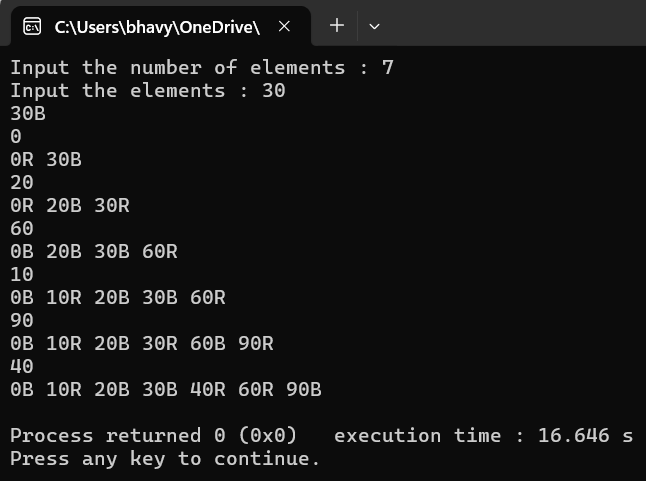
tree.insert(elements[i]);

tree.display();

}

}

**Output :**

****

**Q2. Write a program to calculate the height of a Red-Black Tree.**

**Sample Input: Insert elements [20,15,30,10,25,35],then compute the height**

**of the tree.**

**Output: Display the height of the Red-Black Tree**

#include <iostream>

using namespace std;

enum Color { RED,BLACK };

struct Node

{

int data;

Node \*left,\*right,\*parent;

Color color;

Node(int data)

{

this->data=data;

left=right=parent=nullptr;

this->color=RED;

}

};

class RedBlackTree

{

Node \*root;

void rotateLeft(Node \*&node)

{

Node \*rightChild=node->right;

node->right=rightChild->left;

if(node->right != nullptr)

node->right->parent=node;

rightChild->parent=node->parent;

if(node->parent==nullptr)

root=rightChild;

else if(node==node->parent->left)

node->parent->left=rightChild;

else

node->parent->right=rightChild;

rightChild->left=node;

node->parent=rightChild;

}

void rotateRight(Node \*&node)

{

Node \*leftChild=node->left;

node->left=leftChild->right;

if(node->left != nullptr)

node->left->parent=node;

leftChild->parent=node->parent;

if(node->parent==nullptr)

root=leftChild;

else if(node==node->parent->left)

node->parent->left=leftChild;

else

node->parent->right=leftChild;

leftChild->right=node;

node->parent=leftChild;

}

void fixViolation(Node \*&node)

{

Node \*parent=nullptr;

Node \*grandparent=nullptr;

while(node != root&&node->color != BLACK&&node->parent->color==RED)

{

parent=node->parent;

grandparent=node->parent->parent;

if(parent==grandparent->left)

{

Node \*uncle=grandparent->right;

if(uncle != nullptr&&uncle->color==RED)

{

grandparent->color=RED;

parent->color=BLACK;

uncle->color=BLACK;

node=grandparent;

}

else

{

if(node==parent->right)

{

rotateLeft(parent);

node=parent;

parent=node->parent;

}

rotateRight(grandparent);

swap(parent->color,grandparent->color);

node=parent;

}

}

else

{

Node \*uncle=grandparent->left;

if(uncle != nullptr&&uncle->color==RED)

{

grandparent->color=RED;

parent->color=BLACK;

uncle->color=BLACK;

node=grandparent;

}

else

{

if(node==parent->left)

{

rotateRight(parent);

node=parent;

parent=node->parent;

}

rotateLeft(grandparent);

swap(parent->color,grandparent->color);

node=parent;

}

}

}

root->color=BLACK;

}

void inorderHelper(Node \*node)

{

if(node==nullptr)

return;

inorderHelper(node->left);

cout<<node->data <<(node->color==RED?"R " : "B ");

inorderHelper(node->right);

}

int calculateHeight(Node\* node)

{

if(node==nullptr)

return -1;

int leftHeight=calculateHeight(node->left);

int rightHeight=calculateHeight(node->right);

return max(leftHeight,rightHeight)+1;

}

public:

RedBlackTree() { root=nullptr;}

void insert(const int &data)

{

Node \*node=new Node(data);

Node \*parent=nullptr;

Node \*current=root;

while(current != nullptr)

{

parent=current;

if(node->data<current->data)

current=current->left;

else

current=current->right;

}

node->parent=parent;

if(parent==nullptr)

root=node;

else if(node->data<parent->data)

parent->left=node;

else

parent->right=node;

if(node->parent==nullptr)

{

node->color=BLACK;

return;

}

if(node->parent->parent==nullptr)

return;

fixViolation(node);

}

void display()

{

inorderHelper(root);

cout<<endl;

}

void displayHeight()

{

cout<<"Height of the Red-Black Tree: "<<calculateHeight(root)<<endl;

}

};

int main()

{

RedBlackTree tree;

int n;

cout<<"Input the number of elements : ";

cin>>n;

int elements[n];

cout<<"Input the elements : ";

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

{

cin>>elements[i];

tree.insert(elements[i]);

}

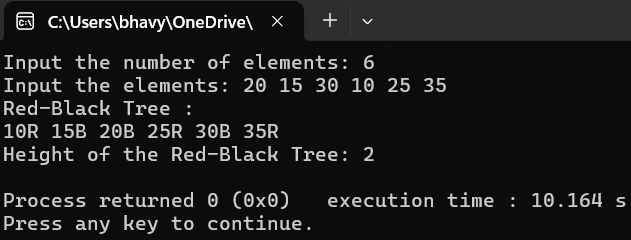
cout<<"Red-Black Tree :\n";

tree.display();

tree.displayHeight();

}

**Output :**

****

**Q3. Create an RB Tree by inserting the nodes in following sequence:**

**20,30,40,50,60,70,80,90,100,110,120,130.**

#include <iostream>

using namespace std;

enum Color { RED,BLACK };

struct Node

{

int data;

Node \*left,\*right,\*parent;

Color color;

Node(int data)

{

this->data=data;

left=right=parent=nullptr;

this->color=RED;

}

};

class RedBlackTree

{

Node \*root;

void rotateLeft(Node \*&node)

{

Node \*rightChild=node->right;

node->right=rightChild->left;

if(node->right != nullptr)

node->right->parent=node;

rightChild->parent=node->parent;

if(node->parent==nullptr)

root=rightChild;

else if(node==node->parent->left)

node->parent->left=rightChild;

else

node->parent->right=rightChild;

rightChild->left=node;

node->parent=rightChild;

}

void rotateRight(Node \*&node)

{

Node \*leftChild=node->left;

node->left=leftChild->right;

if(node->left != nullptr)

node->left->parent=node;

leftChild->parent=node->parent;

if(node->parent==nullptr)

root=leftChild;

else if(node==node->parent->left)

node->parent->left=leftChild;

else

node->parent->right=leftChild;

leftChild->right=node;

node->parent=leftChild;

}

void fixViolation(Node \*&node)

{

Node \*parent=nullptr;

Node \*grandparent=nullptr;

while(node != root&&node->color != BLACK&&node->parent->color==RED)

{

parent=node->parent;

grandparent=node->parent->parent;

if(parent==grandparent->left)

{

Node \*uncle=grandparent->right;

if(uncle != nullptr&&uncle->color==RED)

{

grandparent->color=RED;

parent->color=BLACK;

uncle->color=BLACK;

node=grandparent;

}

else

{

if(node==parent->right)

{

rotateLeft(parent);

node=parent;

parent=node->parent;

}

rotateRight(grandparent);

swap(parent->color,grandparent->color);

node=parent;

}

}

else

{

Node \*uncle=grandparent->left;

if(uncle != nullptr&&uncle->color==RED)

{

grandparent->color=RED;

parent->color=BLACK;

uncle->color=BLACK;

node=grandparent;

}

else

{

if(node==parent->left)

{

rotateRight(parent);

node=parent;

parent=node->parent;

}

rotateLeft(grandparent);

swap(parent->color,grandparent->color);

node=parent;

}

}

}

root->color=BLACK;

}

void inorderHelper(Node \*node)

{

if(node==nullptr)

return;

inorderHelper(node->left);

cout<<node->data <<(node->color==RED?"R " : "B ");

inorderHelper(node->right);

}

public:

RedBlackTree() { root=nullptr;}

void insert(const int &data)

{

Node \*node=new Node(data);

Node \*parent=nullptr;

Node \*current=root;

while(current != nullptr)

{

parent=current;

if(node->data<current->data)

current=current->left;

else

current=current->right;

}

node->parent=parent;

if(parent==nullptr)

root=node;

else if(node->data<parent->data)

parent->left=node;

else

parent->right=node;

if(node->parent==nullptr)

{

node->color=BLACK;

return;

}

if(node->parent->parent==nullptr)

return;

fixViolation(node);

}

void display()

{

inorderHelper(root);

cout<<endl;

}

};

int main()

{

RedBlackTree tree;

int n;

cout<<"Input the number of elements : ";

cin>>n;

int elements[n];

cout<<"Input the elements : ";

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

{

cin>>elements[i];

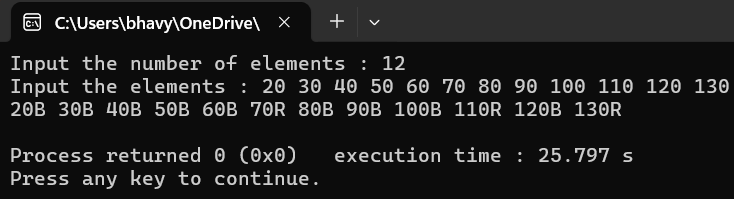
tree.insert(elements[i]);

}

tree.display();

}

**Output :**

****

**Q5. Implement a data structure that supports dynamic order statistics using a Red-**

**Black Tree. The data structure should support the following operations:**

**Insert(x): Insert an integer x into the data structure.**

**Select(k): Find the kth smallest element in the data structure.**

**Rank(x): Find the rank of integer x in the data structure(i.e.,the number**

**of elements less than or equal to x).**

**Your program should implement the Red-Black Tree as the underlying data structure to**

**achieve efficient operations. Make sure to maintain the Red-Black Tree properties after**

**insertions and deletions.**

#include <iostream>

using namespace std;

enum Color { RED,BLACK };

struct Node

{

int data;

Node \*left,\*right,\*parent;

Color color;

int size;

Node(int data)

{

this->data=data;

left=right=parent=nullptr;

this->color=RED;

this->size=1;

}

};

class RedBlackTree

{

Node \*root;

void rotateLeft(Node \*&node)

{

Node \*rightChild=node->right;

node->right=rightChild->left;

if(node->right != nullptr)

node->right->parent=node;

rightChild->parent=node->parent;

if(node->parent==nullptr)

root=rightChild;

else if(node==node->parent->left)

node->parent->left=rightChild;

else

node->parent->right=rightChild;

rightChild->left=node;

node->parent=rightChild;

rightChild->size=node->size;

node->size=1+(node->left?node->left->size : 0)+(node->right?node->right->size : 0);

}

void rotateRight(Node \*&node)

{

Node \*leftChild=node->left;

node->left=leftChild->right;

if(node->left != nullptr)

node->left->parent=node;

leftChild->parent=node->parent;

if(node->parent==nullptr)

root=leftChild;

else if(node==node->parent->left)

node->parent->left=leftChild;

else

node->parent->right=leftChild;

leftChild->right=node;

node->parent=leftChild;

leftChild->size=node->size;

node->size=1+(node->left?node->left->size : 0)+(node->right?node->right->size : 0);

}

void fixViolation(Node \*&node)

{

Node \*parent=nullptr;

Node \*grandparent=nullptr;

while(node != root&&node->color != BLACK&&node->parent->color==RED)

{

parent=node->parent;

grandparent=node->parent->parent;

if(parent==grandparent->left)

{

Node \*uncle=grandparent->right;

if(uncle != nullptr&&uncle->color==RED)

{

grandparent->color=RED;

parent->color=BLACK;

uncle->color=BLACK;

node=grandparent;

}

else

{

if(node==parent->right)

{

rotateLeft(parent);

node=parent;

parent=node->parent;

}

rotateRight(grandparent);

swap(parent->color,grandparent->color);

node=parent;

}

}

else

{

Node \*uncle=grandparent->left;

if(uncle != nullptr&&uncle->color==RED)

{

grandparent->color=RED;

parent->color=BLACK;

uncle->color=BLACK;

node=grandparent;

}

else

{

if(node==parent->left)

{

rotateRight(parent);

node=parent;

parent=node->parent;

}

rotateLeft(grandparent);

swap(parent->color,grandparent->color);

node=parent;

}

}

}

root->color=BLACK;

}

int size(Node\* node)

{

return node==nullptr?0 : node->size;

}

Node\* insertHelper(Node \*root,Node \*node)

{

if(root==nullptr)

return node;

if(node->data<root->data)

{

root->left=insertHelper(root->left,node);

root->left->parent=root;

}

else

{

root->right=insertHelper(root->right,node);

root->right->parent=root;

}

root->size=1+size(root->left)+size(root->right);

return root;

}

Node\* selectHelper(Node\* node,int k)

{

int leftSize=size(node->left);

if(k==leftSize+1)

return node;

else if(k<=leftSize)

return selectHelper(node->left,k);

else

return selectHelper(node->right,k-leftSize-1);

}

int rankHelper(Node\* node,int x)

{

if(node==nullptr)

if(x<node->data)

return rankHelper(node->left,x);

else if(x>node->data)

return size(node->left)+1+rankHelper(node->right,x);

else

return size(node->left)+1;

}

public:

RedBlackTree() { root=nullptr;}

void insert(const int &data)

{

Node \*node=new Node(data);

root=insertHelper(root,node);

fixViolation(node);

}

void inorderHelper(Node \*node)

{

if(node==nullptr)

return;

inorderHelper(node->left);

cout<<node->data <<(node->color==RED?"R " : "B ");

inorderHelper(node->right);

}

int select(int k)

{

Node\* result=selectHelper(root,k);

if(result != nullptr)

return result->data;

return -1;

}

int rank(int x)

{

return rankHelper(root,x);

}

void display()

{

inorderHelper(root);

cout<<endl;

}

};

int main()

{

RedBlackTree tree;

int n,k,x;

cout<<"Input the number of elements : ";

cin>>n;

int elements[n];

cout<<"Input the elements : ";

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

{

cin>>elements[i];

tree.insert(elements[i]);

}

cout<<"Red-Black Tree :\n";

tree.display();

cout<<"Input the value of k : ";

cin>>k;

cout<<"The kth smallest element is : "<<tree.select(k)<<endl;

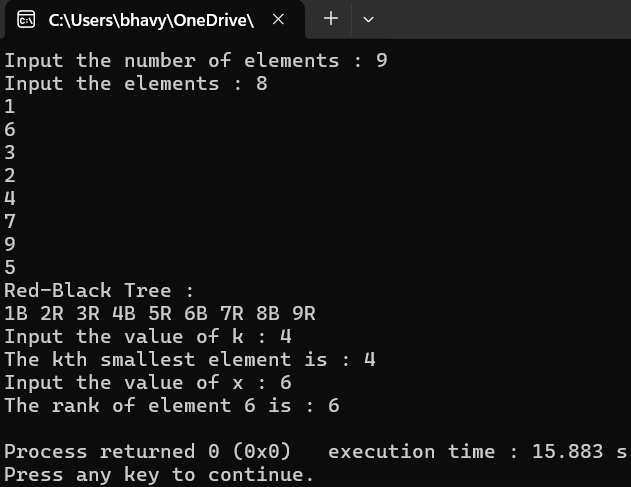
cout<<"Input the value of x : ";

cin>>x;

cout<<"The rank of element "<<x<<" is : "<<tree.rank(x)<<endl;

}

**Output :**

****