#pragma once

#ifndef RED\_BLACK\_TREE\_H

#define RED\_BLACK\_TREE\_H

using namespace std;

#include "coloredNode.h"

#include "tree.h"

#include <iostream>

#include <fstream>

class RedBlackTree : public tree {

private:

ColoredNode\* head;

public:

RedBlackTree() {

head = nullptr;

}

virtual void addNode(long long int iKey, int iRowNumber) {

if (head == nullptr) {

head = new ColoredNode(iKey, iRowNumber, 0, nullptr);

return;

}

ColoredNode\* node = BSTaddColoredNode(iKey, iRowNumber);

ColoredNode\* parent = node->parent;

while (node != head && parent->iColor == 1) {

ColoredNode\* grandparent = parent->parent;

if (grandparent->left == parent) {

ColoredNode\* uncle = grandparent->right;

if (uncle->iRowNumber != -1 && uncle->iColor == 1) {

parent->iColor = 0;

uncle->iColor = 0;

grandparent->iColor = 1;

node = grandparent;

parent = node->parent;

}

else {

if (parent->right == node) {

rotateLeft(parent);

swap(parent, node);

}

rotateRight(grandparent);

parent->iColor = 0;

grandparent->iColor = 1;

break;

}

}

else {

ColoredNode\* uncle = grandparent->left;

if (uncle->iRowNumber != -1 && uncle->iColor == 1) {

grandparent->iColor = 1;

parent->iColor = 0;

uncle->iColor = 0;

node = grandparent;

parent = node->parent;

}

else {

if (parent->left == node) {

rotateRight(parent);

swap(parent, node);

}

rotateLeft(grandparent);

parent->iColor = 0;

grandparent->iColor = 1;

break;

}

}

}

head->iColor = 0;

}

virtual void deleteNode(long long int iKey) {

ColoredNode\* node = BSTdeleteNode(iKey);

while (node!= nullptr && node != head && node->iColor == 0) {

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

ColoredNode\* sibling = node->parent->right;

if (sibling->iColor == 1) {

sibling->iColor = 0;

node->parent->iColor = 1;

rotateLeft(node->parent);

sibling = node->parent->right;

}

else

{

if (sibling->right->iRowNumber != -1 && sibling->right->iColor == 0) {

sibling->left->iColor = 0;

sibling->iColor = 1;

rotateRight(sibling);

sibling = node->parent->right;

}

sibling->iColor = node->parent->iColor;

node->parent->iColor = 0;

sibling->right->iColor = 0;

rotateLeft(node->parent);

node = head;

break;

}

}

else {

ColoredNode\* sibling = node->parent->left;

if (sibling->iColor == 1) {

sibling->iColor = 0;

node->parent->iColor = 1;

rotateLeft(node->parent);

sibling = node->parent->left;

}

if (sibling->left->iColor == 0 && sibling->right->iColor == 0) {

sibling->iColor = 1;

node = node->parent;

}

else

{

if (sibling->left->iColor == 0) {

sibling->right->iColor = 0;

sibling->iColor = 1;

rotateLeft(sibling);

sibling = node->parent->left;

}

sibling->iColor = node->parent->iColor;

node->parent->iColor = 0;

sibling->left->iColor = 0;

rotateRight(node->parent);

node = head;

break;

}

}

}

}

virtual int findNode(long long int iKey) {

ColoredNode\* root = head;

while (root->iRowNumber != -1 && iKey != root->iKey)

{

count++;

(iKey < root->iKey) ? root = root->left : root = root->right;

}

return root->iRowNumber;

}

virtual void print() {

printExecute(head, 0);

}

void printExecute(Node\* root, int level) {

ColoredNode\* rootColored = (ColoredNode\*)root;

if (level == 0) cout << "\n------TREE------\n";

if (rootColored->iRowNumber == -1) return;

printExecute(rootColored->right, level + 1);

for (int i = 0; i < level; i++) cout << "\t";

cout << " " << root->iKey << "(" << rootColored->iColor << ")" << "\n";

printExecute(rootColored->left, level + 1);

}

void rotateRight(ColoredNode\* node) {

ColoredNode\* tmp = node->left;

node->left = tmp->right;

if (tmp->right->iRowNumber == -1) tmp->right->parent = node;

tmp->parent = node->parent;

if (node->parent == nullptr) {

head = tmp;

}

else {

if (node == node->parent->right) node->parent->right = tmp;

else node->parent->left = tmp;

}

tmp->right = node;

node->parent = tmp;

}

void rotateLeft(ColoredNode\* node) {

ColoredNode\* tmp = node->right;

node->right = tmp->left;

if (tmp->left->iRowNumber == -1) tmp->left->parent = node;

tmp->parent = node->parent;

if (node->parent == nullptr) head = tmp;

else {

if (node == node->parent->left) node->parent->left = tmp;

else node->parent->right = tmp;

}

tmp->left = node;

node->parent = tmp;

}

ColoredNode\* BSTaddColoredNode(long long int iKey, int iRowNumber) {

ColoredNode\* root = head;

ColoredNode\* rootParent = head;

while (root->iRowNumber != -1) {

rootParent = root;

(iKey < root->iKey) ? root = root->left : root = root->right;

}

ColoredNode\* tmp = new ColoredNode(iKey, iRowNumber, 1, rootParent);

(rootParent->left->iRowNumber == -1 && rootParent->iKey >= iKey) ? rootParent->left = tmp : (rootParent->right->iRowNumber == -1 && rootParent->iKey < iKey) ? rootParent->right = tmp : 0;

return tmp;

}

ColoredNode\* BSTdeleteNode(long long int iKey) {

ColoredNode\* root = head;

ColoredNode\* tmp = new ColoredNode(0, -1, 0, nullptr);

while (root->iRowNumber != -1 && iKey != root->iKey) (iKey < root->iKey) ? root = root->left : root = root->right;

if (root->iRowNumber == -1) return nullptr;

//if (root == head) return nullptr;

if (root->left->iRowNumber == -1 && root->right->iRowNumber == -1){

(root->parent->left == root) ? root->parent->left->deleteNode() : root->parent->right->deleteNode();

return root->parent;

}

else if (root->left->iRowNumber != -1 && root->right->iRowNumber == -1) {

swap(root, root->left);

root->left->deleteNode();

return root;

}

else if (root->right->iRowNumber != -1 && root->left->iRowNumber == -1) {

swap(root, root->right);

root->right->deleteNode();

return root;

}

else {

ColoredNode\* tmpRight = root->right;

ColoredNode\* tmpParent = root;

while (tmpRight->iRowNumber != -1 && tmpRight->left->iRowNumber != -1)

{

tmpParent = tmpRight;

tmpRight = tmpRight->left;

}

root->ValueSwap(tmpRight);

(tmpParent->right == tmpRight) ? tmpParent->right->deleteNode() : tmpParent->left->deleteNode();

return root;

}

}

ColoredNode\* getHead() {

return head;

}

void generateTree(int size) {

srand(time(NULL));

for (int i = 0; i < size; i++) {

this->addNode(rand() % size + 1, i);

}

}

};

#endif // !RED\_BLACK\_TREE\_H