

EUROPEAN UNIVERSITY

OF LEFKE

***Name :*** *Honest**Chirozva*

***Department :*** *Software Engineering*

***Student Number :*** *154 425*

***Course :*** *Advanced Data Structures (COMP466)*

***Date :*** 29/04/19

*Binary Search Trees*

*Comparison of Morris Algorithm and DSW-Backbone Traversals*

*2018-2019 Spring Semester*

***Tree.hpp***

*#ifndef BINARY\_TREE\_H*

*#define BINARY\_TREE\_H*

*#include <algorithm>*

*#include <cstdlib>*

*#include <ctime>*

*#include <iostream>*

*#include <stack>*

*#include <vector>*

*struct node {*

*int info;*

*struct node \*left, \*right;*

*};*

*typedef struct node Node;*

*class BinaryTree {*

*// private data members*

*Node \*root\_;*

*std::vector<int> data\_;*

*// private methods*

*Node \*makeTree(const int); // make a node*

*void setLeft(Node \*, const int); // set item to the left of given node*

*void setRight(Node \*, const int); // set item to the right of given node*

*Node \*insert(Node \*, const int); // recursive method*

*void inorder(Node \*); // recursive inorder traversal helper*

*Node \*free(Node \*); // delete a node, recursively*

*Node \*getItemNode(const int); // get the node of an item*

*Node \*getPreviousNode(const int); // get the previous node of an item*

*int depth(Node \*); // depth or height of tree*

*void sortVectorItems(); // sorting items before balancing*

*bool removeByMerging(Node \*); // remove item by merging*

*Node \*getMinNode(Node \*); // minimum node of the tree*

*Node \*removeRecursive(Node \*, int); // remove item recursive way*

*void morrisInorder(Node \*); // morris inorder traversal*

*int balanceFactor(Node \*); // balance factor of a node*

*Node \*rightRotation(Node \*, Node \*, Node \*);*

*Node \*leftRotation(Node \*, Node \*, Node \*);*

*public:*

*BinaryTree();*

*void insertIterative(const int);*

*void insertRecursive(const int);*

*void traverseRecursive();*

*void traverseIterative();*

*void insertRandomIterative(const int);*

*void insertRandom(const int);*

*// default M - delete by Merging = M, R for recursive deletion*

*bool deleteItem(const int, const char = 'M');*

*bool deleteRandom(const int, const char = 'M');*

*void vectorInitialisation(std::vector<int>, int, int);*

*void balanceTree();*

*int height();*

*void morrisInorderTraversal();*

*void createBackbone();*

*~BinaryTree();*

*};*

*// implementation*

*// PRIVATE METHODS*

*Node \*BinaryTree::makeTree(const int x) {*

*Node \*temp = new Node;*

*temp->info = x;*

*temp->left = temp->right = nullptr;*

*return temp;*

*};*

*void BinaryTree::setLeft(Node \*p, const int x) {*

*if (p == nullptr)*

*std::cout << ".....EMPTY NODE.....\n";*

*else if (p->left != nullptr)*

*std::cout << ".....NON-EMPTY LEFT NODE.....\n";*

*else*

*p->left = makeTree(x);*

*}*

*void BinaryTree::setRight(Node \*p, const int x) {*

*if (p == nullptr)*

*std::cout << ".....EMPTY NODE.....\n";*

*else if (p->right != nullptr)*

*std::cout << ".....NON-EMPTY RIGHT NODE.....\n";*

*else*

*p->right = makeTree(x);*

*}*

*Node \*BinaryTree::insert(Node \*node, const int x) {*

*if (node == nullptr)*

*return makeTree(x);*

*else if (x < node->info)*

*node->left = insert(node->left, x);*

*else if (x > node->info)*

*node->right = insert(node->right, x);*

*else {*

*std::cout << x << " : is a duplicate therefore deprecated!\n";*

*data\_.pop\_back(); // pop item from vector if it's duplicate*

*}*

*return node;*

*}*

*Node \*BinaryTree::free(Node \*p) {*

*if (p == nullptr) return nullptr;*

*{*

*free(p->left);*

*free(p->right);*

*delete p;*

*}*

*return nullptr;*

*}*

*void BinaryTree::morrisInorder(Node \*root) {*

*Node \*p = root, \*tmp;*

*while (p != nullptr) {*

*if (p->left == nullptr) {*

*// std::cout << p->info << " ";*

*p = p->right;*

*} else {*

*tmp = p->left;*

*while (tmp->right != nullptr && tmp->right != p) tmp = tmp->right;*

*if (tmp->right == nullptr) {*

*tmp->right = p;*

*p = p->left;*

*} else {*

*// std::cout << p->info << " ";*

*tmp->right = nullptr;*

*p = p->right;*

*}*

*}*

*}*

*std::cout << std::endl;*

*}*

*int BinaryTree::balanceFactor(Node \*p) {*

*return depth(p->left) - depth(p->right);*

*}*

*Node \*BinaryTree::rightRotation(Node \*grand\_parent, Node \*parent, Node \*child) {*

*if (parent != root\_ && grand\_parent != nullptr)*

*grand\_parent->right = child;*

*else*

*root\_ = child;*

*parent->left = child->right;*

*child->right = parent;*

*return grand\_parent;*

*}*

*Node \*BinaryTree::leftRotation(Node \*grand\_parent, Node \*parent, Node \*child) {*

*if (parent != root\_ && grand\_parent != nullptr)*

*grand\_parent->right = child;*

*else*

*root\_ = child;*

*parent->right = child->left;*

*child->left = parent;*

*return grand\_parent;*

*}*

*// PUBLIC METHODS*

*BinaryTree::BinaryTree() {*

*root\_ = nullptr;*

*srand(time(NULL));*

*};*

*void BinaryTree::insertRecursive(const int x) {*

*// store the items in vector for sorting, balancing and deletion purposes*

*data\_.push\_back(x);*

*root\_ = insert(root\_, x);*

*}*

*void BinaryTree::insertRandom(const int size) {*

*for (int i = 0; i < size; ++i) insertRecursive(rand());*

*}*

*void BinaryTree::morrisInorderTraversal() { morrisInorder(root\_); }*

*void BinaryTree::createBackbone() {*

*Node \*grand\_parent = nullptr, \*parent = root\_, \*child;*

*while (parent != nullptr) {*

*child = parent->left;*

*if (child != nullptr) {*

*grand\_parent = rightRotation(grand\_parent, parent, child);*

*parent = child;*

*} else {*

*grand\_parent = parent;*

*parent = parent->right;*

*}*

*}*

*// traversing*

*Node \*current = root\_;*

*while (current != nullptr) {*

*// std::cout << current->info << " ";*

*current = current->right;*

*}*

*std::cout << std::endl;*

*}*

*BinaryTree::~BinaryTree() { root\_ = free(root\_); };*

*#endif*

***Main.cpp***

*#include <cstdlib>*

*#include <ctime>*

*#include <iomanip>*

*#include <iostream>*

*#include "Tree.hpp"*

*int main(void) {*

*clock\_t start, end;*

*BinaryTree tree\_1000, tree\_10000, tree\_100000, tree\_500000;*

*tree\_1000.insertRandom(1000);*

*tree\_10000.insertRandom(10000);*

*tree\_100000.insertRandom(100000);*

*tree\_500000.insertRandom(500000);*

*std::cout << std::setprecision(9) << std::fixed;*

*// COMPARISON FOR 1,000 NODES*

*start = clock();*

*tree\_1000.morrisInorderTraversal();*

*end = clock();*

*std::cout << "-------------------1 000 Nodes--------------------"<<std::endl;*

*std::cout << "Morris Alg Traversal - 1000 : "*

*<< static\_cast<double>(end - start) / CLOCKS\_PER\_SEC << std::endl;*

*start = clock();*

*tree\_1000.createBackbone();*

*end = clock();*

*std::cout << "DSW-Backbone Traversal - 1000 : "*

*<< static\_cast<double>(end - start) / CLOCKS\_PER\_SEC << std::endl;*

*// COMPARISON FOR 10,000 NODES*

*start = clock();*

*tree\_10000.morrisInorderTraversal();*

*end = clock();*

*std::cout << "-------------------10 000 Nodes-------------------"<<std::endl;*

*std::cout << "Morris Alg Traversal - 10000 : "*

*<< static\_cast<double>(end - start) / CLOCKS\_PER\_SEC << std::endl;*

*start = clock();*

*tree\_10000.createBackbone();*

*end = clock();*

*std::cout << "DSW-Backbone Traversal - 10000 : "*

*<< static\_cast<double>(end - start) / CLOCKS\_PER\_SEC << std::endl;*

*// COMPARISON FOR 100,000 NODES*

*start = clock();*

*tree\_100000.morrisInorderTraversal();*

*end = clock();*

*std::cout << "------------------100 000 Nodes-------------------"<<std::endl;*

*std::cout << "Morris Alg Traversal - 100000 : "*

*<< static\_cast<double>(end - start) / CLOCKS\_PER\_SEC << std::endl;*

*start = clock();*

*tree\_100000.createBackbone();*

*end = clock();*

*std::cout << "DSW-Backbone Traversal - 100000 : "*

*<< static\_cast<double>(end - start) / CLOCKS\_PER\_SEC << std::endl;*

*// COMPARISON FOR 500,000 NODES*

*start = clock();*

*tree\_500000.morrisInorderTraversal();*

*end = clock();*

*std::cout << "------------------500 000 Nodes-------------------"<<std::endl;*

*std::cout << "Morris Alg Traversal - 500000 : "*

*<< static\_cast<double>(end - start) / CLOCKS\_PER\_SEC << std::endl;*

*start = clock();*

*tree\_500000.createBackbone();*

*end = clock();*

*std::cout << "DSW-Backbone Traversal - 500000 : "*

*<< static\_cast<double>(end - start) / CLOCKS\_PER\_SEC << std::endl;*

*return 0;*

*}*

***OUTPUT***

*-------------------1 000 Nodes--------------------*

*Morris Alg Traversal - 1000 : 0.003666000*

*DSW-Backbone Traversal - 1000 : 0.002505000*

*-------------------10 000 Nodes-------------------*

*Morris Alg Traversal - 10000 : 0.004414000*

*DSW-Backbone Traversal - 10000 : 0.003795000*

*------------------100 000 Nodes-------------------*

*Morris Alg Traversal - 100000 : 0.025775000*

*DSW-Backbone Traversal - 100000 : 0.042998000*

*------------------500 000 Nodes-------------------*

*Morris Alg Traversal - 500000 : 0.111209000*

*DSW-Backbone Traversal - 500000 : 0.199364000*