Christiaan Cronje

COSC3319.01

8 A.M.

March 4, 2017

Lab 5 Grading Option “B”

**Program.cs**

using System;

namespace ThreadedTree

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Starting...");

BinarySearchNode root = CreateThreadedTree();

Console.WriteLine("Insert Initial Nodes");

//1) insert into tree

InsertBinarySearchTree(ref root, "Ajose", "295-1492");

InsertBinarySearchTree(ref root, "Munoz", "291-1864");

InsertBinarySearchTree(ref root, "Kong", "295-1601");

InsertBinarySearchTree(ref root, "Saleem", "293-6122");

InsertBinarySearchTree(ref root, "Seddon", "295-1882");

InsertBinarySearchTree(ref root, "Najar", "291-7890");

InsertBinarySearchTree(ref root, "Voorhees", "294-8075");

InsertBinarySearchTree(ref root, "Sparks", "584-3622");

Console.WriteLine("\nSearch for Saleem");

//2) search using iteration

BinarySearchNode cust;

cust = FindCustomerIterative(root, "Saleem");

//3) search using recursion

cust = FindCustomerRecursive(root, "Saleem");

//if (cust == null) Console.WriteLine("\nKey not found (Recursive)");

Console.WriteLine("\nSearch for Acevedo");

//4) search using iteration again

cust = FindCustomerIterative(root, "Acevedo");

//5) search using recursion

cust = FindCustomerRecursive(root, "Acevedo");

//if (cust == null) Console.WriteLine("\nKey not found (Recursive)");

Console.WriteLine("\nTraverse Inorder starting at Najar");

//6) traverse inorder starting at Najar, print each encounter

cust = FindCustomerIterative(root, "Najar", false);

// traverse inorder starting at cust

TraverseInOrder(cust);

Console.WriteLine("\nInsert 3 more");

//7) insert 3 more entries

InsertBinarySearchTree(ref root, "Devin", "294-1568");

InsertBinarySearchTree(ref root, "Morah", "294-1882");

InsertBinarySearchTree(ref root, "Zembo", "295-6622");

Console.WriteLine("\nTraverse Inorder Iterative");

//8) traverse inorder starting at root, print each encounter

TraverseInOrder(root);

Console.WriteLine("\nTry delete 3 entries");

//7) delete 3 entries with binary search

// FIXME: needs to actually delete

DeleteRandomNode(root, "Ajose");

DeleteRandomNode(root, "Najar");

DeleteRandomNode(root, "Aguirra");

Console.WriteLine("\nInsert 2 nodes");

//8) insert 2 entries

InsertBinarySearchTree(ref root, "Novak", "294-1666");

InsertBinarySearchTree(ref root, "Gonzales", "295-1882");

Console.WriteLine("\nTraverse Inorder Iterative from root");

//9) traverse inorder starting at root, using inordersuccessor

TraverseInOrder(root);

Console.WriteLine("\nTraverse Reverse Inorder Recursive");

//10) traverse using reverse inorder from root

// FIXME: fix recursion

ReverseInOrder(root);

Console.WriteLine("\nTraverse Preorder Iterative");

//11) traverse preorder using iteration and threads

PreOrderIterative(root);

Console.WriteLine();

Console.Write("Press any key to continue...");

Console.Read();

}

static BinarySearchNode CreateThreadedTree()

{

BinarySearchNode root = new BinarySearchNode();

root.Llink = root;

root.Rlink = root;

root.Ltag = false;

root.Rtag = true;

root.info = new Customer("Head", "0");

return root;

}

static void InsertBinarySearchTree(ref BinarySearchNode root, string name, string phone)

{

if (root == null)

{

BinarySearchNode node = new BinarySearchNode(new Customer(name, phone));

root = node;

}

else

{

BinarySearchNode p = root;

while (true)

{

if (p.info.Compare(name) > 0) // search left

{

if (p.Ltag != false)

{

p = p.Llink;

}

else

{

BinarySearchNode q = new BinarySearchNode(new Customer(name, phone));

// insert into left subtree

q.Llink = p.Llink;

q.Ltag = p.Ltag;

p.Llink = q;

p.Ltag = true;

q.Rlink = p;

q.Rtag = false;

if (q.Ltag == true)

{

GetInOrderPredecessor(q).Rlink = q;

}

Console.WriteLine("Inserted " + q.info.ToString());

break;

}

}

else

{

if (p.Rtag != false)

{

p = p.Rlink;

}

else

{

BinarySearchNode q = new BinarySearchNode(new Customer(name, phone));

// insert into right subtree

q.Rlink = p.Rlink;

q.Rtag = p.Rtag;

p.Rlink = q;

p.Rtag = true;

q.Llink = p;

q.Ltag = false;

if (q.Rtag == true)

{

GetInOrderSuccessor(q).Llink = q;

}

Console.WriteLine("Inserted " + q.info.ToString());

break;

}

}

}

}

}

static BinarySearchNode FindCustomerIterative(BinarySearchNode root, string name, bool verbose = true)

{

BinarySearchNode p = root;

while (true)

{

if (p.info.Compare(name) > 0) // search left

{

if (p.Ltag != false)

{

p = p.Llink;

}

else break;

}

else if (p.info.Compare(name) < 0)

{

if (p.Rtag != false)

{

p = p.Rlink;

}

else break;

}

else

{

// node has been found

// return node

if (verbose) Console.WriteLine("Found " + p.info.ToString() + " (Iterative)");

return p;

}

}

if (verbose) Console.WriteLine(name + " not found (Iterative)");

return null;

}

static BinarySearchNode FindCustomerRecursive(BinarySearchNode root, string name, bool verbose = true) // recursive

{

if (root == null) return null;

if (root.info.Compare(name) > 0) // search left subtree

{

if (root.Ltag == true)

{

FindCustomerRecursive(root.Llink, name);

return null;

}

}

else if(root.info.Compare(name) < 0) // search left subtree

{

if (root.Rtag == true)

{

FindCustomerRecursive(root.Rlink, name);

return null;

}

}

else

{

if (verbose) Console.WriteLine("Found " + root.info.ToString() + " (Recursive)");

return root;

}

if (verbose) Console.WriteLine(name + " not found (Recursive)");

return null;

}

static void TraverseInOrder(BinarySearchNode q)

{

if (q == null) return;

BinarySearchNode curr = q;

do

{

Console.WriteLine(curr.info.ToString());

curr = GetInOrderSuccessor(curr);

}

while (curr != q);

}

static BinarySearchNode GetInOrderSuccessor(BinarySearchNode p)

{

BinarySearchNode q = p.Rlink;

if (p.Rtag == false)

{

return q;

}

else

{

while (q.Ltag == true)

{

q = q.Llink;

}

}

return q;

}

static BinarySearchNode GetInOrderPredecessor(BinarySearchNode p)

{

BinarySearchNode q = p.Llink;

if (p.Ltag != false)

{

while (q.Rtag == true)

{

q = q.Rlink;

}

}

return q;

}

static void PreOrderIterative(BinarySearchNode q)

{

if (q == null) return;

BinarySearchNode curr = q;

do

{

Console.WriteLine(curr.info.ToString());

curr = GetPreOrderSuccessor(curr);

}

while (curr != q);

}

static BinarySearchNode GetPreOrderSuccessor(BinarySearchNode p)

{

BinarySearchNode q;// = p;

if (p.Ltag == true)

{

q = p.Llink;

}

else

{

q = p;

while (q.Rtag != true)

{

q = q.Rlink;

}

q = q.Rlink;

}

return q;

}

static void ReverseInOrder(BinarySearchNode q) // recursive

{

if (q != null)

{

if (q.Rtag == true && q != q.Rlink)

{

ReverseInOrder(q.Rlink);

}

Console.WriteLine(q.info.ToString());

if (q.Ltag == true && q != q.Llink)

{

ReverseInOrder(q.Llink);

}

}

}

static void DeleteRandomNode(BinarySearchNode root, string name)

{

BinarySearchNode q = FindCustomerIterative(root, name, false);

if (q == null)

{

Console.WriteLine("Node not in tree");

return;

}

BinarySearchNode t = q;

if (t.Rtag == false)

{

q = t.Llink;

// t => avail

}

else

{

if (t.Ltag == false)

{

q = t.Rlink;

// t => avail

}

BinarySearchNode r = t.Rlink;

if (r.Ltag == false)

{

r.Llink = t.Llink;

q = r;

// t => avail

}

else

{

BinarySearchNode s = r.Llink;

while (s.Ltag == true)

{

r = s;

s = r.Llink;

}

s.Llink = t.Llink;

s.Ltag = t.Ltag;

r.Llink = s.Rlink;

r.Ltag = s.Rtag;

s.Rlink = t.Rlink;

s.Rtag = t.Rtag;

q = s;

// t => avail;

}

}

Console.WriteLine("Deleted: " + name);

}

}

public class Customer

{

public string customerName;

public string customerNumber;

public override string ToString()

{

return customerName + ": " + customerNumber;

}

public Customer()

{

customerName = "";

customerNumber = "";

}

public Customer(string name, string number)

{

customerName = name;

customerNumber = number;

}

public int Compare(string name)

{

if (name.Equals("Head"))

{

return -1;

}

else if (customerName.Equals("Head"))

{

return 1;

}

else

{

return string.Compare(customerName, name);

}

}

}

public class BinarySearchNode

{

public BinarySearchNode Llink;

public BinarySearchNode Rlink;

public bool Ltag;

public bool Rtag;

public Customer info;

public BinarySearchNode()

{

Llink = null;

Rlink = null;

Ltag = false;

Rtag = false;

info = new Customer();

}

public BinarySearchNode(Customer custInfo)

{

Llink = null;

Rlink = null;

Ltag = false;

Rtag = false;

info = custInfo;

}

}

}

Starting...

Insert Initial Nodes

Inserted Ajose: 295-1492

Inserted Munoz: 291-1864

Inserted Kong: 295-1601

Inserted Saleem: 293-6122

Inserted Seddon: 295-1882

Inserted Najar: 291-7890

Inserted Voorhees: 294-8075

Inserted Sparks: 584-3622

Search for Saleem

Found Saleem: 293-6122 (Iterative)

Found Saleem: 293-6122 (Recursive)

Search for Acevedo

Acevedo not found (Iterative)

Acevedo not found (Recursive)

Traverse Inorder starting at Najar

Najar: 291-7890

Saleem: 293-6122

Seddon: 295-1882

Sparks: 584-3622

Voorhees: 294-8075

Head: 0

Ajose: 295-1492

Kong: 295-1601

Munoz: 291-1864

Insert 3 more

Inserted Devin: 294-1568

Inserted Morah: 294-1882

Inserted Zembo: 295-6622

Traverse Inorder Iterative

Head: 0

Ajose: 295-1492

Devin: 294-1568

Kong: 295-1601

Morah: 294-1882

Munoz: 291-1864

Najar: 291-7890

Saleem: 293-6122

Seddon: 295-1882

Sparks: 584-3622

Voorhees: 294-8075

Zembo: 295-6622

Try delete 3 entries

Deleted: Ajose

Deleted: Najar

Node not in tree: Aguirra

Insert 2 nodes

Inserted Novak: 294-1666

Inserted Gonzales: 295-1882

Traverse Inorder Iterative from root

Head: 0

Ajose: 295-1492

Gonzales: 295-1882

Kong: 295-1601

Morah: 294-1882

Munoz: 291-1864

Najar: 291-7890

Novak: 294-1666

Saleem: 293-6122

Seddon: 295-1882

Sparks: 584-3622

Voorhees: 294-8075

Zembo: 295-6622

Traverse Reverse Inorder Recursive

Head: 0

Zembo: 295-6622

Voorhees: 294-8075

Sparks: 584-3622

Seddon: 295-1882

Saleem: 293-6122

Novak: 294-1666

Najar: 291-7890

Munoz: 291-1864

Morah: 294-1882

Kong: 295-1601

Gonzales: 295-1882

Ajose: 295-1492

Traverse Preorder Iterative

Head: 0

Ajose: 295-1492

Munoz: 291-1864

Kong: 295-1601

Gonzales: 295-1882

Morah: 294-1882

Saleem: 293-6122

Najar: 291-7890

Novak: 294-1666

Seddon: 295-1882

Voorhees: 294-8075

Sparks: 584-3622

Zembo: 295-6622

Press any key to continue...