哈尔滨工业大学计算机科学与技术学院

实验报告

|  |  |
| --- | --- |
| 课程名称： | 数据结构与算法 |
| 课程类型： | 必修 |
| 实验名称： | 查找结构的实验比较 |
|  |  |
| 班级： | 1703005 |
| 学号： | 1170300520 |
| 姓名： | 郭子阳 |

1. 实验目的

熟悉查找结构的操作（建立、插入、删除、搜索），熟悉折半查找。

1. 实验要求及实验环境

实验要求：编写程序实现BST存储结构的建立（插入）、删除、查找和排序算法，实现折半查找算法，比较BST查找结构与折半查找的时间性能

实验环境：macOS，VSCode，JDK 11

1. 设计思想（本程序中用到的所有数据类型的定义，核心算法的流程图等）

BST查找结构的节点定义：

class NodeType {

private int record;

private NodeType parentNode;

private NodeType leftChild;

private NodeType rightChild;

public int getRecord() {

return record;

}

public NodeType getParentNode() {

return parentNode;

}

public void setParentNode(NodeType *parentNode*) {

this.parentNode = parentNode;

}

public NodeType getRightChild() {

return rightChild;

}

public void setRightChild(NodeType *rightChild*) {

this.rightChild = rightChild;

}

public NodeType getLeftChild() {

return leftChild;

}

public void setLeftChild(NodeType *leftChild*) {

this.leftChild = leftChild;

}

public void setRecord(int *record*) {

this.record = record;

}

}

折半查找判定树的节点定义：

class HalfNode {

private int value;

private HalfNode leftNode;

private HalfNode rightNode;

public HalfNode(int *value*) {

this.value = value;

}

public int getValue() {

return value;

}

public HalfNode getLeftNode() {

return leftNode;

}

public void setLeftNode(HalfNode *leftNode*) {

this.leftNode = leftNode;

}

public HalfNode getRightNode() {

return rightNode;

}

public void setRightNode(HalfNode *rightNode*) {

this.rightNode = rightNode;

}

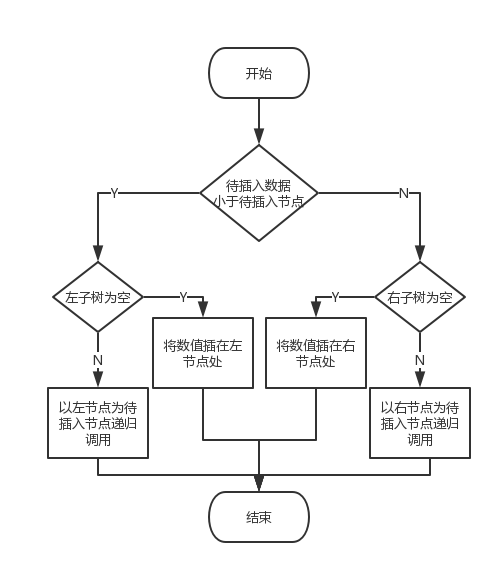
public void setValue(int *value*) {

this.value = value;

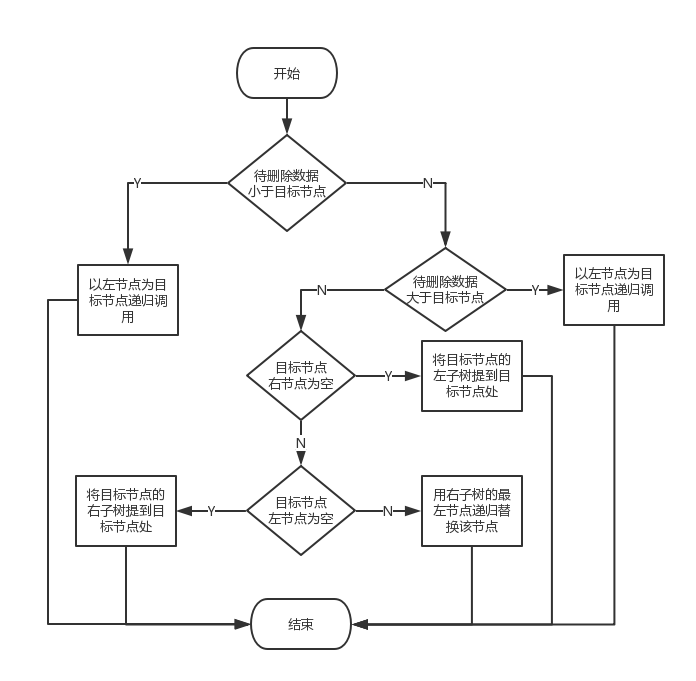
}

}

BST插入节点：



BST删除节点：



1. 测试结果





1. 系统不足与经验体会

深刻理解了二叉搜索树的各种操作，了解了平均查找长度的计算方法。

1. 源代码

import java.util.List;

import java.io.BufferedReader;

import java.util.ArrayList;

import java.util.Scanner;

import java.io.FileReader;

import java.io.IOException;

public class Main {

private static Scanner scanner = new Scanner(System.in);

private static NodeType orderBST;

private static NodeType randomBST;

private static int[] halfSerachArray = new int[1024];

private static ArrayList<Integer> allElements;

private static int orderSuccess;

private static int randomSuccess;

private static int halfSuccess;

private static int halfFailure;

private static int orderFailure;

private static int randomFailure;

private static int tempTimes;

private static int nullNode;

private static HalfNode halfTree;

public static void main(String[] args) {

buildOrderBST();

buildRandomBST();

while(true) {

menu();

}

}

private static void menu() {

System.out.println();

System.out.println();

System.out.println("1. 已排序序列BST树");

System.out.println("2. 随机序列BST树");

System.out.println("3. 折半查找");

System.out.println("4. BST平均查找长度");

System.out.println("5. 折半查找的平均查找长度");

System.out.println("6. 退出");

System.out.print("请输入选择：");

int choice = scanner.nextInt();

System.out.println();

System.out.println();

switch(choice) {

case 1:

orderBST();

break;

case 2:

randomBST();

break;

case 3:

halfSearch();

break;

case 4:

BSTAverage();

break;

case 5:

halfAverage();

break;

case 6:

System.exit(0);

break;

default:

System.out.println("选择有误！");

break;

}

}

private static void orderBST() {

System.out.println("1. 插入数据");

System.out.println("2. 删除数据");

System.out.println("3. 查找数据");

System.out.println("4. 排序数据");

System.out.print("请输入选择：");

int choice = scanner.nextInt();

switch(choice) {

case 1:

System.out.print("请输入要插入的数据：");

int value1 = scanner.nextInt();

insertBST(orderBST, null, value1);

break;

case 2:

System.out.println("请输入要删除的数据：");

int value2 = scanner.nextInt();

deleteBST(orderBST, value2);

break;

case 3:

System.out.println("请输入要查找的数据：");

int value3 = scanner.nextInt();

if(searchBST(orderBST, value3) == -1){

System.out.println("无此数据。");

} else {

System.out.println("已查找到此数据！");

}

break;

case 4:

allElements = new ArrayList<>();

sortBST(orderBST);

System.out.println(allElements.toString());

break;

default:

System.out.println("选择有误！");

break;

}

}

private static void randomBST() {

System.out.println("1. 插入数据");

System.out.println("2. 删除数据");

System.out.println("3. 查找数据");

System.out.println("4. 排序数据");

System.out.print("请输入选择：");

int choice = scanner.nextInt();

switch(choice) {

case 1:

System.out.print("请输入要插入的数据：");

int value1 = scanner.nextInt();

insertBST(randomBST, null, value1);

break;

case 2:

System.out.println("请输入要删除的数据：");

int value2 = scanner.nextInt();

deleteBST(randomBST, value2);

break;

case 3:

System.out.println("请输入要查找的数据：");

int value3 = scanner.nextInt();

if(searchBST(randomBST, value3) == -1){

System.out.println("无此数据。");

} else {

System.out.println("已查找到此数据！");

}

break;

case 4:

allElements = new ArrayList<>();

sortBST(randomBST);

System.out.println(allElements.toString());

break;

default:

System.out.println("选择有误！");

break;

}

}

private static void buildOrderBST() {

orderBST = null;

BufferedReader reader = null;

try {

reader = new BufferedReader(new FileReader("order.txt"));

String line = reader.readLine();

orderBST = new NodeType();

orderBST.setRecord(Integer.parseInt(line));

while((line = reader.readLine()) != null) {

insertBST(orderBST, null, Integer.parseInt(line));

}

System.out.println("有序序列的查找树建立完成。");

} catch(IOException e) {

System.out.println("读取数据错误！");

e.printStackTrace();

} finally {

if(reader != null) {

try {

reader.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

private static void buildRandomBST() {

randomBST = null;

BufferedReader reader = null;

try {

reader = new BufferedReader(new FileReader("random.txt"));

String line = reader.readLine();

randomBST = new NodeType();

randomBST.setRecord(Integer.parseInt(line));

while((line = reader.readLine()) != null) {

insertBST(randomBST, null, Integer.parseInt(line));

}

System.out.println("随机序列的查找树建立完成。");

} catch(IOException e) {

System.out.println("读取数据错误！");

e.printStackTrace();

} finally {

if(reader != null) {

try {

reader.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

private static void insertBST(NodeType node, NodeType parent, int value) {

if(value < node.getRecord()) {

if(node.getLeftChild() == null) {

node.setLeftChild(new NodeType());

NodeType tempNode = node.getLeftChild();

tempNode.setRecord(value);

tempNode.setParentNode(node);

} else {

insertBST(node.getLeftChild(), node, value);

}

} else {

if(node.getRightChild() == null) {

node.setRightChild(new NodeType());

NodeType tempNode = node.getRightChild();

tempNode.setRecord(value);

tempNode.setParentNode(node);

} else{

insertBST(node.getRightChild(), node, value);

}

}

}

private static void deleteBST(NodeType node, int value) {

if(searchBST(node, value) == -1) {

System.out.println("无此数据！");

return;

}

if(node != null) {

if(value < node.getRecord()) {

deleteBST(node.getLeftChild(), value);

} else if(value > node.getRecord()) {

deleteBST(node.getRightChild(), value);

} else {

if(node.getRightChild() == null) {

NodeType tempParent = node.getParentNode();

if(tempParent.getLeftChild() == null || tempParent.getLeftChild().getRecord() != value) {

tempParent.setRightChild(node.getLeftChild());

} else {

tempParent.setLeftChild(node.getLeftChild());

}

} else if(node.getLeftChild() == null) {

NodeType tempParent = node.getParentNode();

if(tempParent.getLeftChild() == null || tempParent.getLeftChild().getRecord() != value) {

tempParent.setRightChild(node.getRightChild());

} else {

tempParent.setLeftChild(node.getRightChild());

}

} else {

node.setRecord(deletemin(node.getRightChild()));

}

}

}

}

private static int deletemin(NodeType node) {

int tmp;

NodeType tempNode;

if(node.getLeftChild() == null) {

tempNode = node;

tmp = node.getRecord();

node = node.getRightChild();

NodeType tempParent = tempNode.getParentNode();

if(tempNode == tempParent.getLeftChild()) {

tempParent.setLeftChild(null);

} else {

tempParent.setRightChild(null);

}

return tmp;

} else {

return deletemin(node.getLeftChild());

}

}

private static int searchBST(NodeType node, int value) {

if(node == null) {

return -1;

} else if(value == node.getRecord()) {

return value;

}

if(value < node.getRecord()) {

return searchBST(node.getLeftChild(), value);

} else {

return searchBST(node.getRightChild(), value);

}

}

private static void sortBST(NodeType node) {

if(node != null) {

sortBST(node.getLeftChild());

allElements.add(node.getRecord());

sortBST(node.getRightChild());

}

}

private static void halfSearch() {

buildHalfSearchArray();

System.out.print("请输入你要查找的数值：");

int key = scanner.nextInt();

boolean result = binarySearch(0, 1023, key);

if(result) {

System.out.println("已查找到该数值！");

} else {

System.out.println("未查找到该数值！");

}

}

private static boolean binarySearch(int start, int end, int key) {

int mid;

if(start > end) {

return false;

} else {

mid = (start + end) / 2;

if(key < halfSerachArray[mid]) {

return binarySearch(start, mid - 1, key);

} else if(key > halfSerachArray[mid]) {

return binarySearch(mid + 1, end, key);

} else

return true;

}

}

private static void buildHalfSearchArray() {

BufferedReader reader = null;

int i = 0;

try {

reader = new BufferedReader(new FileReader("order.txt"));

String line = null;

while((line = reader.readLine()) != null) {

halfSerachArray[i] = Integer.parseInt(line);

i ++;

}

} catch(IOException e) {

System.out.println("读取数据错误！");

e.printStackTrace();

} finally {

if(reader != null) {

try {

reader.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

private static void BSTAverage() {

orderSuccess = 0;

allElements = new ArrayList<>();

sortBST(orderBST);

for(Integer i : allElements) {

searchOrderSuccess(orderBST, i);

}

System.out.println("有序BST查找成功的平均查找长度为" + (orderSuccess\*1.0)/allElements.size());

randomSuccess = 0;

allElements = new ArrayList<>();

sortBST(randomBST);

for(Integer i : allElements) {

searchRandomSuccess(randomBST, i);

}

System.out.println("随机BST查找成功的平均查找长度为" + (randomSuccess\*1.0)/allElements.size());

orderFailure = 0;

nullNode = 0;

allElements = new ArrayList<>();

sortBST(orderBST);

for(Integer i : allElements) {

tempTimes = 0;

searchOrderFailure(orderBST, i);

}

System.out.println("有序BST查找失败的平均查找长度为" + (orderFailure\*1.0)/nullNode);

randomFailure = 0;

nullNode = 0;

allElements = new ArrayList<>();

sortBST(randomBST);

for(Integer i : allElements) {

tempTimes = 0;

searchRandomFailure(randomBST, i);

}

System.out.println("随机BST查找失败的平均查找长度为" + (randomFailure\*1.0)/nullNode);

}

private static int searchRandomFailure(NodeType node, int value) {

if(node == null) {

return -1;

} else if(value == node.getRecord()) {

if(node.getLeftChild() == null && node.getRightChild() == null) {

tempTimes ++;

nullNode += 2;

randomFailure = randomFailure + 2 \* tempTimes;

} else {

tempTimes ++;

nullNode += 1;

randomFailure = randomFailure + tempTimes;

}

return value;

}

if(value < node.getRecord()) {

tempTimes ++;

return searchRandomFailure(node.getLeftChild(), value);

} else {

tempTimes ++;

return searchRandomFailure(node.getRightChild(), value);

}

}

private static int searchOrderFailure(NodeType node, int value) {

if(node == null) {

return -1;

} else if(value == node.getRecord()) {

if(node.getLeftChild() == null && node.getRightChild() == null) {

tempTimes ++;

nullNode += 2;

orderFailure = orderFailure + 2 \* tempTimes;

} else {

tempTimes ++;

nullNode += 1;

orderFailure = orderFailure + tempTimes;

}

return value;

}

if(value < node.getRecord()) {

tempTimes ++;

return searchOrderFailure(node.getLeftChild(), value);

} else {

tempTimes ++;

return searchOrderFailure(node.getRightChild(), value);

}

}

private static int searchRandomSuccess(NodeType node, int value) {

if(node == null) {

return -1;

} else if(value == node.getRecord()) {

return value;

}

if(value < node.getRecord()) {

randomSuccess ++;

return searchRandomSuccess(node.getLeftChild(), value);

} else {

randomSuccess ++;

return searchRandomSuccess(node.getRightChild(), value);

}

}

private static int searchOrderSuccess(NodeType node, int value) {

if(node == null) {

return -1;

} else if(value == node.getRecord()) {

return value;

}

if(value < node.getRecord()) {

orderSuccess ++;

return searchOrderSuccess(node.getLeftChild(), value);

} else {

orderSuccess ++;

return searchOrderSuccess(node.getRightChild(), value);

}

}

private static void halfAverage() {

allElements = new ArrayList<>();

for(int i = 0; i < 2048; i ++) {

if(i % 2 == 1) {

allElements.add(i);

}

}

halfTree = new HalfNode(allElements.get(1023 / 2));

buildHalfTree(allElements.subList(0, 1023 / 2), true, halfTree);

buildHalfTree(allElements.subList(1023 / 2 + 1, allElements.size()), false, halfTree);

halfSuccess = 0;

for(Integer i : allElements) {

halfSuccess(halfTree, i);

}

System.out.println("折半查找成功的平均查找长度为" + (halfSuccess \* 1.0) / allElements.size());

halfFailure = 0;

nullNode = 0;

for(Integer i : allElements) {

tempTimes = 0;

halfFailure(halfTree, i);

}

System.out.println("折半查找失败的平均查找长度为" + (halfFailure\*1.0) / nullNode);

}

private static int halfFailure(HalfNode node, int value) {

if(node == null) {

return -1;

} else if(value == node.getValue()) {

if(node.getLeftNode() == null && node.getRightNode() == null) {

tempTimes ++;

nullNode += 2;

halfFailure = halfFailure + 2 \* tempTimes;

} else {

tempTimes ++;

nullNode += 1;

halfFailure = halfFailure + tempTimes;

}

return value;

}

if(value < node.getValue()) {

tempTimes ++;

return halfFailure(node.getLeftNode(), value);

} else {

tempTimes ++;

return halfFailure(node.getRightNode(), value);

}

}

private static int halfSuccess(HalfNode node, int value) {

if(node == null) {

return -1;

} else if(value == node.getValue()) {

return value;

}

if(value < node.getValue()) {

halfSuccess ++;

return halfSuccess(node.getLeftNode(), value);

} else {

halfSuccess ++;

return halfSuccess(node.getRightNode(), value);

}

}

private static void buildHalfTree(List<Integer> list, boolean left, HalfNode parent) {

if(list.size() == 1) {

if(left) {

parent.setLeftNode(new HalfNode(list.get(0)));

return;

} else {

parent.setRightNode(new HalfNode(list.get(0)));

return;

}

}

if(list.size() == 2) {

if(left) {

if(list.get(0) > list.get(1)) {

parent.setLeftNode(new HalfNode(list.get(1)));

parent.getLeftNode().setRightNode(new HalfNode(list.get(0)));

return;

} else {

parent.setLeftNode(new HalfNode(list.get(0)));

parent.getLeftNode().setRightNode(new HalfNode(list.get(1)));

return;

}

} else {

if(list.get(0) > list.get(1)) {

parent.setRightNode(new HalfNode(list.get(1)));

parent.getRightNode().setRightNode(new HalfNode(list.get(0)));

return;

} else {

parent.setRightNode(new HalfNode(list.get(0)));

parent.getRightNode().setRightNode(new HalfNode(list.get(1)));

return;

}

}

}

if(left) {

parent.setLeftNode(new HalfNode(list.get(list.size() / 2)));

buildHalfTree(list.subList(0, list.size() / 2), true, parent.getLeftNode());

buildHalfTree(list.subList((list.size() / 2) + 1, list.size()), false, parent.getLeftNode());

} else {

parent.setRightNode(new HalfNode(list.get(list.size() / 2)));

buildHalfTree(list.subList(0, list.size() / 2), true, parent.getRightNode());

buildHalfTree(list.subList((list.size() / 2) + 1, list.size()), false, parent.getRightNode());

}

}

}

class NodeType {

private int record;

private NodeType parentNode;

private NodeType leftChild;

private NodeType rightChild;

public int getRecord() {

return record;

}

public NodeType getParentNode() {

return parentNode;

}

public void setParentNode(NodeType parentNode) {

this.parentNode = parentNode;

}

public NodeType getRightChild() {

return rightChild;

}

public void setRightChild(NodeType rightChild) {

this.rightChild = rightChild;

}

public NodeType getLeftChild() {

return leftChild;

}

public void setLeftChild(NodeType leftChild) {

this.leftChild = leftChild;

}

public void setRecord(int record) {

this.record = record;

}

}

class HalfNode {

private int value;

private HalfNode leftNode;

private HalfNode rightNode;

public HalfNode(int value) {

this.value = value;

}

public int getValue() {

return value;

}

public HalfNode getLeftNode() {

return leftNode;

}

public void setLeftNode(HalfNode leftNode) {

this.leftNode = leftNode;

}

public HalfNode getRightNode() {

return rightNode;

}

public void setRightNode(HalfNode rightNode) {

this.rightNode = rightNode;

}

public void setValue(int value) {

this.value = value;

}

}