

**CSC248**

**FUNDAMENTALS OF DATA  
STRUCTURE**

**LAB ASSIGNMENT 7**

NAME : MUHAMMAD REDZA BIN MAHAYADIN

STUDENT ID : 2022676696

GROUP : RCDCS1103B

LECTURER : SIR MOHD NIZAM BIN OSMAN

**QUESTION 1**

BOOK CLASS

package Q1;

public class Book {

    private int serialNum;

    private String title;

    private String author;

    private char code;

    private String publisher;

    private int year;

    public Book() {

    }

    public void setData(int sn, String t, String a, char c, String p, int y) {

        // method definition

        serialNum = sn;

        title = t;

        author = a;

        code = c;

        publisher = p;

        year = y;

    }

    public int getSerialNum() {

        return serialNum;

    }

    public String getTitle() {

        return title;

    }

    public String getAuthor() {

        return author;

    }

    public char getCode() {

        return code;

    }

    public String getPublisher() {

        return publisher;

    }

    public int getYear() {

        return year;

    }

    public String toString() {

        return "Serial Number: " + serialNum + "\nTitle: " + title + "\nAuthor: " + author + "\nCode: " + code

                + "\nPublisher: " + publisher + "\nYear: " + year;

    }

}

BOOKRECORD CLASS

package Q1;

public class bookRecord {

    private TreeNode root;

    public bookRecord() {

        root = null;

    }

    // getter and setter

    public TreeNode getRoot() {

        return root;

    }

    public void setRoot(TreeNode root) {

        this.root = root;

    }

    // count the number of books in the tree

    // the data is already stored in the tree

    // book code is A, B, C, D

    public void countBookCode() {

        // theres code A, B, C, D in the tree

        int[] counts = new int[4];

        // count the book code

        countBookCode(root, counts);

        // print the result

        System.out.println("Book code A: " + counts[0]);

        System.out.println("Book code B: " + counts[1]);

        System.out.println("Book code C: " + counts[2]);

        System.out.println("Book code D: " + counts[3]);

    }

    public void countBookCode(TreeNode node, int[] counts) {

        // if the node is null, return

        if (node == null) {

            return;

        }

        // get the book code

        char code = ((Book) node.getData()).getCode();

        // increment the count

        switch (code) {

            case 'A':

                counts[0]++;

                break;

            case 'B':

                counts[1]++;

                break;

            case 'C':

                counts[2]++;

                break;

            case 'D':

                counts[3]++;

                break;

        }

        // count the left and right node

        countBookCode(node.getLeft(), counts);

        countBookCode(node.getRight(), counts);

    }

    public void searchBook(int serialNum) {

        // search for a book with the given serial number

        // if found, print the book

        // if not found, print not found

        TreeNode node = searchBook(root, serialNum);

        if (node == null) {

            System.out.println("Book not found");

        } else {

            System.out.println(node.getData());

        }

    }

    public TreeNode searchBook(TreeNode node, int serialNum) {

        // if the node is null, return null

        if (node == null) {

            return null;

        }

        // get the serial number

        int nodeSerialNum = ((Book) node.getData()).getSerialNum();

        // if the serial number is the same, return the node

        if (nodeSerialNum == serialNum) {

            return node;

        }

        // search the left and right node

        TreeNode left = searchBook(node.getLeft(), serialNum);

        TreeNode right = searchBook(node.getRight(), serialNum);

        // if the left node is not null, return the left node

        if (left != null) {

            return left;

        }

        // if the right node is not null, return the right node

        if (right != null) {

            return right;

        }

        // if not found, return null

        return null;

    }

    public void displayAll() {

        // display all the books in the tree

        // dont use linked list

        displayAll(root);

    }

    public void displayAll(TreeNode node) {

        // if the node is null, return

        if (node == null) {

            return;

        }

        // print the book

        System.out.println(node.getData() + "\n");

        // display the left and right node

        displayAll(node.getLeft());

        displayAll(node.getRight());

    }

}

TREENODE CLASS

package Q1;

public class TreeNode {

    // data declaration

    private Object element;

    private TreeNode left;

    private TreeNode right;

    public TreeNode(Object elem) {

        // method definition

        this.element = elem;

        this.left = null;

        this.right = null;

    }

    // this is done recursively

    public void insert(Object elem) {

        // if book serial number is less than the current node

        Book book = (Book) elem;

        Book currentBook = (Book) this.element;

        if (book.getSerialNum() < currentBook.getSerialNum()) {

            // if left node is null, insert new node

            if (this.left == null) {

                this.left = new TreeNode(elem);

            } else {

                // else, insert to left node

                this.left.insert(elem);

            }

        } else {

            // if right node is null, insert new node

            if (this.right == null) {

                this.right = new TreeNode(elem);

            } else {

                // else, insert to right node

                this.right.insert(elem);

            }

        }

    }

    public Book getData() {

        // method definition

        return (Book) this.element;

    }

    public TreeNode getLeft() {

        // method definition

        return this.left;

    }

    public TreeNode getRight() {

        // method definition

        return this.right;

    }

}

MAIN CLASS

package Q1;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        TreeNode root = null;

        bookRecord record = new bookRecord();

        Book[] books = new Book[8];

        // initialize the book

        for (int i = 0; i < books.length; i++) {

            books[i] = new Book();

        }

        // insert the book into the tree

        books[0].setData(1217, "Bunga Dedap", "Daud Kamal", 'A', "Sejana", 1998);

        books[1].setData(1324, "Fizik", "Prof. Bun Tat", 'C', "Mc Graw", 2000);

        books[2].setData(1001, "Kimia", "Prof. Kamarul", 'C', "Anderson", 2001);

        books[3].setData(1009, "Botani", "Puan Salmah", 'E', "Mutiara", 1999);

        books[4].setData(781, "Komputer", "Dr Abu ", 'D', "Deitel", 2001);

        books[5].setData(4320, "Sosial", "Dr Kamariah", 'B', "Mutiara", 1998);

        books[6].setData(2700, "Ilmu Alam ", "Dr Kamarudin ", 'A', "Mutiara ", 1999);

        books[7].setData(1243, "Sejarah ", "Puan Kalsom ", 'B', "Tamadun ", 1989);

        // insert the book into the tree using the insert method

        for (int i = 0; i < books.length; i++) {

            if (root == null) {

                root = new TreeNode(books[i]);

            } else {

                root.insert(books[i]);

            }

        }

        // set the root

        record.setRoot(root);

        while (true) {

            System.out.println("1. Count the number of books in the tree");

            System.out.println("2. Search for a book");

            System.out.println("3. Display all books");

            System.out.println("4. Exit");

            System.out.print("\nEnter your choice: ");

            int choice = input.nextInt();

            System.out.println();

            switch (choice) {

                case 1:

                    // record.countBookCode(root);

                    record.countBookCode();

                    break;

                case 2:

                    System.out.print("Enter the serial number: ");

                    int serialNum = input.nextInt();

                    System.out.println();

                    record.searchBook(serialNum);

                    break;

                case 3:

                    record.displayAll();

                    break;

                case 4:

                    input.close();

                    System.out.println("Thank you for using this program!");

                    System.exit(0);

                    break;

                default:

                    System.out.println("Invalid choice!");

                    break;

            }

            System.out.println();

        }

    }

}

**QUESTION 2**

BSTTOCRIMECOMPLAINT CLASS

package Q2;

public class BSTcCrimeComplaint {

    TreeNode root;

    public BSTcCrimeComplaint() {

        root = null;

    }

    public void insertNode(CCrimeComplaint info) {

        if (root == null) {

            root = new TreeNode(info);

        } else {

            TreeNode current = root;

            TreeNode parent = null;

            while (true) {

                parent = current;

                if (info.getcElement().compareToIgnoreCase(current.getInfo().getcElement()) < 0) {

                    current = current.getLeft();

                    if (current == null) {

                        parent.setLeft(new TreeNode(info));

                        return;

                    }

                } else {

                    current = current.getRight();

                    if (current == null) {

                        parent.setRight(new TreeNode(info));

                        return;

                    }

                }

            }

        }

    }

    // display all but prevent duplicate

    public void cElementDisplayAll() {

        cElementDisplayAll(root);

    }

    public void cElementDisplayAll(TreeNode root) {

        if (root != null) {

            cElementDisplayAll(root.getLeft());

            System.out.println(root.getInfo().getcElement());

            cElementDisplayAll(root.getRight());

        }

    }

    // display specific

    public void displayBySpesific(String cElement) {

        displayBySpesific(root, cElement);

    }

    public void displayBySpesific(TreeNode root, String cElement) {

        if (root != null) {

            displayBySpesific(root.getLeft(), cElement);

            if (root.getInfo().getcElement().equalsIgnoreCase(cElement)) {

                System.out.println(root.getInfo().toString() + "\n");

            }

            displayBySpesific(root.getRight(), cElement);

        }

    }

    public int calTotComplaint(int year) {

        return calTotComplaint(root, year);

    }

    public int calTotComplaint(TreeNode root, int year) {

        if (root == null) {

            return 0;

        } else {

            if (root.getInfo().getYear() == year) {

                return root.getInfo().getNoOfComplaint() + calTotComplaint(root.getLeft(), year)

                        + calTotComplaint(root.getRight(), year);

            } else {

                return calTotComplaint(root.getLeft(), year) + calTotComplaint(root.getRight(), year);

            }

        }

    }

    // other method

    public int countNode() {

        return countNode(root);

    }

    public int countNode(TreeNode root) {

        if (root == null) {

            return 0;

        } else {

            return 1 + countNode(root.getLeft()) + countNode(root.getRight());

        }

    }

    public int countLeaf() {

        return countLeaf(root);

    }

    public int countLeaf(TreeNode root) {

        if (root == null) {

            return 0;

        } else {

            if (root.getLeft() == null && root.getRight() == null) {

                return 1;

            } else {

                return countLeaf(root.getLeft()) + countLeaf(root.getRight());

            }

        }

    }

    // count height

    public int countHeight() {

        return countHeight(root);

    }

    public int countHeight(TreeNode root) {

        if (root == null) {

            return 0;

        } else {

            return 1 + Math.max(countHeight(root.getLeft()), countHeight(root.getRight()));

        }

    }

    // delete node

    public void deleteNode(String cElement) {

        root = deleteNode(root, cElement);

    }

    public TreeNode deleteNode(TreeNode root, String cElement) {

        if (root == null) {

            return root;

        } else {

            if (cElement.compareToIgnoreCase(root.getInfo().getcElement()) < 0) {

                root.setLeft(deleteNode(root.getLeft(), cElement));

            } else if (cElement.compareToIgnoreCase(root.getInfo().getcElement()) > 0) {

                root.setRight(deleteNode(root.getRight(), cElement));

            } else {

                if (root.getLeft() == null) {

                    return root.getRight();

                } else if (root.getRight() == null) {

                    return root.getLeft();

                } else {

                    root.setInfo(findMin(root.getRight()));

                    root.setRight(deleteNode(root.getRight(), root.getInfo().getcElement()));

                }

            }

        }

        return root;

    }

    // delete specific

    public void deleteSpecific(String cElement, int year) {

        root = deleteSpecific(root, cElement, year);

    }

    public TreeNode deleteSpecific(TreeNode root, String cElement, int year) {

        if (root == null) {

            return root;

        } else {

            if (cElement.compareToIgnoreCase(root.getInfo().getcElement()) < 0) {

                root.setLeft(deleteSpecific(root.getLeft(), cElement, year));

            } else if (cElement.compareToIgnoreCase(root.getInfo().getcElement()) > 0) {

                root.setRight(deleteSpecific(root.getRight(), cElement, year));

            } else {

                if (root.getInfo().getYear() == year) {

                    if (root.getLeft() == null) {

                        return root.getRight();

                    } else if (root.getRight() == null) {

                        return root.getLeft();

                    } else {

                        root.setInfo(findMin(root.getRight()));

                        root.setRight(deleteSpecific(root.getRight(), root.getInfo().getcElement(), year));

                    }

                } else {

                    root.setLeft(deleteSpecific(root.getLeft(), cElement, year));

                    root.setRight(deleteSpecific(root.getRight(), cElement, year));

                }

            }

        }

        return root;

    }

    // find min

    public CCrimeComplaint findMin(TreeNode root) {

        if (root == null) {

            return null;

        } else {

            if (root.getLeft() == null) {

                return root.getInfo();

            } else {

                return findMin(root.getLeft());

            }

        }

    }

}

CCRIMECOMPLAINT CLASS

package Q2;

public class CCrimeComplaint {

    private String cElement;

    private int noOfComplaint;

    private int year;

    public CCrimeComplaint(String cElement, int noOfComplaint, int year) {

        this.cElement = cElement;

        this.noOfComplaint = noOfComplaint;

        this.year = year;

    }

    public String getcElement() {

        return cElement;

    }

    public void setcElement(String cElement) {

        this.cElement = cElement;

    }

    public int getNoOfComplaint() {

        return noOfComplaint;

    }

    public void setNoOfComplaint(int noOfComplaint) {

        this.noOfComplaint = noOfComplaint;

    }

    public int getYear() {

        return year;

    }

    public void setYear(int year) {

        this.year = year;

    }

    public String toString() {

        return "Element: " + cElement + "\nNumber of Complaint: " + noOfComplaint + "\nYear: " + year;

    }

}

TREENODE CLASS

package Q2;

public class TreeNode {

    CCrimeComplaint info;

    TreeNode left, right;

    public TreeNode(CCrimeComplaint info) {

        this.info = info;

        left = null;

        right = null;

    }

    public TreeNode(CCrimeComplaint info, TreeNode left, TreeNode right) {

        this.info = info;

        this.left = left;

        this.right = right;

    }

    public CCrimeComplaint getInfo() {

        return info;

    }

    public void setInfo(CCrimeComplaint info) {

        this.info = info;

    }

    public TreeNode getLeft() {

        return left;

    }

    public void setLeft(TreeNode left) {

        this.left = left;

    }

    public TreeNode getRight() {

        return right;

    }

    public void setRight(TreeNode right) {

        this.right = right;

    }

    // tostring method

    public String toString() {

        return info.toString();

    }

}

MAIN CLASS

package Q2;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner strInput = new Scanner(System.in);

        Scanner intInput = new Scanner(System.in);

        BSTcCrimeComplaint bst = new BSTcCrimeComplaint();

        String[][] complaints = {

                { "Complaint Elements", "Year", "Total Complaints" },

                { "Obscene", "2019", "969" },

                { "Obscene", "2020", "850" },

                { "False", "2019", "2117" },

                { "False", "2020", "3050" },

                { "Offensive", "2019", "1311" },

                { "Offensive", "2020", "2312" },

                { "Indecent", "2019", "139" },

                { "Indecent", "2020", "188" },

                { "Menacing", "2019", "45" },

                { "Menacing", "2020", "88" },

                { "Others", "2019", "3938" },

                { "Others", "2020", "7472" }

        };

        for (int i = 1; i < complaints.length; i++) {

            bst.insertNode(new CCrimeComplaint(complaints[i][0], Integer.parseInt(complaints[i][2]),

                    Integer.parseInt(complaints[i][1])));

        }

        while (true) {

            System.out.println("1. Display all complaint elements");

            System.out.println("2. Display specific complaint elements");

            System.out.println("3. Calculate total complaints for a specific year");

            System.out.println("4. Calculate increment percentage for total number of complaints from 2019 to 2020");

            System.out.println("5. Exit");

            System.out.print("\nEnter your choice: ");

            int choice = intInput.nextInt();

            switch (choice) {

                case 1:

                    bst.cElementDisplayAll();

                    break;

                case 2:

                    System.out.print("Enter complaint element: ");

                    String cElement = strInput.nextLine();

                    bst.displayBySpesific(cElement);

                    break;

                case 3:

                    System.out.print("Enter year: ");

                    int year = intInput.nextInt();

                    System.out.println("\nTotal complaints for year " + year + ": " + bst.calTotComplaint(year));

                    break;

                case 4:

                    System.out.println("\n2020 complaint amount: " + bst.calTotComplaint(2020));

                    System.out.println("2019 complaint amount: " + bst.calTotComplaint(2019));

                    // change to double to get decimal

                    double percentage = ((double) (bst.calTotComplaint(2020) - bst.calTotComplaint(2019))

                            / (double) bst.calTotComplaint(2019)) \* 100;

                    System.out.println("\n((" + bst.calTotComplaint(2020) + " - " + bst.calTotComplaint(2019) + ") / "

                            + bst.calTotComplaint(2019) + ") \* 100 = " + percentage + "%");

                    System.out.println("\nIncrement percentage for total number of complaints from 2019 to 2020: "

                            + String.format("%.2f", percentage) + "%");

                    break;

                case 5:

                    System.out.println("Thank you for using this program");

                    System.exit(0);

                    break;

                default:

                    System.out.println("Invalid choice");

                    break;

            }

            System.out.println();

        }

    }

}

**QUESTION 3**

BSTCANDIDATE CLASS

package Q3;

public class BSTCandidate {

    TreeNode root;

    public BSTCandidate() {

        root = null;

    }

    // setter and getter for root

    public TreeNode getRoot() {

        return this.root;

    }

    public void setRoot(TreeNode root) {

        this.root = root;

    }

    // insert new candidate into the tree

    public void insert(JobCandidate data) {

        root = insert(root, data);

    }

    private TreeNode insert(TreeNode root, JobCandidate data) {

        if (root == null) {

            root = new TreeNode(data);

        } else {

            if (data.getRegNo() < root.getData().getRegNo()) {

                root.setLeft(insert(root.getLeft(), data));

            } else {

                root.setRight(insert(root.getRight(), data));

            }

        }

        return root;

    }

    // display details of all candidates using recursive method to display detail of

    // candidate name in descending order

    public void displayDetails() {

        displayDetails(root);

    }

    private void displayDetails(TreeNode root) {

        if (root != null) {

            displayDetails(root.getRight());

            System.out.println(root.getData());

            displayDetails(root.getLeft());

        }

    }

    public int countCandidate(char qualification) {

        return countCandidate(root, qualification);

    }

    private int countCandidate(TreeNode root, char qualification) {

        if (root == null) {

            return 0;

        } else {

            int count = 0;

            if (root.getData().getQualification() == qualification) {

                count++;

            }

            count += countCandidate(root.getLeft(), qualification);

            count += countCandidate(root.getRight(), qualification);

            return count;

        }

    }

}

JOBCANDIDATE CLASS

package Q3;

public class JobCandidate {

    private int regNo;

    private String name;

    private char qualification; // D for Diploma, B for Bachelor, M for Master, P for PhD

    private int age;

    private char gender; // M - Male, F - Female

    public JobCandidate(int regNo, String name, char qualification, int age, char gender) {

        this.regNo = regNo;

        this.name = name;

        this.qualification = qualification;

        this.age = age;

        this.gender = gender;

    }

    public int getRegNo() {

        return this.regNo;

    }

    public void setRegNo(int regNo) {

        this.regNo = regNo;

    }

    public String getName() {

        return this.name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public char getQualification() {

        return this.qualification;

    }

    public void setQualification(char qualification) {

        this.qualification = qualification;

    }

    public int getAge() {

        return this.age;

    }

    public void setAge(int age) {

        this.age = age;

    }

    public char getGender() {

        return this.gender;

    }

    public void setGender(char gender) {

        this.gender = gender;

    }

    // toString() method

    public String toString() {

        return "Registration Number: " + regNo + "\nName: " + name + "\nQualification: " + qualification

                + "\nAge: " + age + "\nGender: " + gender;

    }

}

TREENODE CLASS

package Q3;

public class TreeNode {

    JobCandidate data;

    TreeNode left, right;

    public TreeNode(JobCandidate data) {

        this.data = data;

        this.left = null;

        this.right = null;

    }

    public JobCandidate getData() {

        return this.data;

    }

    public void setData(JobCandidate data) {

        this.data = data;

    }

    public TreeNode getLeft() {

        return this.left;

    }

    public void setLeft(TreeNode left) {

        this.left = left;

    }

    public TreeNode getRight() {

        return this.right;

    }

    public void setRight(TreeNode right) {

        this.right = right;

    }

    // toString() method

    public String toString() {

        return data.toString();

    }

}

MAIN CLASS

package Q3;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner intInput = new Scanner(System.in);

        Scanner strInput = new Scanner(System.in);

        BSTCandidate candidateTree = new BSTCandidate();

        String[][] jobCandidates = {

                { "3358", "MUHAMMAD AZMIL BIN AHIMAD", "D", "22", "M" },

                { "5262", "SYAHIRAH BINTI ISMAIL", "P", "32", "F" },

                { "4221", "HUSNA BT ROHA", "M", "28", "F" },

                { "3395", "MUHAMMAD DANIAL BIN NAZIM", "S", "26", "M" },

                { "3222", "DIYANA NUR BINTI HASBI", "S", "24", "F" },

                { "5256", "BATRISYA BINTI DIN", "P", "35", "F" },

                { "3345", "AMIR HAKIM BIN DANIAL", "D", "25", "M" },

                { "3353", "LUQMAN BIN AHMAD", "D", "25", "M" }

        };

        for (int i = 0; i < jobCandidates.length; i++) {

            JobCandidate candidate = new JobCandidate(Integer.parseInt(jobCandidates[i][0]), jobCandidates[i][1],

                    jobCandidates[i][2].charAt(0), Integer.parseInt(jobCandidates[i][3]),

                    jobCandidates[i][4].charAt(0));

            candidateTree.insert(candidate);

        }

        // calculate and display total candidates with masters and phd qualification

        int total = candidateTree.countCandidate('M') + candidateTree.countCandidate('P');

        System.out.println("Total candidates with masters and phd qualification: " + total);

    }

}