**Đề 1:**

**Câu 1:**

**package** Goi\_2;

**import** java.util.Scanner;

**class** Car

{

**int** id;

String name;

Car link;

**public** Car ()

{

**this**.id =0; **this**.name=**null**; **this**.link= **null**;

}

**public** Car(**int** id, String name, Car link)

{

**this**.id =id; **this**.name=name; **this**.link= link;

}

**public** **void** setLink(Car phantu)

{

**this**.link = phantu;

}

**public** Car getLink()

{

**return** **this**.link;

}

}

**public** **class** dslk1

{

Car head, tail;

**int** size;

**public** dslk1()

{

head=**null**; tail=**null**; size =0;

}

**public** **boolean** timNhanHieu(String name)

{

Car current = head; // tìm từ đầu

**while** (current != **null**)

{

**if** (current.name.equalsIgnoreCase(name))

**return** **true**; // Tìm thấy

current = current.link;

}

**return** **false**; // Tìm không thấy

}

**public** **void** insertAtEnd(**int** id, String name)

{

**if** (timNhanHieu(name))

System.***out***.println("Name trung, không the them!");

**else**

{

Car phantu = **new** Car(id, name, **null**);

size++ ;

**if**(head == **null**)

{

head = phantu;

tail = head;

}

**else**

{

tail.setLink(phantu);

tail = phantu;

}

}

}

**public** **void** printList()

{

Car temp = **new** Car();

temp = **this**.head;

**if**(temp != **null**)

{

System.***out***.println("Danh sach lien ket: ");

**while**(temp != **null**)

{

System.***out***.print(temp.id + " " + temp.name + "; ");

temp = temp.link;

}

System.***out***.println();

}

**else**

System.***out***.println("Danh sach lien ket rong!.");

}

**public** **static** **void** main(String[] args)

{

Scanner scan = **new** Scanner(System.***in***);

dslk1 l1 = **new** dslk1();

**int** id; **char** tt;

String name;

**do**

{

System.***out***.print("Nhap id: "); id = scan.nextInt();

scan.nextLine();

System.***out***.print("Nhap name: "); name =scan.nextLine();

l1.insertAtEnd(id, name);

System.***out***.print("Co tiep tuc khong (Y/N): ");

tt = (**char**)scan.next().charAt(0);

} **while** (Character.*toLowerCase*(tt) == 'y');

l1.printList();

scan.close();

System.*out*.println("Ket thuc Chuong trinh.");

}

}

**2/. Cây tìm kiếm nhị phân:**

**A/.** Viết phương thức nhận một tham số là một mảng chứa các số nguyên, thực hiện tạo cây tìm kiếm nhị phân từ mảng đã cho (có kiểm tra trùng khóa) (2.0 đ)

**public** **void** arraytoBST(**int** arr[], BST t)

{ //phương thức lấy dữ liệu từ array --> bst, có loại bỏ key trùng

**for**(**int** i=0; i<arr.length; i++)

**if** (!t.search(arr[i]))

t.insert(arr[i]);

//t.inorder();

}

**B/.** Viết phương thức nhận một tham số x có kiểu int, thực hiện đếm số node trong cây tìm kiếm nhị phân có key > x. (1.5 đ)

**public** **int** countX(**int** x)

{

**return** countX(root,x);

}

**private** **int** countX(BSTNode root, **int** x)

{ **int** count=0;

**if**(root==**null**)

**return** 0;

**if**(root.data > x) count++;

**int** countLeft = countX(root.left, x);

**int** countRight = countX(root.right, x);

**return** count + countLeft + countRight;

}

**C/.** Viết phương thức đếm số node có 1 con trái của cây tìm kiếm nhị phân. Viết phương thức main để test chương trình (1.5 đ)

**public** **int** countLeftChild() // đếm số node có 1 con trái

{

**return** countLeftChild(root);

}

**private** **int** countLeftChild(BSTNode root)

{

**int** left\_count, right\_count, count =0;

**if** (root == **null**)

**return** 0;

**if** (root.left != **null** && root.right == **null**)

count = count + 1;

left\_count = countLeftChild(root.left);

right\_count = countLeftChild(root.right);

count = count + left\_count + right\_count;

**return** count;

}

**Đề 3:**

**Câu 1:**

**import** java.util.Scanner;

**class** Nuoc

{

**int** stt;

String tennuoc;

Nuoc link;

**public** Nuoc ()

{

**this**.stt =0; **this**.tennuoc=**null**; **this**.link= **null**;

}

**public** Nuoc(**int** stt, String tennuoc, Nuoc link)

{

**this**.stt =stt; **this**.tennuoc=tennuoc; **this**.link= link;

}

**public** **void** setLink(Nuoc phantu)

{

**this**.link = phantu;

}

**public** Nuoc getLink()

{

**return** **this**.link;

}

}

**public** **class** dslk2

{

Nuoc head, tail;

**int** size;

**public** dslk2()

{

head=**null**; tail=**null**; size =0;

}

**public** **int** timTenNuoc(String tennuoc)

{

Nuoc current = head; // tìm từ đầu

**while** (current != **null**)

{

**if** (current.tennuoc.equalsIgnoreCase(tennuoc))

**return** 1; // Tìm thấy

current = current.link;

}

**return** **0**; // Tìm không thấy

}

**public** **void** insertAtEnd(**int** stt, String tennuoc)

{

**if** (timTenNuoc(tennuoc)==1)

System.***out***.println("Ten Nuoc trung, khong the them!");

**else**

{

Nuoc phantu = **new** Nuoc(stt, tennuoc, **null**);

size++ ;

**if**(head == **null**)

{

head = phantu;

tail = head;

}

**else**

{

tail.setLink(phantu);

tail = phantu;

}

}

}

**public** **void** printList()

{

Nuoc temp = **new** Nuoc();

temp = **this**.head;

**if**(temp != **null**)

{

System.***out***.println("Danh sach lien ket: ");

**while**(temp != **null**)

{

System.***out***.print(temp.stt + " " + temp.tennuoc + "; ");

temp = temp.link;

}

System.***out***.println();

}

**else**

System.***out***.println("Danh sach lien ket rong!.");

}

**public** **static** **void** main(String[] args)

{

Scanner scan = **new** Scanner(System.***in***);

dslk2 l2 = **new** dslk2();

**int** stt; **char** tt;

String tennuoc;

**do**

{

System.***out***.print("Nhap So thu tu: "); stt = scan.nextInt();

scan.nextLine();

System.***out***.print("Nhap Ten nuoc: "); tennuoc =scan.nextLine();

l2.insertAtEnd(stt, tennuoc);

System.***out***.print("Co tiep tuc khong (Y/N): ");

tt = (**char**)scan.next().charAt(0);

} **while** (Character.*toLowerCase*(tt) == 'y');

l2.printList();

scan.close();

System.***out***.println("Ket thuc Chuong trinh.");

}

}

**Câu 2/. Cây tìm kiếm nhị phân:**

**A/.** Viết phương thức nhận một tham số là một mảng chứa các số nguyên, thực hiện tạo cây tìm kiếm nhị phân từ các giá trị > 0 của mảng đã cho (có kiểm tra trùng khóa) (2.0 đ)

**public** **void** arraytoBST(**int** arr[], BST t)

{ //phương thức lấy dữ liệu từ array --> bst, có loại bỏ key trùng

**for**(**int** i=0; i<arr.length; i++)

**if** (arr[i]>0)

**if** (!t.search(arr[i]))

t.insert(arr[i]);

//t.inorder();

}

**B/.** Viết phương thức duyệt các node có giá trị lẻ theo inorder. (1.5 đ)

**public** **void** inorderle()

{

inorderle(root);

}

**private** **void** inorderle(BSTNode root)

{

**if** (root != **null**)

{

inorderle(root.getLeft());

**if** (root.data %2 != 0)

System.***out***.print(root.getData() +" ");

inorderle(root.getRight());

}

}

**C/.** Viết phương thức đếm số node có 1 con phải của cây tìm kiếm nhị phân. Viết phương thức main để test chương trình (1.5 đ)

**public** **int** countRightChild() //đếm số node có 1 con phải

{

**return** countRightChild(root);

}

**private** **int** countRightChild(BSTNode root)

{

**int** left\_count, right\_count, count =0;

**if** (root == **null**)

**return** 0;

**if** (root.left==**null** && root.right != **null**)

count = count + 1;

left\_count = countRightChild(root.left);

right\_count = countRightChild(root.right);

count = count + left\_count + right\_count;

**return** count;

}

7/. Duyệt Inorde, Preorder không đệ quy (dùng vòng lặp)

import java.io.\*; import java.util.\*;

class Node1

{

int data;

Node1 left, right;

public Node1(int d)

{

data = d;

left = right = null;

}

public int getData()

{

return data;

}

}

public class BinarySearchTree

{

Node1 root;

public BinarySearchTree()

{

root=null;

}

public void insert(int d)

{

root = insert(root, d);

}

private Node1 insert(Node1 node, int d)

{

if (node == null)

node = new Node1(d); //đệ qui

else

{

if (d <= node.getData())

node.left = insert(node.left, d);

else

node.right = insert(node.right, d);

}

return node;

}

public void inOrder()

{ //Duyệt bst theo inorder (LNR) bằng vòng lặp

Node1 cur = root;

while (cur != null)

{

if (cur.left == null)

{

System.out.print(cur.data + " ");

cur = cur.right;

}

else

{

Node1 temp = cur.left;

while (temp.right != null && temp.right != cur)

temp = temp.right;

if (temp.right == null)

{

temp.right = cur;

cur = cur.left;

}

else

{

temp.right = null;

System.out.print(cur.data + " ");

cur = cur.right;

}

}

}

}

public void preorder()

{

preorder(root);

}

private void preorder(Node1 node)

{ //Duyệt bst theo preorder (NLR) bằng vòng lặp

while (node != null)

{

if (node.left == null) {

System.out.print(node.data + " ");

node = node.right;

}

else

{

Node1 current = node.left;

while (current.right != null && current.right != node)

{

current = current.right;

}

if (current.right == node)

{

current.right = null;

node = node.right;

}

else

{

System.out.print(node.data + " ");

current.right = node;

node = node.left;

}

}

}

}

public static void main(String[] args)

{

BinarySearchTree bst = new BinarySearchTree();

bst.insert(15); bst.insert(18); bst.insert(20);

bst.insert(13); bst.insert(10); bst.insert(9);

bst.insert(5); bst.insert(11); bst.insert(22);

System.out.println("\nDuyet inorder:");

bst.inOrder();

System.out.println("\nDuyet preorder:");

bst.preorder();

}

}