**CS323-22: Project 3 (JAVA)**

**Yida Tao**

**Due date: Sep. 26, 2017**

Algorithm Steps:

Step 1: open inFile and all outfiles

Step 2.1: listHead 🡨 create a dummy listBinTreeNode for listHead to point to

step 2.2: (chStr and prob) 🡨 get from inFile

step 2.3: spot 🡨 findSpot(prob) // see the method in the above

step 2.4: newNode 🡨 make a new node // use constructor with chStr and prob

step 2.5: listInsert(spot, newNode) // inserting newNode between spot and spot.next.

step 2.6: printList to outFile5 //

step 2.7: repeat step 2.2 to step 2.6 while inFile is NOT empty

step 2.8: close inFile

step 3.0: oldListHead 🡨 create a dummy listBinTreeNode and let oldListHead points

to the dummy node

oldListHead.next 🡨listHead.next // save the original linked list

step 3.1: newNode 🡨 create a listBinTreeNode

newNode’s prob 🡨 the sum of prob of the first and second node of the list

newNode’s chStr 🡨 concatenate chStr of the first node and chStr of the s econd node in the list

newNode’s left 🡨 the first node of the list

newNode’s right 🡨 the second node of the list

listHead 🡨 the third node of the list

printNode (newNode) to outFile5

step 3.2: spot 🡨 findSpot(newNode’prob)

listInsert (spot, newNode)

printList to outFile5 //for debugging purpose

step 3.3: repeat step 3.1 – 3.2 until the list only has one node left which is the newNode

step 3.4: Root 🡨 newNode

step 4: Construct Huffman char-code pairs in recursion

step 5: Print the Huffman Tree in Pre-ordering, in-ordering, post-ordering

step 6: Close files

**Source Code**

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

**public** **class** project3 {

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

HuffmanLinkedList LL = **new** HuffmanLinkedList();

HuffmanBinaryTree tree = **new** HuffmanBinaryTree();

FileOutputStream fos = **null**;

//constructHuffmanList

//constructHuffmanTree

**try** {

fos = **new** FileOutputStream(args[5]);

System.*setOut*(**new** PrintStream(fos));

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

LL.constructHuffmanLList(args[0], args[5]);

tree.constructHuffmanBinTree(LL);

//Call constructCharCode ()

**try** {

fos = **new** FileOutputStream(args[1]);

System.*setOut*(**new** PrintStream(fos));

tree.constructCharCode();

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

//Call preOrderTraveral ()

**try** {

fos = **new** FileOutputStream(args[2]);

System.*setOut*(**new** PrintStream(fos));

tree.preOrderTraveral();

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

//Call inOrderTraversal ()

**try** {

fos = **new** FileOutputStream(args[3]);

System.*setOut*(**new** PrintStream(fos));

tree.inOrderTraveral();;

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

//Call postOrderTraveral ()

**try** {

fos = **new** FileOutputStream(args[4]);

System.*setOut*(**new** PrintStream(fos));

tree.postOrderTraveral();;

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

//finish up

**try** {

fos.close();

System.*setOut*(**new** PrintStream(**new** FileOutputStream(FileDescriptor.***out***)));

} **catch** (IOException e) {

e.printStackTrace();

}

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

}//main

}//project3

**public** **class** listBinTreeNode {

String chStr;

**int** prob;

String code;

listBinTreeNode left;

listBinTreeNode right;

listBinTreeNode next;

**public** listBinTreeNode(){

chStr = "";

prob = 0;

code = "";

left = **null**;

right = **null**;

next = **null**;

}

**public** listBinTreeNode(String s, **int** i){

chStr = s;

prob = i;

code = "";

left = **null**;

right = **null**;

next = **null**;

}

**public** **void** printNode(){

String s = "";

**if**(**this**.chStr != **null**) s += "Node:" + **this**.chStr + ", Node's prob:" + **this**.prob;

**else**

s+= "Node:Null" + ", Node's prob:" + **this**.prob;

**if**(**this**.next != **null**) s += ", Node.next:" + **this**.next.chStr;

**else**

s += ", Node.next:null";

**if**(**this**.left != **null**) s += ", Node.left:" + **this**.left.chStr;

**else**

s += ", Node.left:null";

**if**(**this**.right != **null**) s += ", Node.right:" + **this**.right.chStr;

**else**

s += ", Node.right:null";

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

}

}//node

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.Scanner;

**public** **class** HuffmanLinkedList {

listBinTreeNode head;

**public** HuffmanLinkedList() {

head = **new** listBinTreeNode("dummy",0);

}

**public** **void** constructHuffmanLList(String in,String out){

listBinTreeNode newNode = **null**;

Scanner sc = **null**;

String s1 = "";

**int** i1 = 0;

**try** {

sc = **new** Scanner(**new** File(in));

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

**while**(sc.hasNext()) {

s1 = sc.next();

i1 = sc.nextInt();

newNode = **new** listBinTreeNode(s1,i1);

newNode.printNode();

**this**.listInsert(**this**.findSpot(newNode.prob), newNode);

**this**.printList();

}

sc.close();

}

**public** **int** findSpot(**int** prob){

**int** i=0;

listBinTreeNode walker = head;

**while**(walker.next != **null** && (prob > walker.next.prob)) {

//System.out.println(i + " " + walker.next.prob);

walker = walker.next;

i++;

}

**return** i;

}

**public** **void** listInsert(**int** spot, listBinTreeNode nNode){

listBinTreeNode n = nNode;

listBinTreeNode walker = head;

**for**(**int** i = 0; i < spot ; i++){

walker = walker.next;

}

n.next = walker.next;

walker.next = n;

}

**public** **boolean** isEmpty(){

**boolean** b = **false**;

**if**(**this**.head.next == **null**) b= **true**;

**return** b;

}

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

String s ="";

listBinTreeNode walker = head;

s += "listHead-->(" + walker.chStr + "," + walker.prob + ",";

**while**((walker = walker.next) != **null**) {

s += walker.chStr + ")-->("+ walker.chStr + "," + walker.prob + ",";

}

s += "Null)";

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

}

}//HuffmanLinkedList

**public** **class** HuffmanBinaryTree {

**private** listBinTreeNode root;

**public** HuffmanBinaryTree(){

root = **null**;

}

**public** HuffmanBinaryTree(listBinTreeNode t) {

root = t;

}

**public** **void** constructHuffmanBinTree(HuffmanLinkedList LL){

listBinTreeNode newHead = LL.head;

listBinTreeNode first = **null**;

listBinTreeNode second = **null**;

String comb ="";

**int** totalCount = 0;

listBinTreeNode nNode = **null**;

**while**(newHead.next.next != **null**){

first = newHead.next;

second = newHead.next.next;

comb = first.chStr + second.chStr;

totalCount = first.prob + second.prob;

nNode = **new** listBinTreeNode(comb , totalCount);

nNode.left = first;

nNode.right = second;

LL.listInsert(LL.findSpot(nNode.prob), nNode);

nNode.printNode();

LL.printList();

newHead = newHead.next.next;

}

root = nNode;

}

**private** **static** **boolean** isLeaf(listBinTreeNode t){

**boolean** b = **false**;

**if**(t.left == **null** && t.right == **null**) b = **true**;

**return** b;

}

**public** **void** constructCharCode(){

*constructCharCode*(root,"");

}

**private** **static** **void** constructCharCode(listBinTreeNode t,String s){

**if**(t!=**null**){

t.code = s;

*constructCharCode*(t.left,s+"0");

**if**(*isLeaf*(t)){

System.***out***.println(t.chStr + ", code:" + t.code);

}

*constructCharCode*(t.right,s+"1");

}

**return**;

}

**public** **void** preOrderTraveral(){

*preOrderTraveral*(root);

}

**private** **static** **void** preOrderTraveral(listBinTreeNode t){

**if**(t!=**null**){

System.***out***.println(t.chStr + ", prob:"+ t.prob);

*preOrderTraveral*(t.left);

*preOrderTraveral*(t.right);

}

**return**;

}

**public** **void** inOrderTraveral(){

*inOrderTraveral*(root);

}

**private** **static** **void** inOrderTraveral(listBinTreeNode t){

**if**(t!=**null**){

*inOrderTraveral*(t.left);

System.***out***.println(t.chStr + ", prob:"+ t.prob);

*inOrderTraveral*(t.right);

}

**return**;

}

**public** **void** postOrderTraveral(){

*postOrderTraveral*(root);

}

**private** **static** **void** postOrderTraveral(listBinTreeNode t){

**if**(t!=**null**){

*postOrderTraveral*(t.left);

*postOrderTraveral*(t.right);

System.***out***.println(t.chStr + ", prob:"+ t.prob);

}

**return**;

}

}//huffmanbinarytree

**Input**

d 2

f 4

h 15

k 6

m 47

p 25

w 1

**Output 1**

m, code:0

p, code:10

k, code:1100

w, code:110100

d, code:110101

f, code:11011

h, code:111

**Output 2**

preOrderTravesal:

mpkwdfh, prob:100

m, prob:47

pkwdfh, prob:53

p, prob:25

kwdfh, prob:28

kwdf, prob:13

k, prob:6

wdf, prob:7

wd, prob:3

w, prob:1

d, prob:2

f, prob:4

h, prob:15

**Output 3**

inOrderTravesal:

m, prob:47

mpkwdfh, prob:100

p, prob:25

pkwdfh, prob:53

k, prob:6

kwdf, prob:13

w, prob:1

wd, prob:3

d, prob:2

wdf, prob:7

f, prob:4

kwdfh, prob:28

h, prob:15

**Output 4**

postOrderTravesal:

m, prob:47

p, prob:25

k, prob:6

w, prob:1

d, prob:2

wd, prob:3

f, prob:4

wdf, prob:7

kwdf, prob:13

h, prob:15

kwdfh, prob:28

pkwdfh, prob:53

mpkwdfh, prob:100

**Output 5**

Node:d, Node's prob:2, Node->next:null, Node->left:null, Node->right:null

listHead-->(dummy,0,d)-->(d,2,Null)

Node:f, Node's prob:4, Node->next:null, Node->left:null, Node->right:null

listHead-->(dummy,0,d)-->(d,2,f)-->(f,4,Null)

Node:h, Node's prob:15, Node->next:null, Node->left:null, Node->right:null

listHead-->(dummy,0,d)-->(d,2,f)-->(f,4,h)-->(h,15,Null)

Node:k, Node's prob:6, Node->next:null, Node->left:null, Node->right:null

listHead-->(dummy,0,d)-->(d,2,f)-->(f,4,k)-->(k,6,h)-->(h,15,Null)

Node:m, Node's prob:47, Node->next:null, Node->left:null, Node->right:null

listHead-->(dummy,0,d)-->(d,2,f)-->(f,4,k)-->(k,6,h)-->(h,15,m)-->(m,47,Null)

Node:p, Node's prob:25, Node->next:null, Node->left:null, Node->right:null

listHead-->(dummy,0,d)-->(d,2,f)-->(f,4,k)-->(k,6,h)-->(h,15,p)-->(p,25,m)-->(m,47,Null)

Node:w, Node's prob:1, Node->next:null, Node->left:null, Node->right:null

listHead-->(dummy,0,w)-->(w,1,d)-->(d,2,f)-->(f,4,k)-->(k,6,h)-->(h,15,p)-->(p,25,m)-->(m,47,Null)

Node:wd, Node's prob:3, Node->next:f, Node->left:w, Node->right:d

listHead-->(dummy,0,w)-->(w,1,d)-->(d,2,wd)-->(wd,3,f)-->(f,4,k)-->(k,6,h)-->(h,15,p)-->(p,25,m)-->(m,47,Null)

Node:wdf, Node's prob:7, Node->next:h, Node->left:wd, Node->right:f

listHead-->(dummy,0,w)-->(w,1,d)-->(d,2,wd)-->(wd,3,f)-->(f,4,k)-->(k,6,wdf)-->(wdf,7,h)-->(h,15,p)-->(p,25,m)-->(m,47,Null)

Node:kwdf, Node's prob:13, Node->next:h, Node->left:k, Node->right:wdf

listHead-->(dummy,0,w)-->(w,1,d)-->(d,2,wd)-->(wd,3,f)-->(f,4,k)-->(k,6,wdf)-->(wdf,7,kwdf)-->(kwdf,13,h)-->(h,15,p)-->(p,25,m)-->(m,47,Null)

Node:kwdfh, Node's prob:28, Node->next:m, Node->left:kwdf, Node->right:h

listHead-->(dummy,0,w)-->(w,1,d)-->(d,2,wd)-->(wd,3,f)-->(f,4,k)-->(k,6,wdf)-->(wdf,7,kwdf)-->(kwdf,13,h)-->(h,15,p)-->(p,25,kwdfh)-->(kwdfh,28,m)-->(m,47,Null)

Node:pkwdfh, Node's prob:53, Node->next:null, Node->left:p, Node->right:kwdfh

listHead-->(dummy,0,w)-->(w,1,d)-->(d,2,wd)-->(wd,3,f)-->(f,4,k)-->(k,6,wdf)-->(wdf,7,kwdf)-->(kwdf,13,h)-->(h,15,p)-->(p,25,kwdfh)-->(kwdfh,28,m)-->(m,47,pkwdfh)-->(pkwdfh,53,Null)

Node:mpkwdfh, Node's prob:100, Node->next:null, Node->left:m, Node->right:pkwdfh

listHead-->(dummy,0,w)-->(w,1,d)-->(d,2,wd)-->(wd,3,f)-->(f,4,k)-->(k,6,wdf)-->(wdf,7,kwdf)-->(kwdf,13,h)-->(h,15,p)-->(p,25,kwdfh)-->(kwdfh,28,m)-->(m,47,pkwdfh)-->(pkwdfh,53,mpkwdfh)-->(mpkwdfh,100,Null)