**CS 700-34 HuffmanTreeCoding**

**Student:** Shuhua Song

**Due Date: Submission Data:**

Soft Copy: 02/25/2020 Soft Copy: 02/25/2020

Hard Copy: 02/27/2020 Hard Copy: 02/27/2020

**Algorithm Steps:**

1. **Main()**

Step 0: Open input file ‘inFile’ from argv[1]

Open output file ‘outFile1’ , ‘outFile2’ , ‘outFile3’ from g argv[2], argv[3] and argv[4];

Step 1: create a listHead, set listHead = constructHuffmanLList(InFile, outFile3)

Step 2: printLIst(listHead, outFile2)

Step 3: set the root= constructHuffmanBinTree(listHead, outFile3)

Step 4: preOrderTraversal(Root, outFile2)

Step 5: inOrderTraversal(Root, outFile2)

Step 6: postOrderTraversal(Root, outFile2)

Step 7: constructCharCode(Root, ‘ ‘, outFile1)

Step 8: Close all files;

1. Function **constructHuffmanLList(inFile, outFile)** Steps;

Step 1: listHead <- get a newNode as the dummy (“dummy”, 0, …), which is TreeNode for ListHead to point to;

Step 2: get character, probability from inFile

Step 3: create a newNode by using the data (char, prob,’ ‘, null, null, null)

Step 4: listInsert(listHead, newNode)

Step 5: printList(listHead, outFile)

Step 6: repeat step 2-step4 until the end of inFile

Step 7: return listHead

1. Function **constructHuffmanBinTree(listHead, outFile)** steps

Step 1: create a treeNode called newNode

newNode’s prob is the firstNode’s prob and the secondNode’s prob of the list;

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

newNode’s left = firstNode

newNode’s right = secondNode

newNode’next = null

Step 2: listInsert(listHead, newNode)

Step 3: listHead’s next = the third node after dummy node

Step 4: printList(listHead, outFile)

Step 5: repeat step 1- step 4 until the list only has one node left after the dummy node

Step 6: return listHead’s next

1. Function **findspot(newNode)** Steps

Step 1: set the spot = listHead

Step 2: if spot’s next != null and spot’s next’s data < newNode’s data

spot = spot’next

Step 3: Repeat step2 until condition failed

Step 4: if spot’s next’s data = newNode’s data

spot’s next count++;

spot = null;

Step 5: return spot;

V. Function **listInsert(newNode**) steps

Step 1: Find the proper position ‘spot’ in the linkedlist for this new node to insert

Step 2: If spot != nullptr, then newNode’s next = spot.next, spot.next = newNode

VI. Funciton **printList (index,outFile)**

Step 0: output the completed to outFile: “hashTable[index]->”

Step 1: printSpot = hashTableAry[index]

Step 2: printNode(printSpot)

printSpot=printSpot’s next

Step3: repeat step 2 while printNode!=null

VII. Function **bool isLeaf(node)**

Return node.left == null && node.right==right

VIII. Function **constructCharCode(node, code, outFile)**

If isLeaf(node)

Node’s code = code;

Output node’s chStr and node’s code to outFile

Else

constructCharCode(node’left, code+”0”, outFile)

constructCharCode(node’right, code+”1”, outFile)

VIIII. Function **preOrderTraversal(node, outFile)**

If isLeaf(node)

printNode(node)

Else

printNode(node)

preOrderTraversal(node.left, outFile)

preOrderTraversal(node.right, outFile)

X. Function **inOrderTraversal(root, outFile)**

If isLeaf(node)

printNode(node)

Else

inOrderTraversal(node.left, outFile)

printNode(node)

inOrderTraversal(node.right, outFile)

XI. Function **postOrderTraversal(root, outFile)**

If isLeaf(node)

printNode(node)

Else

inOrderTraversal(node.left, outFile)

inOrderTraversal(node.right, outFile)

printNode(node)

**Code:**

*// project4.cpp*

*// Created by Shuhua Song on 2/22/20.*

*// Copyright © 2020 Shuhua Song. All rights reserved.*

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

class HtreeNode {

public:

string chStr;

int prob;

string code;

HtreeNode \*next;

HtreeNode \*left;

HtreeNode \*right;

HtreeNode(){

chStr = "dummy";

prob = 0;

code = "";

next = nullptr;

left = nullptr;

right = nullptr;

}

HtreeNode(string chStr, int prob, string code, HtreeNode \*next, HtreeNode \*left, HtreeNode \*right){

this->chStr = chStr;

this->prob = prob;

this->code = code;

this->left = left;

this->right = right;

this->next = next;

}

void printNode( ofstream& outFile){

if(this->left != nullptr || this->right != nullptr){

outFile << "(" << this->chStr << "," << this->prob << "," << this->code << "," << "NULL" //there is no next pointer in the Tree

<< "," << this->left->chStr << "," << this->right->chStr << "," << ")->";

}

}

};

class linkedList {

public:

HtreeNode \*listHead;

linkedList(){

listHead = new HtreeNode(); }

void listInsert( HtreeNode \*listHead, HtreeNode \*newNode){

HtreeNode \*spot = findSpot(listHead,newNode);

newNode->next = spot->next;

spot->next = newNode;

}

HtreeNode \*findSpot(HtreeNode \*listHead, HtreeNode \*newNode){

HtreeNode \*spot = listHead;

while(spot->next != nullptr && spot->next->prob <= newNode->prob){

spot = spot->next;

}

return spot;

}

void printList(HtreeNode \*listHead, ofstream& outFile){

outFile << "listHead-> " ;

HtreeNode \*cur = listHead;

while(cur != nullptr && cur->next != nullptr) {

outFile << "(" << cur->chStr << "," << cur->prob << "," << cur->code << "," << cur->next->chStr

<< "," << "NULL" << "," << "NULL)->";

cur = cur->next;

}

outFile << "(" << cur->chStr << "," << cur->prob << "," << cur->code << "," << "NULL"

<< "," << "NULL" << "," << "NULL)->";

outFile << "NULL\n";

}

};

class HuffmanBinaryTree {

public:

HtreeNode \*root;

//linkedList \*list;

HtreeNode \*listHead;

HuffmanBinaryTree(){

root = new HtreeNode();

//list = new linkedList();

}

HtreeNode \*constructHuffmanLList(ifstream& inFile, ofstream& outFile){

linkedList \*list = new linkedList();

//HtreeNode \*listHead = new HtreeNode();

//listHead = list->listHead;

if(inFile.is\_open()){

while(!inFile.eof()){

string chr;

int prob = 0;

inFile >> chr;

inFile >> prob;

if(inFile.eof()) break;

HtreeNode \*newNode = new HtreeNode(chr, prob,"" ,nullptr, nullptr, nullptr);

list->listInsert(list->listHead,newNode);

list->printList(list->listHead, outFile) ;

}

}

return list->listHead;

}

HtreeNode \*constructHuffmanBinTree(HtreeNode \*listHead, ofstream& outFile){

linkedList \*list = new linkedList();

//cout<<list->listHead->chStr<<endl;

//HtreeNode \*listHead = list->listHead;

while(listHead->next != nullptr && listHead->next->next != nullptr){

HtreeNode \*newNode = new HtreeNode();

newNode->prob = listHead->next->prob + listHead->next->next->prob;

newNode->chStr = listHead->next->chStr + listHead->next->next->chStr;

newNode->left = listHead->next;

newNode->right = listHead->next->next;

newNode->next = nullptr;

listHead->next = listHead->next->next->next;//set the list->head first before the newNode insert

list->listInsert( listHead, newNode);

list->printList(listHead, outFile);

}

//root = list->listHead->next;

return listHead->next;

}

void preOrderTraversal(HtreeNode \*root, ofstream& outFile){

if(isLeaf(root)){

//outFile << root->chStr;

return;

}else{

root->printNode(outFile);

preOrderTraversal(root->left, outFile);

preOrderTraversal(root->right, outFile);

}

}

void inOrderTraversal(HtreeNode \*root, ofstream& outFile){

if(root==nullptr) return;

if(isLeaf(root)){

return;

}else{

preOrderTraversal(root->left, outFile);

root->printNode(outFile);

preOrderTraversal(root->right, outFile);

}

}

void postOrderTraversal(HtreeNode \*root, ofstream& outFile){

if(root==nullptr) return;

if(isLeaf(root)){

return;

}else{

preOrderTraversal(root->left, outFile);

preOrderTraversal(root->right, outFile);

root->printNode(outFile);

}

}

void constructCharCode(HtreeNode \*root, string code, ofstream& outFile){

if(isLeaf(root)){

root->code = code;

outFile << root->chStr << " " << root->code << endl;

//cout << root->chStr << " " << root->code << endl;

}else{

root->code = code;

constructCharCode(root->left, root->code+ "0", outFile);

constructCharCode(root->right, root->code+ "1", outFile);

}

}

bool isLeaf(HtreeNode \*root){

if(root->left==nullptr && root->right==nullptr)

return true;

return false;

}

};

int main(int argc, char \*argv[]){

ifstream inFile(argv[1]);

ofstream outFile1(argv[2]); //contain Huffman <char, code> pairs

ofstream outFile2(argv[3]); //

ofstream outFile3(argv[4]);

HuffmanBinaryTree \*tree = new HuffmanBinaryTree();

linkedList \*list = new linkedList();

list->listHead = tree->constructHuffmanLList(inFile,outFile3);

list->printList(list->listHead,outFile2);

tree->root = tree->constructHuffmanBinTree(list->listHead, outFile3);

outFile2 << "PreOrder: " ;

tree->preOrderTraversal(tree->root, outFile2);

outFile2 << "\ninOrder: " ;

tree->inOrderTraversal(tree->root, outFile2);

outFile2 << "\npostOrder: " ;

tree->postOrderTraversal(tree->root, outFile2);

//outFile2 << "\nChar Code table: " ;

tree->constructCharCode(tree->root, "", outFile1);

inFile.close();

outFile1.close();

outFile2.close();

outFile3.close();

return 0;

}

**OutFile 1:**

z 000

a 00100

d 00101

b 00110

c 00111

e 010

y 011

x 1

**OutFile 2:**

listHead-> (dummy,0,,a,NULL,NULL)->(a,2,,d,NULL,NULL)->(d,2,,b,NULL,NULL)->(b,3,,c,NULL,NULL)->(c,3,,z,NULL,NULL)->(z,7,,e,NULL,NULL)->(e,12,,y,NULL,NULL)->(y,13,,x,NULL,NULL)->(x,58,,NULL,NULL,NULL)->NULL

PreOrder: (zadbceyx,100,,NULL,zadbcey,x,)->(zadbcey,42,,NULL,zadbc,ey,)->(zadbc,17,,NULL,z,adbc,)->(adbc,10,,NULL,ad,bc,)->(ad,4,,NULL,a,d,)->(bc,6,,NULL,b,c,)->(ey,25,,NULL,e,y,)->

inOrder: (zadbcey,42,,NULL,zadbc,ey,)->(zadbc,17,,NULL,z,adbc,)->(adbc,10,,NULL,ad,bc,)->(ad,4,,NULL,a,d,)->(bc,6,,NULL,b,c,)->(ey,25,,NULL,e,y,)->(zadbceyx,100,,NULL,zadbcey,x,)->

postOrder: (zadbcey,42,,NULL,zadbc,ey,)->(zadbc,17,,NULL,z,adbc,)->(adbc,10,,NULL,ad,bc,)->(ad,4,,NULL,a,d,)->(bc,6,,NULL,b,c,)->(ey,25,,NULL,e,y,)->(zadbceyx,100,,NULL,zadbcey,x,)->

**OutFile 3:**

listHead-> (dummy,0,,a,NULL,NULL)->(a,2,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,a,NULL,NULL)->(a,2,,b,NULL,NULL)->(b,3,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,a,NULL,NULL)->(a,2,,b,NULL,NULL)->(b,3,,c,NULL,NULL)->(c,3,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,a,NULL,NULL)->(a,2,,d,NULL,NULL)->(d,2,,b,NULL,NULL)->(b,3,,c,NULL,NULL)->(c,3,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,a,NULL,NULL)->(a,2,,d,NULL,NULL)->(d,2,,b,NULL,NULL)->(b,3,,c,NULL,NULL)->(c,3,,e,NULL,NULL)->(e,12,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,a,NULL,NULL)->(a,2,,d,NULL,NULL)->(d,2,,b,NULL,NULL)->(b,3,,c,NULL,NULL)->(c,3,,e,NULL,NULL)->(e,12,,x,NULL,NULL)->(x,58,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,a,NULL,NULL)->(a,2,,d,NULL,NULL)->(d,2,,b,NULL,NULL)->(b,3,,c,NULL,NULL)->(c,3,,e,NULL,NULL)->(e,12,,y,NULL,NULL)->(y,13,,x,NULL,NULL)->(x,58,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,a,NULL,NULL)->(a,2,,d,NULL,NULL)->(d,2,,b,NULL,NULL)->(b,3,,c,NULL,NULL)->(c,3,,z,NULL,NULL)->(z,7,,e,NULL,NULL)->(e,12,,y,NULL,NULL)->(y,13,,x,NULL,NULL)->(x,58,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,b,NULL,NULL)->(b,3,,c,NULL,NULL)->(c,3,,ad,NULL,NULL)->(ad,4,,z,NULL,NULL)->(z,7,,e,NULL,NULL)->(e,12,,y,NULL,NULL)->(y,13,,x,NULL,NULL)->(x,58,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,ad,NULL,NULL)->(ad,4,,bc,NULL,NULL)->(bc,6,,z,NULL,NULL)->(z,7,,e,NULL,NULL)->(e,12,,y,NULL,NULL)->(y,13,,x,NULL,NULL)->(x,58,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,z,NULL,NULL)->(z,7,,adbc,NULL,NULL)->(adbc,10,,e,NULL,NULL)->(e,12,,y,NULL,NULL)->(y,13,,x,NULL,NULL)->(x,58,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,e,NULL,NULL)->(e,12,,y,NULL,NULL)->(y,13,,zadbc,NULL,NULL)->(zadbc,17,,x,NULL,NULL)->(x,58,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,zadbc,NULL,NULL)->(zadbc,17,,ey,NULL,NULL)->(ey,25,,x,NULL,NULL)->(x,58,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,zadbcey,NULL,NULL)->(zadbcey,42,,x,NULL,NULL)->(x,58,,NULL,NULL,NULL)->NULL

listHead-> (dummy,0,,zadbceyx,NULL,NULL)->(zadbceyx,100,,NULL,NULL,NULL)->NULL