CS 323\_33 Programming Language: C++

Project #4 Huffman Encoding

Andres Quintero

Due Date:

Soft copy: 2/25/2020

Hard copy: 2/27/2020

Submission:

Soft copy: 2/25/2020

Hard copy: 2/27/2020

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Main():

Step 0: inFile 🡨 open input file from argv[1]

outFile1, outFile2, outFile3 🡨 open from argv[2], argv[3], argv[4]

Step 1: listHead 🡨 constructHuffmanLList (inFile, outFile3)

Step 2: printList(listHead, outFile2)

Step 3: Root 🡨 constructHuffmanBinTree (listHead, outFile3)

Step 4: preOrderTraversal (Root, outFile2)

Step 5: inOrderTraversal (Root, outFile2)

Step 6: postOrderTraversal (Root, outFile2)

Step 7: constructCharCode (Root, ‘’, outFile1) // ‘’ is an empty string

Step 8: close all files

**Source code:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

//Data Structures

class HTreeNode{

public:

string charStr;

int prob;

string code;

HTreeNode\* left;

HTreeNode\* right;

HTreeNode\* next;

// Constructor

HTreeNode(string charStr, int prob, string code, HTreeNode\* left, HTreeNode\* right, HTreeNode\* next){

this->charStr = charStr;

this->prob = prob;

this->code = code;

this->left = left;

this->right = right;

this->next = next;

};

};

class linkedList {

public:

HTreeNode\* listHead;

linkedList(){

this->listHead = NULL;

};

};

class HuffmanBinaryTree {

public:

HTreeNode\* root;

HuffmanBinaryTree(){

this->root = NULL;

};

};

// Prototypes

void printNode(HTreeNode\* node, fstream& outFile);

void listInsert(HTreeNode\* listHead, HTreeNode\* newNode);

HTreeNode\* findSpot(HTreeNode\* listHead, HTreeNode\* newNode);

void printList(HTreeNode\* listHead, fstream& outFile);

HTreeNode\* constructHuffmanLL(fstream& inFile, fstream& outFile);

HTreeNode\* constructHuffmanBinTree(HTreeNode\* listHead, fstream& outFile);

bool isLeaf(HTreeNode\* node);

void preOrderTraversal(HTreeNode\* node, fstream& outFile);

void inOrderTraversal(HTreeNode\* node, fstream& outFile);

void postOrderTraversal(HTreeNode\* node, fstream& outFile);

void constructCharCode(HTreeNode\* node, string code, fstream& outFile);

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

// Opening files

fstream inFile(argv[1]);

fstream outFile1(argv[2], fstream::out);

fstream outFile2(argv[3], fstream::out);

fstream outFile3(argv[4], fstream::out);

linkedList\* list = new linkedList();

HuffmanBinaryTree\* tree = new HuffmanBinaryTree();

list->listHead = constructHuffmanLL(inFile, outFile3);

printList(list->listHead, outFile2);

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

preOrderTraversal(tree->root, outFile2);

outFile2 << endl;

inOrderTraversal(tree->root, outFile2);

outFile2 << endl;

postOrderTraversal(tree->root, outFile2);

// char and value seprated by a single space

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

// Closing Files

inFile.close();

outFile1.close();

outFile2.close();

outFile3.close();

}

//Functions

void constructCharCode(HTreeNode\* node, string code, fstream& outFile){

if(isLeaf(node)){

node->code = code;

outFile << node->charStr << " " <<node->code << endl;

} else {

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

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

}

}

void postOrderTraversal(HTreeNode\* node, fstream& outFile){

if (isLeaf(node)){

printNode(node, outFile);

} else {

postOrderTraversal(node->left, outFile);

postOrderTraversal(node->right, outFile);

printNode(node, outFile);

}

}

void inOrderTraversal(HTreeNode\* node, fstream& outFile){

if (isLeaf(node)){

printNode(node, outFile);

} else {

inOrderTraversal(node->left, outFile);

printNode(node, outFile);

inOrderTraversal(node->right, outFile);

}

}

void preOrderTraversal(HTreeNode\* node, fstream& outFile){

if (isLeaf(node)){

printNode(node, outFile);

} else {

printNode(node, outFile);

preOrderTraversal(node->left, outFile);

preOrderTraversal(node->right, outFile);

}

}

bool isLeaf(HTreeNode\* node){

if(node->left == NULL && node->right == NULL){

return true;

} else {

return false;

}

}

HTreeNode\* constructHuffmanBinTree(HTreeNode\* listHead, fstream& outFile){

while(listHead->next->next != NULL){//only one left

HTreeNode\* firstNode = listHead->next;

HTreeNode\* secondNode = listHead->next->next;

HTreeNode\* newNode = new HTreeNode(firstNode->charStr + secondNode->charStr, firstNode->prob + secondNode->prob, "", firstNode, secondNode, NULL);

listInsert(listHead, newNode);

listHead->next = listHead->next->next->next;

printList(listHead, outFile);

}

return listHead->next;

}

HTreeNode\* constructHuffmanLL(fstream& inFile, fstream& outFile){

HTreeNode\* listHead = new HTreeNode("dummy", 0, "", NULL, NULL, NULL);

string charStr;

int prob;

while(inFile >> charStr){

inFile >> prob;

HTreeNode\* newNode = new HTreeNode(charStr, prob, "", NULL, NULL, NULL);

listInsert(listHead, newNode);

printList(listHead, outFile);

}

return listHead;

}

void printList(HTreeNode\* node, fstream& outFile) {

outFile << "listHead ->";

while(node != NULL){

printNode(node, outFile);

node = node->next;

}

outFile << endl;

}

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

HTreeNode\* Spot = listHead;

while(Spot->next != NULL && Spot->next->prob < newNode->prob){

Spot = Spot->next;

}

return Spot;

}

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

HTreeNode\* Spot = findSpot(listHead, newNode);

if(Spot != NULL){

newNode->next = Spot->next;

Spot->next = newNode;

}

}

void printNode(HTreeNode\* node, fstream& outFile){

if(node->next == NULL){

outFile << "(" << node->charStr << ", "

<< node->prob << ", "

<< node->code << ", "

<< (node->next ? node->next->charStr : "NULL") << ", "

<< (node->left ? node->left->charStr : "NULL") << ", "

<< (node->right ? node->right->charStr : "NULL") << ") -> NULL";

} else {

outFile << "(" << node->charStr << ", "

<< node->prob << ", "

<< node->code << ", "

<< (node->next ? node->next->charStr : "NULL") << ", "

<< (node->left ? node->left->charStr : "NULL") << ", "

<< (node->right ? node->right->charStr : "NULL") << ") ->";

}

}

**outFile1: Huffman code pairs**

A screenshot of a cell phone

Description automatically generated

**outFile1: LinkedList,**

**pre-order, in-order, post-order**

A screenshot of a cell phone

Description automatically generated

**outFile3: LinkList Build**

A screenshot of a computer

Description automatically generated