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

**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**

**#include <iostream>**

**#include <fstream>**

**#include <string>**

**#include <sstream>**

**using namespace std;**

**class listBinTreeNode {**

**public:**

**string chStr;**

**int prob;**

**string code;**

**listBinTreeNode \*left;**

**listBinTreeNode \*right;**

**listBinTreeNode \*next;**

**//constructor**

**listBinTreeNode() {**

**chStr = "dummy";**

**prob = 0;**

**code = "";**

**next = NULL;**

**left = NULL;**

**right = NULL;**

**}**

**//constructor**

**listBinTreeNode(string s, int i){**

**chStr = s;**

**prob = i;**

**code = "";**

**next = NULL;**

**left = NULL;**

**right = NULL;**

**}**

**void printNode(){**

**cout<< "Node:" + chStr + ", Node's prob:" << prob;**

**if(next != NULL) {**

**cout<< ", Node->next:" + next->chStr;**

**}else {**

**cout<<", Node->next:null";**

**}**

**if(left != NULL) {**

**cout<< ", Node->left:" + left->chStr;**

**}else {**

**cout<<", Node->left:null";**

**}**

**if(right != NULL) {**

**cout<< ", Node->right:" + right->chStr;**

**}else {**

**cout<<", Node->right:null";**

**}**

**cout<<endl;**

**}**

**}; //class**

**class HuffmanLinkedList {**

**public:**

**listBinTreeNode \*head;**

**HuffmanLinkedList(){**

**head = new listBinTreeNode();**

**}**

**//destructor**

**~HuffmanLinkedList(){**

**listBinTreeNode \*walker = head;**

**while(walker != NULL){**

**walker = walker->next;**

**delete head;**

**head = walker;**

**}**

**}**

**void constructHuffmanLList(char\* in){**

**string s1 = "";**

**int i1 = 0;**

**listBinTreeNode \*newNode = NULL;**

**//open file**

**ifstream inFile;**

**inFile.open(in);**

**string s;**

**while(inFile>>s){**

**s1 = s;**

**if(inFile>>s){**

**istringstream ( s ) >> i1;**

**}**

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

**newNode->printNode();**

**listInsert((findSpot(newNode->prob)),newNode);**

**printList();**

**}**

**inFile.close();**

**}**

**int findSpot(int prob) {**

**int i=0;**

**listBinTreeNode \*walker = head;**

**while(walker->next != NULL && (prob > walker->next->prob)) {**

**walker = walker->next;**

**i++;**

**}**

**return i;**

**}**

**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;**

**}**

**bool isEmpty(){**

**bool b = false;**

**if(head->next == NULL) b = true;**

**return b;**

**}**

**void printList() {**

**listBinTreeNode \*walker =head;**

**cout<< "listHead-->(" + walker->chStr + "," << walker->prob << ",";**

**while((walker = walker->next) != NULL) {**

**cout << walker->chStr + ")-->("+ walker->chStr + "," << walker->prob << ",";**

**}**

**cout << "Null)"<<endl;**

**}**

**}; //class**

**class HuffmanBinaryTree{**

**public:**

**listBinTreeNode \*root;**

**HuffmanBinaryTree(){**

**root = NULL;**

**}**

**HuffmanBinaryTree(listBinTreeNode \*t){**

**root = t;**

**}**

**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;**

**}**

**//tree is deleted by deleting Linked List**

**~HuffmanBinaryTree(){**

**}**

**bool isLeaf(listBinTreeNode \*t){**

**bool b = false;**

**if(t->left == NULL && t->right == NULL) b = true;**

**return b;**

**}**

**void constructCharCode(){**

**constructCharCode(root,"");**

**}**

**void preOrderTravesal(){**

**cout<<"preOrderTravesal:"<<endl;**

**preOrderTravesal(root);**

**}**

**void inOrderTravesal(){**

**cout<<"inOrderTravesal:"<<endl;**

**inOrderTravesal(root);**

**}**

**void postOrderTravesal(){**

**cout<<"postOrderTravesal:"<<endl;**

**postOrderTravesal(root);**

**}**

**/\***

**void deleteTree(){**

**if(root!=NULL){**

**deleteTree(root);**

**}**

**}**

**\*/**

**private:**

**void constructCharCode(listBinTreeNode \*t,string s){**

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

**t->code = s;**

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

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

**cout<< t->chStr + ", code:" + t->code <<endl;**

**}**

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

**}**

**}**

**void inOrderTravesal(listBinTreeNode \*t){**

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

**inOrderTravesal(t->left);**

**cout<< t->chStr + ", prob:"<< t->prob <<endl;**

**inOrderTravesal(t->right);**

**}**

**return;**

**}**

**void preOrderTravesal(listBinTreeNode \*t){**

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

**cout<< t->chStr + ", prob:"<< t->prob <<endl;**

**preOrderTravesal(t->left);**

**preOrderTravesal(t->right);**

**}**

**return;**

**}**

**void postOrderTravesal(listBinTreeNode \*t){**

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

**postOrderTravesal(t->left);**

**postOrderTravesal(t->right);**

**cout<< t->chStr + ", prob:"<< t->prob <<endl;**

**}**

**return;**

**}**

**/\***

**void deleteTree(listBinTreeNode \*t){**

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

**deleteTree(t->left);**

**deleteTree(t->right);**

**delete(t);**

**}**

**return;**

**}**

**\*/**

**};**

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

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

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

**if(argv[1]==NULL) {**

**cout<<"no parameter"<<endl;**

**return 0;**

**}**

**//open file and do check**

**ifstream inFile;**

**inFile.open(argv[1]);**

**if(!inFile.is\_open()){**

**cout<<"cant find file"<<endl;**

**return 0;**

**}else {**

**inFile.close();**

**}**

**//im going to do redirect for cout**

**streambuf \*console = cout.rdbuf();**

**ofstream out1;**

**out1.open(argv[2]);**

**ofstream out2;**

**out2.open(argv[3]);**

**ofstream out3;**

**out3.open(argv[4]);**

**ofstream out4;**

**out4.open(argv[5]);**

**ofstream out5;**

**out5.open(argv[6]);**

**cout.rdbuf(out5.rdbuf());**

**LL->constructHuffmanLList(argv[1]);**

**tree->constructHuffmanBinTree(LL);**

**cout.rdbuf(out1.rdbuf());**

**tree->constructCharCode();**

**cout.rdbuf(out2.rdbuf());**

**tree->preOrderTravesal();**

**cout.rdbuf(out3.rdbuf());**

**tree->inOrderTravesal();**

**cout.rdbuf(out4.rdbuf());**

**tree->postOrderTravesal();**

**cout.rdbuf(console);**

**cout<<"done"<<endl;**

**out1.close();**

**out2.close();**

**out3.close();**

**out4.close();**

**out5.close();**

**//tree is deleted by deleting linked list**

**delete LL;**

**return 0;**

**}**

**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)