323.25 Project 3: Huffman Tree Coding

Student: Jingshi Liu

Due Date: 3/05/2022

Algorithm Steps for the Project:

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, outFile2)

Step 2: printList (listHead, outFile1)

Step 3: Root <— constructHuffmanBinTree (listHead, outFile2)

Step 4: constructCharCode (Root, ‘’, outFile1) // ‘’ is an empty string

Step 5: preOrderTraversal (Root, outFile1)

Step 6: inOrderTraversal (Root, outFile1)

Step 7: postOrderTraversal (Root, outFile1)

Step 8: close all files

Source Code:

```
//
//  main.cpp
//  CS320_Project3
//
//  Created by Jingshi Liu on 2/28/22.
//

#include <iostream>
#include <fstream>
using namespace std;

class HuffmanTreeNode{
public:
    string char_Str;
```

```
    int prob;
    string bin_code;
    HuffmanTreeNode *left;
    HuffmanTreeNode *right;
    HuffmanTreeNode *next;

    HuffmanTreeNode(string char_str, int prob, HuffmanTreeNode
*left, HuffmanTreeNode *right, HuffmanTreeNode *next){
        this->left = left;
        this->right = right;
        this->next = next;
        this->char_Str = char_str;
        this->prob = prob;
    }

    // print char, prob, bin_code, next_char, left_char,
right_char
    string print_huffman_node(){
        string node_info = "(" + char_Str + ", " +
to_string(prob) + ", ";

        if(this->bin_code.length() == 0){
            node_info += " ";
        }else{
            node_info += this->bin_code + ", ";
        }

        if(this->next == NULL){
            node_info += "NULL, ";
        }else{
            node_info += next->char_Str + ", ";
        }

        if(this->left == NULL){
            node_info += "NULL, ";
        }else{
            node_info += left->char_Str + ", ";
        }

        if(this->right == NULL){
            node_info += "NULL)->";
        }else{
            node_info += right->char_Str + ")->";
        }

        return node_info;
```

```
    }
};

class HuffmanTree{
public:
    HuffmanTreeNode* listhead;
    HuffmanTreeNode* root;

    HuffmanTree(){
        this->listhead = new HuffmanTreeNode("dummy", 0, NULL,
NULL, NULL);
        this->root = listhead;
    }

    HuffmanTreeNode* findSpot(HuffmanTreeNode* new_node){
        HuffmanTreeNode* spot = this->listhead;

        while(spot->next != NULL && spot->next->prob < new_node-
>prob){
            spot = spot->next;
        }
        return spot;
    }

    void listInsert(HuffmanTreeNode* new_node){
        HuffmanTreeNode* spot = findSpot(new_node);
        new_node->next = spot->next;
        spot->next = new_node;
    }

    HuffmanTreeNode* construct_huffman_linked_list(ifstream&
inFile, ofstream& outFile){
        string character = "";
        int prob;
        while(inFile >> character && inFile >> prob){
            listInsert(new HuffmanTreeNode(character, prob,
NULL, NULL, NULL));
            print_list(outFile);
            outFile << "\n\n";
        }
        return this->listhead;
    }

    HuffmanTreeNode* construct_huffman_tree(ofstream& outFile){
        while(listhead->next != NULL && listhead->next->next!=
NULL){
```

```
            HuffmanTreeNode* new_node = new HuffmanTreeNode(
                                    listhead->next->char_Str +
listhead->next->next->char_Str,
                                    listhead->next->prob + listhead-
>next->next->prob,
                                    listhead->next,
                                    listhead->next->next,
                                    NULL);
            listInsert(new_node);
            listhead->next = listhead->next->next->next;
            new_node->left->next = nullptr;
            new_node->right->next = nullptr;
            print_list(outFile);
            outFile << "\n\n";
        }
        root = listhead->next;
        return root;
    }

    void construct_char_code(HuffmanTreeNode* current_node,
string code_of_parent, ofstream& outFile){
        if(isLeaf(current_node)){
            current_node->bin_code = code_of_parent;
            outFile << current_node->char_Str + " " +
current_node->bin_code<< endl;
        }else{
            construct_char_code(current_node->left,
code_of_parent + "0", outFile);
            construct_char_code(current_node->right,
code_of_parent + "1", outFile);
        }
    }

    void preorder_traversal(HuffmanTreeNode* current_node,
ofstream& outFile){
        if(isLeaf(current_node))
            outFile << current_node->print_huffman_node();
        else{
            outFile << current_node->print_huffman_node();
            preorder_traversal(current_node->left, outFile);
            preorder_traversal(current_node->right, outFile);
        }
    }

    void inorder_traversal(HuffmanTreeNode* current_node,
ofstream& outFile){
```

```
        if(isLeaf(current_node))
            outFile << current_node->print_huffman_node();
        else{
            inorder_traversal(current_node->left, outFile);
            outFile << current_node->print_huffman_node();
            inorder_traversal(current_node->right, outFile);
        }
    }

    void postorder_traversal(HuffmanTreeNode* current_node,
ofstream& outFile){
        if(isLeaf(current_node))
            outFile << current_node->print_huffman_node();
        else{
            postorder_traversal(current_node->left, outFile);
            postorder_traversal(current_node->right, outFile);
            outFile << current_node->print_huffman_node();
        }
    }

    void print_list(ofstream& outFile){
        HuffmanTreeNode* head = listhead;

        while(head != NULL){
            outFile << head->print_huffman_node();
            head = head->next;
        }
        outFile << endl;
    }

    bool isLeaf(HuffmanTreeNode* node){
        return node->left == NULL && node->right == NULL;
    }


};


int main(int argc, const char * argv[]) {
    ifstream inFile;
    ofstream outFile1, outFile2;
    if(argc != 4){
        cout<< "Please pass in one data file and 2 output
file"<<endl;
    }else{
        inFile.open(argv[1]);
```

```
        outFile1.open(argv[2]);
        outFile2.open(argv[3]);
    }
    HuffmanTree* huffman_tree = new HuffmanTree();
    outFile2 << "LL\n\n";
    huffman_tree->construct_huffman_linked_list(inFile,
outFile2);
    outFile1 << "\n\nHuffman Linked List\n\n";
    huffman_tree->print_list(outFile1);

    outFile2 << "\n\nTree\n";
    huffman_tree->construct_huffman_tree(outFile2);

    outFile1 << "\n\nBinary Code of Each Character\n\n";
    huffman_tree->construct_char_code(huffman_tree->root, "",
outFile1);

    outFile1 << "\n\nPreorder Traversal\n\n";
    huffman_tree->preorder_traversal(huffman_tree->root,
outFile1);

    outFile1 << "\n\nInorder Traversal\n\n";
    huffman_tree->inorder_traversal(huffman_tree->root,
outFile1);

    outFile1 << "\n\nPostorder Traversal\n\n";
    huffman_tree->postorder_traversal(huffman_tree->root,
outFile1);

    inFile.close();
    outFile1.close();
    outFile2.close();

    return 0;
}
```

Note: A few titles and \n were added in outFiles for better readability.

Output:

Data 1, outFile1, and outFile2

Data 1:
a 40
o 5
e 20
h 5
i 10
g 15
d 5

outFile1 of data 1:

Huffman Linked List

(dummy, 0, d, NULL, NULL)->(d, 5, h, NULL, NULL)->(h, 5, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, g, NULL, NULL)->(g, 15, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

Binary Code of Each Character

a 0
i 100
o 1010
d 10110
h 10111
g 110
e 111

Preorder Traversal

(aiodhge, 100, NULL, a, iodhge)->(a, 40, 0, NULL, NULL, NULL)->(iodhge, 60, NULL, iodh, ge)->(iodh, 25, NULL, i, odh)->(i, 10, 100, NULL, NULL, NULL)->(odh, 15, NULL, o, dh)->(o, 5, 1010, NULL, NULL, NULL)->(dh, 10, NULL, d, h)->(d, 5, 10110, NULL, NULL, NULL)->(h, 5, 10111, NULL, NULL, NULL)->(ge, 35, NULL, g, e)->(g, 15, 110, NULL, NULL, NULL)->(e, 20, 111, NULL, NULL, NULL)->

Inorder Traversal

(a, 40, 0, NULL, NULL, NULL)->(aiodhge, 100, NULL, a, iodhge)->(i, 10, 100, NULL, NULL, NULL)->(iodh, 25, NULL, i, odh)->(o, 5, 1010, NULL, NULL, NULL)->(odh, 15, NULL, o, dh)->(d, 5, 10110, NULL, NULL, NULL)->(dh, 10, NULL, d, h)->(h, 5, 10111, NULL, NULL, NULL)->(iodhge, 60, NULL, iodh, ge)->(g, 15, 110, NULL, NULL, NULL)->(ge, 35, NULL, g, e)->(e, 20, 111, NULL, NULL, NULL)->

Postorder Traversal

(a, 40, 0, NULL, NULL, NULL)->(i, 10, 100, NULL, NULL, NULL)->(o, 5, 1010, NULL, NULL, NULL)->(d, 5, 10110, NULL, NULL, NULL)->(h, 5, 10111, NULL, NULL, NULL)->(dh, 10, NULL, d, h)->(odh, 15, NULL, o, dh)->(iodh, 25, NULL, i, odh)->(g, 15, 110, NULL, NULL, NULL)->(e, 20, 111, NULL, NULL, NULL)->(ge, 35, NULL, g, e)->(iodhge, 60, NULL, iodh, ge)->(aiodhge, 100, NULL, a, iodhge)->

outFile2 of data 1:

LL

(dummy, 0, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, o, NULL, NULL)->(o, 5, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, o, NULL, NULL)->(o, 5, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, h, NULL, NULL)->(h, 5, o, NULL, NULL)->(o, 5, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, h, NULL, NULL)->(h, 5, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, h, NULL, NULL)->(h, 5, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, g, NULL, NULL)->(g, 15, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, d, NULL, NULL)->(d, 5, h, NULL, NULL)->(h, 5, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, g, NULL, NULL)->(g, 15, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

Tree
(dummy, 0, o, NULL, NULL)->(o, 5, dh, NULL, NULL)->(dh, 10, i, d, h)->(i, 10, g, NULL, NULL)->(g, 15, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, i, NULL, NULL)->(i, 10, odh, NULL, NULL)->(odh, 15, g, o, dh)->(g, 15, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, g, NULL, NULL)->(g, 15, e, NULL, NULL)->(e, 20, iodh, NULL, NULL)->(iodh, 25, a, i, odh)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, iodh, NULL, NULL)->(iodh, 25, ge, i, odh)->(ge, 35, a, g, e)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, a, NULL, NULL)->(a, 40, iodhge, NULL, NULL)->(iodhge, 60, NULL, iodh, ge)->

(dummy, 0, aiodhge, NULL, NULL)->(aiodhge, 100, NULL, a, iodhge)->

Data 2, outFile1, and outFile2

Data 2:

a 50
z 2
i 10
o 5
u 3
e 30

outFile1 of data 2:

Huffman Linked List

(dummy, 0, z, NULL, NULL)->(z, 2, u, NULL, NULL)->(u, 3, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 30, a, NULL, NULL)->(a, 50, NULL, NULL, NULL)->

Binary Code of Each Character

z 00000
u 00001
o 0001
i 001
e 01
a 1

Preorder Traversal

(zuoiea, 100, NULL, zuoie, a)->(zuoie, 50, NULL, zuoi, e)->(zuoi, 20, NULL, zuo, i)->(zuo, 10, NULL, zu, o)->(zu, 5, NULL, z, u)->(z, 2, 00000, NULL, NULL, NULL)->(u, 3, 00001, NULL,

NULL, NULL)->(o, 5, 0001, NULL, NULL, NULL)->(i, 10, 001, NULL, NULL, NULL)->(e, 30, 01, NULL, NULL, NULL)->(a, 50, 1, NULL, NULL, NULL)->

Inorder Traversal

(z, 2, 00000, NULL, NULL, NULL)->(zu, 5, NULL, z, u)->(u, 3, 00001, NULL, NULL, NULL)->(zuo, 10, NULL, zu, o)->(o, 5, 0001, NULL, NULL, NULL)->(zuoi, 20, NULL, zuo, i)->(i, 10, 001, NULL, NULL, NULL)->(zuoie, 50, NULL, zuoi, e)->(e, 30, 01, NULL, NULL, NULL)->(zuoiea, 100, NULL, zuoie, a)->(a, 50, 1, NULL, NULL, NULL)->

Postorder Traversal

(z, 2, 00000, NULL, NULL, NULL)->(u, 3, 00001, NULL, NULL, NULL)->(zu, 5, NULL, z, u)->(o, 5, 0001, NULL, NULL, NULL)->(zuo, 10, NULL, zu, o)->(i, 10, 001, NULL, NULL, NULL)->(zuoi, 20, NULL, zuo, i)->(e, 30, 01, NULL, NULL, NULL)->(zuoie, 50, NULL, zuoi, e)->(a, 50, 1, NULL, NULL, NULL)->(zuoiea, 100, NULL, zuoie, a)->

outFile2 of data 2:

LL

(dummy, 0, a, NULL, NULL)->(a, 50, NULL, NULL, NULL)->

(dummy, 0, z, NULL, NULL)->(z, 2, a, NULL, NULL)->(a, 50, NULL, NULL, NULL)->

(dummy, 0, z, NULL, NULL)->(z, 2, i, NULL, NULL)->(i, 10, a, NULL, NULL)->(a, 50, NULL, NULL, NULL)->

(dummy, 0, z, NULL, NULL)->(z, 2, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, a, NULL, NULL)->(a, 50, NULL, NULL, NULL)->

(dummy, 0, z, NULL, NULL)->(z, 2, u, NULL, NULL)->(u, 3, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, a, NULL, NULL)->(a, 50, NULL, NULL, NULL)->

(dummy, 0, z, NULL, NULL)->(z, 2, u, NULL, NULL)->(u, 3, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 30, a, NULL, NULL)->(a, 50, NULL, NULL, NULL)->

Tree
(dummy, 0, zu, NULL, NULL)->(zu, 5, o, z, u)->(o, 5, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 30, a, NULL, NULL)->(a, 50, NULL, NULL, NULL)->

(dummy, 0, zuo, NULL, NULL)->(zuo, 10, i, zu, o)->(i, 10, e, NULL, NULL)->(e, 30, a, NULL, NULL)->(a, 50, NULL, NULL, NULL)->

(dummy, 0, zuoi, NULL, NULL)->(zuoi, 20, e, zuo, i)->(e, 30, a, NULL, NULL)->(a, 50, NULL, NULL, NULL)->

(dummy, 0, zuoie, NULL, NULL)->(zuoie, 50, a, zuoi, e)->(a, 50, NULL, NULL, NULL)->

(dummy, 0, zuoiea, NULL, NULL)->(zuoiea, 100, NULL, zuoie, a)->

Data 3, outFile1, and outFile2

Data 3:

a 40
d 3
e 20
h 1
i 10
g 1
o 5
s 2
t 5
c 1
m 1
v 1
b 10

outFile1 for data 3:

Huffman Linked List

(dummy, 0, v, NULL, NULL)->(v, 1, m, NULL, NULL)->(m, 1, c, NULL, NULL)->(c, 1, g, NULL, NULL)->(g, 1, h, NULL, NULL)->(h, 1, s, NULL, NULL)->(s, 2, d, NULL, NULL)->(d, 3, t, NULL, NULL)->(t, 5, o, NULL, NULL)->(o, 5, b, NULL, NULL)->(b, 10, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

Binary Code of Each Character

a 0
i 100
o 1010
h 101100
c 1011010
g 1011011
d 10111
v 1100000
m 1100001
s 110001
t 11001
b 1101
e 111

Preorder Traversal

(aiohcgdvmstbe, 100, NULL, a, iohcgdvmstbe)->(a, 40, 0, NULL, NULL, NULL)->(iohcgdvmstbe, 60, NULL, iohcgd, vmstbe)->(iohcgd, 21, NULL, i, ohcgd)->(i, 10, 100, NULL, NULL, NULL)->(ohcgd, 11, NULL, o, hcgd)->(o, 5, 1010, NULL, NULL, NULL)->(hcgd, 6, NULL, hcg, d)->(hcg, 3, NULL, h, cg)->(h, 1, 101100, NULL, NULL, NULL)->(cg, 2, NULL, c, g)->(c, 1, 1011010, NULL, NULL, NULL)->(g, 1, 1011011, NULL, NULL, NULL)->(d, 3, 10111, NULL, NULL, NULL)->(vmstbe, 39, NULL, vmstb, e)->(vmstb, 19, NULL, vmst, b)->(vmst, 9, NULL, vms, t)->(vms, 4, NULL, vm, s)->(vm, 2, NULL, v, m)->(v, 1, 1100000, NULL, NULL, NULL)->(m, 1, 1100001, NULL, NULL, NULL)->(s, 2, 110001, NULL, NULL, NULL)->(t, 5, 11001, NULL, NULL, NULL)->(b, 10, 1101, NULL, NULL, NULL)->(e, 20, 111, NULL, NULL, NULL)->

Inorder Traversal

(a, 40, 0, NULL, NULL, NULL)->(aiohcgdvmstbe, 100, NULL, a, iohcgdvmstbe)->(i, 10, 100, NULL, NULL, NULL)->(iohcgd, 21, NULL, i, ohcgd)->(o, 5, 1010, NULL, NULL, NULL)->(ohcgd, 11, NULL, o, hcgd)->(h, 1, 101100, NULL, NULL, NULL)->(hcg, 3, NULL, h, cg)->(c, 1, 1011010, NULL, NULL, NULL)->(cg, 2, NULL, c, g)->(g, 1, 1011011, NULL, NULL, NULL)->(hcgd, 6, NULL, hcg, d)->(d, 3, 10111, NULL, NULL, NULL)->(iohcgdvmstbe, 60, NULL, iohcgd, vmstbe)->(v, 1, 1100000, NULL, NULL, NULL)->(vm, 2, NULL, v, m)->(m, 1, 1100001, NULL, NULL, NULL)->(vms, 4, NULL, vm, s)->(s, 2, 110001, NULL, NULL, NULL)->(vmst, 9, NULL, vms, t)->(t, 5, 11001, NULL, NULL, NULL)->(vmstb, 19, NULL, vmst, b)->(b, 10, 1101, NULL, NULL, NULL)->(vmstbe, 39, NULL, vmstb, e)->(e, 20, 111, NULL, NULL, NULL)->

Postorder Traversal

(a, 40, 0, NULL, NULL, NULL)->(i, 10, 100, NULL, NULL, NULL)->(o, 5, 1010, NULL, NULL, NULL)->(h, 1, 101100, NULL, NULL, NULL)->(c, 1, 1011010, NULL, NULL, NULL)->(g, 1, 1011011, NULL, NULL, NULL)->(cg, 2, NULL, c, g)->(hcg, 3, NULL, h, cg)->(d, 3, 10111, NULL, NULL, NULL)->(hcgd, 6, NULL, hcg, d)->(ohcgd, 11, NULL, o, hcgd)->(iohcgd, 21, NULL, i, ohcgd)->(v, 1, 1100000, NULL, NULL, NULL)->(m, 1, 1100001, NULL, NULL, NULL)->(vm, 2, NULL, v, m)->(s, 2, 110001, NULL, NULL, NULL)->(vms, 4, NULL, vm, s)->(t, 5, 11001, NULL, NULL, NULL)->(vmst, 9, NULL, vms, t)->(b, 10, 1101, NULL, NULL, NULL)->(vmstb, 19, NULL, vmst, b)->(e, 20, 111, NULL, NULL, NULL)->(vmstbe, 39, NULL, vmstb,

e)->(iohcgdvmstbe, 60, NULL, iohcgd, vmstbe)->(aiohcgdvmstbe, 100, NULL, a, iohcgdvmstbe)->

outFile 2 for data 3:

LL

(dummy, 0, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, d, NULL, NULL)->(d, 3, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, d, NULL, NULL)->(d, 3, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, h, NULL, NULL)->(h, 1, d, NULL, NULL)->(d, 3, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, h, NULL, NULL)->(h, 1, d, NULL, NULL)->(d, 3, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, g, NULL, NULL)->(g, 1, h, NULL, NULL)->(h, 1, d, NULL, NULL)->(d, 3, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, g, NULL, NULL)->(g, 1, h, NULL, NULL)->(h, 1, d, NULL, NULL)->(d, 3, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, g, NULL, NULL)->(g, 1, h, NULL, NULL)->(h, 1, s, NULL, NULL)->(s, 2, d, NULL, NULL)->(d, 3, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, g, NULL, NULL)->(g, 1, h, NULL, NULL)->(h, 1, s, NULL, NULL)->(s, 2, d, NULL, NULL)->(d, 3, t, NULL, NULL)->(t, 5, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, c, NULL, NULL)->(c, 1, g, NULL, NULL)->(g, 1, h, NULL, NULL)->(h, 1, s, NULL, NULL)->(s, 2, d, NULL, NULL)->(d, 3, t, NULL, NULL)->(t, 5, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, m, NULL, NULL)->(m, 1, c, NULL, NULL)->(c, 1, g, NULL, NULL)->(g, 1, h, NULL, NULL)->(h, 1, s, NULL, NULL)->(s, 2, d, NULL, NULL)->(d, 3, t, NULL, NULL)->(t, 5, o, NULL,

NULL)->(o, 5, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, v, NULL, NULL)->(v, 1, m, NULL, NULL)->(m, 1, c, NULL, NULL)->(c, 1, g, NULL, NULL)->(g, 1, h, NULL, NULL)->(h, 1, s, NULL, NULL)->(s, 2, d, NULL, NULL)->(d, 3, t, NULL, NULL)->(t, 5, o, NULL, NULL)->(o, 5, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, v, NULL, NULL)->(v, 1, m, NULL, NULL)->(m, 1, c, NULL, NULL)->(c, 1, g, NULL, NULL)->(g, 1, h, NULL, NULL)->(h, 1, s, NULL, NULL)->(s, 2, d, NULL, NULL)->(d, 3, t, NULL, NULL)->(t, 5, o, NULL, NULL)->(o, 5, b, NULL, NULL)->(b, 10, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

Tree
(dummy, 0, c, NULL, NULL)->(c, 1, g, NULL, NULL)->(g, 1, h, NULL, NULL)->(h, 1, vm, NULL, NULL)->(vm, 2, s, v, m)->(s, 2, d, NULL, NULL)->(d, 3, t, NULL, NULL)->(t, 5, o, NULL, NULL)->(o, 5, b, NULL, NULL)->(b, 10, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, h, NULL, NULL)->(h, 1, cg, NULL, NULL)->(cg, 2, vm, c, g)->(vm, 2, s, v, m)->(s, 2, d, NULL, NULL)->(d, 3, t, NULL, NULL)->(t, 5, o, NULL, NULL)->(o, 5, b, NULL, NULL)->(b, 10, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, vm, NULL, NULL)->(vm, 2, s, v, m)->(s, 2, hcg, NULL, NULL)->(hcg, 3, d, h, cg)->(d, 3, t, NULL, NULL)->(t, 5, o, NULL, NULL)->(o, 5, b, NULL, NULL)->(b, 10, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, hcg, NULL, NULL)->(hcg, 3, d, h, cg)->(d, 3, vms, NULL, NULL)->(vms, 4, t, vm, s)->(t, 5, o, NULL, NULL)->(o, 5, b, NULL, NULL)->(b, 10, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, vms, NULL, NULL)->(vms, 4, t, vm, s)->(t, 5, o, NULL, NULL)->(o, 5, hcgd, NULL, NULL)->(hcgd, 6, b, hcg, d)->(b, 10, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, o, NULL, NULL)->(o, 5, hcgd, NULL, NULL)->(hcgd, 6, vmst, hcg, d)->(vmst, 9, b, vms, t)->(b, 10, i, NULL, NULL)->(i, 10, e, NULL, NULL)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, vmst, NULL, NULL)->(vmst, 9, b, vms, t)->(b, 10, i, NULL, NULL)->(i, 10, ohcgd, NULL, NULL)->(ohcgd, 11, e, o, hcgd)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, i, NULL, NULL)->(i, 10, ohcgd, NULL, NULL)->(ohcgd, 11, vmstb, o, hcgd)->(vmstb, 19, e, vmst, b)->(e, 20, a, NULL, NULL)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, vmstb, NULL, NULL)->(vmstb, 19, e, vmst, b)->(e, 20, iohcgd, NULL, NULL)->(iohcgd, 21, a, i, ohcgd)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, iohcgd, NULL, NULL)->(iohcgd, 21, vmstbe, i, ohcgd)->(vmstbe, 39, a, vmstb, e)->(a, 40, NULL, NULL, NULL)->

(dummy, 0, a, NULL, NULL)->(a, 40, iohcgdvmstbe, NULL, NULL)->(iohcgdvmstbe, 60, NULL, iohcgd, vmstbe)->

(dummy, 0, aiohcgdvmstbe, NULL, NULL)->(aiohcgdvmstbe, 100, NULL, a, iohcgdvmstbe)->