**CSC 323-21: Project5: Huffman Encode and Decode (CPP)**

**Adil Abuwani**

**Due date: Oct. 9, 2018**

Algorithm steps for computing:

Step1: Open a text file to compute the probabilities of characters in the file;

and store the char-prob pairs in two 1-D arrays (charCounts[256]

as defined in project 1 with additional charCode [256]) in this project,

Output the char-prob pairs to an output file, as in your project 1.

Step 2: Construct the ordered linked list from the char-prob pair and then,

from the ordered linked list to construct the Huffman binary tree

similar to your project 3. Output debugging print as in your project 3.

Step 3: Open a text file to be encoded (see algorithm below)

and outputs the encoded result (zero’s and 1’s ) to a text file.

Step 4: Next, asks the user for the name of the text file (1’s and 0’s) that is to be decode.

Then, do the decoding; then,

output the result of the decode text to an output file.

Main()

Step 0: inFile1 🡨 open input file from argv[1]

outFile1, outFile2, …, outFile4 🡨 open from argv[3], …, argv[6]

Step 1: computeCount (inFile1) // see below for method update

Step 2: constructHuffmanLList (charCounts, outFile4) // see below for method update

Step 3: constructHuffmanBinTree (listHead, outFile4) // No update, same as project 3

Step 4: getCode (Root, ‘’, outFile1) // ‘’ is an empty string // see below for method update

\*\*\* below are new for this project \*\*\*

Step 5: inFile2 🡨 open input file from argv[2] // the text file for encoding

Step 6: Encode (inFile2, outFile2) // new method, see algorithm steps below

Step 7: decodeFileName 🡨 ask user from console to provide the name of the file to be decode

// decodeFileName is a string variable!

Step 8: decodeFile 🡨 open the decodeFileName

Step 9: Decode (decodeFile, outFile3, Root) // new method, see algorithm steps below

Step 10: close all files

computeCount (inFile)

step 1: charIn 🡨 read the next character from inFile, **one character at a time**

**// declare charIn as unsign char**

step 2: index 🡨 cast charIn to integer // you may use atoi (charIn) function in C++

step 3: charCounts[index]++

step 4: repeat step 1 to step 3 while inFile is not empty

step 5: printAry (outFile4)

printAry (outFile)

step 1: index 🡨0

step 2: if charCounts[index] > 0

symbol 🡨 cast index to character // you may use itoa (index) function in C++

write symbol charCounts[index] per text line to outFile

step 3: index ++

step 4: repeat step 2 to step 3 while i < 256

. constructHuffmanLList (charCounts[], outFile)

Step 1: listHead 🡨 get a newNode as the dummy listNode with (“dummy” ,0), listHead to point to.

Step 2: index 🡨 0

Step 3: if charCount[index] > 0

chr 🡨 cast index to asci // you may use C++ itoa function

Prob 🡨 charCounts [index]

newNode 🡨 get a new listNode

newNode.chStr 🡨 chr

newNode.prob 🡨 Prob

newNode.next 🡨 null

spot 🡨 findSpot (listHead, newNode) // use algorithm steps given in class

insertOneNode(spot, newNode) // insert newNode after spot

Step 4: index++

Step 5: repeat step 3 – step 4 while index <= 255

Step 6: printList (listHead, outFile)

constructHuffmanBinTree (listHead, outFile)

Step 1: newNode 🡨 create a treeNode

newNode’s prob 🡨 the sum of prob of the first and second node of the list // first is the node after dummy

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

newNode’s left 🡨 the first node of the list

newNode’s right 🡨 the second node of the list

Step 2: spot 🡨 findSpot(listHead, newNode)

insertOneNode (spot, newNode) // inserting newNode between spot and spot.next.

// only need two statements.

listHead.next 🡨 listHead .next.next.next //

printList (listHead, outFile) //

Step 3: repeat step 1 – step 2 until the list only has one node left which is the newNode

Step 4: Root 🡨 newNode

getCode (T, code)

if isLeaf (T)

T.code 🡨 code;

Index 🡨cast T. chStr to integer, you may use C++ atoi function

charCode[index] 🡨 code

else

constructCharCode (T->left, code + “0”) //string concatenation

constructCharCode (T->right, code + “1”) //string concatenation

Encode (inFile, outFile)

step 1: charIn 🡨 read the next character from inFile, **one character at a time**

**// including spaces!!!**

step 2: index 🡨 cast charIn to integer // You may use C++ atoi function

step 3: code 🡨 charCode[index]

step 4: write code to outFile

step 5: repeat step 1 to step 4 while inFile is not empty

Decode (inFile, outFile, node) // a recursive function!

if isLeaf (node)

Write node.chr to outFile

Decode (inFile, outFile, Root)

else

if eof inFile // and node is NOT a leaf

write error message “the encode file is a corrupted file”

else

nextBit 🡨 read the next char from inFile

if nextBit is ‘0’

Decode (inFile, outFile, node’s left)

If nextBit is ‘1’

Decode (inFile, outFile, node’s right)

b) Source code in CPP:

**#include** <iostream>

**#include** <string>

**#include** <fstream>

**using** **namespace** std;

string charCode[256]; //will contain the charCode array of strings

**class** frequencyCounter{

**public**:

**int** charCounts[256];

//default constructor

**frequencyCounter**(){

**for**(**int** i=0; i<256; i++){

**this**->charCounts[i]=0;

}//inatilze to 0

}

**void** **computeCount**(ifstream& inFile){

//datastructures

**unsigned** **char** charIn;

**int** index=0;

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

inFile>>noskipws;

//Space between words need to include in the counting!!!

**while**(inFile>>charIn){ //read single character at a time

//step 2- index <- cast charIn to integer

index=(**int**)charIn;

//step 3: charCounts[index]++, and ignore those greatr then 256

**if**(index<256) { //ignore those greater then 255

**this**->charCounts[index]++;

}

}//end while //step 4: repeat step 1 to step 3 while infile is not empty

}**else**{

cerr<<"Can not open the file"<<**endl**;

}

}//end computeCount

**void** **printAry**(ofstream& outFile){

**if**(outFile.is\_open()){

**for**(**int** index=0;index<256;index++) {

//step2

**if**(charCounts[index]>0) { //Do not print any charactors that have zero count.

//cast index to character

**unsigned** **char** c=(**char**)index;

string temp(1, c); //convert to string

outFile<<c<<" "<<**this**->charCounts[index]<<**endl**;

}//end if

}//end for, step 4: repeat step 2 to step 3 while i < 256

}**else**{

cerr<<"Can not open the file"<<**endl**;

}

}//end printAry

}; //end frequencyCounter

**class** treeNode{

**public**:

string chStr;

**int** prob;

treeNode\* next;

treeNode\* left;

treeNode\* right;

string code;

string charCode[256];

// default-constructor with no paramater

**treeNode**(){

**this**->chStr="";

**this**->prob=0;

**this**->next=NULL;

**this**->left=NULL;

**this**->right=NULL;

**this**->code="";

}

//constructor with two paramaters

**treeNode**(string ch, **int** p){

**this**->chStr=ch;

**this**->prob=p;

**this**->next=NULL;

**this**->left=NULL;

**this**->right=NULL;

**this**->code="";

}//end constructor

**void** **printNode**(treeNode\* T, ofstream &outFile){

outFile << " (\"" <<T->chStr << "\", " << T->prob << "): "; //first one is empty data

**if**(T->next!= NULL){

outFile<<"(Next: \""<<T->next->chStr<<"\", "<<T->next->prob << ") ";

}

**else**{

outFile<<"(Next: NULL) ";

}

**if**(T->left!= NULL){

outFile<<"(Left: \""<<T->left->chStr<<"\", "<<T->left->prob << ") ";

}

**else**{

outFile<<"(Left: NULL) ";

}

**if**(T->right!= NULL){

outFile<<"(Right: \""<<T->right->chStr << "\", " << T->right->prob << ") --> ";

}

**else**{

outFile<<"(Right: NULL) --> ";

}

outFile<<**endl**; //one print per text line

}//end printTree-node

///////////////////////////////////////////////////////////////////////////////////

};//end treeNode

**class** linkedList{

**public**:

treeNode\* listHead;

**linkedList**(){

treeNode\* myHeadNode;

//Step 1: listHead get a newNode as the dummy listNode with (“dummy” ,0), listHead to point to.

myHeadNode= **new** treeNode("dummy", 0); //new since its a pointer constructor

**this**->listHead=myHeadNode;

}//end consructor

treeNode\* **findSpot**(treeNode \*newNode) {

treeNode\* mySpot = listHead; //start from the treeHead

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

mySpot=mySpot->next;

} //end while

**return** mySpot;

}//end findSpot

**void** **insertNewNode** (treeNode \*newNode) {

treeNode\* mySpot = findSpot(newNode); //find spot, and insert the node

newNode->next = mySpot->next;

mySpot->next = newNode;

}//end inserNewtNode

**void** **printList**(ofstream& outFile) {

treeNode\* currentNode=listHead;

outFile<<"listHead -->";

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

outFile << "(\""<<currentNode->chStr << "\", " <<currentNode->prob<< ", \"" << currentNode->next->chStr << "\")-->";

currentNode=currentNode->next; //move to the next node

}//end while

//last element

outFile << "(\""<<currentNode->chStr << "\", " <<currentNode->prob<< ", NULL)-->NULL"<<**endl**;

}//end printList

**void** **constructHuffmanLList**(**int** charCount[], ofstream& outFile, linkedList\* myLinkedList){

string ch;

**int** prob;

// newNode;

**int** index=0;

**while**(index<256){

**if**(charCount[index]>0){

prob=charCount[index];

**unsigned** **char** c=(**char**)index; //cast to char

string temp(1, c);

ch=temp;

treeNode\* newNode = **new** treeNode(ch,prob); //create new node

myLinkedList->insertNewNode(newNode); //find spot and insert it

}//end if

index++;

}//end while

//print list-// Note: This printList is outside of the loop, therefore, you will only print the list once!!!

myLinkedList->printList(outFile);

} //end constructHuffmanLinList

};

**class** HuffmanBinaryTree{

**public**:

treeNode\* root; //root of tree

//constructors

**HuffmanBinaryTree**(){

**this**->root=NULL;

}//end constructor

**void** **constructHuffmanBinTree** (ofstream& outFile, linkedList\* myLinkedList ){

**while**(myLinkedList->listHead->next->next!=NULL){

//create a new node, and set as a tree node

treeNode\* myNode= **new** treeNode(); //create a tree node, and set as a tree node, with empty par

myNode->chStr=myLinkedList->listHead->next->chStr + myLinkedList->listHead->next->next->chStr; //conconate 1node+ 2nd node

myNode->prob=myLinkedList->listHead->next->prob + myLinkedList->listHead->next->next->prob; //1node + 2 node

myNode->left=myLinkedList->listHead->next; //left is the first node

myNode->right=myLinkedList->listHead->next->next; //right is the second node

outFile<<**endl**; //go to new line

//print hear

myLinkedList->insertNewNode(myNode); //find the spot, and insert one node (one method for both operation baby)

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

myNode->printNode(myNode, outFile);

myLinkedList->printList(outFile); //print the linkedlist

}//end while repeat step 1 – step 2 until the list only has one node left which is the newNode

**this**->root=myLinkedList->listHead->next; //root is the new node

}//end constructhuffmanLList

**bool** **isLeaf**(treeNode\* T){ // given node is a leaf if both left and right are null.

**if**(T->left==NULL && T->right==NULL){

**return** **true**;

}

**return** **false**;

}//end isLeaf

**void** **preOrderTraversal**(treeNode\* T, ofstream& outFile){

//if (T is null) do nothing

**if**(T==NULL){

**return**; //do nothing!

}**else**{

T->printNode(T, outFile);

preOrderTraversal(T->left, outFile);

preOrderTraversal(T->right, outFile);

}

} //end preOrderTraversal

**void** **inOrderTraversal**(treeNode\* T, ofstream& outFile){

//if (T is null) do nothing

**if**(T==NULL){

**return**; //do nothing!

}**else**{

inOrderTraversal(T->left, outFile);

T->printNode(T, outFile);

inOrderTraversal(T->right, outFile);

}

} //end ineOrderTraversal

**void** **postOrderTraversal**(treeNode\* T, ofstream& outFile){

//if (T is null) do nothing

**if**(T==NULL){

**return**; //do nothing!

}**else**{

postOrderTraversal(T->left, outFile);

postOrderTraversal(T->right, outFile);

T->printNode(T, outFile);

}

} //end ineOrderTraversal

**void** **getCode**(treeNode\* T, string code, ofstream& outFile){

**if**(isLeaf(T)){

T->code=code; //set the tree code

**unsigned** **char** chStr=T->chStr.at(0); //the character

**int** index=(**int**)chStr;

T->charCode[index]=code;

outFile<<chStr<<"\t"<<T->charCode[index]<<**endl**; // new line after

charCode[index]=T->charCode[index];

}**else**{

getCode(T->left, code+"0", outFile);

getCode(T->right, code+"1", outFile);

}//end else

}//end getCode

**void** **Encode**(ifstream& inFile, ofstream& outFile){

**unsigned** **char** charIn;

**int** index=0;

inFile>>noskipws; //dont skip whitespace

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

**while**(inFile>>charIn){ //read single character at a time

//step 2- index <- cast charIn to integer

index=(**int**)charIn;

//step 3: code = charCode[index]

//write code to outFile

outFile<<charCode[index];

}//end while //step 5: repeat step 1 to step 3 while infile is not empty

}**else**{

cerr<<"Can not open the file"<<**endl**;

}

}//end Encode

**void** **Decode**(ifstream& inFile, ofstream& outFile, treeNode\* T, treeNode\* myRoot){

**unsigned** **char** nextBit;

**if**(isLeaf(T)){

outFile<<T->chStr;

Decode(inFile, outFile, myRoot, myRoot); //decode from the root

}**else** **if**(inFile.eof()&&!isLeaf(T)){

cerr<<"the encode file is a corrupted file"<<**endl**;

}**else**{

inFile>>nextBit;

**if**(nextBit=='0'){

Decode(inFile, outFile, T->left, myRoot);

}

**if**(nextBit=='1'){

Decode(inFile, outFile, T->right, myRoot);

}

}

}//end Decode

}; //end classhufmanBinaryTree

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

ifstream inFile1;

ifstream inFile2;

inFile1.open(argv[1]); //Input-1 (argv[1] ): An English text file to compute char-prob pairs

inFile2.open(argv[2]); //

ofstream outFile1;

ofstream outFile2;

ofstream outFile3; //for decode

ofstream outFile4;

outFile1.open(argv[3]);

outFile2.open(argv[4]);

outFile3.open(argv[5]);

outFile4.open(argv[6]); //outFile 4 (argv[6]): for all the debugging outputs.

frequencyCounter myfrequencyCounter;

//Step 1: computeCount (inFile1)

myfrequencyCounter.computeCount(inFile1);

myfrequencyCounter.printAry(outFile4); //printAry

//Step 2: constructHuffmanLList (charCounts, outFile4) // see below for method update

linkedList\* myLinkedList= **new** linkedList();

outFile4<<**endl**;

myLinkedList->constructHuffmanLList(myfrequencyCounter.charCounts, outFile4, myLinkedList);

//Step 3: constructHuffmanBinTree (listHead, outFile4) // No update, same as project 3

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

myBinaryTree->constructHuffmanBinTree(outFile4, myLinkedList);

myBinaryTree->getCode(myBinaryTree->root,"",outFile1); //outFile1 (argv[3]): A text file contains the Huffman <char , code> pairs

//Step 5: inFile2  open input file from argv[2] // the text file for encoding

//Step 6: Encode (inFile2, outFile2) // new method, see algorithm steps below

myBinaryTree->Encode(inFile2, outFile2); //outFile2 (argv[4]): A text file contains the encoded result.

//close input2file

inFile2.close(); //close inputfile after encoding is complete

outFile2.close(); //outputfile2 contains 1 and 0, that has to be decoded

**char** userInput[100];

cout<<"Enter file name to be Decoded (Ex: foo.txt)"<<**endl**;

cin>>userInput;

//Step 8: decodeFile  open the decodeFileName

ifstream inFile3;

inFile3.open(userInput);

treeNode\* myRoot=myBinaryTree->root; //point at the root

//Step 9: Decode (decodeFile, outFile3, Root)

myBinaryTree->Decode(inFile3, outFile3, myBinaryTree->root, myRoot);

//step10-close all files

inFile1.close();

inFile3.close();

//output-files

outFile1.close();

outFile3.close();

outFile4.close();

**return** 0;

}//end main

c) Input- a text file argv[1]: HuffmanEncodeDecode\_data1.txt

In fall (autumn), New England is famous for its glorious foliage as billions

of leaves change from green to a kaleidoscope

of colors. The air is crisp and cool — perfect for hiking, biking or

a classic fly-drive break along back roads, where farm stands are piled high

with crunchy apples and orange pumpkins. When is the best time to come?

The leaves start turning colors in the northern regions of Maine, Vermont,

and New Hampshire typically around mid-to-late September and peaking around mid-October.

In the more southern states of Massachusetts and Rhode Island

and along the coast of New Hampshire and southern Maine, color starts later

and often lasts up until November.

Connecticut: Drive scenic State Route 169 parallel to the Rhode Island border,

with picture-pretty villages such as Canterbury, Pomphrey, Brooklyn, and Woodstock.

In Old Mystic, tour B. F. Clyde Cider Mill, America’s oldest steam-powered cider mill,

family owned and run since 1881. Watch cider-making demonstrations

on weekends in October and November. During fall harvest time,

learn the difference between Merlot and Chardonnay, Cayuga and Seyval

at the 25 wineries along the Connecticut Wine Trail. Connecticut is one big orchard:

in fall, pick your own pumpkins and apples Or, hike and bike in the Last Green Valley,

the 35-town National Heritage Corridor in the northeast corner of the state:

five state parks, seven state forests, 80 ponds and lakes, and 130 miles of trails,

including the East Coast Greenway.

Maine: In the midcoast, tour the Maine Maritime Museum, where you can

see old boats, a shipyard and art, as well as take part in hands-on activities.

Drive down a rocky promontory to Five Islands Lobster in Georgetown for a lobster roll:

from Route 1, take 127 South. Local directions suggest:

"keep going until your hat floats, and then just back up a bit."

Enjoy fall color all over Maine: go hiking and biking in

Acadia National Park canoe and listen to loons in the Rangeley Lakes region.

And, for a spectacular view, take the elevator to the observatory

high up on the Penobscot Narrows Bridge.

Masssachusetts:  Drive along the Mohawk Trail in Western Massachusetts — America’s

first officially designated "scenic road."

It runs up and over the Berkshire Hills and leads to state parks

and forests for hiking and biking. Drive to the top of Mount Greylock

for fabulous foliage views. At Natural Bridge State Park, see a true wonder,

North America’s only naturally formed white marble arch.

Or, zipline through the tree canopy on one of the ziplines

found throughout the state. North of Boston,

go out to Cape Ann and take a boat tour from Gloucester or Essex.

New Hampshire:In the beautiful White Mountains region,

you can stand on the top of New England: the summit of Mount Washington.

Get there on the Auto Road (self-drive or van tours)

or on the historic Cog Railway. Want to climb an easy mountain?

Near Jaffrey, Mount Monadnock is the "most climbed mountain in North America,"

with great views from the top. In the Lakes Region,

take a cruise on the M/S Mount Washington.

With mountains as a backdrop, the foliage seems to wrap around Lake Winnipesaukee.

Up in northern New Hampshire,

Dixville Notch is an area covered with endless unspoiled scenery,

exceptional hiking, mountain biking.

Rhode Island: The USA’s smallest state is making a name

for itself as a producer of wine.

And, when better to visit the vineyards than at harvest time?

With lunch and a tasting, this is a great day out.

But, across the state, Rhode Island farms are also celebrating their bounty;

stop at farm stands, pick-your-own and join-in-the-harvest festivals.

Or, go back in time to the start of America’s industrial revolution.

At Slater Mill, Samuel Slater built North America’s first successful water-powered,

cotton-spinning mill on the banks of the Blackstone River.

Tour this historic area by boat and see the fall color on the Blackstone Valley Explorer.

Vermont:For birds-eye views of the foliage, head for Bennington.

At the Battle Monument, you can scan the landscape from the top, more than 300 ft.

(91m) above the ground. And, you don’t have to climb any stairs; just take the elevator!

Or, drive up the very steep Skyline Drive to the top of Mount Equinox, south of Manchester.

On a clear day, you can see across New England, into New York State and even into Canada.

Burlington makes a great base for seeing the foliage:

take a Lake Champlain cruise, or go up in a hot-air balloon!

In fall (autumn), New England is famous for its glorious foliage as billions

of leaves change from green to a kaleidoscope

of colors. The air is crisp and cool — perfect for hiking, biking or

a classic fly-drive break along back roads, where farm stands are piled high

with crunchy apples and orange pumpkins. When is the best time to come?

The leaves start turning colors in the northern regions of Maine, Vermont,

and New Hampshire typically around mid-to-late September and peaking around mid-October.

In the more southern states of Massachusetts and Rhode Island

and along the coast of New Hampshire and southern Maine, color starts later

and often lasts up until November.

Connecticut: Drive scenic State Route 169 parallel to the Rhode Island border,

with picture-pretty villages such as Canterbury, Pomphrey, Brooklyn, and Woodstock.

In Old Mystic, tour B. F. Clyde Cider Mill, America’s oldest steam-powered cider mill,

family owned and run since 1881. Watch cider-making demonstrations

on weekends in October and November. During fall harvest time,

learn the difference between Merlot and Chardonnay, Cayuga and Seyval

at the 25 wineries along the Connecticut Wine Trail. Connecticut is one big orchard:

in fall, pick your own pumpkins and apples Or, hike and bike in the Last Green Valley,

the 35-town National Heritage Corridor in the northeast corner of the state:

five state parks, seven state forests, 80 ponds and lakes, and 130 miles of trails,

including the East Coast Greenway.

Maine: In the midcoast, tour the Maine Maritime Museum, where you can

see old boats, a shipyard and art, as well as take part in hands-on activities.

Drive down a rocky promontory to Five Islands Lobster in Georgetown for a lobster roll:

from Route 1, take 127 South. Local directions suggest:

"keep going until your hat floats, and then just back up a bit."

Enjoy fall color all over Maine: go hiking and biking in

Acadia National Park canoe and listen to loons in the Rangeley Lakes region.

And, for a spectacular view, take the elevator to the observatory

high up on the Penobscot Narrows Bridge.

Masssachusetts:  Drive along the Mohawk Trail in Western Massachusetts — America’s

first officially designated "scenic road."

It runs up and over the Berkshire Hills and leads to state parks

and forests for hiking and biking. Drive to the top of Mount Greylock

for fabulous foliage views. At Natural Bridge State Park, see a true wonder,

North America’s only naturally formed white marble arch.

Or, zipline through the tree canopy on one of the ziplines

found throughout the state. North of Boston,

go out to Cape Ann and take a boat tour from Gloucester or Essex.

New Hampshire:In the beautiful White Mountains region,

you can stand on the top of New England: the summit of Mount Washington.

Get there on the Auto Road (self-drive or van tours)

or on the historic Cog Railway. Want to climb an easy mountain?

Near Jaffrey, Mount Monadnock is the "most climbed mountain in North America,"

with great views from the top. In the Lakes Region,

take a cruise on the M/S Mount Washington.

With mountains as a backdrop, the foliage seems to wrap around Lake Winnipesaukee.

Up in northern New Hampshire,

Dixville Notch is an area covered with endless unspoiled scenery,

exceptional hiking, mountain biking.

Rhode Island: The USA’s smallest state is making a name

for itself as a producer of wine.

And, when better to visit the vineyards than at harvest time?

With lunch and a tasting, this is a great day out.

But, across the state, Rhode Island farms are also celebrating their bounty;

stop at farm stands, pick-your-own and join-in-the-harvest festivals.

Or, go back in time to the start of America’s industrial revolution.

At Slater Mill, Samuel Slater built North America’s first successful water-powered,

cotton-spinning mill on the banks of the Blackstone River.

Tour this historic area by boat and see the fall color on the Blackstone Valley Explorer.

Vermont:For birds-eye views of the foliage, head for Bennington.

At the Battle Monument, you can scan the landscape from the top, more than 300 ft.

(91m) above the ground. And, you don’t have to climb any stairs; just take the elevator!

Or, drive up the very steep Skyline Drive to the top of Mount Equinox, south of Manchester.

On a clear day, you can see across New England, into New York State and even into Canada.

Burlington makes a great base for seeing the foliage:

take a Lake Champlain cruise, or go up in a hot-air balloon!

In fall (autumn), New England is famous for its glorious foliage as billions

of leaves change from green to a kaleidoscope

of colors. The air is crisp and cool — perfect for hiking, biking or

a classic fly-drive break along back roads, where farm stands are piled high

with crunchy apples and orange pumpkins. When is the best time to come?

The leaves start turning colors in the northern regions of Maine, Vermont,

and New Hampshire typically around mid-to-late September and peaking around mid-October.

In the more southern states of Massachusetts and Rhode Island

and along the coast of New Hampshire and southern Maine, color starts later

and often lasts up until November.

Connecticut: Drive scenic State Route 169 parallel to the Rhode Island border,

with picture-pretty villages such as Canterbury, Pomphrey, Brooklyn, and Woodstock.

In Old Mystic, tour B. F. Clyde Cider Mill, America’s oldest steam-powered cider mill,

family owned and run since 1881. Watch cider-making demonstrations

on weekends in October and November. During fall harvest time,

learn the difference between Merlot and Chardonnay, Cayuga and Seyval

at the 25 wineries along the Connecticut Wine Trail. Connecticut is one big orchard:

in fall, pick your own pumpkins and apples Or, hike and bike in the Last Green Valley,

the 35-town National Heritage Corridor in the northeast corner of the state:

five state parks, seven state forests, 80 ponds and lakes, and 130 miles of trails,

including the East Coast Greenway.

Maine: In the midcoast, tour the Maine Maritime Museum, where you can

see old boats, a shipyard and art, as well as take part in hands-on activities.

Drive down a rocky promontory to Five Islands Lobster in Georgetown for a lobster roll:

from Route 1, take 127 South. Local directions suggest:

"keep going until your hat floats, and then just back up a bit."

Enjoy fall color all over Maine: go hiking and biking in

Acadia National Park canoe and listen to loons in the Rangeley Lakes region.

And, for a spectacular view, take the elevator to the observatory

high up on the Penobscot Narrows Bridge.

Masssachusetts:  Drive along the Mohawk Trail in Western Massachusetts — America’s

first officially designated "scenic road."

It runs up and over the Berkshire Hills and leads to state parks

and forests for hiking and biking. Drive to the top of Mount Greylock

for fabulous foliage views. At Natural Bridge State Park, see a true wonder,

North America’s only naturally formed white marble arch.

Or, zipline through the tree canopy on one of the ziplines

found throughout the state. North of Boston,

go out to Cape Ann and take a boat tour from Gloucester or Essex.

New Hampshire:In the beautiful White Mountains region,

you can stand on the top of New England: the summit of Mount Washington.

Get there on the Auto Road (self-drive or van tours)

or on the historic Cog Railway. Want to climb an easy mountain?

Near Jaffrey, Mount Monadnock is the "most climbed mountain in North America,"

with great views from the top. In the Lakes Region,

take a cruise on the M/S Mount Washington.

With mountains as a backdrop, the foliage seems to wrap around Lake Winnipesaukee.

Up in northern New Hampshire,

Dixville Notch is an area covered with endless unspoiled scenery,

exceptional hiking, mountain biking.

Rhode Island: The USA’s smallest state is making a name

for itself as a producer of wine.

And, when better to visit the vineyards than at harvest time?

With lunch and a tasting, this is a great day out.

But, across the state, Rhode Island farms are also celebrating their bounty;

stop at farm stands, pick-your-own and join-in-the-harvest festivals.

Or, go back in time to the start of America’s industrial revolution.

At Slater Mill, Samuel Slater built North America’s first successful water-powered,

cotton-spinning mill on the banks of the Blackstone River.

Tour this historic area by boat and see the fall color on the Blackstone Valley Explorer.

Vermont:For birds-eye views of the foliage, head for Bennington.

At the Battle Monument, you can scan the landscape from the top, more than 300 ft.

(91m) above the ground. And, you don’t have to climb any stairs; just take the elevator!

Or, drive up the very steep Skyline Drive to the top of Mount Equinox, south of Manchester.

On a clear day, you can see across New England, into New York State and even into Canada.

Burlington makes a great base for seeing the foliage:

take a Lake Champlain cruise, or go up in a hot-air balloon!

Input2- a text file argv[2]: HuffmanEncodeDecode\_EncodeData.txt

Enjoy fall color all over Maine: go hiking and biking in

Acadia National Park canoe and listen to loons in the Rangeley Lakes region.

And, for a spectacular view, take the elevator to the observatory

high up on the Penobscot Narrows Bridge.

Up in northern New Hampshire,

Dixville Notch is an area covered with endless unspoiled scenery,

exceptional hiking, mountain biking.

Input3- A text file from the user console, which is the result from the encoded input\_2 (outputFile2) which will be used to determine if I could get back the same result:

0111101000110011110000010001110111110100100101110011100111100000010001001110000011110101110011100111101000111001011110011110111011011011010001101111001000001100101111000110011100100001001010001100101111101011011000101110111010001000010010100011001011111001000110110100101011110010000010110010101001011110111001101011101001001000011010111001111001010100101011001100100111000000101101101000111111010110110001011101001101000001101011110110110101010001101001110001000011000011100100011011010100111011111101110011101011011001011111111001111111110111110001000010101100100111110001010101100111111010111010010000110111000011010010101111001011000101010100110100100100000111101011110000101011011110000010101011000000000110011101100111101110010010011111110001010100110101010110010011111110101001110111111011111001111111110010101110101000001111010101000110101001110111111010001110100000111110011111001010111010100000111110111110100101011100100010111011101100000101011011010000110110101001110111111001010100101111011010001110100000100000100010101101110011010110011001110001110001000111011101010100110100001010101111111111000011010010110010101010101011010110110010001101100110100000111010011101111001101101101110011011111110001110111011000110110111110101100001011100100001111110101000101011100101001000011010001111010111110010010010011100111111110111001101000101000000011100101011010000011101011011011010110011111110110101011000001000111001011110011111100101110111000101001010011101101111011000101100111111000100011100000101100001010110100001001001111110010111000010000011110110111100111110111010100110100101111101111010110000011110101101010010010000110101110011110011100100001001010001100101110101001100111111000000010110101010110100011011011101000100001001010001100101111110000100101

OutputFile1 (argv[3]): A text file contains the Huffman <char , code>

c 00000

u 00001

s 0001

: 00100000

L 001000010

D 001000011

( 0010001000

5 00100010010

2 00100010011

’ 001000101

C 00100011

k 001001

d 00101

r 0011

i 0100

, 010100

1 010101000

P 0101010010

0 0101010011

; 01010101000

9 01010101001

z 01010101010

U 01010101011

O 010101011

  0101011

p 010110

g 010111

n 0110

h 01110

j 0111100000

V 0111100001

q 011110001000

Y 011110001001

— 01111000101

6 011110001100

/ 011110001101

J 011110001110

7 011110001111

A 01111001

E 011110100

! 01111010100

F 01111010101

x 0111101011

- 01111011

m 011111

o 1000

f 100100

100101

l 10011

t 1010

a 1011

110

. 1110000

w 1110001

v 1110010

N 11100110

R 111001110

G 1110011110

" 1110011111

b 1110100

I 111010100

B 111010101

W 111010110

S 111010111

T 1110110000

H 1110110001

3 11101100100

) 11101100101

? 11101100110

8 11101100111

M 11101101

y 1110111

e 1111

OutputFile2 (argv[4]): A text file contains the encoded result from the inputFile2

0111101000110011110000010001110111110100100101110011100111100000010001001110000011110101110011100111101000111001011110011110111011011011010001101111001000001100101111000110011100100001001010001100101111101011011000101110111010001000010010100011001011111001000110110100101011110010000010110010101001011110111001101011101001001000011010111001111001010100101011001100100111000000101101101000111111010110110001011101001101000001101011110110110101010001101001110001000011000011100100011011010100111011111101110011101011011001011111111001111111110111110001000010101100100111110001010101100111111010111010010000110111000011010010101111001011000101010100110100100100000111101011110000101011011110000010101011000000000110011101100111101110010010011111110001010100110101010110010011111110101001110111111011111001111111110010101110101000001111010101000110101001110111111010001110100000111110011111001010111010100000111110111110100101011100100010111011101100000101011011010000110110101001110111111001010100101111011010001110100000100000100010101101110011010110011001110001110001000111011101010100110100001010101111111111000011010010110010101010101011010110110010001101100110100000111010011101111001101101101110011011111110001110111011000110110111110101100001011100100001111110101000101011100101001000011010001111010111110010010010011100111111110111001101000101000000011100101011010000011101011011011010110011111110110101011000001000111001011110011111100101110111000101001010011101101111011000101100111111000100011100000101100001010110100001001001111110010111000010000011110110111100111110111010100110100101111101111010110000011110101101010010010000110101110011110011100100001001010001100101110101001100111111000000010110101010110100011011011101000100001001010001100101111110000100101

OutputFile3: A text file contains the decoded result, which is got from the outputFile2 that contained the encoded result:

Enjoy fall color all over Maine: go hiking and biking in

Acadia National Park canoe and listen to loons in the Rangeley Lakes region.

And, for a spectacular view, take the elevator to the observatory

high up on the Penobscot Narrows Bridge.

Up in northern New Hampshire,

Dixville Notch is an area covered with endless unspoiled scenery,

exceptional hiking, mountain biking.

OutputFile4: print only the \*first 6 pages\* of outFile4 using font size 9

219

2157

! 6

" 18

( 9

) 9

, 177

- 57

. 117

/ 3

0 12

1 21

2 6

3 9

5 6

6 3

7 3

8 9

9 6

: 39

; 6

? 9

A 48

B 36

C 42

D 21

E 24

F 9

G 18

H 18

I 36

J 3

L 21

M 75

N 60

O 24

P 12

R 33

S 36

T 18

U 6

V 12

W 36

Y 3

a 936

b 135

c 303

d 336

e 1137

f 213

g 204

h 408

i 690

j 12

k 165

l 453

m 207

n 813

o 855

p 189

q 3

r 687

s 645

t 936

u 306

v 126

w 120

x 15

y 147

z 6

’ 21

— 6

  93

listHead -->("dummy", 0, "q")-->("q", 3, "Y")-->("Y", 3, "J")-->("J", 3, "7")-->("7", 3, "6")-->("6", 3, "/")-->("/", 3, "—")-->("—", 6, "z")-->("z", 6, "U")-->("U", 6, ";")-->(";", 6, "9")-->("9", 6, "5")-->("5", 6, "2")-->("2", 6, "!")-->("!", 6, "F")-->("F", 9, "?")-->("?", 9, "8")-->("8", 9, "3")-->("3", 9, ")")-->(")", 9, "(")-->("(", 9, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "x")-->("x", 15, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("qY", 6): (Next: "—", 6) (Left: "q", 3) (Right: "Y", 3) -->

listHead -->("dummy", 0, "J")-->("J", 3, "7")-->("7", 3, "6")-->("6", 3, "/")-->("/", 3, "qY")-->("qY", 6, "—")-->("—", 6, "z")-->("z", 6, "U")-->("U", 6, ";")-->(";", 6, "9")-->("9", 6, "5")-->("5", 6, "2")-->("2", 6, "!")-->("!", 6, "F")-->("F", 9, "?")-->("?", 9, "8")-->("8", 9, "3")-->("3", 9, ")")-->(")", 9, "(")-->("(", 9, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "x")-->("x", 15, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("J7", 6): (Next: "qY", 6) (Left: "J", 3) (Right: "7", 3) -->

listHead -->("dummy", 0, "6")-->("6", 3, "/")-->("/", 3, "J7")-->("J7", 6, "qY")-->("qY", 6, "—")-->("—", 6, "z")-->("z", 6, "U")-->("U", 6, ";")-->(";", 6, "9")-->("9", 6, "5")-->("5", 6, "2")-->("2", 6, "!")-->("!", 6, "F")-->("F", 9, "?")-->("?", 9, "8")-->("8", 9, "3")-->("3", 9, ")")-->(")", 9, "(")-->("(", 9, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "x")-->("x", 15, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("6/", 6): (Next: "J7", 6) (Left: "6", 3) (Right: "/", 3) -->

listHead -->("dummy", 0, "6/")-->("6/", 6, "J7")-->("J7", 6, "qY")-->("qY", 6, "—")-->("—", 6, "z")-->("z", 6, "U")-->("U", 6, ";")-->(";", 6, "9")-->("9", 6, "5")-->("5", 6, "2")-->("2", 6, "!")-->("!", 6, "F")-->("F", 9, "?")-->("?", 9, "8")-->("8", 9, "3")-->("3", 9, ")")-->(")", 9, "(")-->("(", 9, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "x")-->("x", 15, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("6/J7", 12): (Next: "j", 12) (Left: "6/", 6) (Right: "J7", 6) -->

listHead -->("dummy", 0, "qY")-->("qY", 6, "—")-->("—", 6, "z")-->("z", 6, "U")-->("U", 6, ";")-->(";", 6, "9")-->("9", 6, "5")-->("5", 6, "2")-->("2", 6, "!")-->("!", 6, "F")-->("F", 9, "?")-->("?", 9, "8")-->("8", 9, "3")-->("3", 9, ")")-->(")", 9, "(")-->("(", 9, "6/J7")-->("6/J7", 12, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "x")-->("x", 15, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("qY—", 12): (Next: "6/J7", 12) (Left: "qY", 6) (Right: "—", 6) -->

listHead -->("dummy", 0, "z")-->("z", 6, "U")-->("U", 6, ";")-->(";", 6, "9")-->("9", 6, "5")-->("5", 6, "2")-->("2", 6, "!")-->("!", 6, "F")-->("F", 9, "?")-->("?", 9, "8")-->("8", 9, "3")-->("3", 9, ")")-->(")", 9, "(")-->("(", 9, "qY—")-->("qY—", 12, "6/J7")-->("6/J7", 12, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "x")-->("x", 15, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("zU", 12): (Next: "qY—", 12) (Left: "z", 6) (Right: "U", 6) -->

listHead -->("dummy", 0, ";")-->(";", 6, "9")-->("9", 6, "5")-->("5", 6, "2")-->("2", 6, "!")-->("!", 6, "F")-->("F", 9, "?")-->("?", 9, "8")-->("8", 9, "3")-->("3", 9, ")")-->(")", 9, "(")-->("(", 9, "zU")-->("zU", 12, "qY—")-->("qY—", 12, "6/J7")-->("6/J7", 12, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "x")-->("x", 15, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

(";9", 12): (Next: "zU", 12) (Left: ";", 6) (Right: "9", 6) -->

listHead -->("dummy", 0, "5")-->("5", 6, "2")-->("2", 6, "!")-->("!", 6, "F")-->("F", 9, "?")-->("?", 9, "8")-->("8", 9, "3")-->("3", 9, ")")-->(")", 9, "(")-->("(", 9, ";9")-->(";9", 12, "zU")-->("zU", 12, "qY—")-->("qY—", 12, "6/J7")-->("6/J7", 12, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "x")-->("x", 15, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("52", 12): (Next: ";9", 12) (Left: "5", 6) (Right: "2", 6) -->

listHead -->("dummy", 0, "!")-->("!", 6, "F")-->("F", 9, "?")-->("?", 9, "8")-->("8", 9, "3")-->("3", 9, ")")-->(")", 9, "(")-->("(", 9, "52")-->("52", 12, ";9")-->(";9", 12, "zU")-->("zU", 12, "qY—")-->("qY—", 12, "6/J7")-->("6/J7", 12, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "x")-->("x", 15, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("!F", 15): (Next: "x", 15) (Left: "!", 6) (Right: "F", 9) -->

listHead -->("dummy", 0, "?")-->("?", 9, "8")-->("8", 9, "3")-->("3", 9, ")")-->(")", 9, "(")-->("(", 9, "52")-->("52", 12, ";9")-->(";9", 12, "zU")-->("zU", 12, "qY—")-->("qY—", 12, "6/J7")-->("6/J7", 12, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "!F")-->("!F", 15, "x")-->("x", 15, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("?8", 18): (Next: "T", 18) (Left: "?", 9) (Right: "8", 9) -->

listHead -->("dummy", 0, "3")-->("3", 9, ")")-->(")", 9, "(")-->("(", 9, "52")-->("52", 12, ";9")-->(";9", 12, "zU")-->("zU", 12, "qY—")-->("qY—", 12, "6/J7")-->("6/J7", 12, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "!F")-->("!F", 15, "x")-->("x", 15, "?8")-->("?8", 18, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("3)", 18): (Next: "?8", 18) (Left: "3", 9) (Right: ")", 9) -->

listHead -->("dummy", 0, "(")-->("(", 9, "52")-->("52", 12, ";9")-->(";9", 12, "zU")-->("zU", 12, "qY—")-->("qY—", 12, "6/J7")-->("6/J7", 12, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "!F")-->("!F", 15, "x")-->("x", 15, "3)")-->("3)", 18, "?8")-->("?8", 18, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("(52", 21): (Next: "’", 21) (Left: "(", 9) (Right: "52", 12) -->

listHead -->("dummy", 0, ";9")-->(";9", 12, "zU")-->("zU", 12, "qY—")-->("qY—", 12, "6/J7")-->("6/J7", 12, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "!F")-->("!F", 15, "x")-->("x", 15, "3)")-->("3)", 18, "?8")-->("?8", 18, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "(52")-->("(52", 21, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

(";9zU", 24): (Next: "O", 24) (Left: ";9", 12) (Right: "zU", 12) -->

listHead -->("dummy", 0, "qY—")-->("qY—", 12, "6/J7")-->("6/J7", 12, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "!F")-->("!F", 15, "x")-->("x", 15, "3)")-->("3)", 18, "?8")-->("?8", 18, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "(52")-->("(52", 21, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, ";9zU")-->(";9zU", 24, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("qY—6/J7", 24): (Next: ";9zU", 24) (Left: "qY—", 12) (Right: "6/J7", 12) -->

listHead -->("dummy", 0, "j")-->("j", 12, "V")-->("V", 12, "P")-->("P", 12, "0")-->("0", 12, "!F")-->("!F", 15, "x")-->("x", 15, "3)")-->("3)", 18, "?8")-->("?8", 18, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "(52")-->("(52", 21, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "qY—6/J7")-->("qY—6/J7", 24, ";9zU")-->(";9zU", 24, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("jV", 24): (Next: "qY—6/J7", 24) (Left: "j", 12) (Right: "V", 12) -->

listHead -->("dummy", 0, "P")-->("P", 12, "0")-->("0", 12, "!F")-->("!F", 15, "x")-->("x", 15, "3)")-->("3)", 18, "?8")-->("?8", 18, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "(52")-->("(52", 21, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "jV")-->("jV", 24, "qY—6/J7")-->("qY—6/J7", 24, ";9zU")-->(";9zU", 24, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("P0", 24): (Next: "jV", 24) (Left: "P", 12) (Right: "0", 12) -->

listHead -->("dummy", 0, "!F")-->("!F", 15, "x")-->("x", 15, "3)")-->("3)", 18, "?8")-->("?8", 18, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "(52")-->("(52", 21, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "P0")-->("P0", 24, "jV")-->("jV", 24, "qY—6/J7")-->("qY—6/J7", 24, ";9zU")-->(";9zU", 24, "O")-->("O", 24, "E")-->("E", 24, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("!Fx", 30): (Next: "R", 33) (Left: "!F", 15) (Right: "x", 15) -->

listHead -->("dummy", 0, "3)")-->("3)", 18, "?8")-->("?8", 18, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "(52")-->("(52", 21, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "P0")-->("P0", 24, "jV")-->("jV", 24, "qY—6/J7")-->("qY—6/J7", 24, ";9zU")-->(";9zU", 24, "O")-->("O", 24, "E")-->("E", 24, "!Fx")-->("!Fx", 30, "R")-->("R", 33, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("3)?8", 36): (Next: "W", 36) (Left: "3)", 18) (Right: "?8", 18) -->

listHead -->("dummy", 0, "T")-->("T", 18, "H")-->("H", 18, "G")-->("G", 18, """)-->(""", 18, "(52")-->("(52", 21, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "P0")-->("P0", 24, "jV")-->("jV", 24, "qY—6/J7")-->("qY—6/J7", 24, ";9zU")-->(";9zU", 24, "O")-->("O", 24, "E")-->("E", 24, "!Fx")-->("!Fx", 30, "R")-->("R", 33, "3)?8")-->("3)?8", 36, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("TH", 36): (Next: "3)?8", 36) (Left: "T", 18) (Right: "H", 18) -->

listHead -->("dummy", 0, "G")-->("G", 18, """)-->(""", 18, "(52")-->("(52", 21, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "P0")-->("P0", 24, "jV")-->("jV", 24, "qY—6/J7")-->("qY—6/J7", 24, ";9zU")-->(";9zU", 24, "O")-->("O", 24, "E")-->("E", 24, "!Fx")-->("!Fx", 30, "R")-->("R", 33, "TH")-->("TH", 36, "3)?8")-->("3)?8", 36, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("G"", 36): (Next: "TH", 36) (Left: "G", 18) (Right: """, 18) -->

listHead -->("dummy", 0, "(52")-->("(52", 21, "’")-->("’", 21, "L")-->("L", 21, "D")-->("D", 21, "1")-->("1", 21, "P0")-->("P0", 24, "jV")-->("jV", 24, "qY—6/J7")-->("qY—6/J7", 24, ";9zU")-->(";9zU", 24, "O")-->("O", 24, "E")-->("E", 24, "!Fx")-->("!Fx", 30, "R")-->("R", 33, "G"")-->("G"", 36, "TH")-->("TH", 36, "3)?8")-->("3)?8", 36, "W")-->("W", 36, "S")-->("S", 36, "I")-->("I", 36, "B")-->("B", 36, ":")-->(":", 39, "C")-->("C", 42, "A")-->("A", 48, "-")-->("-", 57, "N")-->("N", 60, "M")-->("M", 75, " ")-->(" ", 93, ".")-->(".", 117, "w")-->("w", 120, "v")-->("v", 126, "b")-->("b", 135, "y")-->("y", 147, "k")-->("k", 165, ",")-->(",", 177, "p")-->("p", 189, "g")-->("g", 204, "m")-->("m", 207, "f")-->("f", 213, "

")-->("

", 219, "c")-->("c", 303, "u")-->("u", 306, "d")-->("d", 336, "h")-->("h", 408, "l")-->("l", 453, "s")-->("s", 645, "r")-->("r", 687, "i")-->("i", 690, "n")-->("n", 813, "o")-->("o", 855, "t")-->("t", 936, "a")-->("a", 936, "e")-->("e", 1137, " ")-->(" ", 2157, NULL)-->NULL

("(52’", 42): (Next: "C", 42) (Left: "(52", 21) (Right: "’", 21) -->