HACETTEPE UNIVERSITY

Computer Science and Engineering Department

BBM203 : Software Laboratory I

Fall 2020, Section: 1

Programming Assignment IV

TREES & HUFFMAN ENCODING

Mehmet Giray Nacakcı
Number: 21989009
Due : 31.12.2020
TA: Yunus Can Bilge

1. Software Design Notes

1.0 Problem Definition

A text compression and decompression system based on Huffman Algorithm is to be developed.

Txt input files will be processed according to commands received from terminal during runtime; results are outputted to terminal.

1.1 Data Structures

- vector<string> allCharactersOfFile;

Character sequence of most recently processed input file. It is a vector since input size is unpredictable.

- LinkedList listOfTreeToBeMerged;

With encode command, a tree leaf is created for each unique character of allCharactersOfFile. Leafs and subtrees are temporarily held in listOfTreeToBeMerged till they are all merged into one encodingTree. A custom linked list class is implemented to provide functionalities such as finding the leafs with least frequencies and merging leafs.

- BinaryTree encodingTree;

Every TreeNode is already a subtree itself; however, this object holds the main root.

- vector<EncodedCharacter*> encodedCharacters;

Leaves of encodingTree are traversed and each character is mapped to its binary encoding determined by its position on the tree. Characters and their encodings are kept in individual EncodedCharacter objects. encodedCharacters is a vector since size is unpredictable.

1.2 Algorithms

1.2.1 Encoding Algorithm

0.1 Clear data (encodedCharacters) remaining from the previous encoding.
0.2 Extract new character sequence (allCharactersOfFile) from input file.
(Compression.cpp :: 24 - 28)

Compression::createTreeLeavesAndTheirFrequencies( ) :

1. For each character of allCharactersOfFile :
1.i.1 If the character already exists in the list of leaves (listOfTreeToBeMerged), increment its frequency value.
1.i.2 Else: Create a leaf (treeNode) for this new character and append it to the list.

Compression::createEncodingTree( ) :

2.1 Until all treeNodes in listOfTreeToBeMerged are merged into one :
2.i.1 Search for the two treeNodes with minimum frequencies among the list.
2.i.2 Merge those two nodes into one.

2.2 Assign this one unified node to the root of encodingTree.

Compression::buildEncodingMap(TreeNode* recursiveNode, ...) :

3. Recursively traverse the encoding tree and fill encodedCharacters :

3.0 Take recursiveNode as argument (root of encodingTree by default).
3.0.1 Return from this call if recursiveNode is null.

3.1 Call step 3 with left child of recursiveNode and append "0" to encoding.

3.2 If reached a leaf : (Since only leaf nodes hold characters,) Get the character of the leaf; along with the encoding, construct an encodedCharacter and append it to encodedCharacters.

3.3 Call step 3 with right child of recursiveNode and append "1" to encoding.

Compression::encodeTheTextSequence( ) :

4.1 For each character of allCharactersOfFile :
4.1.j.1 Find the character in encodedCharacters, and get its binary encoding.
4.1.j.2 Output the character encoding to terminal.
(Compression::printAndGetCharacterEncoding(character))

4.2 Save the encoding to "encoded.txt" so that decoding can be directly done by decoding "encoded.txt".

5. clear the input container "allCharactersOfFile" for upcoming file reads.
(Compression.cpp :: 32)

1.2.2 Decoding Algorithm

0. Extract binary character sequence (allCharactersOfFile) from input file.
(Compression.cpp :: 46)

Compression::decodeTheTextSequence( ) :

1. Assign root of encodingTree to traverseNode.

2. For each character of allCharactersOfFile :

2.i.1 case "0": assign its own left child to traverseNode
2.i.2 case "1": assign its own rigth child to traverseNode

2.i.3 If traverseNode reached a leaf (Only leaf nodes hold characters) :
2.i.3.1 Print to terminal: the character this leaf holds.
2.i.3.2 Assign the root of encodingTree to traverseNode. (Back to traverse origin.)

3. Clear the input container "allCharactersOfFile" for upcoming file reads.
(Compression.cpp :: 50)

1.2.3 Tree Listing (BinaryTree::listTree(TreeNode* recursiveNode = root)) Recursive Algorithm

Example Tree Listing Format:

```
-> Root
   -> R
   |   -> R
   |   |   -> R: "i"
   |   |   -> L: "r"
   |   -> L
   |       -> R: "h"
   |       -> L: "\n"
   -> L
      -> R: " "
      -> L: "a"
```

Tree will be listed in Parent – Right child – Left child order traversal, in order to make right branches appear at top, since I thought this would be intuitive.

1. Print indentation sequence of this line, which has a length proportional to recursion depth of this call.

1.1 callHistoryFor_HorizontalIndentation determines how many recursive calls were made and were they towards right or left child ("1" or "0"). This distinction is useful since the vertical lines are wanted only between a right child and a left child.

1.2 For each character of callHistoryFor_HorizontalIndentation :
1.2.i.1 case "1" : Print vertical line "|" and indentation. ("|" below a "R")
1.2.i.2 case "0" : Print blank indentation.

2. Print treeNode title which is "R" (right child) or "L" (left child) or "Root".

3. If recursiveNode is a leaf, print the character it holds to terminal. (Since only leaf nodes hold characters.)

4. Call listTree with right child of recursiveNode and append "1" to call history.

5. Call listTree with left child of recursiveNode and append "0" to call history.

2. Testing Notes

Build command: g++ -std=c++11 *.cpp -o Main
Run command: ./Main

Terminal commands are running successfully. When test texts are encoded and decoded back, they are unchanged (except being all lowercase).

Terminal Commands

No command other than *encode* or *end* will take action unless anything has been encoded yet. This is checked by checking if the vector encodedCharacters is empty.

2.1 -i fileToCompress.txt -encode

Character sequence gets extracted from file, encoding algorithm (1.2.1) gets invoked. Outputs the encoded text to terminal, also to "encoded.txt" for easier decoding.

2.2 -s character_ToGetTheBinaryEncodingOf

Search and find the character in encodedCharacters vector, then print its encoding. (Compression::printAndGetCharacterEncoding (characterToGetTheBinaryEncodingOf))

2.3 -i encoded.txt -decode

Encoded sequence gets extracted from file, decoding algorithm (1.2.2) gets invoked. Since encoding algorithm saves its output into "encoded.txt", that file can be used directly, for convenience.

2.4 -l

(lowercase L). Invokes listTree function, which was elaborated in 1.2.3.

2.5 end

To safely terminate the program without memory leaks.

3. Submission Notes

I am not confident regarding if the .exe files I provided are properly working. Since I do not own a Windows machine, I compiled the two .exe files below, by using

g++ -std=c++11 *.cpp –o Main.exe

command on DEV machine and also x86-64 MacBook.

- exe_MadeOn_DEV.exe
- exe_MadeOn_MAC.exe