Department of Computer Science

CS260 Data Structures

Winter 2022

Programming Assignment 4

1 Regulations

Due Date : 02/25/2022 - 11:59pm
Late Submission: 10% off for each late day, with at most 2 day late submission
Submission : via Gradescope.
Team : The homework is to be done and turned in individually. No teaming up.
Cheating : All parties involved in cheating get zero from assignment and will be reported to the University.

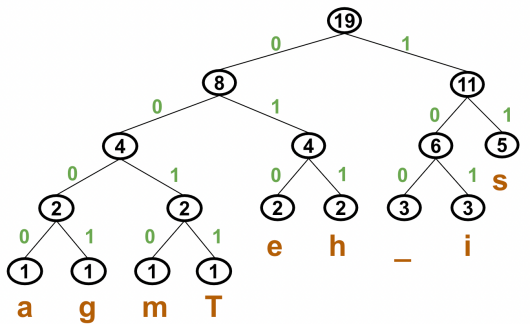
2 Lossless Data Compression and Huffman Coding

ASCII, abbreviated from American Standard Code for Information Interchange, is a character encoding standard for electronic communication. ASCII codes are used in representing text in computers and telecommunications equipment where each character is encoded with 8 bits. Since there are 256 different values that can be encoded with 8 bits, there are potentially 256 different characters in the ASCII character set. As an example, the letters a,b,c are encoded in ASCII as 'a' = 01100001, 'b' = 01100010, and 'c' = 01100011.

Data compression allows encoding information economically, which becomes handy when storing large files as well as for transferring them over a network. An efficient way of compressing text files is by accounting for characters that are more frequent than others in a certain file. Huffman code is one such approach that is commonly used for lossless data compression. You can find information about the logic of Huffman Coding along with a pseudo-code of the encoding algorithm in the recommended reading text Algorithms Unlocked, Thomas H. Cormen, Chapter 9 (available online via Drexel Library). You can also benefit from this website to visualize animation of how Huffman encoding algorithm will work for a given input text. Also, refer to the slides on Huffman Coding from class resources.

Text to be encoded: This is his message

Character	a	g	m	T	e	h	_	i	s
Frequency	1	1	1	1	2	2	3	3	5



Code Table	
s	11
e	010
h	011
_	100
i	101
a	0000
g	0001
m	0010
T	0011

Steps to generate Huffman code table for the sentence *This is his message.*

In this assignment, you will write a C program that encodes a given text file using Huffman Coding and decodes the encoded file that you generated. While doing that, you are strongly suggested to use heaps and binary trees as part of your implementation.

3 Roadmap

You will write a program that does Huffman 1) encoding and 2)decoding of a given text file. Below is a rough roadmap of what is needed to be done to achieve this goal.

1. **Encoder:**
   (a) You will read input from a text file which consist of alphanumeric characters and punctuation marks (a sample text file will be provided).
   (b) Calculate the frequency of each character in this text file.
   (c) Implement a priority queue data structure to be used in the Huffman Coding algorithm. (Note: You need to determine what will be the structure of the data to be kept inside the priority queue)
   (d) Implement a binary tree data structure to be used in the Huffman Coding algorithm. (Note: you need to determine what will be the node structure of the binary tree)
   (e) Implement Huffman Coding algorithm, as described in the referenced text in the previous section, by using the priority queue and binary tree that you implemented.
   (f) Run the Huffman Coding algorithm on the set of character frequencies that you calculated to obtain a Huffman Code tree.
   (g) Parse that tree to generate a Huffman Code table, where each character is assigned a binary code. (Note: you might want to consider storing your code table in a *closed hash table* to speed up your encoding in the next step)
   (h) Parse the input text file and encode the text in binary using the Huffman codes that you saved on your code table.
   (i) Write the Huffman code table and the encoded text into two separate files. You should store the Huffman code table in a way that you can easily reconstruct the code tree when the decoder part of your program is run.
   (j) Calculate compression statistics and print it to standard output. Your statistics should consist of the size of the original file in terms of bits (number of characters * 8, as we assume that the text is initially encoded using ASCII), the size of the encoded text in bits (i.e., count of 0/1 bits that you generated in your encoded text), and the compression ratio (the ratio of these two numbers in percentage).
   (k) For debugging purposes, you should write a print function for your Huffman code tree, that traverses the tree and prints the nodes . (Note: pre-order printing might be a good idea. Anything else that works you visualize the tree is also fine)
2. **Decoder:**
   (a) Read the Huffman code table and reconstruct the Huffman Code tree (which will be a binary tree!).
   (b) Implement Huffman *decoding* algorithm.
   (c) Read the encoded text file, parse it by going through the Huffman tree and decode the characters in file using your algorithm.
   (d) Write decoded text into a new file.

4 Starter Package

- Along with this assignment instructions document, you will be provided with:
  - a **starter code** that has a sample function to read input from file and another sample function to write into file. You might need to change these functions to match the input/output specifications of this assignment. These functions are given to you to merely serve as a helper.
  - a sample **text input file** that consists of a sentence written with alpha numeric characters, space, and punctuation marks. Test cases will not include new line character, so you don't need to worry about that special case.
  - a simple **makefile**, that compiles the code to generate an executable
  - an **executable** *huffman* **program**, which you can compare against your program to match input/output specifications.
  - a sample **code table** output file and a sample **encoded version of the text input** file that are generated by the executable that is provided. You can take these as a reference for what the output would look like.
  - a starter **report file in LaTeX**, which you will fill with the analysis of your assignment.

5 Input/Output Specifications

- Your **executable** "huffman" is going to take **four command line arguments** depending on its **mode** of encode or decode.
  - **Encode:** In order to encode a file, your program will accept **four** command line arguments as follows:

    /> ./huffman encode plainText.txt codeTable.txt encodedText.txt

    where *encode* indicates that the program will be in encoding mode, *plainText.txt* will be the path of the file that contains the text that you will encode (any file path can come here!), *codeTable.txt* will be the path of the file that you will write the code table to, and *encodedText.txt* will be the path of the file that you will write the encoded text to.
  - **Decode:** In order to decode a file, your code will accept **four** command line arguments as follows:

    /> ./huffman decode codeTable.txt encodedText.txt decodedText.txt

    where *decode* indicates that the program will be in decoding mode, *codeTable.txt* will be the path of the file that you will read the code table from, *encodedText.txt* will be the path of the file that you will read the encoded data from, and the *decodedText.txt* will be the path that you will write the decoded text to.
- **Structure of the files** will be as follows:
  - **plainText.txt** : will be a text file that contains alphanumeric characters and punctuation marks. The input text will not include a new line, to make the implementation a bit easier. The text will end with and EOF, which you do not need to encode. A sample file will be provided along with this instruction file. Below is an example text that could be in a text file:

    She sells seashells.
  - **codeTable.txt**: will contain the code table, where each line will consist of a character (space character will be represented as a white space), the binary Huffman code to represent this character, and the frequency of the character in the file represented with an integer number, each separated by comma. The *codeFile.txt* should be as follows for the example sentence above (note that, depending on how you order the ties, Huffman codes can change for characters that has the same frequency).

s,10,5
l,00,4
e,111,4
,010,2
h,1101,2
S,0110,1
a,0111,1
.,1100,1

Note that the fourth line above indicates space character.

- **encodedText.txt** : will be a text file that contains the encoded text in binary. For the example provided above, this file should consist of the following:

  01101101111010101110000100101011101111011011110000101100

- **decodedText.txt** : will be the text file that contains the decoded text, which should look identical to the plainText.txt file above. Note that, you will need to put EOF to the end of each file.

- **Compression Statistics** for encoding will be printed to standard output. For the example provided above, the output should look as follows (where compression ratio is printed up to two decimal places):

  Original: 160 bits
  Compressed: 56 bits
  Compression Ratio: 35.00%

- **Report file** should provide the following information:
  - Give an overview of the roadmap you followed for implementing the encoder.
  - Explain the priority queue and binary tree that you implemented. What role did they play in the overall program?
  - Give an overview of how your decoder works?
  - Analyze and explain the running time of your encoder to encode a text with n characters.

  You will be provided with a starter LaTeX file for your report. You should answer the questions in there, and can add more information if you like.

5.1 Submission

- Your code must be running on tux. So, make sure that it compiles and runs on tux before you submit, even if you develop it elsewhere.
- Before submitting your code, compare the outputs of your program for various test data with that of the provided executable. If there exists a difference in the compression ratio, that would lead to your program failing in test cases while grading. Make sure you match the input/output specifications.
- Your submission package **must include** your **source code**, a **makefile**, and a **report file** written in pdf that explains your approach for solving the problem. Put all these material in a folder named with your userid, (example: abc123), compress it into a zip file named as your student id (example: abc123.zip). Submit this zip file through Gradescope.
- Your makefile needs to produce an executable named "huffman", once we type "make" in command line.

5.2 Grading

- Assignment is going to be graded out of 100 points.
- You will provide your **self assessment** for your effort for the **30 point** portion of the assignment.
- **50 point** portion of the remaining part will be for the **test cases** of the assignment.
- The final **20 point** will be for the **report file**.