CS21120 Assignment

Huffman Encoding System

Release date: Thursday 10th November 2016
Hand in date: Friday 9th December 2016 (16:00 via Blackboard/Turnitin)
Feedback date: Friday 6th January 2017

Aims and Objectives

The aim of this assignment is to give you experience in coding and using some basic data structures.

The objective is to implement the ***Huffman encoding scheme*** (see CS10720 lecture notes, pages 178ff) for encoding a message in a sequence of characters/symbols with the shortest possible average number of bits per character/symbol and thus minimum number of bits for the total message.

Example

The textual message "Hello!" has five different symbols: 'H', 'e', 'l', 'o' and '!'. Three bits are required to encode these five symbols, e.g.: 'H'=000, 'e'=001, 'l'=010, 'o'=011, and '!'=100. This is a brute-force fixed length encoding scheme, and treats different symbols equally, no matter how frequently they appear in the message.

The *Huffman* encoding scheme assigns fewer bits to symbols which have high frequency of occurrence and more bits to symbols which have low frequency of occurrence and is therefore a *variable length encoding scheme.* It achieves an optimal average number of bits per symbol for the whole message. However, the encoding scheme is not unique. For the example given above, it could create an encoding book

H=110, e=111, l=0, o=101, !=100 or H=00, e=01, l=10, o=111, !=110

For further information about Huffman encoding, please refer to the CS10720 lecture notes, pages 178ff (a copy of these notes is included along with this assignment specification on Blackboard).

Overview

Given a message in a sequence of characters/symbols, Huffman encoding is one of the best methods to encode the message in an optimized number of bits per character/symbol. It works by:

(i) Determining the frequency of the occurrence of each symbol inside the message.

(ii) Ranking the symbols in order of increasing frequency.

(iii) Combining the two lowest frequency symbols into a single composite symbol for each step. The frequency of this new symbol is therefore the sum of the two original ones and the process is then repeated until a single composite symbol remains.

(iv) Starting at the final symbol, it breaks-up the composite symbols, assigns 0 to the left branch and 1 to the right branch and then creates a Huffman binary tree.

(v) The binary string from the root node to the leaf node will be used to encode the symbol at the leaf node.

Requirements

You are required to implement a system which will take a text file as input, generate an optimal encoding book for all of the symbols contained in the message and analyse the results obtained. You should use appropriate data structures to accomplish this task. The Java API provides a number of implementations of common data structures, e.g.: Queues, Priority Queues, Maps, Heaps, Stacks, etc. and you can use these to implement your system. A number of files are provided with this assignment for the purpose of testing your Huffman tree implementation. You need to provide statistical textual output which describes your tree (height, number of nodes, average depth). After the optimal Huffman code has been generated, it must be applied to encode the given text and the encoding length must be compared with the encoding length of the brute-force fixed length encoding scheme in order to show the extent of the compression.

Marking

This assignment comprises 50% of the total mark for the module. It is marked out of 100. The following guidelines will be used for marking:

1) *Program design and implementation (50%):*
    a. You should implement some form of file-handling that will allow you to read-in the file to be processed and perform frequency counting of the characters.
    b. Build a Huffman tree. In order to do this, a suggestion is to define a node, which contains information such as the symbol being encoded, frequency, index, left and right children. To build the tree, you should use appropriate data structures and methods in order to allow the searching for the trees with smallest frequencies.
    c. Implement a Huffman encoding scheme and generate an optimal binary string representation for each symbol. To build the code book, you should assign the binary symbols 0 and 1 to the left and right branches of the Huffman tree respectively and recursively.
2) *Program evaluation (30%):*
    a. The Huffman encoding scheme should be implemented correctly. Use the text files provided to test your implementation and print out (in this order) uncompressed file size, compressed file size and the compression ratio (*uncompressed file size / compressed file size*) to the console followed by statistical textual output which describes your tree (height, number of nodes, average depth; in this order). The *uncompressed file size* should be determined based on the brute force fixed length encoding scheme and is the length of the corresponding encoding. The *compressed file size* is the length of the Huffman encoding. Note that this is different from the size of the actual file. You should comment on the approach that you used in order to calculate the result.
    b. You will have used a number of different data structures in order to implement your particular system. You must comment on the suitability of your choices in your report and the time and space complexity related to the operations required in order to fulfil the specification.
    c. Discuss advantages and disadvantages that the brute force fixed length and Huffman variable length encoding schemes have.

3) *Report (20%):*
    a. A report detailing the design and implementation needs to be submitted in .pdf file format.
    b. It should describe in detail the code-structure and design: different classes, methods, definitions, parameters, and relationships (if any).
    c. The report should be split into sections, e.g. Introduction, System design, System implementation-and-operation, time and space complexity of the data structures you have used and conclusion. You should acknowledge any additional 3rd party material that you use in footnotes.
    d. You should include a short passage at the end of your report which contains a self-assessment of your work along with a short description of what you might do if you could implement your system in a different way.
    e. 2,000 (± 10%) words (excluding figures, tables, diagrams, code excerpts, and references) are appropriate for the report. All figures, tables and diagrams should include captions, and be numbered and cited where appropriate in the main text.

Submission

For this assignment you are asked to submit
(1) *.java* files of source code via Blackboard
(2) a report in .pdf format via Turnitin.

The main class ***must*** be named *<uid>CS21120Assign.java, the report must be named <uid>CS21120Assign.pdf*, where*<uid>* should be replaced with your user ID. For example, mine would be called *yylCS21120Assign.java and yylCS21120Assign.pdf*. All the .java files should be zipped together as *yylCS21120Assign*.zip for submission in one go. These files should include your user interface *and* Huffman encoding scheme implementation, expansion and any support classes (these could be inner or just private), such as the node *Comparable* class, and the data you used for testing. To facilitate the marking, make sure that you follow such naming rules and your code can be run using the following command line: java <uid>CS21120Assign. Any deviation from such naming requirement will result in a deduction of 10% of the total marks awarded.

Resources

For this assignment you are permitted to use classes from the java collection framework (java.util.* package). You are not allowed to use code or classes from other sources and must implement your own classes. Various resources can be found online however, you must not just copy and paste any material. Any resources you use to help you to learn must be acknowledged in your report. As with all such assignments, the work must be your own. Do not share code with your colleagues and ensure that all of your own code is securely stored and is not accessible to your classmates. Any sources you use should be properly credited with a citation, and any help you receive (e.g. from demonstrators) should be acknowledged appropriately. Your report must accurately reflect what you have achieved, and any discrepancies between your code and report could be treated as academic fraud. Your attention is drawn to the plagiarism warning given in the student handbook.

Marking grid

(This is just a guide)

Components	70-100%	60-70%	50-60%	40-50%	0-40%
Program design and implementation (50%)	Well documented, explained and presented in the report. Easy to read source code with intuitive structure, naming and coding conventions.	Relatively detailed documentation, with some details missing. Good overall source code structure but with minor omissions or missing detail.	Comprehensible documentation, and source code but with some important missing detail.	Basic documentation, some effort is required to follow both documentation and source code	Poorly documented, explained, or presented, poor source code.
Program evaluation (30%)	Comprehensive and convincing	Very good, showing some insight.	Good, giving explanations	Basic, restatement of results	Largely missing
Report (20%)	Well documented, showing impressive work	Most work has been documented with sufficient detail	Some work has been documented with understandable detail	Basic work has been documented with limited detail.	Incomplete, various parts are missing

Yonghuai Liu, Christine Zarges, and Neil Mac Parthalain
10th November 2016