Project 1: Huffman Coding

326.212, Autumn 2014
Seoul National University

This project is due 6pm Friday, 11/14/2014. Upload your project into the project collection folder on `eTL`. **No late submission is accepted**.

Introduction

Consider data that consists of a stream of symbols from a finite alphabet. **Compression algorithms** are the class of algorithms that encode those data such that they can be stored and transmitted with minimum size. **Huffman coding** is a *lossless*[1] compression algorithm created by David A. Huffman in 1952. This coding algorithm is a theoretical basis of the 'zip' file format. It has several special properties:

- **Variable length code**: Each symbol has different length of code, and frequent symbols are encoded with shorter codes.
- **Prefix code**: Code for a symbol cannot be a prefix of another code. For example, if the symbol 'A' is encoded by '10', no other code for a symbol cannot begin with '10'.
- **Optimality**: In terms of expected value of length of code, Huffman coding is the best coding algorithm among all algorithms that encode each symbol separately.

Assume that the symbols are sorted according to the frequency of occurrence in ascending order (suppose we know the frequencies). The procedure to generate the code involves the following construction of a binary tree.

1. Initially all symbols are leaf nodes.
2. The two symbols with the lowest frequencies are joined to create a composite symbol whose frequency of occurrence is sum of the two symbols joined. This forms a parent node that corresponds to the new composite symbol with its children being two merged symbols. This new node is now treated as a new symbol.
3. The symbols are sorted again by new frequency and the procedure is repeated until there is only one node left. The single remaining node is the root node corresponding to the composition of all the original symbols.

If there were N original symbols, we need $N - 1$ iterations to complete the tree construction. Now, left branch is labelled with '0' and right branch is labelled with '1'. The code for each symbol is the string of '0's and '1's on the path from the root node to the leaf node corresponding to the symbol.

[1] lossless compression: compression algorithm taht allows original data to be perfectly recontructed from the compressed data.

Example

Let the alphabet $X = \{1, 2, 3, 4, 5\}$, and frequency of occurrence is 0.25, 0.25, 0.2, 0.15, 0.15, respectively. First, the two symbols with lowest frequencies, '4' and '5' are merged to form a composite symbol '4 5', whose new frequency is 0.3 . Next, the two nodes with lowest newly calculated frequencies are '3' and '2', forming a new composite symbol '2 3'[2], with frequency of 0.45. Proceeding this way, we obtain the following table and coding tree.

Codeword Length	Codeword	X	Frequency				
2	10	1	0.25	0.3	0.45	0.55	1
2	01	2	0.25	0.25	0.3	0.45	
2	00	3	0.2	0.25	0.25		
3	111	4	0.15	0.2			
3	110	5	0.15				

Table 1: An example of Huffman coding table

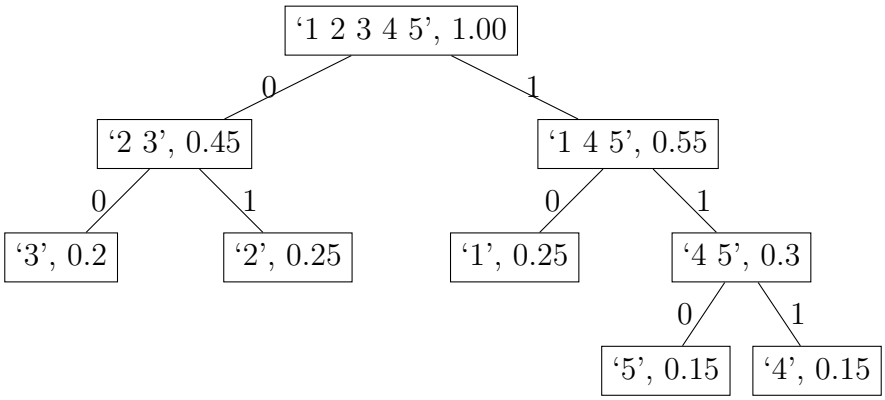

Figure 1: An example of Huffman coding tree

[2]'2' and '1' have the same frequencies here, and '2' was deemed to have lower frequency. Basically, one may use any kind of tiebreaker. However, for this project, we will use the method described in Task B.

Task A: Decoder

In this task, we are to implement a Huffman decoder.

Things to do

From the MSVS solution provided, find the file `decode.c` in the project named `decode`.

1. Read the sample code to understand the overall structure and how it implements the algorithm.
2. Construct a Huffman coding tree from the mapping between symbols(in terms of ASCII code) and its binary string. The mapping is given in `code.txt`.
   (a) Complete the function `build_tree()`.
3. Fill in the missing code to generate the decoded output from the encoded string and symbol tree.
   (a) Hint: for each string, traverse the tree and output a symbol when a leaf node is encountered.
4. Fill in the missing code (if any) to free all the resources and memory before exiting the code.

Outputs

The program prints the output on the file `decoded.txt`. The correct output is `agedeggfacedbadcabbage`.

Task B: Encoder

In this task, we are to implement a Huffman encoder. For simplicity, assume that the only symbols that can be used are 'a', 'b', 'c', 'd', 'e', 'f', 'g'. Also assume that the symbols' frequencies are 0.01, 0.04, 0.05, 0.11, 0.19, 0.20, 0.40, respectively.

Things to do

From the MSVS solution provided, find the file `encode.c` in the project named `encode`.

1. Read the sample code to understand the overall structure and how it implements the algorithm. Especially, pay attention to how a priority queue is used in the program, and how the tree is built bottom-up.

2. *Implementation of priority queue*: During each iteration, we need to keep track of the two (composite) symbols with the lowest frequencies. This can be done by using a priority queue. For its general use, it is implemented with a heap, in which insertion and removal takes $O(\log n)$ time. However, for simplicity, we will use a sorted linear linked list for this project. When we insert a new symbol to the linked list, we will insert it in the correct position so that the linked list will be kept sorted all the time. Then, removing the symbol with lowest frequency is simple. `encode.c` contains template code that implements a priority queue. `pq_pop()` is a function that removes the lowest-frequency symbol, and `pq_insert()` inserts a new symbol to the priority queue. To help debugging, there also is a function `pq_display()`. Complete `pq_pop()` and `pq_insert()`.

   (a) `pq_insert()`: Consider three conditions.
      i. queue is empty
      ii. new element goes before the beginning
      iii. new element goes at the end or in the middle of the queue

   (b) `pq_pop()`: Make sure you update the pointers for the element to be removed.

3. Finally fill in the missing code (if any) to free all resources and memory before exiting the code.

Additional Conditions

1. When merging two nodes, we will assign '0' to the symbol with lower frequency, and '1' to the one with higher frequency.
2. Tiebreaker: when two or more symbols have the same frequency,

   (a) Basic symbols will be deemed to have higher frequencies than composite symbols.
   (b) Basic symbols with lower ASCII code will be deemed to have higher frequencies.
   (c) Composite symbols created earlier will be deemed to have higher frequencies.

Output

The output is printed on the file `code.txt`. Your output should match the reference values below.

ASCII	symbol	code	ASCII	symbol	code
97	a	10000	101	e	110
98	b	10001	102	f	111
99	c	1001	103	g	0
100	d	101			

Task C: Encoding and decoding a text file

So far, we have assumed that symbols and their frequencies are given. However, in this task, we are to compute set of symbols and their frequencies from a large text file.

Things to do

In the MSVS solution provided, there are empty projects `encode_file` and `decode_file`. You are to modify `encode.c` and `decode.c` properly in order to encode and decode a text file. For example, 'pg1342.txt' and 'pg28233.txt' and their desired output is provided in 'Release' folder.

1. `encode_file`
   (a) Read in a text file to compute frequency of each symbol used in the file. Input file names and output file names should be accepted through user input. For example:

   ```
   Input text file name: pg1342.txt
   Input code file name: code_1342.txt
   Input encoded file name: encoded_1342.txt
   ```

   (b) *Set of symbols*: characters that appear in the text whose ASCII code corresponds to 0-255 will be encoded. Characters that do not appear in the file should not be encoded.
   (c) *input file*:
      i. a text file to encode (e.g. `pg1342.txt` provided in the directory named `Release`.)
   (d) *output files*:
      i. a code table (e.g. `code_1342.txt`)
      ii. an encoded text file that contains a string of '0's and '1's (e.g. `encoded_1342.txt`)
2. `decode_file`
   (a) Modify the internal algorithm if necessary
   (b) Input and output file names should be entered as user input. For example:

   ```
   Input encoded file name: encoded_1342.txt
   Input code file name: code_1342.txt
   Input decoded file name: decoded_1342.txt
   ```

   (c) *input files*:
      i. encoded file(e.g. `encoded_1342.txt`)
      ii. code table (e.g. `code_1342.txt`)
   (d) *output file*:
      i. reconstructed (decoded) text file (e.g. `decoded_1342.txt`)

Outputs

`pg1342.txt`[3] and `decoded_1342.txt` should be identical. Also compare your code table to the reference code table provided. This file only includes ASCII characters between 0-127. `pg28233.txt`[4] also includes some symbols from range 128-255. We will try multiple input files for grading.

Further instructions

1. All the programs submitted will be compiled on Microsoft Visual Studio 2010 Professional Edition installed on Windows 7 in **release mode**, not in the debug mode. Please test your program sufficiently on release mode before submission.

2. It is encouraged to verbally discuss ideas with other students. When you do so, **acknowledge the students with whom you discussed your work**. However, sharing code is a serious misconduct. **Do not look at anyone else's code, and do not show anyone your code.**

3. You will have **one and only one chance to submit** your project. Review your work carefully before your final submission.

[3] *Pride and Prejudice* by Jane Austen
[4] *Philosophiæ Naturalis Principia Mathematica* by Isaac Newton