159.333 INDIVIDUAL PROJECT

RESEARCH OF DATA COMPRESSION ALGORITHMS IN C++

SUPERVISED BY: ANDRE BARCZAK
SUBMITTED BY: HAOMIN LIU 12109377
DATE: 24/06/2015

ABSTRACT

Data compression is one of the most important areas of computer science. It enables devices to transmit data with fewer bits (saving time and storage space).

In this project, the two crucial lossless data compression algorithms (Huffman coding and LZW coding) were investigated and implemented.

For Huffman coding program, the entropy (information redundancy), compression ratio and compression time based on different user defined bit lengths have been discussed. An optimum method which can potentially improve the compression ratio has been proposed. For LZW coding program, the entropy, compression ratio and compression time have been discussed as well. Design of how to choose these two algorithms based on different circumstances have been discussed.

Key words: data compression, Huffman coding, LZW coding,, entropy, compression ratio, compression time.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I would like to express my deepest appreciation to my supervisor, Dr. Andre L. C. Barczak for his patient guidance, support and encouragement to complete this 159.333 individual project. Without his leading, I may never have an in-depth research in data compression field. I would also like to express my gratitude to all my lectures for their support and teaching during my studies of bachelor degree at Massey University.

1. INTRODUCTION

1.1 BACKGROUND

This project was conducted for Massey university 159.333 Individual Programming Project paper. The main purpose of this proposed project is about researching and implementing data compression algorithms.

The objectives of the project are:

1. Research and apply data compression algorithms involves C++ and data structures.
2. Implement compression algorithms and compare effectiveness of implementations.
3. Make extensions on current methodological approaches and test designs of developed algorithms.

In this project, two lossless data compression algorithms were implemented and discussed. Lossless data compression uses statistical redundancy to represent data with lossless information.

Lempel-Ziv (LZ) is one of the most popular lossless compression algorithms. This is a string coding algorithm. LZW (Lempel–Ziv–Welch) is mainly used to compress GIF files. In order to minimize the number of bits in a file, the encoders can operate in a variable-rate mode and they can hide in compressed data (A.Lempel, 1978). The original data can be decoded using string table.

Also, there is an optimum method of compression is to transmit messages in their place of symbols. (A.Huffman, 1952) Huffman coding is also a common lossless algorithm which is widely used in entropy encoding. The process of this particular algorithm is to assign variable-length code to characters (using different arrangement of 0 or 1 to represent characters). Length of the code depends on the frequency of the corresponding character, the most frequent character gets the smallest code and the least frequent character gets the longest code. LZW and Huffman coding algorithms are the starting points of this project. After implementation and testing of these two algorithms, if there is a need for comparing other algorithms may be explored as well.

2. LITERATURE REVIEW

In this section some compression-related research in the areas of information theory and data compression algorithm will be discussed. For this project, entropy and lossless compression algorithm are the main topics to be discussed. Therefore, a brief introduction of data compression and lossless compression will be discussed accordingly.

2.1 GENERAL IDEA OF DATA COMPRESSION

2.1.1 SCHEMA OF DATA COMPRESSION

In computer science and communication theory, data compression or source coding is the art of encoding original data with specific mechanism to fewer bits (or other information related units). For example, if the word "compression" in previous sentence is encoding to "comp", then this sentence can be stored in a text file with less data. A popular living example is ZIP file format, which is widely used in PC. It not only provides compress functions, but also offers archive tools (Archiver) that can store many files in a same root file.

Data compress communication works only when information sender and receiver both understand encoding mechanism (G.E.Blelloch, 2013). For instance, this report makes sense if the recipient knows it need to be explained in English characters. Some compression algorithms using this feature (data is encrypted during compression) to ensure that only authorized party can get data accurately.

Data compression is possible because most real-world data contain large amount of statistical redundancy. Redundancy can exist in various forms. For example, in a text file, the letter "e" in English appears more frequently than the letter "q". The possibility is dramatically reduced if the letter "q" followed by "z". There also exists data redundancy in multimedia information. Basically, data redundancy has following types:

1. Spatial redundancy.

(The static architectural background, blue sky and lawn of an image contain many same pixels. If such image is stored pixel by pixel, then it will waste a lot of space.)

2. Temporal redundancy.

(In television, animation images between adjacent frames likely contain the same background, only the position of moving objects have some slightly changes. So it is only worthy to store discrepant portion of the adjacent frame.)

3. Structure redundancy
4. Knowledge redundancy
5. Information entropy redundancy.

(It is also known as encoding redundancy, which is the entropy that data carry.)

Hence, one aspect of compression is redundancy removal. Characterization of redundancy involves some form of modeling. So for the previous English letter redundancy example, the model of this redundancy is English text. This process is also known as de-correlation. After the modelling process the information needs to be encoded into a binary bits representation. This encoding part involves coding algorithms which are applied

From the angle of economic theory, data compression can reduce expensive resources consumption, such as hard disk space and bandwidth connection. However, compression costs information processing resources, which might be expensive as well (G.E.Blelloch, 2013). Therefore, effective compression mechanism always has a tradeoff between compression capability, distortion, required computational resources and other factors.

2.1.2 CLASSIFICATION

There are many methods to compress data. Different sets of data need different compression methods (also known as encoding algorithm). It can be classified in following aspect:

1. Instant compression and non-instant compression
2. Data compression and file compression
3. Lossless compression and lossy compression

This project is mainly involves the third aspect. Firstly, Lossless data compression algorithm normally uses statistical redundancy to represent data with lossless information. This means data from a sender can be represented more succinctly without loss of information. Secondly, lossy compression algorithms use less bits to represent an image, audio or video with irretrievably lost of original data (K.Sayood, 2006). If a certain degree of fidelity loss is allowed, then further compression can be achieved. For example, when a person is watching TV, he may

not notice some details are not perfect. This is due to the image of current TV program above a certain spatial frequency; therefore, the high-frequency information is redundant for TV-like application (K.Sayood, 2006). Also, two audio sample sequence may sound like a same one, but they are not exactly same. In summary, some compression mechanisms are reversible, so the original data can be restored, this is so called lossless data compression. Other mechanisms are allowed a certain degree of data loss for achieving higher compression rate, this is known as lossy data compression.

2.1.3 PRINCIPLE AND THEORY

The theoretical basis of compression is information theory. From the viewpoint of information, compression is a process of removing the redundant information (or removing certain and inferable information) while retaining the uncertain information. That is to say, compression is using a closer description of the information essence to replace the original described redundancy; such essential thing is the quantity of information.

2.2 LOSSLESS COMPRESSION

2.2.1 INFORMATION AND ENTROPY

Entropy is a measure of information (R.M.Gray, 2013). Shanoon came up with a quantitative relationship of information that represents the basis of a mathematical model of communication process. Suppose we have an event X which is a set of outcomes of an experiment. Shanoon defined a quantity named *self-information*, then this quantity is given by

$$i(X) = log\ \frac{1}{P(X)} = -logP(x)$$

The base of the logarithm can be 2, e or 10, their corresponding units of *self-information* are bits, nats or Hartleys (R.M.Gray., 2013).The average amount of information associated with the experiment is called the *entropy H (Greek letter Eta)*, it is defined as:

$$\mathrm{H(M)} = \mathrm{E[i(X)]} =$$

$$\sum_{X=1}^{M} i(X)P(X) = -\sum_{X=1}^{M} i(X)logP(x) =$$

$$\frac{\mathrm{p0}}{\lg \mathrm{p0}} + \frac{\mathrm{p1}}{\lg \mathrm{p1}} + \frac{\mathrm{p2}}{\lg \mathrm{p2}} + \ \ldots \ + \ \mathrm{pX} - \frac{1}{\lg \mathrm{pX}} - 1$$

(M is information source, p is probability of characters appearance)

The H is the entropy of the source. It has following characteristics:

- Entropy is the interface between coding and modelling.
- Entropy is the measure of random events. This is due to it only take another coordinate of probability of such event.
- Equation for calculating the range of Entropy:
- 0 ⩽ Entropy ⩽ log(n), where n is number of outcomes
- – Entropy 0(minimum entropy) occurs when one of the probabilities is 1 and rest are 0's
- – Entropy log(n)(maximum entropy) occurs when all the probabilities have equal values of 1/n.

When each character of a character set of information source have equal probability distributions, then the entropy has a maximum value logM (where M is the number of characters in that character set) (R.M.Gray, 2013). This is known as the principle of maximum entropy.

$$\mathrm{H(M)} = -\sum_{X=1}^{M} i(X) log P(x) \leq log M$$

$$when \sum_{X=1}^{M} P(x) = 1, P(x) = \frac{1}{M}$$

Furthermore, the redundancy of discrete memory-less source implicit in non-equal probability distribution of information source (A.Lesne, 2011). This means once H (M) is less than logM, the data compression is possible. From this idea, the redundancy of information M is the difference value between maximum discrete entropy and real entropy of information source. It is given by:

$$\mathrm{R} = H_{max}(M) - H(M) = log M - H(M)$$

After investigating the mathematical model of the link between entropy of the information source, and the average number of bits required to represent the output of the source. Also, the probabilistic structure of the information source is the key when creating a binary representation. To summarize, correct estimates of the probability will adequately increase the compression efficiency.

The way to create a binary representation for the information source output, or the generation of a code for the information source, is the topic of next part-CODING AND ALGORITHM.

2.2.2 CODING AND ALGORITHM

Entropy coding is the coding without loss of original information (based on entropy principle). Information entropy is the average amount of information (uncertainty Metric) of the source. The common entropy coding are:

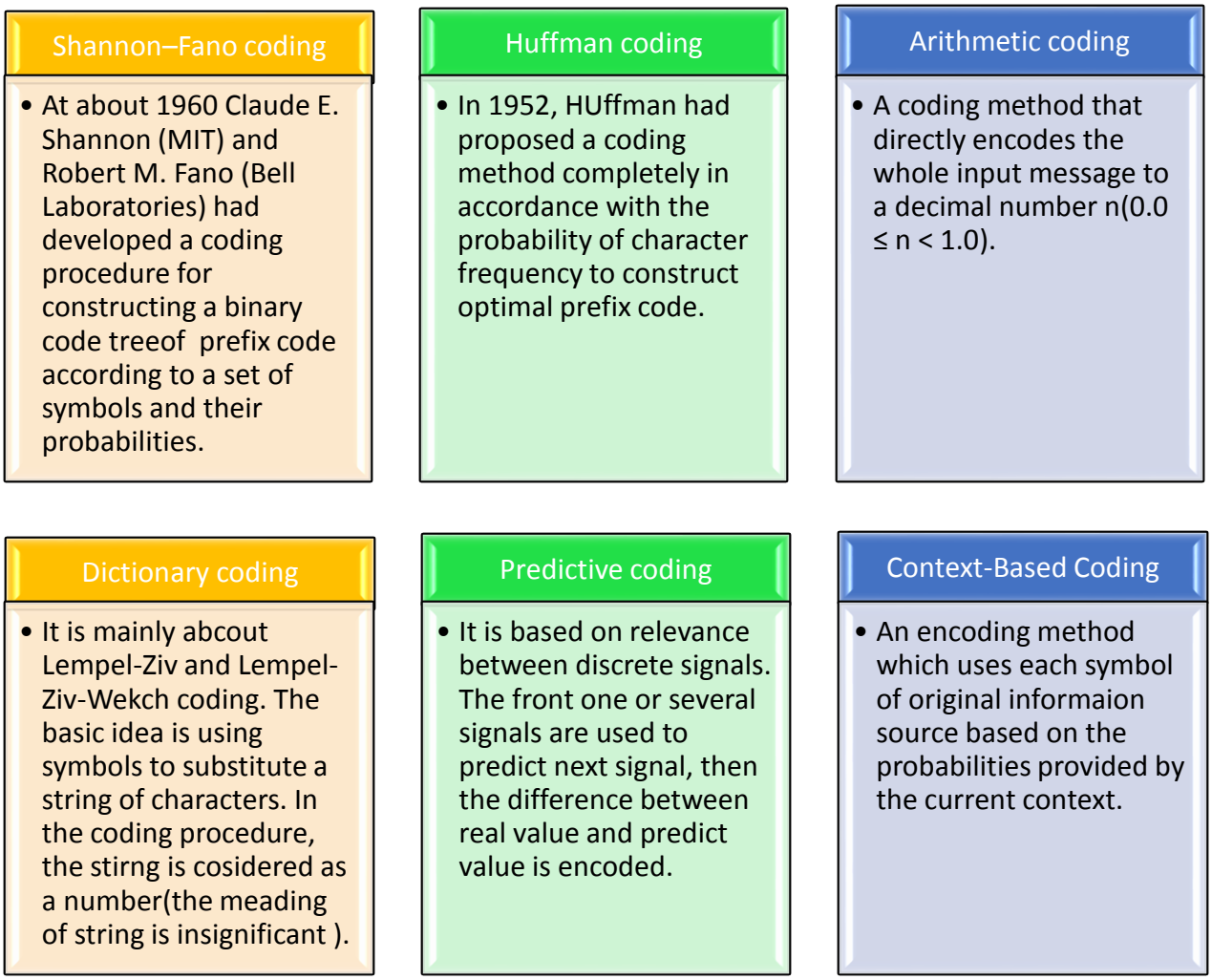


Figure 1. Common entropy coding summary

In video coding, entropy coding transforms a series of symbols of video sequence into a compression stream for transmission or storage. Then, the Input symbols may include the quantized transform coefficients, motion vector, header

information and additional information (e.g. the information of bit marker for decoding).

This project is mostly related to Huffman and LZW coding. Therefore, the details of these two algorithms will be discussed in methodology and implementation sections.

3. METHODOLOGY

This section will mainly present various methods which were used in this project. Implementations and experimental results will be discussed in next sections. All the methods were conducted for testing and making extensions on current data compression algorithms, such as Huffman and LZW algorithm. After implementation, there is a set of compression testing files called Calgary Corpus which were used for testing the efficiency of compression algorithm. A series of programs were written in C++ to make adaptation of already available device, algorithm and design, in order to research and improve the productivity of current compression algorithms. These programs are tested using Visual Studio 2013 IDE; on an Intel Core i7-4790k 4.00GHz equipped PC machine under Windows 8.1 operation system. The overall work process can be defined as following chart:

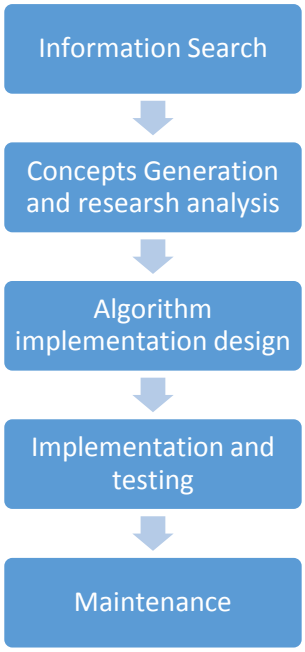


Figure 2.Propose work flow chart for data compression algorithm

3.1 INFORMATION SEARCH

Firstly, an in-depth information research was conducted for idea generation. Comprehensive and broad information was required to understand and implement the compression algorithms, in order to provide a basis to generate implementing concepts as well as the constraints to refine concepts. Therefore, information research has been carried out throughout the whole program development process from different sources by various strategies. Using the guideline given by University Libraries, University at Albany, SUNY(2015), there were two sources of to gather information: primary and secondary sources. . The information obtained from both sources can be further categorized according to their contents and origins (Figure 3).

Figure 3. The categories of information gathered in information search

3.2 CONCEPTS GENERATION AND RESEARCH ANALYSIS

Based on the information, research including schema of data compression and details of mainly focused aspects, in terms of information and entropy, coding and algorithm. A literature review of information research was represented in the previous section.

3.3 ALGORITHM IMPLEMENTATION DESIGN

A.HUFFMAN CODING

There is an optimum method of compression is to transmit messages in their place of symbols. (A.Huffman, 1952) Huffman coding is a common lossless algorithm which is widely used in entropy encoding. The process of this particular algorithm is to assign VLC (variable-length code) to characters (using different arrangement of 0 or 1 to represent characters). Length of the code depends on the frequency of the corresponding character, the most frequent character gets the smallest code and the least frequent character gets the longest code (J.Zelenski, K.Schwarz., 2012).

Huffman coding procedure can be illustrated as the following figure:

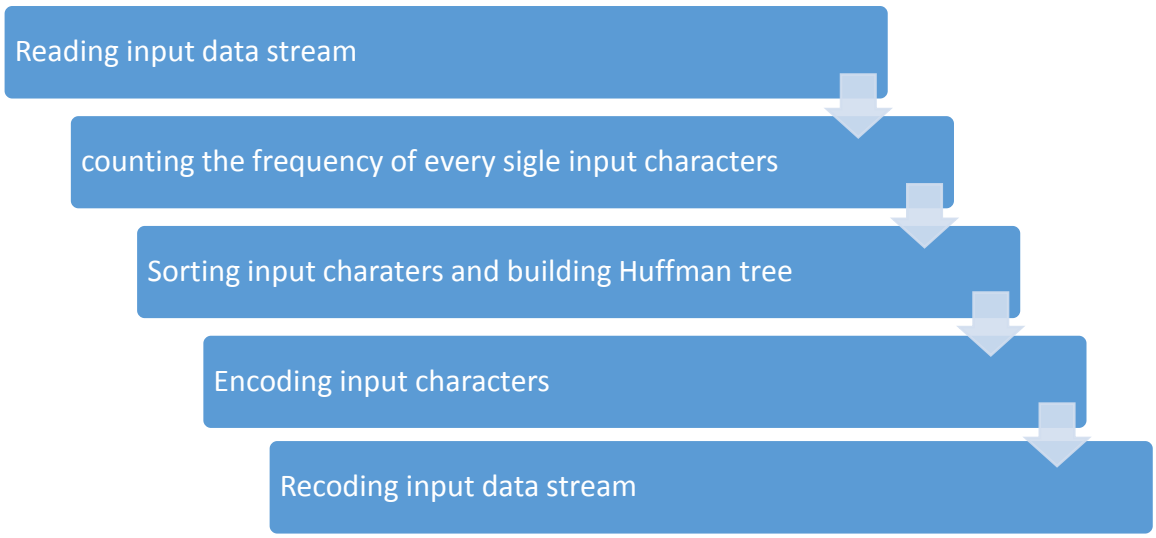


Figure 4.Huffman coding procedure

The essence of the Huffman algorithm is the recoding of original information source based on statistic results of character occurrence frequency; instead of dealing with the repetition of the characters or the repetition of the sub strings. In

practical applications, the occurrence frequency of original symbols cannot be predicted (T.H.Cormen, C.E.Leiserson, R.L.Rivest, C.Stein, 2009). The procedure needs two steps (statistical counting and coding) which are slow and impractical. Conversely, there is an adaptive (or dynamic) Huffman algorithm that needs no statistical counting. It can dynamically adjust Huffman tree during compression which improves the processing rate significantly. Therefore, Huffman coding has high productivity, fast operation speed and flexible implementation procedure.

B.LZW CODING

Lempel-Ziv (LZ) is one of the most popular lossless compression algorithms. This is a string coding algorithm. LZW (Lempel–Ziv–Welch) is mainly used to compress GIF files. In order to minimize the number of bits in a file, the encoders can operate in a variable-rate mode and they can hide in compressed data (A.Lempel, 1978). The original data can be decoded using string table.

LZW compression algorithm has three main objects: data stream, encoded stream and dictionary. At the encoding step, data stream is the input object (e.g. text files according to sequence) while encoded stream is the output object (encoding data after operations) (K.Sutner, 2003). Rather, in the decoding step, encoded stream is the input object and data stream is the output object. Additionally, dictionary is both needed for encoding and decoding stream (T.H.Cormen, C.E.Leiserson, R.L.Rivest, C.Stein, 2009).

The basic principles of LZW algorithm is extraction of characters from the original information source, then a dictionary table is built to create index for referencing and replacing characters corresponding to original data (A.Lempel, 1978).

The implementation details of both algorithms will be discussed in the next section.

4. IMPLEMENTATION RESULTS AND DISCUSSION

4.1 IMPLEMENTATION DETAILS OF HUFFMAN CODING

4.1.1 IMPLEMENTATION PREPARATION

Huffman coding is basically an algorithm of representing data in another representation to reduce the information redundancy. The data is encoded in binary.

The Huffman algorithm can be illustrated as following flow chart:

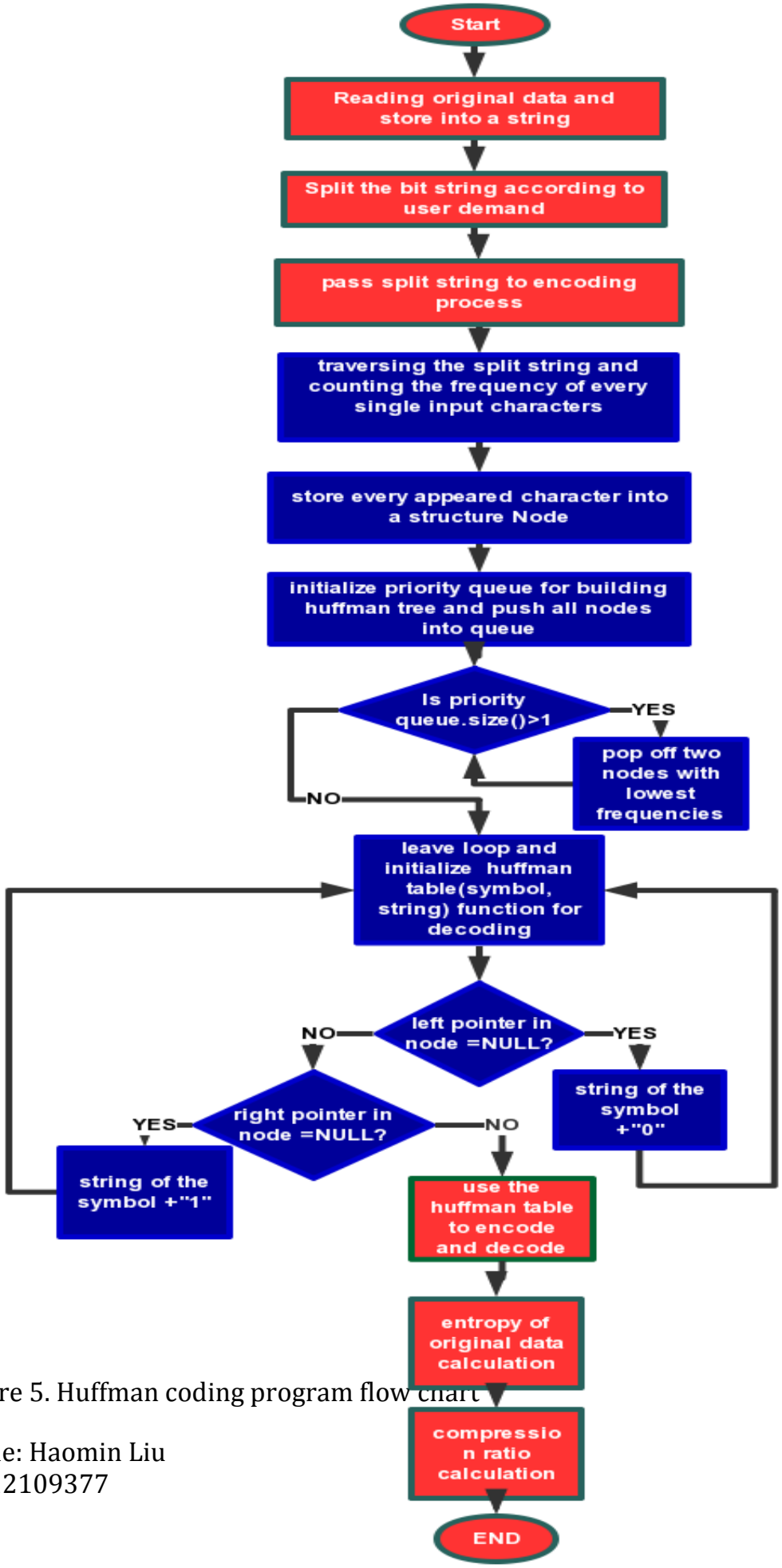


Figure 5. Huffman coding program flow chart

4.1.2 PREPROCESSING: SPLIT THE INPUT FILE STRING ACCORDING TO USER DEMAND

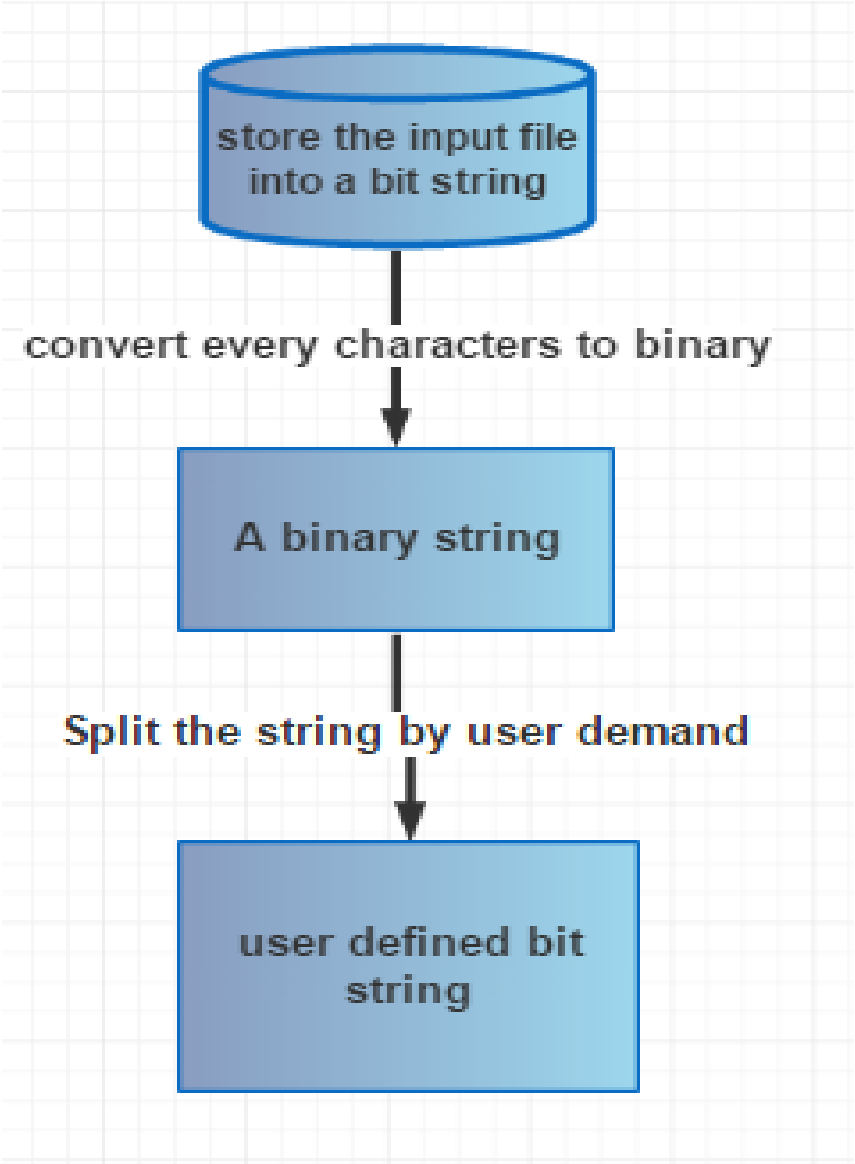


Figure 6. Process of splitting input file string

4.1.3 PROCESSING: HUFFMAN ENCODING AND DECODING PROCESSES

The part is the most important step of the whole process. The details of implementation are divided into several parts and can be illustrated in following steps:

A.BUILDING FREQUENCY TABLE

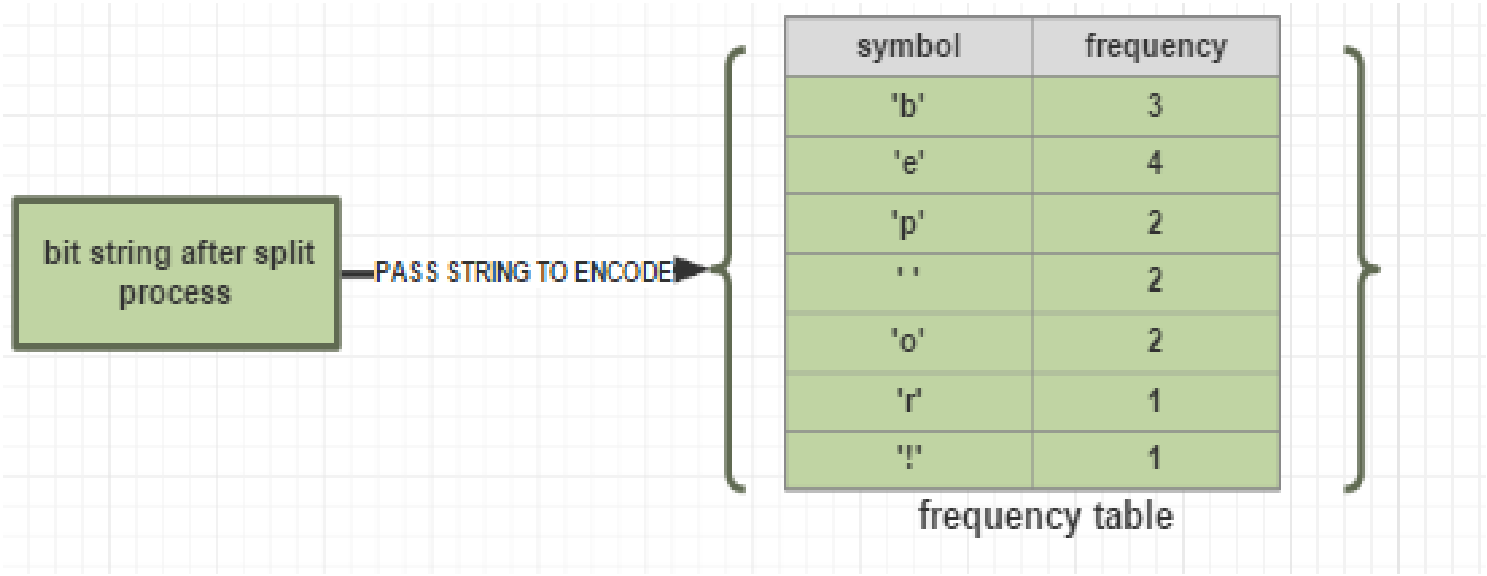
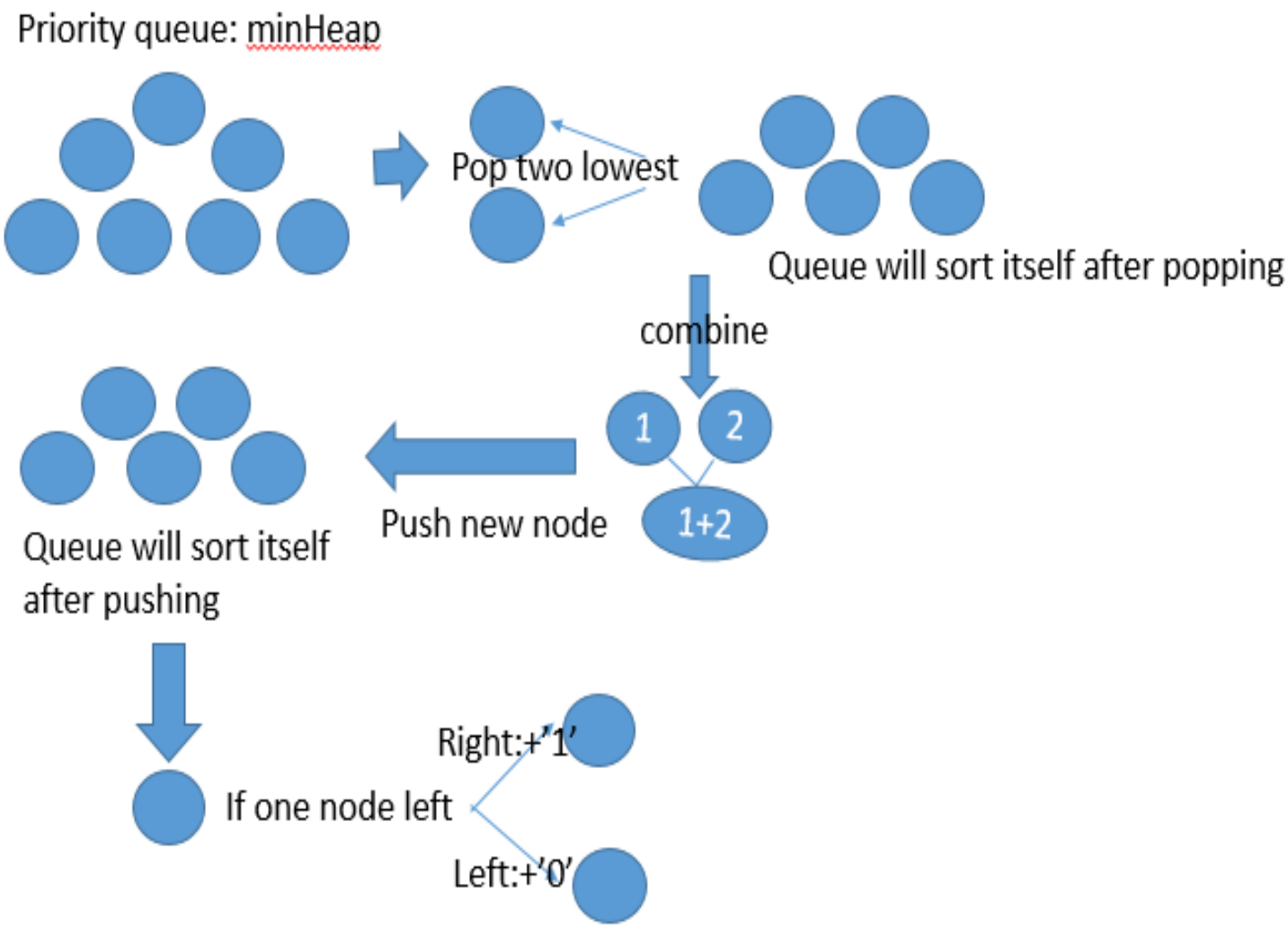


Figure 7. Process of building frequency table

This step is passing the split bit string to first encoding procedure (building frequency table). The STL ordered map is used to store frequency of symbols. Each symbol has its own appearance frequency. In the program, for loop is used to get the frequency count for every single symbol.

B. HUFFMAN TREE BUILDING PROCESS

After building frequency table, next step is building the Huffman tree which is the crucial part for encoding and decoding. A simple visual of this process is given by following figure



:

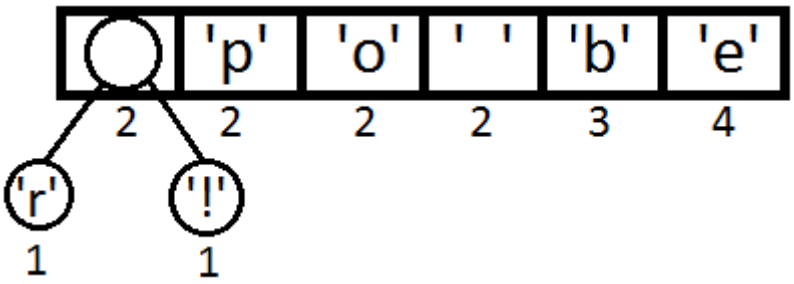
Figure 8. Process of building Huffman tree

Furthermore, the building details will be discussed in next few paragraphs. Firstly, the information of frequency table is passed and stored in minHeap(priority queue). The minHeap is sorted according to priority (for Huffman coding is less one at front) and stored in an array. So following array is the sorted queue according to frequency table in the A part and priority:

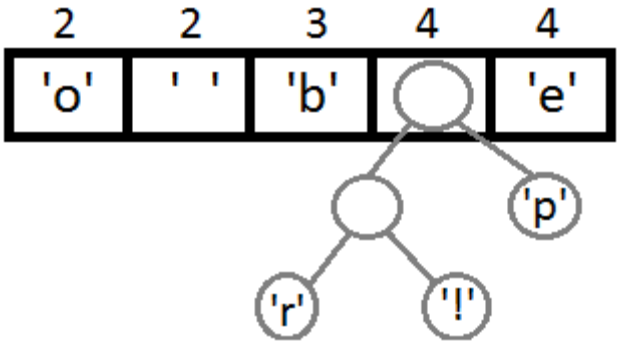
'r'	'!'	'p'	'o'	' '	'b'	'e'

Secondly, the minHeap is needed to be transformed to binary tree. Every two elements of the beginning (or two with lowest frequencies) are pop off to create one binary tree (first one is left child and second one is right child). Then, adding

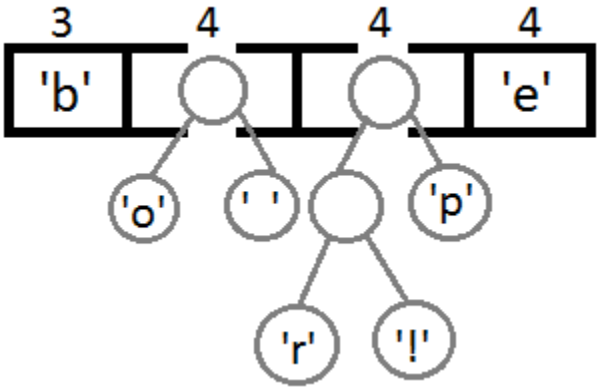
the frequencies of these two elements and pushing back to minHeap. So the data graph can be showed as following:

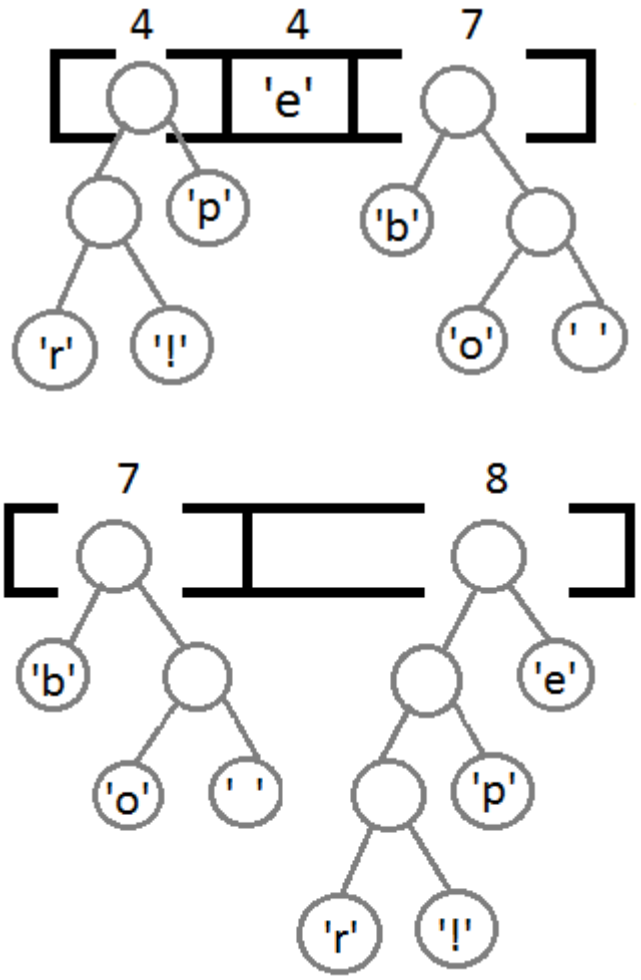


Similarly, popping off another first two elements to form a node with frequency equals four. Then it is pushed back to minHeap again:

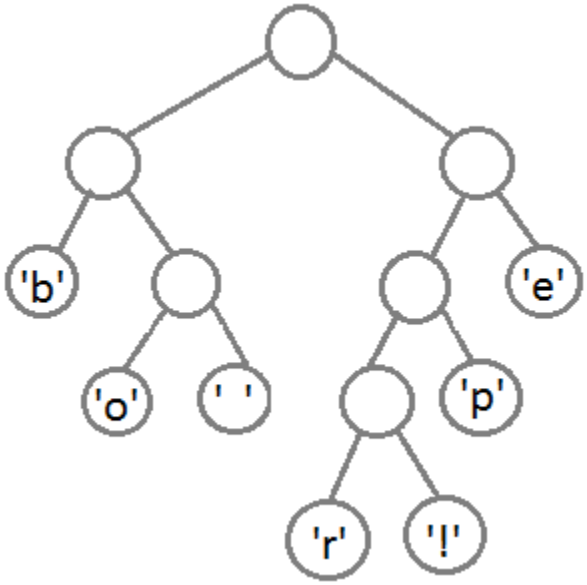


Moreover, continuing the previous algorithm (it can be seen that this is a process of build binary tree from bottom to top):

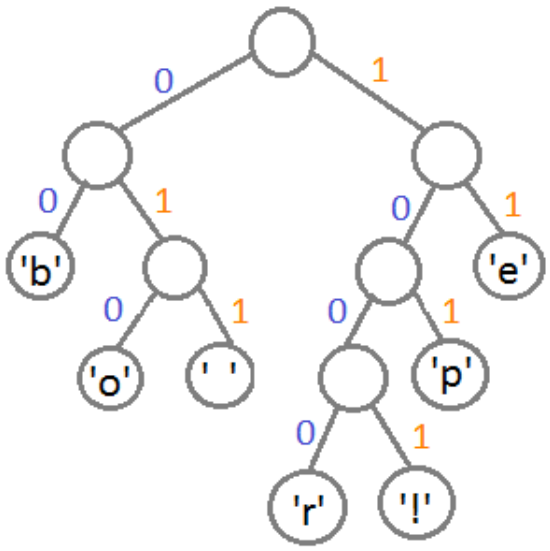




Finally, the following binary tree can be formed:



Meanwhile, the left children can be coded as ’0’ and the right children can be coded as ‘1’. Then we can traverse the Huffman tree to get the encoding string of each symbol. For instance, encoding string of ‘b’ is 00 and encoding string of ‘p’ is 101. Also, we can see that the more frequently a symbol appears, the higher level it will be.



The coding of how to build Huffman trees is given by:

```
//use priority queue to build min heap
priority_queue<huffmanNode*, vector<huffmanNode*>, compareFunctor> minHeap;
huffmanNode *head = NULL, *lowest1 = NULL, *lowest2 = NULL;

//put the character count into the huffman priority queue
for (auto entry : frequencyTable){

    huffmanNode* n = new huffmanNode();
    n->symbol = entry.first;
    n->frequency = entry.second;
    n->left = n->right = nullptr;
    minHeap.push(n);
}

//pop off two lowest at a time
while (minHeap.size()>1)
{
    lowest1 = minHeap.top(); minHeap.pop();
    lowest2 = minHeap.top(); minHeap.pop();
    huffmanNode *temp = new huffmanNode();
    temp->frequency = lowest1->frequency + lowest2->frequency;
    temp->left = lowest1; temp->right = lowest2;
    minHeap.push(temp);
}
//final one left is the root of huffman tree
head = minHeap.top(); minHeap.pop();
```

Figure 9. Coding of building Huffman tree

C.BUILDING HUFFMAN TABLE FOR ENCODING AND DECODING

Next step is building Huffman table for encoding and decoding. The Huffman tree was created from last past will be used to generate the Huffman table. All the codes will be pushed into an ordered map to record each symbol's encoding string. Also, based on this table, an encoding string of the whole data can be generated and encoded data can be decoded to original data. The whole process can be illustrated in following figure:

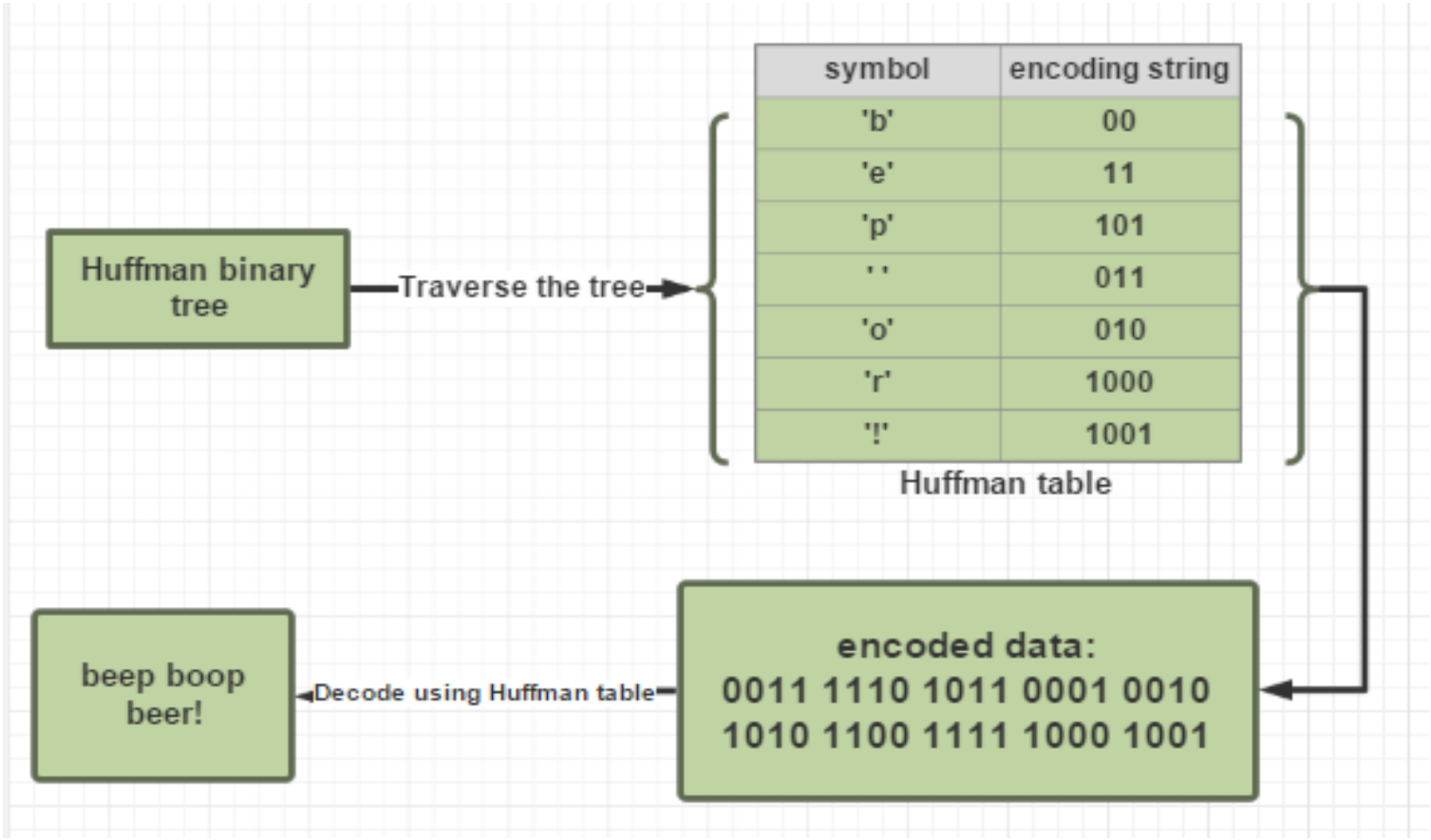


Figure 10. Process of building Huffman table for encoding and decoding

In the process of building Huffman table, depth-first search algorithm and recursive function are used to note down the encoding string for each symbol. Decoding is using encoded data and traversing Huffman tree to get original input string.

4.1.4 POST PROCESSING: ENTROPY AND COMPRESSION RATIO CALCULATION

A. ENTROPY CALCULATION

After Huffman encoding and decoding processes, the information content is calculated in order to get maximum compression possibility.

For example, there are nine balls in a lottery pool and three outcomes possible for choosing a ball, it can be either blue, red or yellow.

The entropy of this particular event can be calculated as following:

$$\text{Entropy} = \text{H(M)} = \text{E[i(X)]} =$$

$$\sum_{X=1}^{M} i(X)P(X) = -\sum_{X=1}^{M} i(X)logP(x) =$$

$$-\left(\frac{4}{9}\right)\log\left(\frac{4}{9}\right) + \left[-\left(\frac{2}{9}\right)\log\left(\frac{2}{9}\right)\right] + \left[-\left(\frac{3}{9}\right)\log\left(\frac{3}{9}\right)\right] = 1.5305$$

```
void huffmanCode::getEntropy(){

    int dataLength = data.length();
    double entropy = 0;
    for (std::pair<char, int> p : frequencyTable) {
        double freq = static_cast<double>(p.second) / dataLength;
        entropy += freq * log2(freq);
    }
    entropy *= -1;
    std::cout << "The information entropy is " << entropy << endl;

}
```

Figure 11. Entropy calculation codes

B. COMPRESSION RATIO

This part is compression ratio calculation. We need this part for comparing different split bit string. The formula and codes are given by:

$$\text{Compression ratio} = \frac{\text{uncompressed size}}{\text{compressed size}}$$

```
//compression ratio calcualtion
double compressionRatio = (double)(string.data()).size() / (double)encodeData.size();
cout <<" The compression ratio is " << compressionRatio << endl << endl;
```

Figure 12. Compression ratio calculation codes

4.1.5 DATA STRUCTURE

```
struct huffmanNode{
    char symbol;
    int frequency;
    huffmanNode *left, *right;
    huffmanNode();
    huffmanNode(char, int);
    ~huffmanNode();
    bool isLeaf();
};
```

Figure 13. Data structure for Huffman coding algorithm

The basic data structure used in Huffman coding program is the HuffmanNode.

The members of struct huffmanNode are:

- char symbol(which holds every single unique character appears in original data).
- int frequency(which holds how many times that an unique character occurs in original data(also known as frequency)).
- Pointers to left and right children (which are assigned with NULL at the beginning until the program start pushing nodes together to form Huffman tree).
- Constructers for the huffmanNode.(which are used to assign initial values of symbol, frequency and huffmanNode pointers)
- Bool isLeaf() function(which is used to check whether current node is leaf or not)

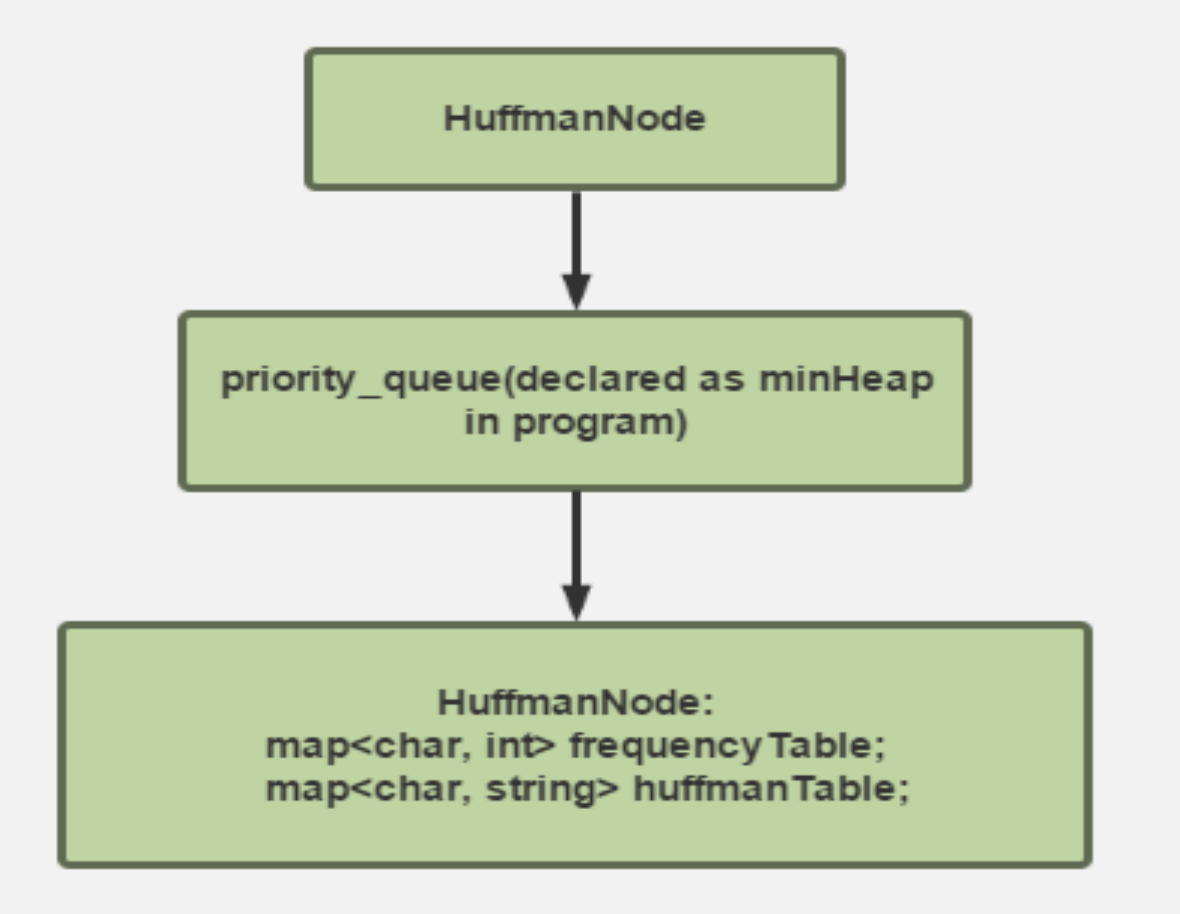


Figure 14. Flow chart of how data structures process in the program

From the figure 14, it can be said that the huffmanNodes are pushed onto a priority queue called minHeap. The minHeap is formed by container (vector) of huffmanNodes. The nodes in minHeap are sorted according to a compare functor which is defined by:

```
class compareFunctor{
public:
    bool operator()(huffmanNode *lhs, huffmanNode *rhs)
    {
        return lhs->frequency > rhs->frequency;
    }
};
```

Figure 15. Compare functors coding

Instead of the largest element of the minHeap at the root, the smallest one is. This is due to one of Huffman encoding steps is popping the two smallest nodes off the heap, such process will be discussed in Huffman tree

Additionally, the STL ordered map (which are huffmanTable and frequencyTable) are used to store character frequency of appearance and generated Huffman encoding data.

The Huffman nodes and all of node-related features described in this section are combined into a HuffmanCode class which takes a user-defined bit string and processes other functions for the user.

The code can be found in attached Visual Studio 2013 project file ‘huff1’.

4.2 IMPLEMENTATION DETAILS OF LZW ALGORITHM CODING

The encoding and decoding processes of LZW are given by figure 16 and figure 17. The implementation of coding details can be found in attached Visual Studio project 'lll'.

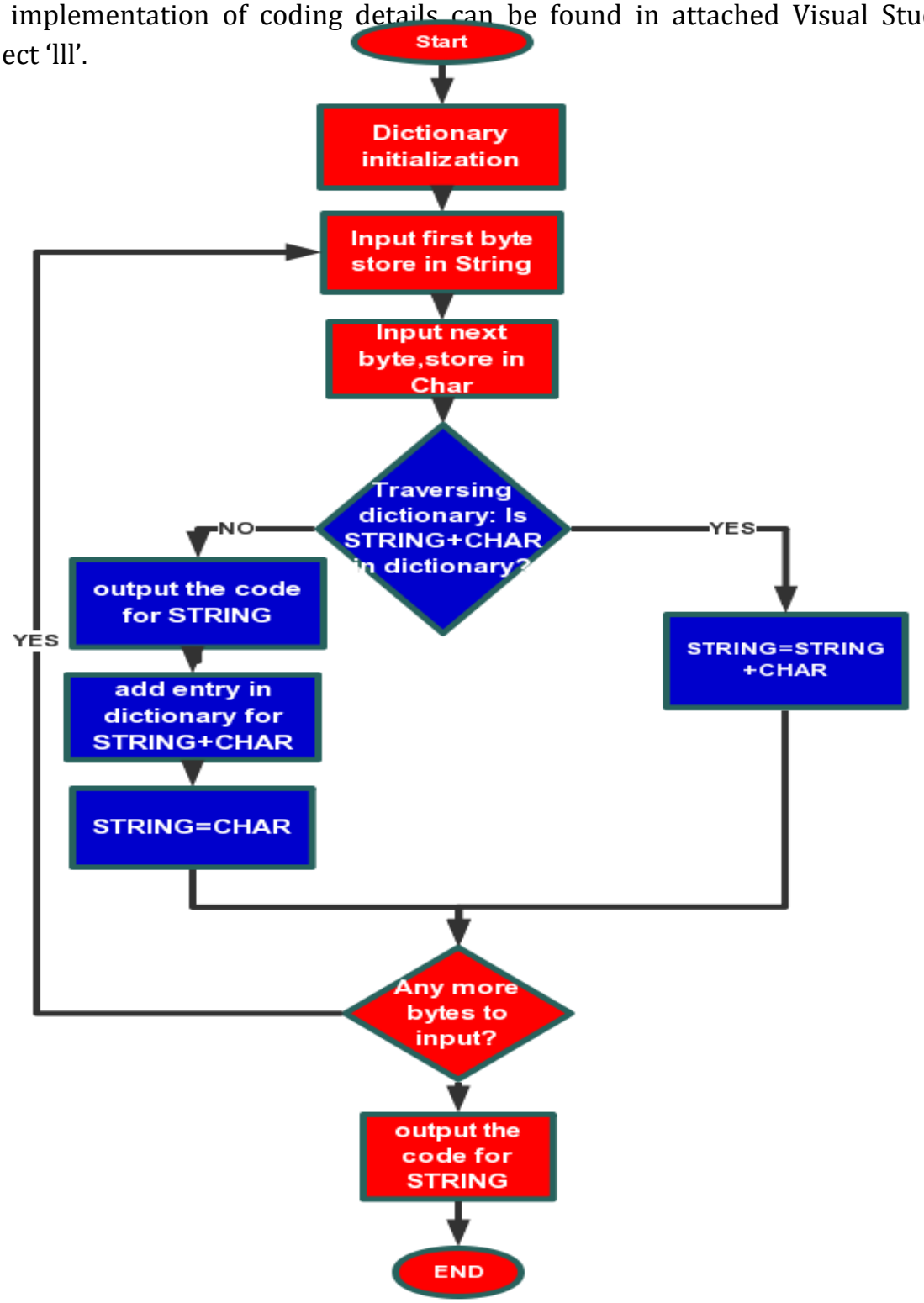


Figure 16. The LZW compression algorithm encoding process (CHAR is a single byte, STRING is a variable length of sequence of bytes)

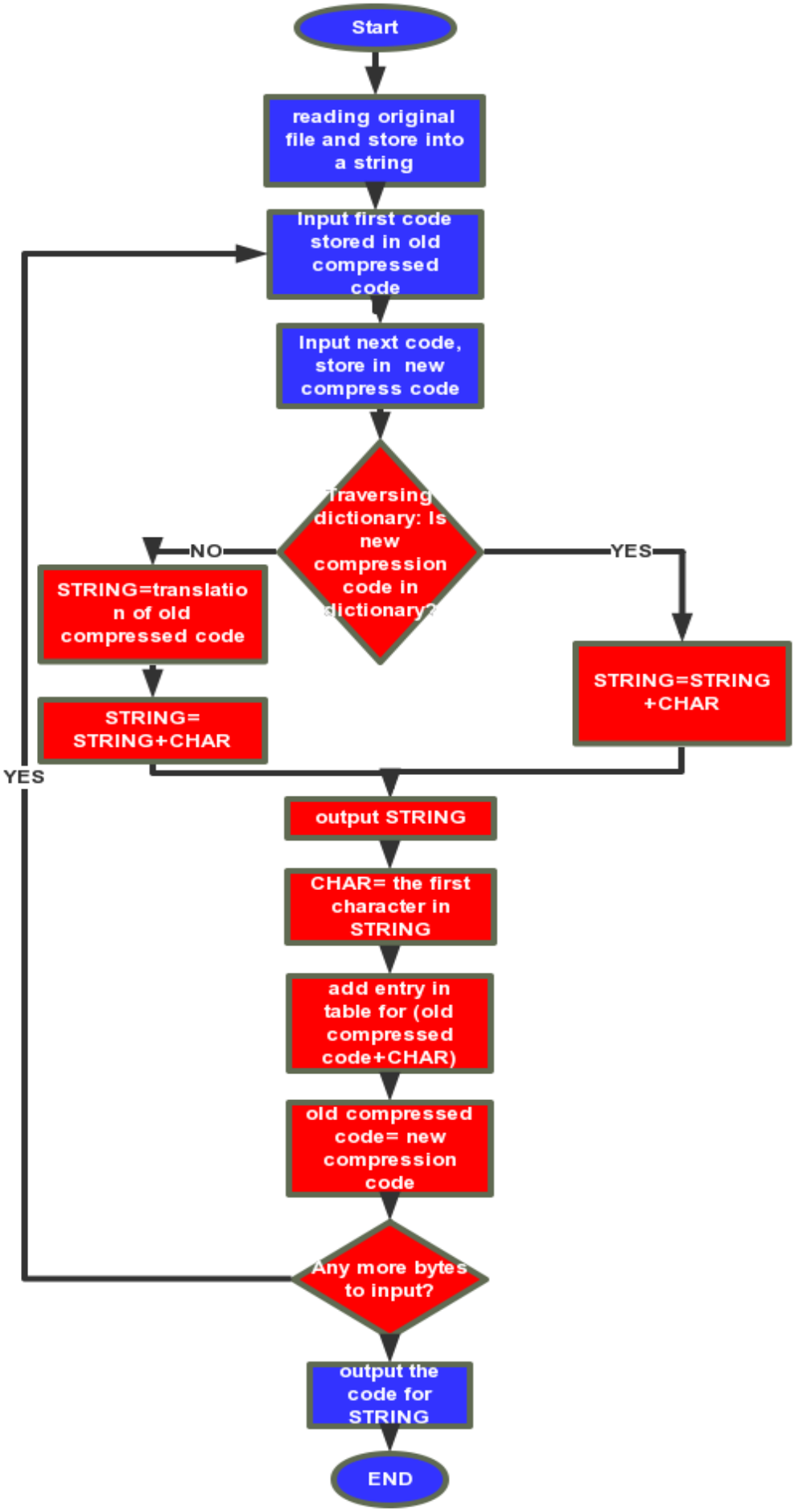


Figure 17. The LZW compression algorithm decoding process (CHAR is a single byte, STRING is a variable length of sequence of byte

4.3 CALGARY CORPUS TEST RESULTS AND DISCUSSION OF HUFFMAN CODING PROGRAM

A series of Huffman coding compression experiments were conducted using Calgary corpus test files. Calgary corpus is mostly used corpus in the data compression (especially in text compression) and is the standard for lossless compression evaluation. Therefore, the test file of Calgary is the most suitable testing files under the circumstance of this project.

During the experiments, four files was tested with Huffman coding program. These files have different information redundancy (especially the test file of a picture has higher redundancy than others). The major data that we want to obtain form the experiments are entropy, compression ratio and compression time (for different bit lengths). The relationship between bit length and compression ratio was compared. Also, the relationship between all aspects of Huffman coding was compared and discussed. The details of comparison figures and discussions will be illustrated in the following section.

EXPERIMENT DETAILS

AIM

- To investigate how different bit lengths influence compression ratio
- To investigate how entropy and compression change for different bit lengths

PROCEDURE

A. Starting Huffman coding program and Input from 1 to 31(bit length) for producing entropy, compression ratio and compression time.

B. Collecting the results of different bit length attempts and drawing the table to illustrate.

C. Making comparison graphs based on result table.

The following figures were conducted for comparing different factors in Huffman coding program.

A. Experiment on book2

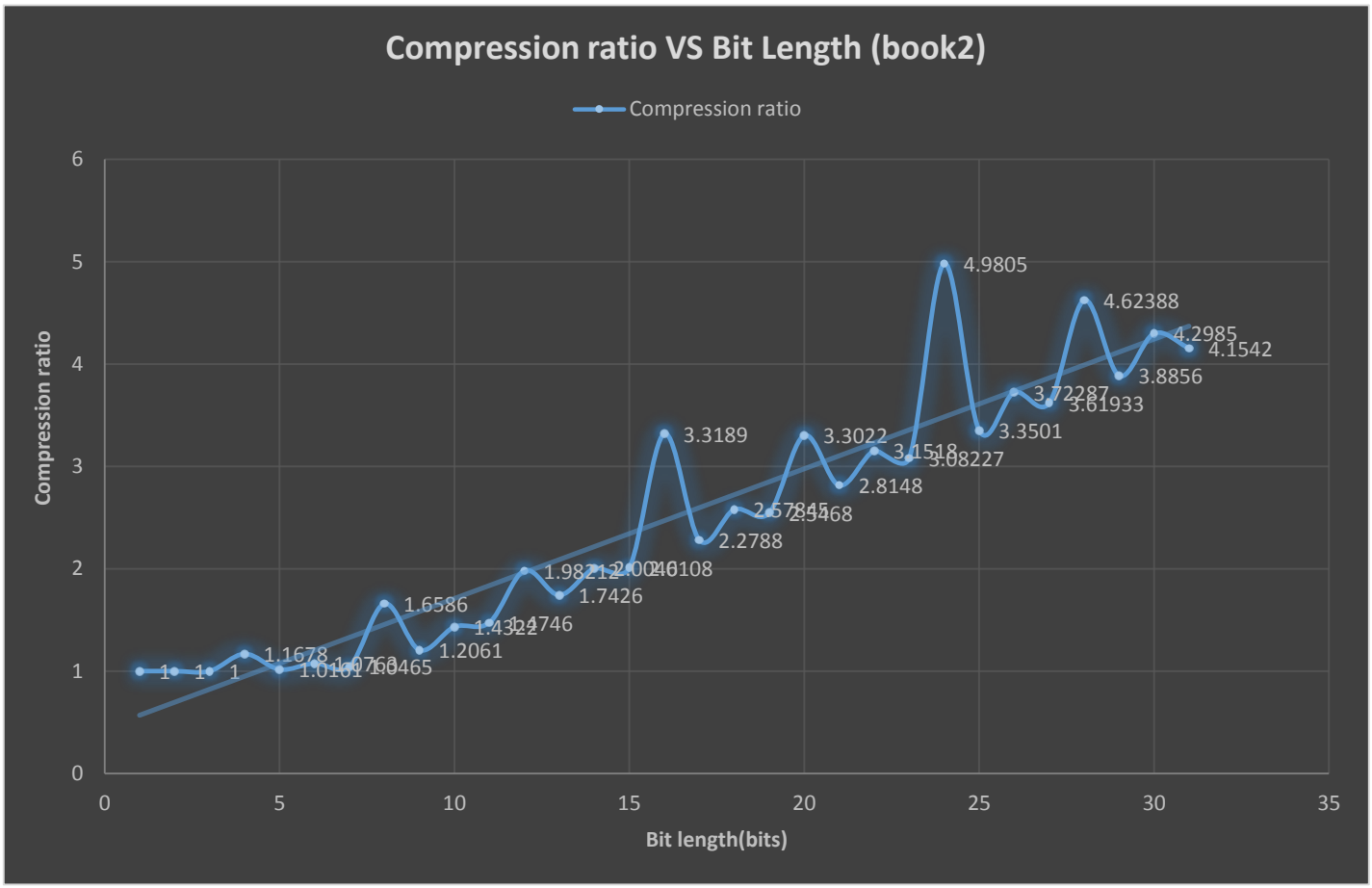


Figure 18. Huffman coding compression ratio against bit length graph (test file is book2 from Calgary corpus)

The size of testing file book2 is 597kb. From the figure, it can be said that the highest compression ratio of this experiment is 4.9805 and corresponding bit length is 24 bits (The survey has a margin of time error about 1.5s; this estimated error is based on many test results).

Trend:

For this book2 file, experiment results shows that the compression ratio is ascending as the bit length is increasing. Obviously, there are 3 high compression ratio points. It is worth noting that their corresponding bit lengths are all even numbers (16, 24 and 28).

Additionally, from the results in the table 1 and 2, it can be said that even bits number has better compression ratio than odd bits number. Therefore, it is recommended that users can choose even bit numbers in order to have better compression ratio.

From several experiment results, the best compression ratio can be reached when the bit length is 24 bits.

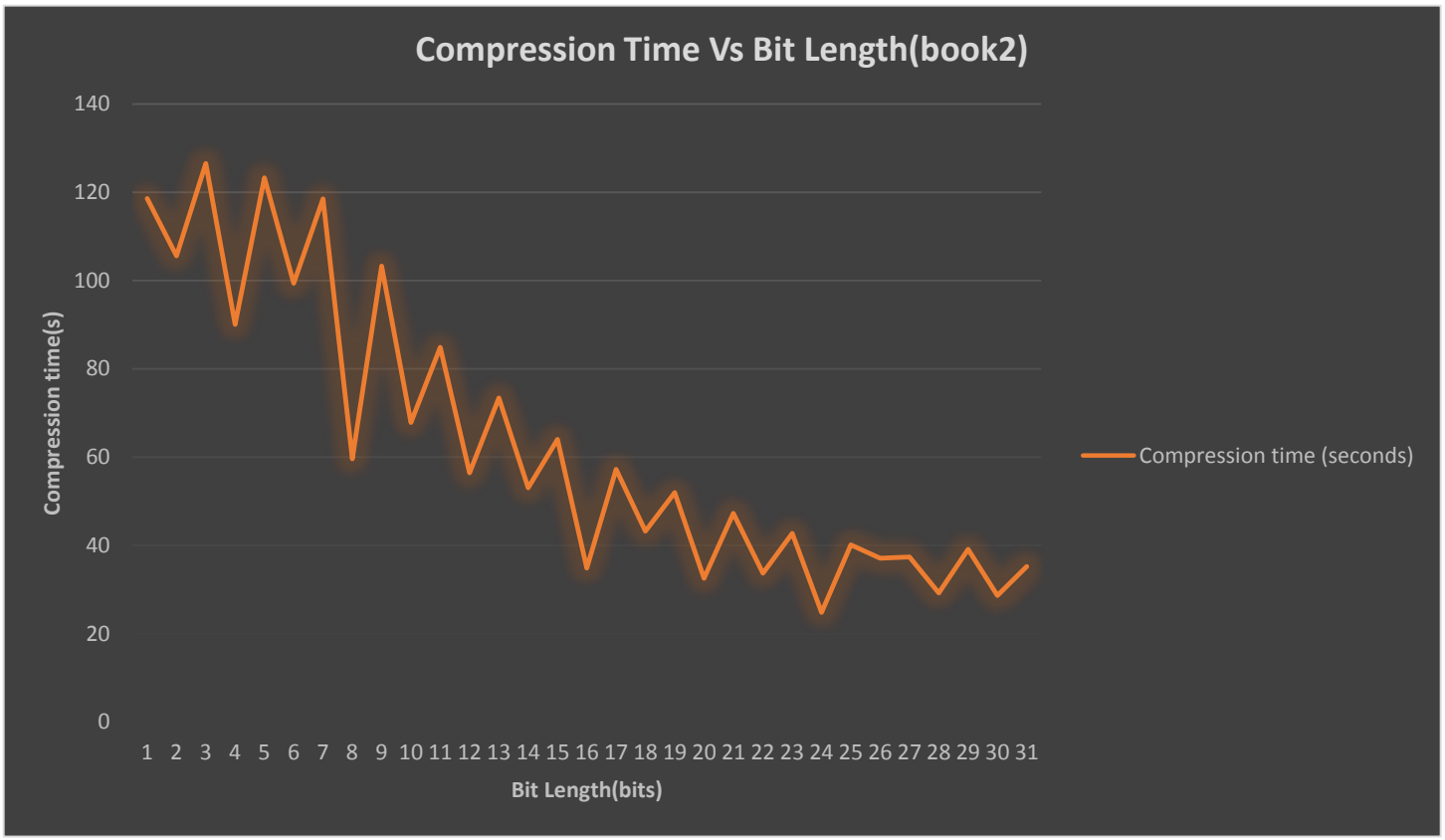


Figure 19. Huffman coding compression time against bit length graph (test file is book2 from Calgary corpus)

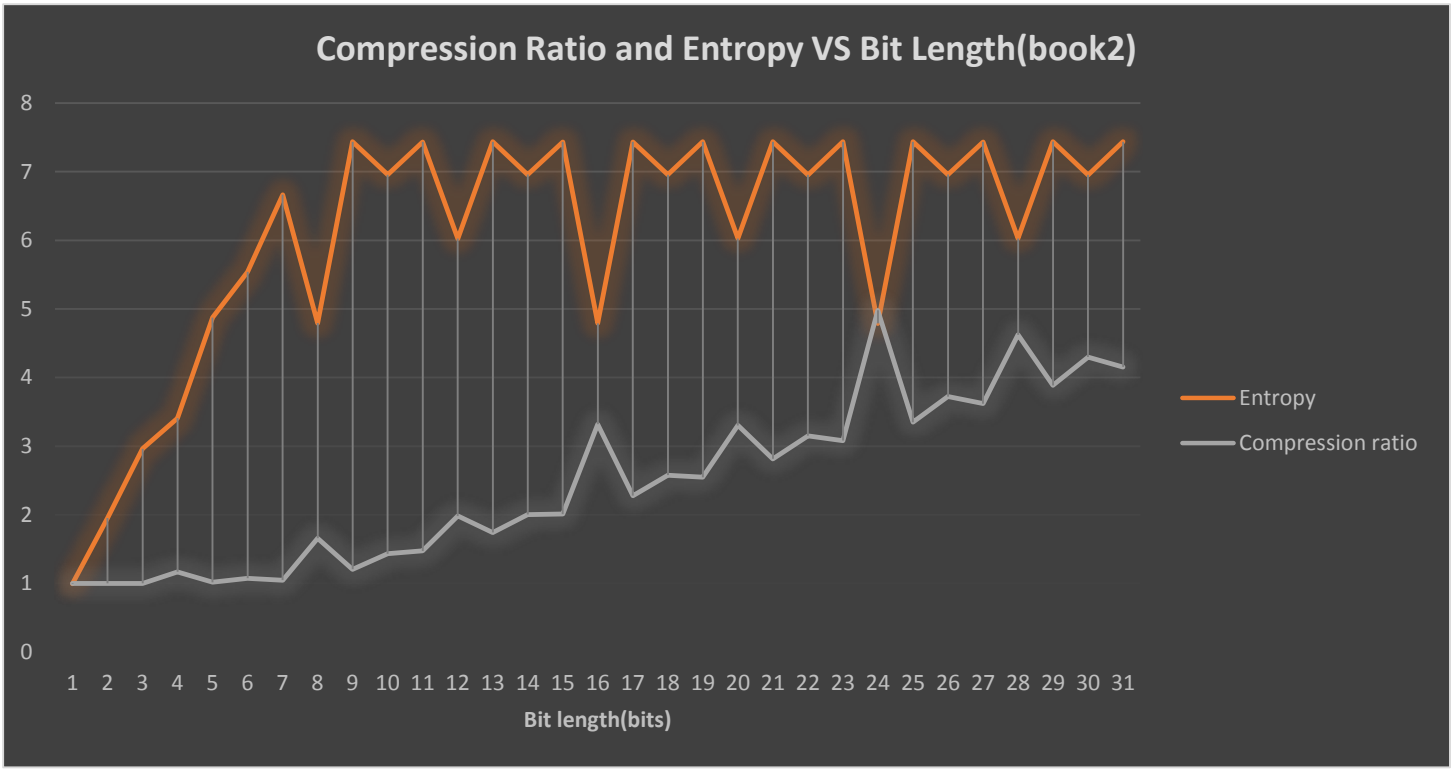


Figure 20. Huffman coding compression ratio and entropy against bit length graph (test file is book2 from Calgary corpus)

From these comparison graphs, it can be summarized that the compression ratio is ascending with the increase of bit length while the compression time is descending with the increase of bit length. Also, the entropy increased a little at the beginning, then reached a maximum value at 7.43. After this point, entropy became stable around 7.

Therefore, choosing bigger bit length will not only increase the compression ratio, but also decrease the compression time.

B. Experiment on paper1

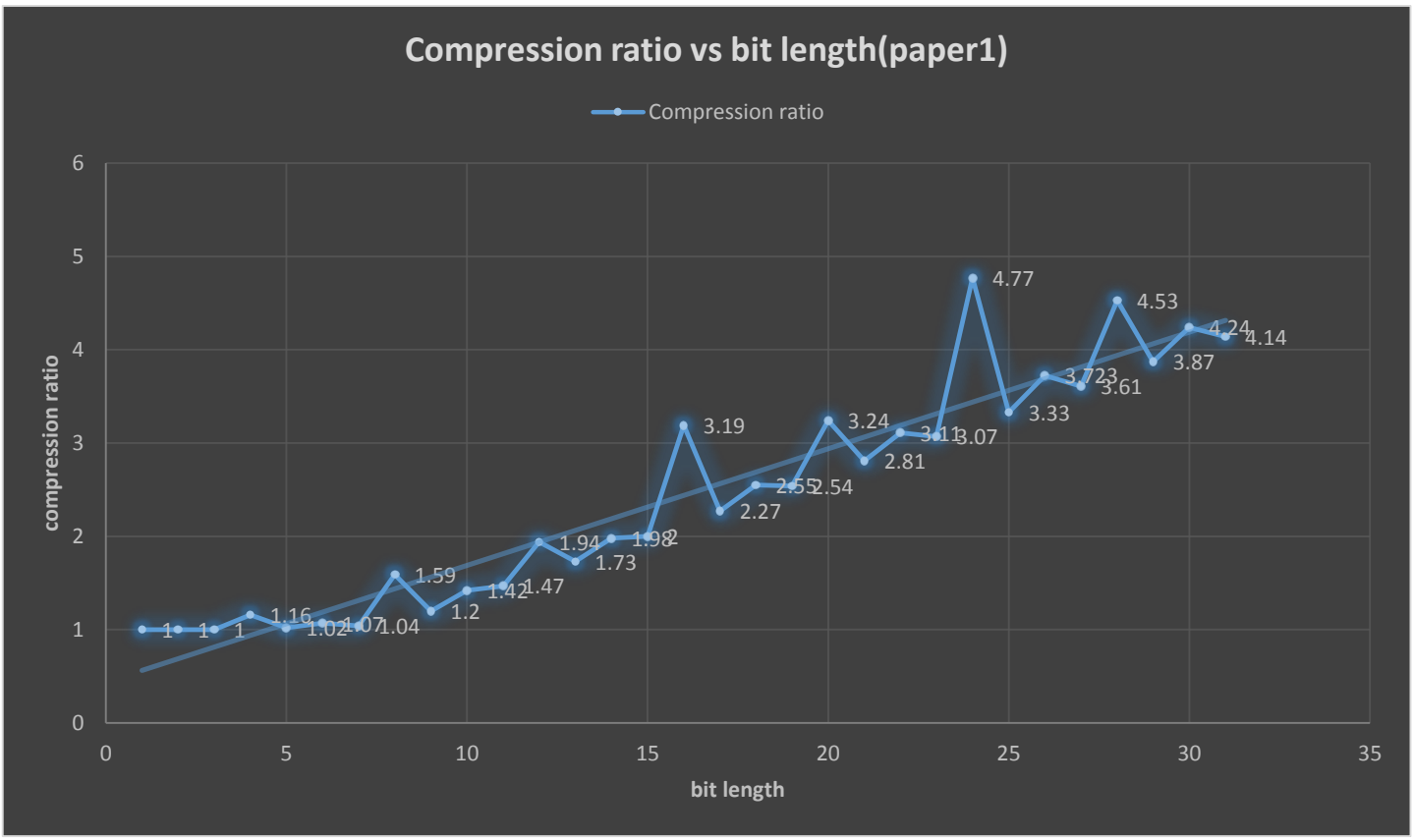


Figure 21. Huffman coding compression against bit length graph (Test file is paper1 from Calgary corpus)

The size of testing file paper1 is 52kb. The above figure shows that the highest compression ratio of this experiment is 4.77 and corresponding bit length is 24 bits (The survey has a margin of time error about 1.5s, this estimated error is based on many test results).

Trend:

For this paper1 file, experiment results shows that the compression ratio is ascending as the bit length is increasing. There are 4 high compression ratio points. It is worth noting that their corresponding bit lengths are all even numbers (16, 20, 24 and 28).

Also, the trend line and high compression points of this test file are relatively similar. This may due to these two files are common English text files with respect to books and papers. That is to say, the information redundancy of these two files are almost the same. Therefore, it is recommended that user can choose even bit length numbers between 16 and 28 to get better compression ratios.

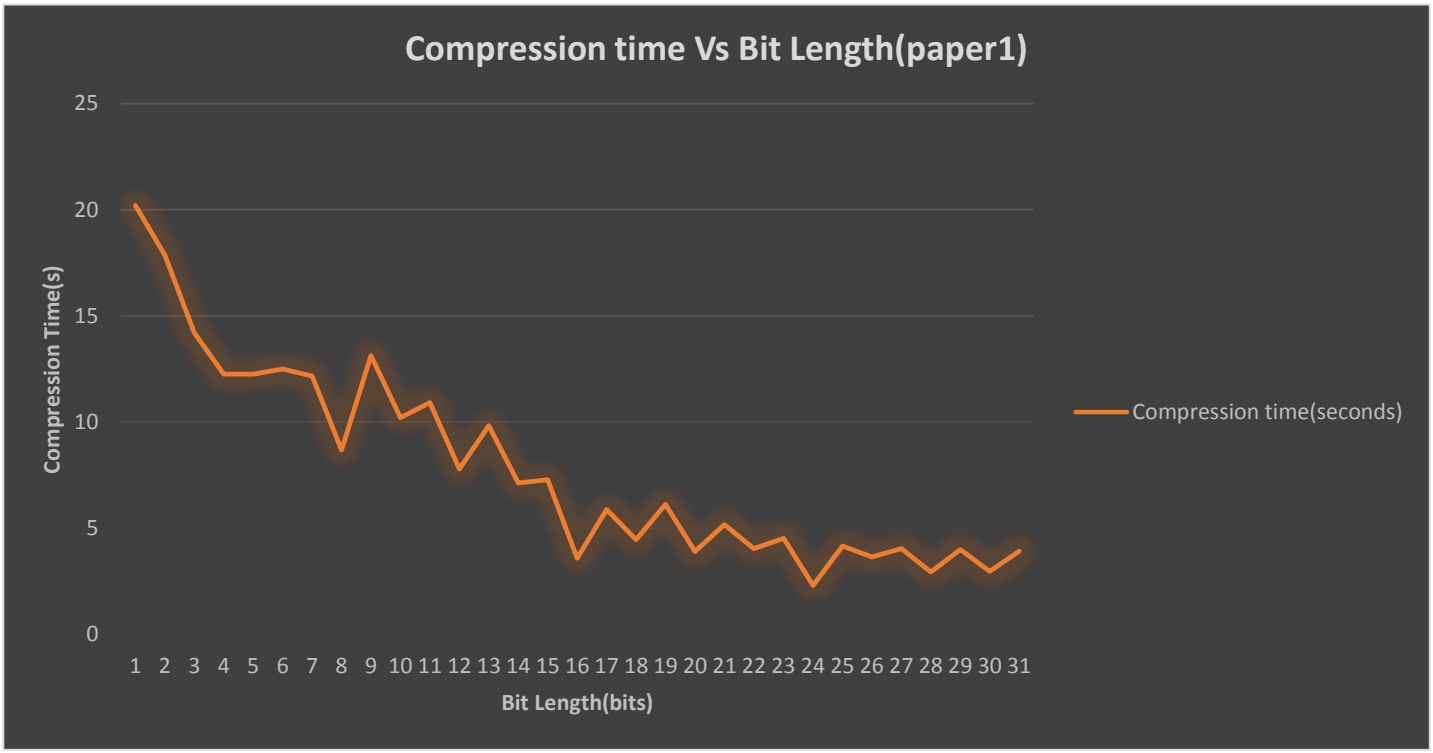


Figure 22. Huffman coding compression time against bit length graph (test file is paper1 from Calgary corpus)

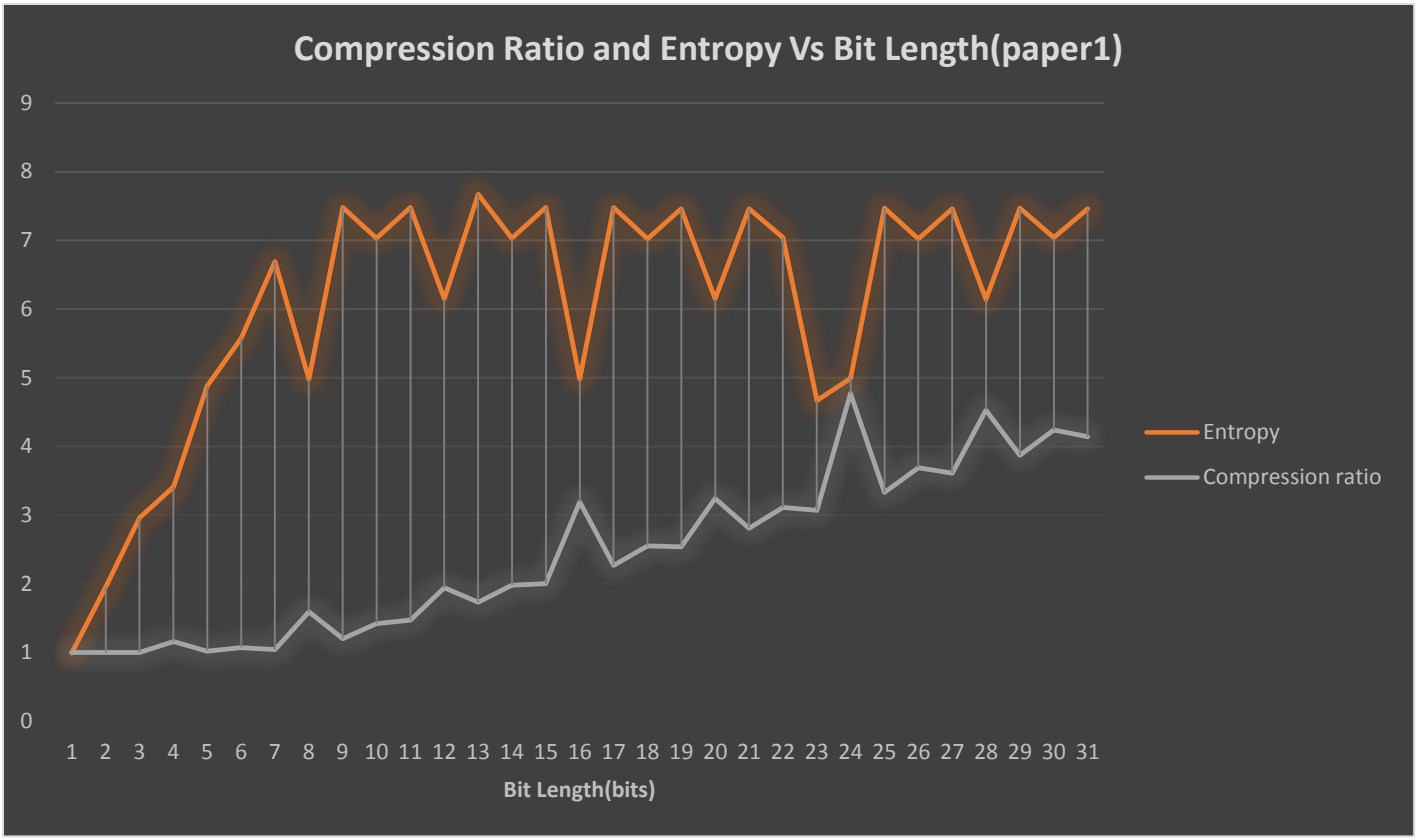


Figure 23. Huffman coding compression ratio and entropy against bit length graph (test file is book2 from Calgary corpus)

From comparison graphs, it can be concluded that the compression ratio is ascending with the increase of bit length while the compression time is descending with the increase of bit length. Also, the entropy does not change much after reaching maximum value. Therefore, it is recommended that best option for user to get highest compression ratio is choosing bit length from 16 to 28 bits.

C. Experiment on pic

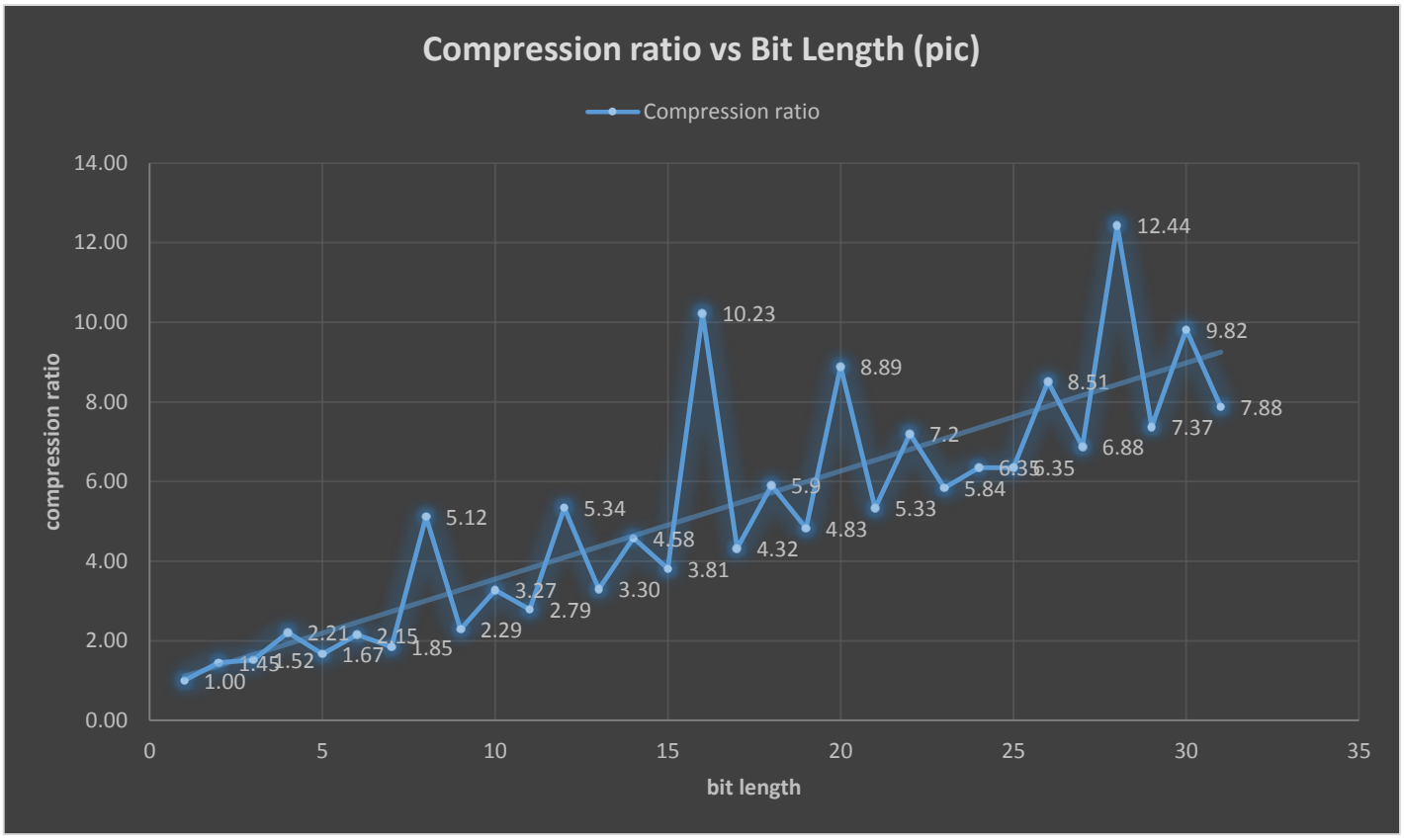


Figure 24. Huffman coding compression against bit length graph (Test file is pic from Calgary corpus)

The size of testing file pic is 502kb. From the above figure, it can be said that the highest compression ratio of this experiment is 12.44 and corresponding bit length is 28 bits (The survey has a margin of time error about 1.5s, this estimated error is based on many test results).

Trend:

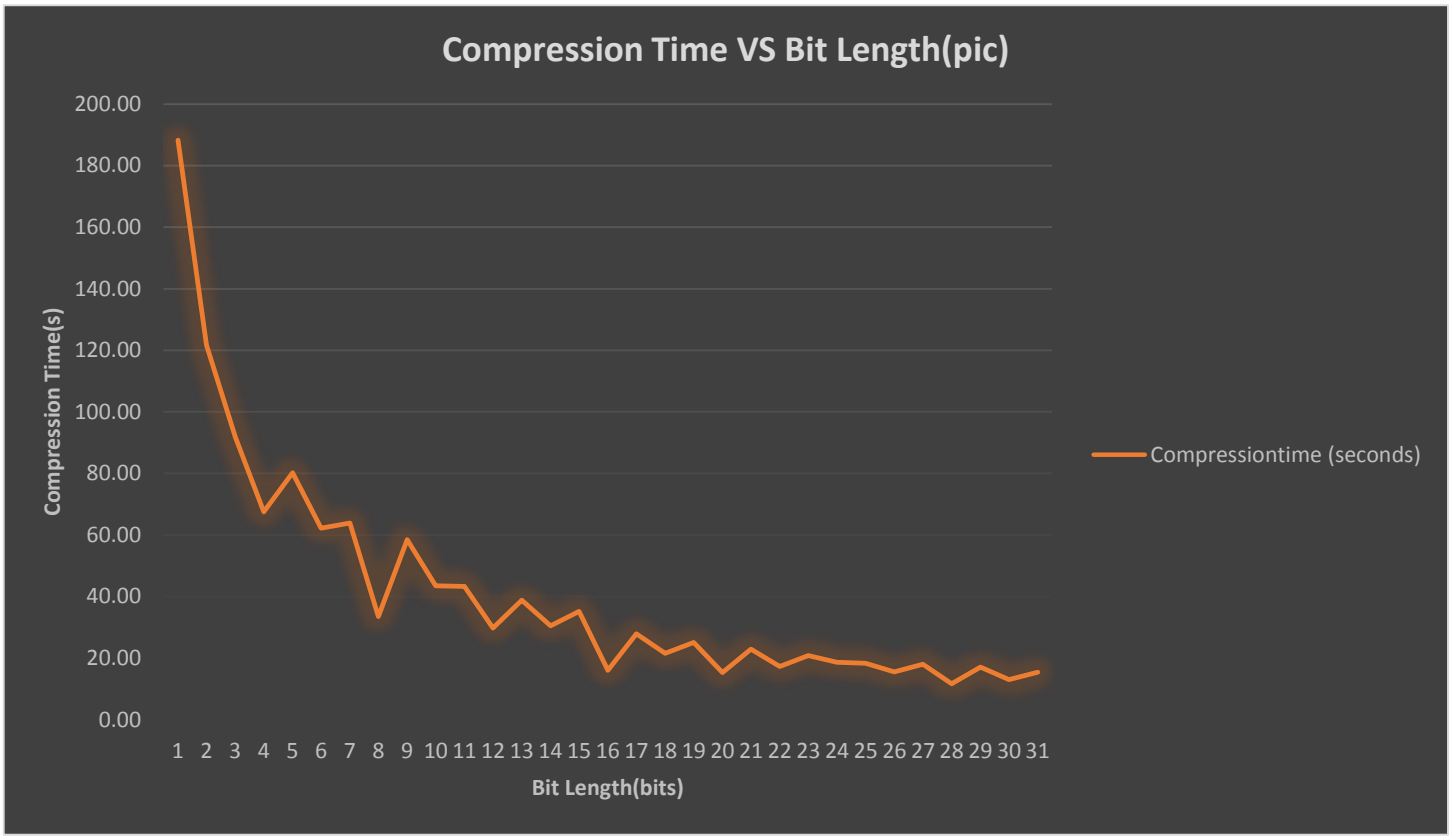
For this pic file, experiment results shows that the compression ratio is ascending as the bit length is increasing. Obviously, there are 3 high compression ratio points. It is worth noting that their corresponding bit lengths are all even numbers (16, 20 and 28).

Clearly, this pic file can be compressed much smaller than book2 and paper1. This is due to pic file normally has high information redundancy, therefore each bit in the compressed message can carry more information. While this file still can reach the highest compression ratio in the range between 16

and 28, this means the user can also choose bit length in this range to get better compression ratio for files with different redundancy.

Additionally, from the results in the table 5 and 6, it can be said that even bits number has better compression ratio than odd bits number. Therefore, user can choose even bit numbers in order to have better compression ratio.

Figure 25. Huffman coding compression time against bit length graph (test file is paper1 from Calgary corpus)



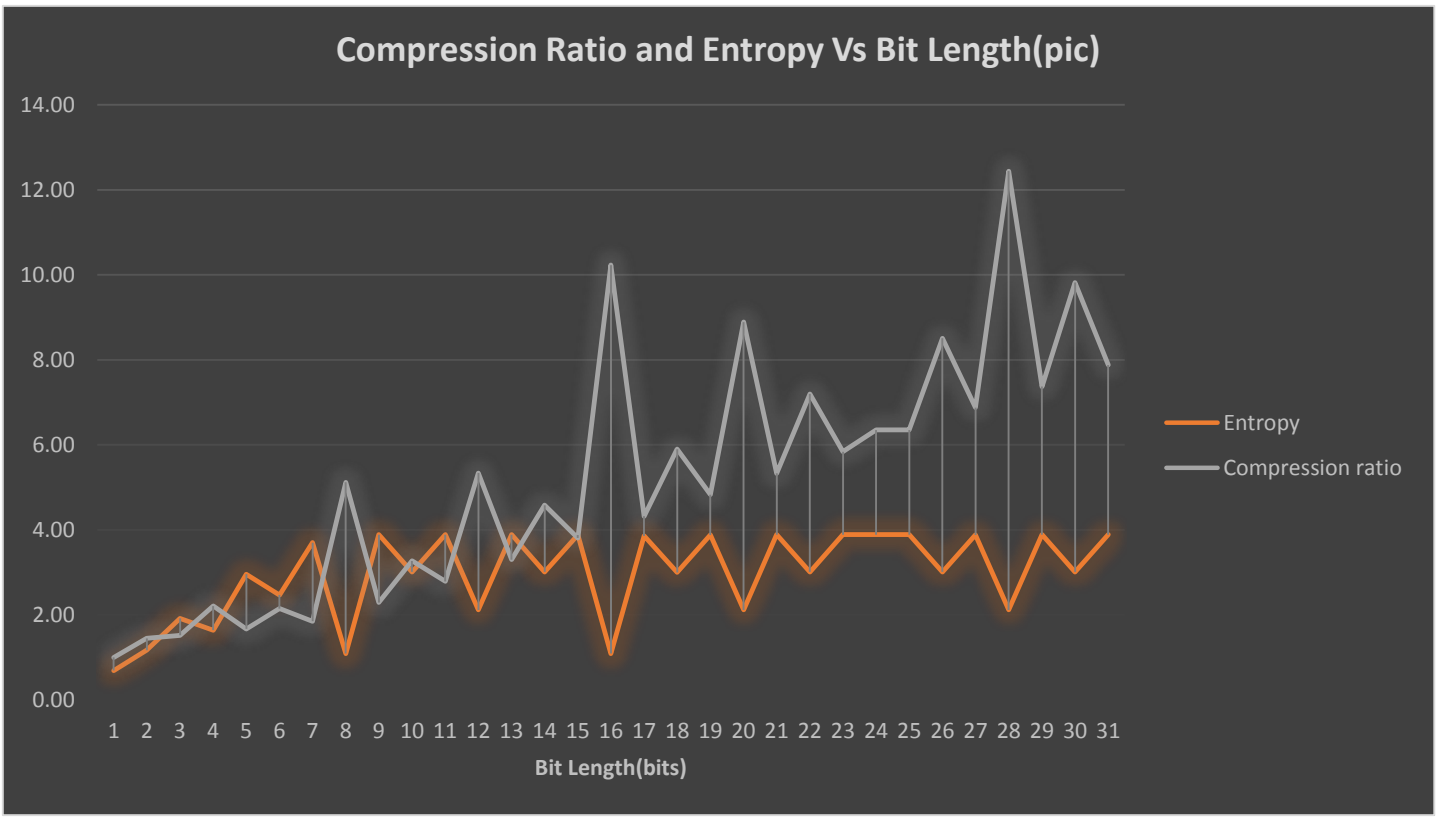


Figure 26. Huffman coding compression ratio and entropy against bit length graph (test file is book2 from Calgary corpus)

From comparison graphs, it can be concluded that the compression ratio is ascending with the increase of bit length while the compression time is descending with the increase of bit length. Also, the entropy does not change much.

Moreover, from the results of listed three experiments, files with different information redundancy (e.g. book2 and pic) do not affect highest compression ratio range (which is between 16 and 28 bit length).

4.4 LZW EXPERIMENTS

EXPERIMENT DETAILS

AIM

- To investigate how entropy influence compression ratio of LZW coding
- To investigate how LZW and Huffman coding can get high compression ratio

PROCEDURE

A. Set Input file and start LZW coding program
B. Collecting the results of different file with different entropies for comparison

RESULTS

A. Experiment on pic.

The file size of pic before compression: 497kb

The file size of pic after compression: 121kb

Compression ratio=file size before compression/file size after compression=497/121= 4.1

B. Experiment on paper1.

The file size of pic before compression: 81kb

The file size of pic after compression: 84kb

Compression ratio=file size before compression/file size after compression=81/84= 0.96

C. Experiment on test1 (extreme case of a repeated symbol)

The file size of pic before compression: 8kb

The file size of pic after compression: 3kb

Compression ratio=file size before compression/file size after compression=8/3= 2.67

The same file tested with Huffman coding program and produced 3.57 compression ratio.

The test results of A and B shows that the pic file with lots of repeated patterns of multiple bytes can get good compression ratio; while the paper1 with less repeated patterns get larger compressed file (which means original file did not compress).

The test results of this extreme case C shows that Huffman coding can get much better compression ratio than LZW. This is due to LZW coding stores repeated symbols in dictionary while Huffman coding just encode most repeated symbol with shortest code. Therefore, the dictionary size of LZW coding is larger than normal cases. It can be concluded that Huffman performs in accordance with the redundancy (which is given a measure using Entropy) while LZW may produce better results for large files with lots of repeated patterns of multiple bytes.

5. CONCLUSION

To conclude, this project has introduced the background of data compression in terms of classification, principle and theory. The lossless data compression and information entropy are mainly discussed in this report for further implementations. Huffman coding and LZW coding algorithms were implemented for investigating how to improve compression ratio. The implementation is based on several methods with respect to information search, concepts generation and research analysis.

The objectives of this project are mostly completed. Two important lossless algorithms were implemented and discussed in details. .

For Huffman part, a user-defined bit length Huffman coding program was designed for investigation the relationship between coding bit length and compression ratio. The experimental results showed that the program can produce better compression ratio if the input file string is split by even number bits. Also, in the range of 16 to 28 split bits, the program can reach the highest compression ratio. Therefore, it is recommended that user can focus on this range to get a productive compression. Additionally, the test files with high information redundancy (e.g. image file) can be compressed in higher compression ratio.

For LZW part, the experimental results showed that LZW may get better compression ratio for large files with lots of repeated patterns of multiple bytes; While Huffman coding performs in accordance with redundancy.

BIBLIOGRAPHY

A.Lesne. (2011). *Shannon entropy: a rigorous mathematical notion at the crossroads between probability, information theory, dynamical systems and statistical physics.* Retrieved from:

http://www.lptmc.jussieu.fr/user/lesne/MSCS-entropy.pdf

A.RRNYI. (1961). *ON MEASURES OF ENTROPY AND INFORMATION.* Retrieved from:

http://l.academicdirect.org/Horticulture/GAs/Refs/Renyi_1961.pdf

A.Shahbahrami, R.Bahrampour, M.S.Rostami, M.A.Mobarhan. (2011). *Evaluation of Huffman and Arithmetic Algorithms for Multimedia Compression Standards.* Retrieved from:

http://arxiv.org/ftp/arxiv/papers/1109/1109.0216.pdf

D.A.Huffman. (1952). *A Method for the Construction of Minimum-Redundancy Codes**. Retrieved from: http://compression.ru/download/articles/huff/huffman_1952_minimum-redundancy-codes.pdf

G.E.Blelloch. (2013). *Introduction to Data Compression.* Retrieved from:

http://www.cs.cmu.edu/afs/cs/project/pscico-guyb/realworld/www/compression.pdf

J.Ziv, A.Lempel. (1978). *Compression of Individual Sequences viaVariable-Rate Coding.* Retrieved from: http://www.cs.duke.edu/courses/spring03/cps296.5/papers/ziv_lempel_1978_variable-rate.pdf

J.Zelenski, K.Schwarz. (2012). *Huffman Encoding and Data Compression.* Retrieved from:

http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1126/handouts/220%20Huffman%20Encoding.pdf

K.Barr, K.Asanovic. (2003). *Energy Aware Lossless Data Compression.* Retrieved from:

http://www.eecs.berkeley.edu/~krste/papers/compression-mobisys2003.pdf

K.Sayood. (2006). *Introduction to data compression 3rd edition*. Retrieved from:
http://rahilshaikh.weebly.com/uploads/1/1/6/3/11635894/data_compression.pdf

K.Sutner. (2003). *LZW Compression.* Retrieved from:

http://www.cs.cmu.edu/~ab/Desktop/15-211%20Archive/res00107/LZW.pdf

M.Al-laham, I.M.M.El Emary. (2007). *Comparative Study Between Various Algorithms of Data Compression Techniques.* Retrieved from:

http://www.iaeng.org/publication/WCECS2007/WCECS2007_pp326-336.pdf

M.Nelson, J.L.Gailly. (Unknown). The Data Compression Book Second Edition. Retrieved from:

http://lib.mdp.ac.id/ebook/Karya%20Umum/The-Data-Compression-Book.pdf

R.C.Gonzalez, R.E.Woods. *Digital Image Processing.* Prentice Hall, 2002.

R.M.Gray. (2013). *Entropy and Information Theory First Edition, Corrected.* Retrieved from:

http://www-ee.stanford.edu/~gray/it.pdf

S.Pigeon. (Unknown) *Huffman Coding.* Retrieved from:

http://www.stevenpigeon.com/Publications/publications/HuffmanChapter.pdf

S.R. KODITUWAKKU, U. S.AMARASINGHE. (2009). *COMPARISON OF LOSSLESS DATA COMPRESSION ALGORITHMS FOR TEXT DATA.* Retrieved from:

http://www.ijcse.com/docs/IJCSE10-01-04-23.pdf

T.H.Cormen, C.E.Leiserson, R.L.Rivest, C.Stein. (2009) *INTRODUCTION TO ALGORITHMS THIRD EDITION.* Retrieved form:

https://robot.bolink.org/ebooks/Introduction%20to%20Algorithms%203e%20-%20Thomas%20Cormen,%20Charles%20Leiserson%20and%20Ronald%20Rivest.pdf

6. APPENDIX

6.1 LIST OF FIGURES

6.2 LIST OF TABLES

Table 1. Huffman coding experiments based on Calgary corpus test file (book2) part 1

Bit Length(bits)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Entropy	0.9937	1.95482	2.9627	3.4085	4.8758	5.5394	6.6651	4.7926	7.4365	6.95614	7.43461	6.0251	7.4355	6.9586	7.4346	4.790
Compression ratio	1	1	1	1.1678	1.0161	1.0763	1.0465	1.6586	1.2061	1.4322	1.4746	1.98212	1.7426	2.0046	2.0108	3.3189
Compression time (seconds)	118.61	105.58	126.54	90.08	123.28	99.38	118.53	59.58	103.30	67.81	84.89	56.41	73.38	53.04	63.96	34.84

Bit Length(bits)	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Entropy	7.4348	6.9553	7.4355	6.0275	7.43589	6.9539	7.4371	4.7879	7.4368	6.95747	7.4349	6.0280	7.4390	6.9531	7.43719
Compression ratio	2.2788	2.57845	2.5468	3.3022	2.8148	3.1518	3.08227	4.9805	3.3501	3.72287	3.61933	4.62388	3.8856	4.2985	4.1542
Compression time (seconds)	57.19	43.17	51.96	32.52	47.26	33.66	42.73	24.82	40.11	37.06	37.32	29.20	39.11	28.59	35.22

Table 2. Huffman coding experiments based on Calgary corpus test file (book2) part 2

Bit Length(bits)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Entropy	0.99	1.95	2.96	3.41	4.88	5.57	6.69	4.98	7.48	7.03	7.48	6.15	7.67	7.03	7.48	4.98
Compression ratio	1.00	1.00	1.00	1.16	1.02	1.07	1.04	1.59	1.20	1.42	1.47	1.94	1.73	1.98	2.00	3.19
Compression time (seconds)	20.20	17.87	14.21	12.26	12.26	12.50	12.17	8.69	13.14	10.21	10.92	7.80	9.84	7.13	7.28	3.59

Table 3. Huffman coding experiments based on Calgary corpus test file (paper1.txt) part 1

Bit Length(bits)	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Entropy	7.48	7.02	7.46	6.15	7.46	7.04	4.67	4.99	7.47	6.957	7.46	6.14	7.47	7.04	7.46
Compression ratio	2.27	2.55	2.54	3.24	2.81	3.11	3.07	4.77	3.33	3.723	3.61	4.53	3.87	4.24	4.14
Compression time (seconds)	5.89	4.47	6.13	3.91	5.18	4.05	4.53	2.31	4.17	37.06	4.04	2.94	4	2.98	3.92

Table 4. Huffman coding experiments based on Calgary corpus test file (paper1) part 2

Bit Length(bits)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Entropy	0.69	1.17	1.92	1.64	2.96	2.47	3.70	1.09	3.89	3.01	3.89	2.12	3.89	3.01	3.88	1.09
Compression ratio	1.00	1.45	1.52	2.21	1.67	2.15	1.85	5.12	2.29	3.27	2.79	5.34	3.30	4.58	3.81	10.23
Compression time (seconds)	188.27	121.69	91.90	67.50	80.20	62.25	63.83	33.54	58.42	43.44	43.31	29.72	38.84	30.49	35.16	16.01

Table 5. Huffman coding experiments based on Calgary corpus test file (pic) part 1

Bit Length(bits)	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Entropy	3.86	3.00	3.88	2.12	3.89	3.01	3.89	3.89	3.89	3.01	3.88	2.12	3.89	3.01	3.89
Compression ratio	4.32	5.9	4.83	8.89	5.33	7.2	5.84	6.35	6.35	8.51	6.88	12.44	7.37	9.82	7.88
Compression time (seconds)	27.86	21.51	25.02	15.28	22.91	17.26	16.81	18.6	18.27	15.46	17.99	11.64	17.1	12.97	15.38

Table 6. Huffman coding experiments based on Calgary corpus test file (pic) part 2