Multimedia Project Report

Course： Fundamentals of Multimedia

Project Name: Compressor

Name: LiuBin(刘彬)

StudentID: 3062211106

Class: Software Engineering 0604

目录

1.Members

序号	学号	专业班级	姓名	性别	分工
1	3062211106	软件工程	刘彬	男	全部工作

2.Project Introduction

2.1 Background

In multimedia application, compression is a wildly used field because of its advantage of saving disk storage and reducing transfer burden. The compression can be divided in two categories, lossless and lossy based on the algorithm it used. The both categories have a wild usage, but the lossy algorithm is more suitable for graphic or video compression because they have certain format and the lost information can be retrieved easily without harming the quality of original picture. On the hand, lossless algorithm can be used for compressing a wider range of files. Besides, some of the data indeed need to be stored without any loss. In my project, I focus my attention on the lossless compression which can compress all kinds of files.

There are many compression algorithms; one of the most famous algorithms is Huffman coding which is first presented by David A. Huffman in a 1952 paper, this method attracted an overwhelming amount of research and has been adopted in many important applications, such as fax machines, JPEG, and MPEG. Another widely used algorithm is LZW dictionary-based coding which employs an adaptive, dictionary-based compression technique. Unlike variable-length coding, in which the lengths of the codeword are different, LZW uses fixed-length codeword to represent variable-length string of character that commonly occur together.

In my project, I will mainly focus on Huffman algorithm; Huffman algorithm has advantages in compressing the file which contains much duplicate information. In order to analysis the Huffman algorithm, my program should have the ability to compress all kinds of files, and I will have to analysis the time and effect it performance.

2.2 Project Description

This is a program which can compress all kinds of files; the program can retrieve data from a text file, then analysis the frequency of each character and construct a Huffman code table. This program will display the code table and the codeword for study.

Besides, this program can indeed do the compression work just like the famous software WinRAR does. That means, the program can open any files and compress it to a specific file format, and can decompress it later. I need to record the efficiency of the compression work.

3. Technique Details

3.1 Algorithm Description

In Huffman coding algorithm, the key feature is Huffman Tree. To some extent, all you need to do is construct a tree according to Huffman algorithm. To do this, the first step you need to do is:

1. Sort the symbols according to the frequency count of their occurrences.

For example, let's have look in word "Hello", the word count as follows:

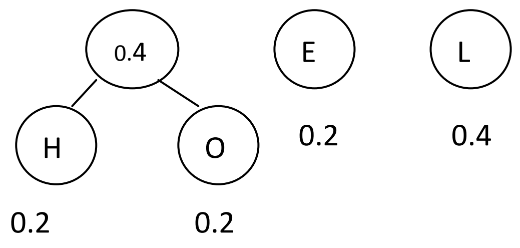
Word	**H**	**E**	**L**	**O**
Count	1	1	2	1

So we need to sort these characters in a rising order according to their frequency of occurrences.

The second step is:

2. Merge the first two items to form a new node whose frequency is the sum of the two items. Then sort the symbols again.

Let's see the example again:



By repeating the step 1 and step 2, finally we can get a Tree whose root value is 1. Then we can transverse the tree from the root, in which the left branches are coded 0 and right branches 1.

The Huffman Algothrim as follows:

```
1. Initialization: put all symbols on the list sorted according to their frequency
   counts
2. Repeat until the list has only one symbol left.
   (a) From the list, pick two symbols with the lowest frequency counts. Form a
       Huffman sub tree that has these two symbols as child nodes and create a
       parent node for them.
   (b) Assign the sum of the children's frequency counts to the parent and insert
       it into the list, such that the order is maintained
   (c) Delete the children from the list.
3. Assign a code word for each leaf based on the path from the root.
```

Algorithm 1

3.2 File Operation

Why I mentioned the file operation? Because read/write as a bit is also an important technique trick in this project. If we ignore this technique, which means we do file operation based on byte, then we will not get desired result—then compressed may be even larger than the original file. So, in order to get a better result, my program will do file operation based on bit.

In order to achieve this, I used two methods:

- Divide the code word to strings with length of 8 and then covert it to byte which values between 0 and 255.
- Store a byte in an array, then get one bit each time, there is a variable capacity to record how many bits the byte left, if we collect 8 bits, then we covert it to a byte and write to the destination file.

The first method is mainly used in writing and getting the code table because it is easier to implement. In compression process, I first write 'H' and 'U' at the beginning of the compressed file, then followed the file length. Then, it's time to write code table. The procedure is list in the diagram below. The function DivideCode () is key function because it need to add 0 bits to the separated part to ensure each part has a length of 8. The same job is repeated when we decompress a file and get the code table.

```
WriteCodeTable()
{
    Foreach(key in codetable)
    {
        Write(key);
        Write(codelengh);
        If(codelength>8)
        {
            DivideCode(code);
            Write(first_8bits);
            Write(second_8bits);
        }
        Else
            Write(code);
    }
}
```

Algorithm 2

After the Huffman Tree is constructed and codetable is maintained, we need to write the content of the original file to the compressed file according to the code table. This time, we need to use the second method to write bit stream into a file because we cannot ensure each code word has the fixed length.

In order to achieve this, I write two classes: BitstreamReader and BitstreamWriter to do bit operations. Both share the same meaning, first store bits in an array, keep an variable capacity to record how many bits left in the array if the capacity falls to zero, then read/write the bit array. The structure of BitstreamReader is shown in Data Structure 1:

```
Class BitsreamReader
{
      private byte WriteByte;
      private int Capacity;
      public byte Read()
       {
          if (Capacity == 0)
          {
              ReadByte = BinaryFileReader.ReadByte();
              Capacity = 8;
          }
          Capacity--;
          return (byte)((ReadByte & (1 << (Capacity))) >> Capacity);
       }
}
```

Data Structure 1

As the BitstreamWriter is just opposite to the above structure, I won't list it.

After solving the problems of file operation, we could concentrate on compression and decompression.

3.3 Compression

3.3.1 Analysis Original File

In order to compress a file, we should first get the original file and analysis it. The program read the file by bytes, thus we needn't care the file format. All we need to do is sort the elements range from 0 to 255 according to its frequency.

In order to achieve this, I used a Dictionary<char, int> to calculate the appear count (int) of each byte. The Algorithm as follows:

```
while (readbyte() != -1)
{
        if (binaryInfo.ContainsKey(readbyte) == false)
                binaryInfo.Add(nByte, 1);
        else
                binaryInfo[nByte]++;
         count++;
          ReadByte();
}
```

Algorithm 3

After this, the Dictionary-binaryInfo records the code and how many times it occurs. Then we could use it to construct the code table which is also a Dictionary.
The Constructor of HuffmanItem has two arguments: frequency and key as shown in Algorithm 4. So we could construct the Huffman Items of according to the analyses result.

```
    foreach (int key in _codec.binaryInfo.Keys)
          items[i++] = new HuffmanItem((double)binaryInfo[key] / total, key);
```

Algorithm 4

3.3.2 Construct Huffman Tree and Code Table

Now we get an array of Huffman Items, the structure of Huffman item is listed in Data Structure 2:

```
class HuffmanItem
{
    private double _frequency;
    private string _codeword;
    private HuffmanItem _Left;
    private HuffmanItem _Right;
    private string _stringvalue;

    public static HuffmanItem Merge(HuffmanItem left, HuffmanItem right)
    public static void UpdateTree(HuffmanItem[] HuffmanArray)
    public static void UpdateCodeword(HuffmanItem root)
    public static HuffmanItem ReconstructHuffmanTree(Dictionary<string, string> codetable
}
```

Data Structure 2

There four static methods in this class:

- Merge ()—combine first two Huffman Items
- UpdateTree ()----construct a Huffman tree according to HuffmanArray
- UpdateCodeword()----get the codeword from the root
- ReconstructHuffmanTree()—reconstruct a Huffman tree according to a code table

The key steps in construct the Huffman tree is UpdateTree. The pseudo code as follows

```
UpdateTree()
{
    While(not finished)
    {
        Sort()
        Merge()
        Updatecodeword()
    }
}
```

Algorithm 5

This is very easy to understand which follows the Huffman Algorithm closely.

Next step we need to construct code table using this Huffman Tree. So we need to traverse the Huffman Tree and get both the code value and the keywords.

This algorithm is also very easy to implement with the help of recursion, see Algorithm 6:

```
ShowHuffmanresult (Root)
{
    if (root.stringvalue != "")
        _strcodetable[root.stringvalue] = root.codeword;

    if (root.Left != null)
         ShowHuffmanresult(root.Left, listview);
    if (root.Right != null)
         ShowHuffmanresult(root.Right, listview);
}
```

Algorithm 6

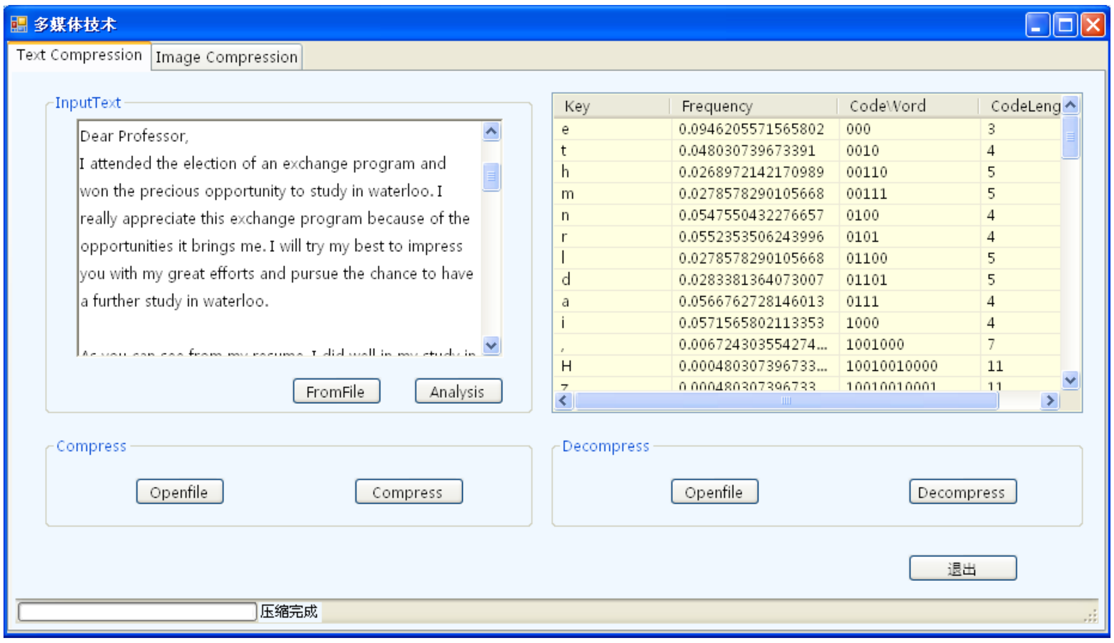
3.4 Decompression

After finishing the Compression work, the decompression seems to be very easy. The first thing to do is also analysis compressed file. We just follow the rules we compressed the file: first two character ‘H’ and ‘U’, follows the file length, and then follows the code table. The most import thing in decompression is reconstruct codetable, if you reconstruct wrong codetable, you will fail to decompression the file.

After you get the code table, you can read the bit stream from the rest of the file and recover it according to the Huffman Tree.

4. Experiments Results

4.1 User Interface



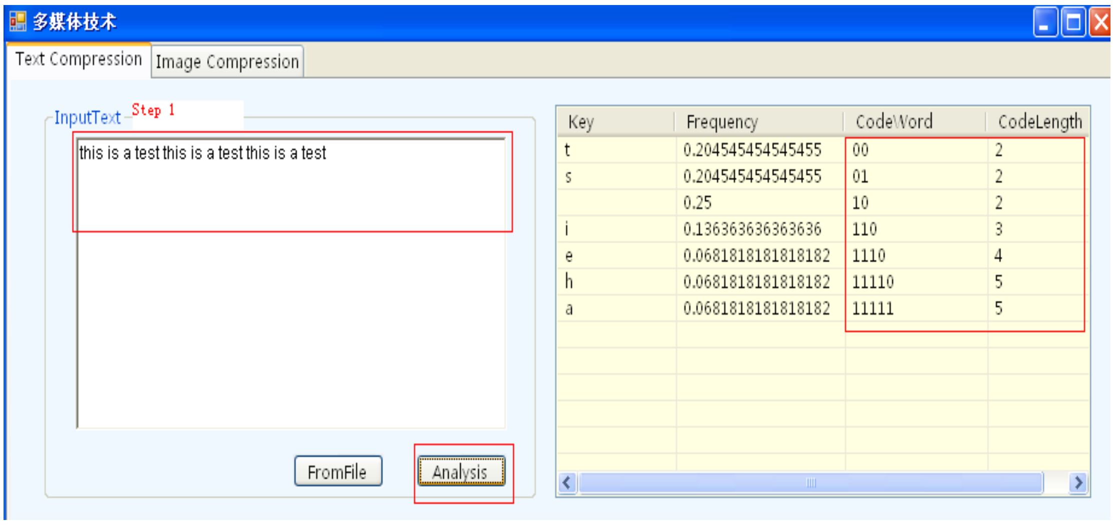
Dear Professor,
I attended the election of an exchange program and won the precious opportunity to study in waterloo. I really appreciate this exchange program because of the opportunities it brings me. I will try my best to impress you with my great efforts and pursue the chance to have a further study in waterloo.

Key	Frequency	CodeWord	CodeLeng
e	0.0946205571565802	000	3
t	0.048030739673391	0010	4
h	0.0268972142170989	00110	5
m	0.0278578290105668	00111	5
n	0.0547550432276657	0100	4
r	0.0552353506243996	0101	4
l	0.0278578290105668	01100	5
d	0.0283381364073007	01101	5
a	0.0566762728146013	0111	4
i	0.0571565802113353	1000	4
,	0.006724303554274...	1001000	7
H	0.000480307396733...	10010010000	11
z	0.000480307396733	10010010001	11

The user interface of the program is list above. As you can see, there is a input text area, this area is used to show your compressed text, you can either type whatever you like or get the text from a text file. Then if you type analysis button, you could get the result from the list view on the right side which record the information of key, frequency, codeword and code length.
On the bottom of the interface is the main function—compress and decompress. You should first open a file and then click compress or decompress button to get expected result.

4.2 Analysis Text

- **From Text Box**
  First we should test whether the program gives the right Huffman Tree. As we could see from picture 2, we could type any text in the text box, then, we click Analysis button, we could get the coding information from the listview on the right.



this is a test this is a test this is a test

Key	Frequency	CodeWord	CodeLength
t	0.204545454545455	00	2
s	0.204545454545455	01	2
	0.25	10	2
i	0.136363636363636	110	3
e	0.0681818181818182	1110	4
h	0.0681818181818182	11110	5
a	0.0681818181818182	11111	5

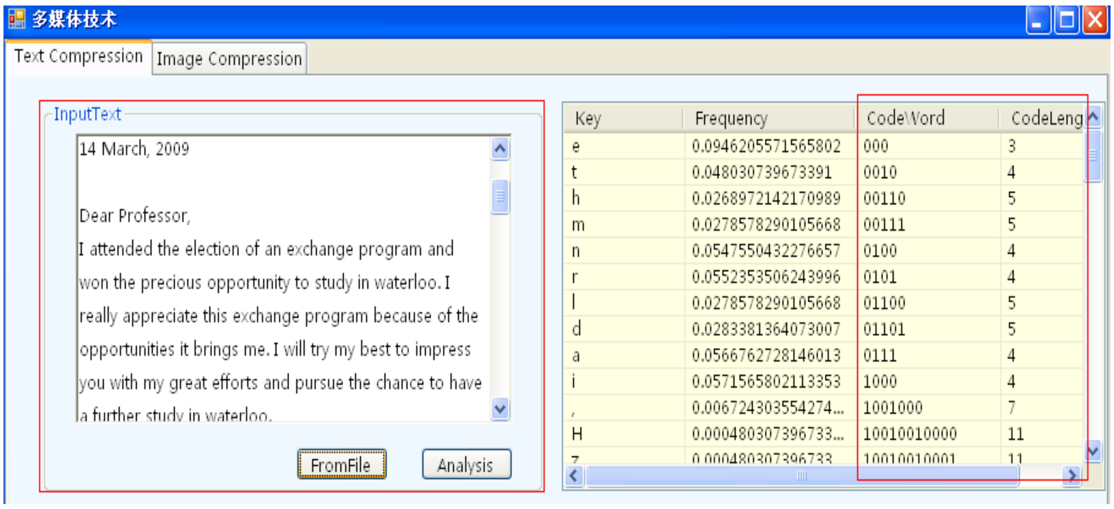
Picture 2

We could see from the result, when I type "this is a test" for three times and press the analysis button, the coding result is on the right and the result is right. The average code length is 3.28, but the original code length is only 3. This is because there are 8 characters thus only need 3 bit to represent.

- **From Text file**

We could also get text from text file. As shown in picture 3, we could click FromFile button and then we could got the coding information on the right.

From the picture, we could see the analysis coding result is on the right side according to the original text file. The average code length is shorter than the original code length because there are more sample characters this time.



14 March, 2009

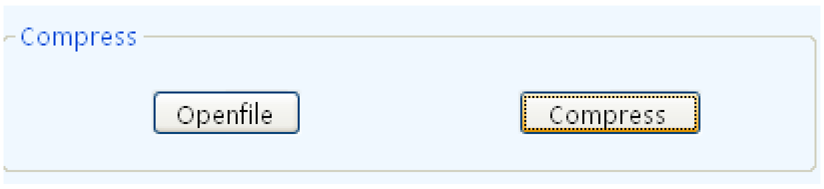
Dear Professor,
I attended the election of an exchange program and won the precious opportunity to study in waterloo. I really appreciate this exchange program because of the opportunities it brings me. I will try my best to impress you with my great efforts and pursue the chance to have a further study in waterloo.

Key	Frequency	CodeWord	CodeLeng
e	0.0946205571565802	000	3
t	0.048030739673391	0010	4
h	0.0268972142170989	00110	5
m	0.0278578290105668	00111	5
n	0.0547550432276657	0100	4
r	0.0552353506243996	0101	4
l	0.0278578290105668	01100	5
d	0.0283381364073007	01101	5
a	0.0566762728146013	0111	4
i	0.0571565802113353	1000	4
,	0.006724303554274...	1001000	7
H	0.000480307396733...	10010010000	11
z	0.000480307396733	10010010001	11

Picture 3

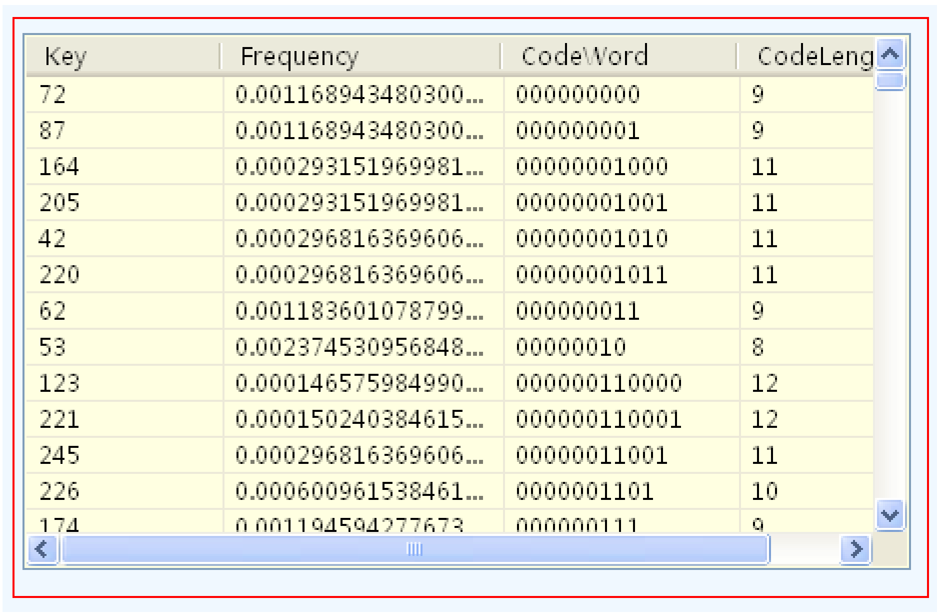
4.3 Compress file

The first process of compression is open file and analysis is, according to Picture 4, you just need to click open file and then click compress, type the saved file name, and wait.



Picture 4

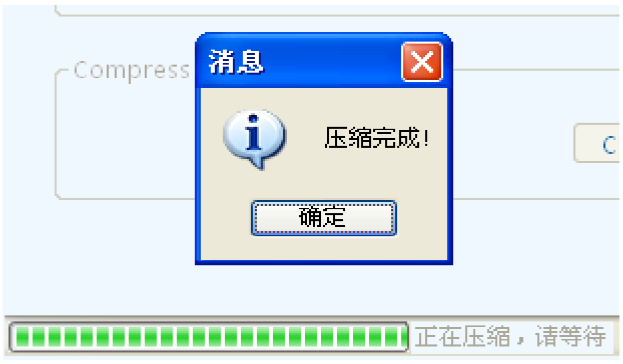
Then the analysis information will appear in the list view. From Picture 5, we could see the Huffman coding information, Ker refers to the byte value of each 8 bits, Frequency refers to its frequency, code word refers to the coding result after analysis the file and code length refers to the length of the code.



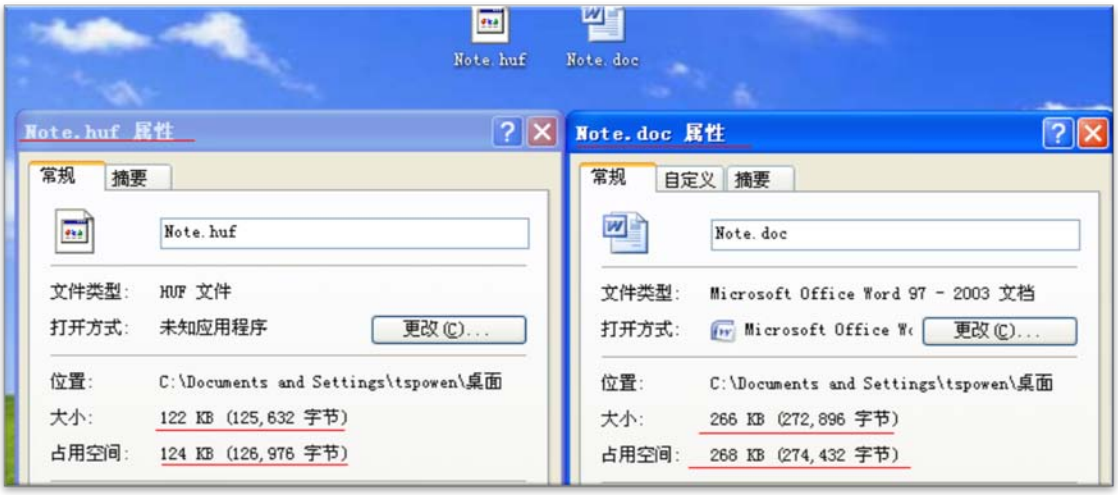
Key	Frequency	CodeWord	CodeLeng
72	0.001168943480300...	000000000	9
87	0.001168943480300...	000000001	9
164	0.000293151969981...	00000001000	11
205	0.000293151969981...	00000001001	11
42	0.000296816369606...	00000001010	11
220	0.000296816369606...	00000001011	11
62	0.001183601078799...	000000011	9
53	0.002374530956848...	00000010	8
123	0.000146575984990...	000000110000	12
221	0.000150240384615...	000000110001	12
245	0.000296816369606...	00000011001	11
226	0.000600961538461...	0000001101	10
174	0.001194594277673	000000111	9

Picture 5

We could now press compress button, and get compress finished information. This work may take seconds according to the size of file.



Let’s see the compress efficiency. I compress a Microsoft word file—note.doc which has the size of 266KB, I save the compressed file as Note.huf, and the size of the file is 122KB, the compress ration is 144/266 = 54%



Note.huf 属性

常规 摘要

Note.huf

文件类型: HUF 文件
打开方式: 未知应用程序 更改(C)...
位置: C:\Documents and Settings\tspowen\桌面
大小: 122 KB (125,632 字节)
占用空间: 124 KB (126,976 字节)

Note.doc 属性

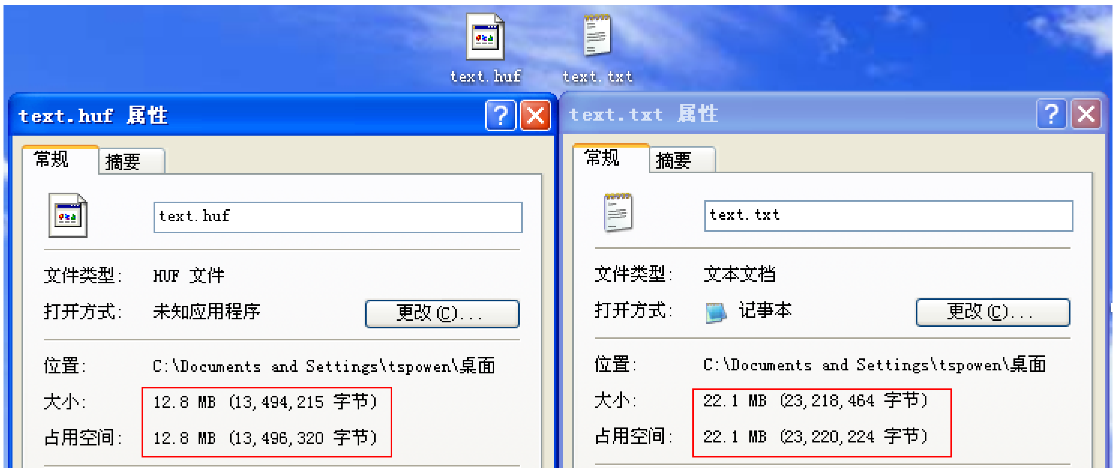
常规 自定义 摘要

Note.doc

文件类型: Microsoft Office Word 97 - 2003 文档
打开方式: Microsoft Office W 更改(C)...
位置: C:\Documents and Settings\tspowen\桌面
大小: 266 KB (272,896 字节)
占用空间: 268 KB (274,432 字节)

Picture 6

Let’s see the compression of text file, I compressed a txt file whose size is 22.1MB, after compression, the size of compressed file is only 12.8MB, the compression ratio is 44%. The result is wonderful.



text.huf 属性

常规 摘要

text.huf

文件类型: HUF 文件
打开方式: 未知应用程序 更改(C)...
位置: C:\Documents and Settings\tspowen\桌面
大小: 12.8 MB (13,494,215 字节)
占用空间: 12.8 MB (13,496,320 字节)

text.txt 属性

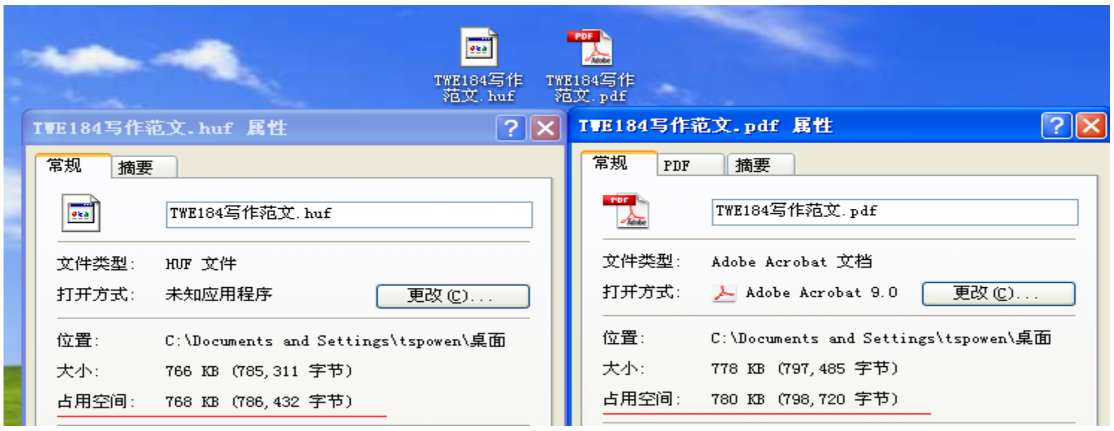
常规 摘要

text.txt

文件类型: 文本文档
打开方式: 记事本 更改(C)...
位置: C:\Documents and Settings\tspowen\桌面
大小: 22.1 MB (23,218,464 字节)
占用空间: 22.1 MB (23,220,224 字节)

Picture 7

Now let’s have a look on the compression of another text file—PDF, we compress a PDF file—“TWE184 写作范文” whose size is 780KB, but the compressed file, as shown in the picture 7 is 768KB, only compressed 12KB, and the compress ration is 1.5%



TWE184写作范文.huf
TWE184写作范文.pdf

TWE184写作范文.huf 属性

常规 | 摘要

TWE184写作范文.huf

文件类型: HUF 文件
打开方式: 未知应用程序 更改(C)...
位置: C:\Documents and Settings\tspowen\桌面
大小: 766 KB (785,311 字节)
占用空间: 768 KB (786,432 字节)

TWE184写作范文.pdf 属性

常规 | PDF | 摘要

TWE184写作范文.pdf

文件类型: Adobe Acrobat 文档
打开方式: Adobe Acrobat 9.0 更改(C)...
位置: C:\Documents and Settings\tspowen\桌面
大小: 778 KB (797,485 字节)
占用空间: 780 KB (798,720 字节)

Picture 7

4.4 Compress Multimedia file

We have already seen the result of compress text file, now have a look at multimedia files, such as BMP, AVI, MP3 .
I will show the result briefly in a diagram:

File Type	Mp3	AVI	JPG	BMP
Original File Size	10MB	18.7MB	137KB	300KB
Compressed File Size	9.2MB	18.4MB	102KB	200KB
Compress Ratio	8%	1%	25%	33%

So the compression result is not so well as text compression. Later I will do analysis.

4.5 Compress compressed file

There are also a category of files which is already compressed, such as WinRAR or WinZip. It is very funny that the compressed file is even larger than the original file. A WinZip file with size of 1.53KB comes to 2.25KB after compression. So it is of no use to compress a compressed file.

4.6 Conclusion

- **Phenomenon**

1. From the result list above, we can see that Huffman coding is not suitable for all kind of file, the compress ratio range from 1% to 54%. Typically, the compress ratio is higher among text files, while the result is poor among multimedia files.
2. There are also differences in compressing the same file format—the more repeat contents in the file, the better compress ratio can we get.

- **Reason**

The reason why Huffman coding is not suitable for some file format is those files have already been processed, some of them are already compressed such as JPG or MP3. No wonder why the compress ratio is very low.

Second, the more repeat contents in the file, the shorter code word it can be assigned, because the frequency is very high according to Huffman Algorithm, so the compression ratio is better.

References:

1. <Fundamentals of Multimedia> ---- Ze-Nian Li   Mark S. Drew
2. <Professional C# 2005> ---- Jay Glynn

Enclosure：

Source Code