ISSN:0975-9646

Simultaneous Data Compression and Encryption

T.SubhamastanRao [1], M.Soujanya,[2] T.Hemalatha,[3] T.Revathi,[4]

[#1,3,4], *Dept of IST, KL University,Guntur Dt, Andhra Pradesh,India*
*[*2]Dept of CSE, MRECW College, Hyderabad, Andhra Pradesh,India*

***ABSTRACT*-Data compression is known for reducing storage and communication costs. It involves transforming data of a given format, called source message to data of a smaller sized format called codeword. Data encryption is known for protecting formation from eavesdropping. It transforms data of a given format, called plaintext, to another format, called cipher text, using an encryption key. The major problem existing with the current compression and encryption methods is the speed, i.e. the processing time required by a computer. To lessen the problem, combine the two processes into one. The basic idea of the combining the two processes is to add a pseudo random shuffle into a data compression. The aim behind the shuffling process is to get the different Huffman table for the original Huffman tree by shuffling nodes in the tree. Using the numbers generated by pseudo random generator, nodes in the Huffman tree are shuffled. After the encryption we will get one mapping Huffman table, which is not identical to the original. Finally this table only will send across the network. Once the Huffman table is encrypted no one having the decompression module can decrypt it. So in this new algorithm compression and encryption is done simultaneously.**

KEYWORDS:Encryption,Decryption,Compression,Datacompression,Decompression,Huffman compression.

1. INTRODUCTION

Security of network communications is arguably the most important issue in the world today given the vast amount of valuable information that is passed around in various networks. While larger files need to be sent on network with security, it has to be encrypted. The files larger in size when encrypted still increases in size. Hence normally its compressed and sent across the network. Data compression is known for reducing storage and communication costs. It involves transforming data of a given format, called source message, to data of a smaller sized format, called codeword. Data encryption is known for protecting information from eavesdropping. It transforms data of a given format, called plaintext, to another format, called cipher text, using an encryption key. The major problem existing with the current compression and encryption methods are

1. Low Speed.
2. More processing time.
3. More Cost.

To lessen the problem, our approach combines the two processes (Compression and Encryption) into one process. In the new approach both encryption and compression are done at the same time. It takes less processing time and more speed. For compression Huffman compression algorithm is used. Add pseudo Random shuffle into the data compressed. Nodes in the tree are shuffled. Shuffling is done to get different Huffman table. After encryption we will get one shuffled Huffman table. i.e. Not identical to original .Finally this table is send across the network. Once encrypted no other than the intended receiver can decrypt it. Hence compression and encryption is done simultaneously.

1.1. Existing system:

Currently compression and encryption methods are done separately The major problem existing with the current compression and encryption methods is the speed, i.e. the processing time required by a computer. Because doing two processes takes more time.

1.2. Proposed system:

To lessen the problem, this approach combines the two processes into one. i.e. proposed a new approach which will perform both encryption and compression at the same time. This approach lessons the processing time required by a computer to do the compression and encryption processes.

2. SYSTEM ANALYSIS AND DESIGN

2.1. PROBLEM DEFINITION:

Currently compression and encryption methods are done separately. The major problem existing with the current compression and encryption methods is the speed, i.e. the processing time required by a computer .Because doing two processes takes more time. To lessen the problem, our approach combines the two processes (Compression and Encryption) into one process. In the new approach both encryption and compression are done at the same time. It takes less processing time and more speed. For compression Huffman compression algorithm is used. Add pseudo Random shuffle into the data compressed. Node in the tree is shuffled. Shuffling is done to get different Huffman table. After encryption we will get one shuffled Huffman table i.e.; not identical to original .Finally this table is send across the network. Once encrypted no other than the intended receiver can decrypt it. Hence compression and encryption is done simultaneously.

2.2. DESIGN METHODOLOGY

2.2.1. INTRODUCTION TO DATA COMPRESSION:

Data compression is often referred to as coding, where coding is a very general term encompassing any special representation of data which satisfies a given need. Information theory is defined to be the study of efficient coding and its consequences, in the form of speed of transmission and probability of error. Data compression may be viewed as a branch of information theory in which the primary objective is to minimize the amount of data to be transmitted. The purpose of this paper is to present and analyze a variety of data compression algorithms. A simple characterization of data compression is that it involves transforming a string of characters in some representation

www.manaraa.com

(such as ASCII) into a new string (of bits, for example) which contains the same information but whose length is as small as possible. Data compression has important application in the areas of data transmission and data storage. Many data processing applications require storage of large volumes of data, and the number of such applications is constantly increasing as the use of computers extends to new disciplines. At the same time, the proliferation of computer communication networks is resulting in massive transfer of data over communication links. Compressing data to be stored or transmitted reduces storage and/or communication costs amount of compression that can be obtained with lossless compression. Lossless compression ratios are generally in the range of 2:1 to 8:1. Lossy compression, in contrast, works on the assumption that the data doesn't have to be stored perfectly. Much information can be simply thrown away from images, video data, and audio data, and the when uncompressed, the data will still be of acceptable quality. Compression ratios can be an order of magnitude greater than those available from lossless methods. The question of which is "better", lossless or lossy techniques, is pointless. Each has its own uses, with lossless techniques better in some cases and lossy techniques better in others. In fact, lossless and lossy techniques are often used together to obtain the highest compression ratios. Even given a specific type of file, the contents of the file, particularly the orderliness and redundancy of the data, can strongly influence the compression ratio. In some cases, using a particular data compression technique on a data file where there isn't a good match between the two can actually result in a *bigger* file.

2.2.2. HUFFMAN COMPRESSION ALGORITHM

Huffman compression is a primitive data compression scheme invented by Huffman in 1952. The Huffman compression algorithm is named after its inventor, David Huffman, formerly a professor at MIT. This code has become a favorite of mathematicians and academics, resulting in volumes of intellectual double-talk. Huffman's patent has long since expired and no license is required. There are however many variations of this method still being patented. The code can easily be implemented in very high speed compression systems. The Huffman code assumes "prior knowledge" of the relative character frequencies stored in a table or library. A secret table made available only to authorized users can be used for data encryption. A more sophisticated and efficient lossless compression technique is known as "Huffman coding", in which the characters in a data file are converted to a binary code, where the most common characters in the file have the shortest binary codes, and the least common have the longest. The Huffman Compression algorithm is an algorithm used to compress files. It does this by assigning smaller codes to frequently used characters and longer codes for characters that are less frequently used. Huffman Compression, also known as Huffman Encoding, is one of many compression techniques in use today. Others are LZW, Arithmetic Encoding, RLE and many more. One of the main benefits of Huffman Compression is how easy it is to understand and implement yet still gets a decent compression ratio on average files.

. Example-1 the Huffman tree is as shown below

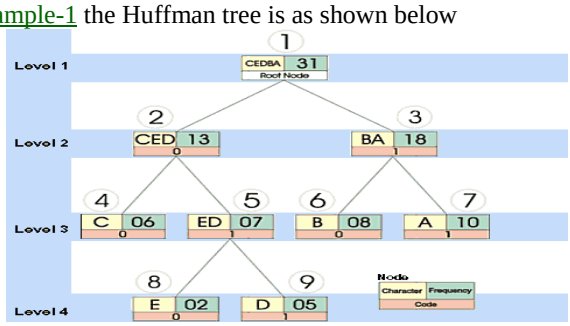

Figure1

Each non leaf node of the tree has two child nodes, the left child node and the right child node. The non leaf node is known as the parent node of these two child nodes. Similarly the parent of a parent node is the grandparent of the child nodes. The parent, grand parent, great grandparent etc. are collectively called the ancestors of a child node. The child nodes are called the descendants of the ancestors. In the tree above node 3 is the parent of nodes 6 and 7. Node 1 is the parent of 3. It is also the grandparent of nodes 6 and 7. Generally nodes 3 and 1 are the ancestors of nodes 6 and 7. Nodes 6 and 7 are the descendants of nodes 3 and 1. Note that if n characters are present in a file then the number of nodes in the Huffman tree is 2n-1. If there are n nodes in a tree then there can be at most (n+1)/2 levels, and at least log2(n+1) levels. The number of levels in a Huffman tree indicates the maximum length of code required to represent a character.

The code length of a character indicates the level in which the character lies. If the code length of a character is n then it lies in the (n+1)th level of the tree. For example the code length of character D is 011. the code length is 3. Therefore this character must lie in the 4th level of the tree. The code for each character is obtained by starting from the root node and traveling down to the leaf that represents the character. When moving to a left child node a '0' is appended to the code and when moving to a right child node a '1' is appended to the code. To get the code for the character 'A' from the tree, we first start at the root node (i.e node 1). Since the character 'A' is the descendant of the right child node (We decide which branch to follow by testing to see which branch either is the leaf node for the character or is its ancestor) we move to the right and append a '1' to the code for character 'A'. Now we are on node 3. The leaf node for character 'A' lies to the right of this node, so we again move to the right and append a '1' to its code. We have now reached node 7 which is the leaf node for the character 'A'. Thus the code for character 'A' is 11. In similar fashion the codes for other characters can also be obtained. You can see that codes of characters having higher frequencies are shorter than those having lower frequencies,. There are two types of Huffman coding: static and adaptive. In a static Huffman coding, the Huffman tree stays the same in the entire coding process. In an adaptive Huffman coding, the Huffman tree changes according to the data processed.

www.manaraa.com

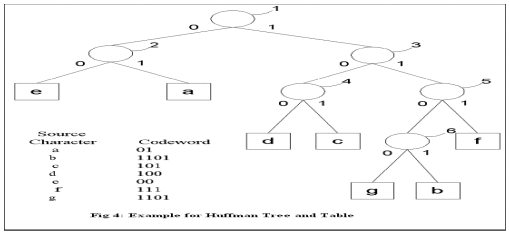

Figure2

Once the Huffman tree is built, regardless of its type, the encoding process is identical. The codeword for each source character is the sequence of labels along the path from the root to the leave node representing that character. For example, in above FIG., the codeword for 'a' is '01', 'b ' is '1101', etc.

2.3. SYSTEM ARCHITECTURE:

Data compression is known for reducing storage and communication costs. It involves transforming data of a given format, called source message to data of a smaller sized format called codeword. Data encryption is known for protecting information from eavesdropping. It transforms data of a given format, called plaintext, to another format, called cipher text, using an encryption key. The major problem existing with the current compression and encryption methods is the speed, i.e. the processing time required by a computer. To lessen the problem, combine the two processes into one. The basic idea of the combining the two processes is to add a pseudo random shuffle into a data compression. The aim behind the shuffling process is to get the different Huffman table for the original Huffman tree by shuffling nodes in the tree. Using the numbers generated by pseudo random generator, nodes in the Huffman tree are shuffled. After the encryption we will get one mapping Huffman table, which is not identical to the original. Finally this table only will send across the network. Once the Huffman table is encrypted no one having the decompression module can decrypt it. So in this new algorithm compression and encryption is done simultaneously.

2.4. DATA FLOW DIAGRAM:

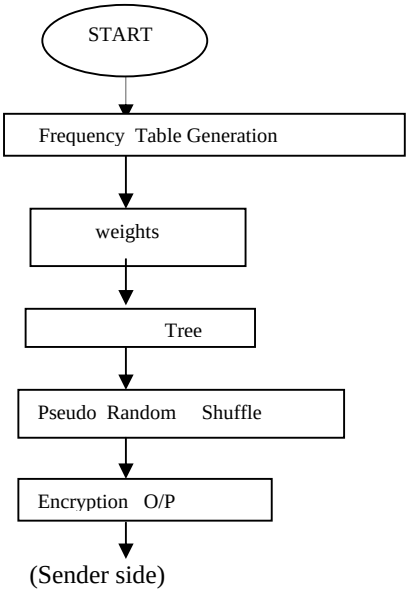

Figure3

A data flow diagram (DFD) is a graphical reprasenttion that depicts information flow and the transform that are applied that are applied as data moves from input to output. The DFD may be used to represent a system or software at the level of abstraction.DFDs may be partitioned into level that represents information and functional details. Hence DFD provides a mechanism for functional modeling as well as information modeling. A level-0 DFD also called as fundamental system model or a context model represents a entire software element as single bubble with input and output processes(bubble)and information flow paths are represented as level-0 DFD is partitioned to reveal more details that is level-1 may contain 5 or 6 bubbles with inter connecting arrows.Each is the process represented at level-1 is a sub function of overall system depictured in the ontext model. A Rectangle is used to represent an External Entity i.e., a system element(e.g. Hardware, a person, another program)or another system produce information for transformation by the software or receives the information produced by the software. A circle(also called bubble) represents a process or transform that is applied to data(or control) and changes it in some way. An Arrow represents one or more data iems, the double line represents the data store-store information that is used by software. The simplicity of DFD notation is one reason why structures analysis techniques are widely used.

2.5. COMPRESSION DIAGRAM

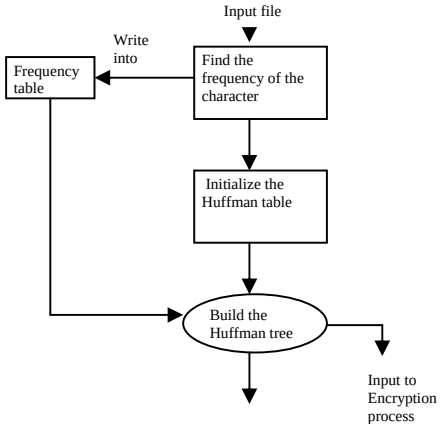

Figure 4.

2.6. DECOMPRESSION DIAGRAM:

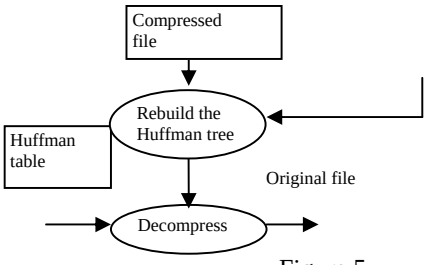

Figure 5.

www.manaraa.com

2.7. CODE DESIGN

This paper basically consists of four modules. They are

- Compression/Encryption module
- Decryption module/ Decompression

2.7.1. COMPRESSION & ENCRYPTION MODULE:

In this paper, to compress the file Huffman compression algorithm is used. To compress the file first we need to generate the Huffman tree for the given input file.

ALGORITHM FOR HUFFMAN ENCODING

Step I: Generate the table of frequencies

Make one pass through the original file to determine:

- The number of bytes in the file (original file size).
- The number of unique bytes (the alphabet size).
- The number of times each unique byte appears in the file (the weights).

Step II: Build the encoding tree

- Put all of the leaf nodes in a priority queue based on their weights.
- Do the following until the priority queue is empty:
  - Delete the smallest two nodes.
  - Create a new node with these deleted nodes as children.
  - Set the parent pointers of the deleted nodes in the table.
  - Set the child type of the deleted nodes in the table.

Step III: Encoding the characters

For each character in the input file, find the leaf node corresponding to that character and then traverse the tree to its root, pretending the bits for that character. If current node is a left child, pretend a 0.If current node is a right child, pretend a 1.If current node has no parent, return the string for this character.Otherwise, the parent the current node. After the compression process the Huffman tree for the input file is generated as well as the Huffman table is created. Once the Huffman tree is generated, the file is scanned again and each character in the file is replaced by its corresponding code from the tree. Once this is done, the character and the codes for each character along with the length of each code must be saved in the file in the form of a table. This table is needed during the decompression process. The frequency of the characters does not have to be saved because it is not needed for the decompression process. After the compression process the Huffman tree are encrypted using the pseudo random shuffle. Pseudo random shuffle provides a seed with that a mapping Huffman table will be generated.

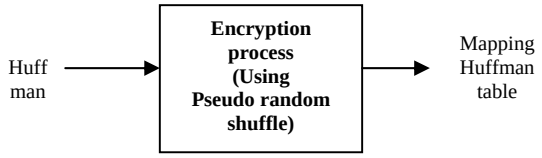

Figure 6 - Encryption Process Diagram

In this encryption module the Huffman table, which is generated in the compression process is different from the existing table. With that a new tree is generated and fixed length key is used to encrypt the Huffman table. In this Huffman table contains maximum of 256 entries. The output of this module is the mapping Huffman table. This Huffman table is saved along with the compressed file. So, once the Huffman table is encrypted no one having the decompression module can decrypt it.

- **Set the children pointers in the decoding table**
- For each entry in the table starting with index 0: (Do not include the last one because it is the root).
- Get the parent of this entry
- If the current entry is a right child , set the right child of the parent to this entry
- Otherwise set the left child of the parent to this entry
- **Set the parent of the entry to -1.**

ALGORITHM FOR WRITE UNENCODEDFILE:

For each character in the encoded file:

- Start at the root of the tree (entry currenttableSize -1 in the table).
- Do
- Read the bit
- If the bit is a 0 take the left branch.
- If the bit is a 1 take the right branch.
- Until child is -1(leaf).
- Output the leaf's byte values to the file.

CODE TO IMPLEMENT HUFFMAN ENCODING ALGORITHM:

- **Code to implement Step 1 of the Huffman Encoding algorithm :**

```
private void computeCharacterCounts() throwsIOException
{
FileInputStream f=null;
Try
{ // we catch and rethrow the exception to be sure to close file
f=new FileInputStream(unencodedfileName);
int thisByte;
while((thisbyte=f.read())!=-1) {
insertInTable(thisByte,1); {
unencodedFileSize++;
}}
catch(Exception e) {
throw new
IOException("Exception"+e.getMessage()+"reading"+unencodedFil
eName);
}
finally
{  // always want to close the file no matter what
if(f!=null)
f.close();
}
alhabetsize=currenttablesize; // now have the unique characters}
private void insertInTable(int v,int weight) {
if(invertedTable[v]==-1)
{ //if v is not in the table add it
encdingTable[currentTableSize]=newHuffmanCodingEntry(v,
currentTableSize,weight,-1,false);
invertedTable[v]=currentTableSize;
currentTableSize++;}
else // add weight to the total weight of the character v
encodingTable[invertedTable[v]].addWeight(weight); }
```


www.manaraa.com

- **Code to implement Step II of the Huffman Encoding algorithm :**

```
Private void buildTree(){
int nextInternalvalue=MAXIMUM_UNIQUE_CHARCTERS;
PriorityQueueADT q=new LinkedPriorityQueue();
for(int i=0;i<alphabetSize;i++)
q.add(encodingTable[i]);
try{
while(true)
{
HuffmanCodingEntry T1=(HuffmanCodingEntry)(q.removeMin());
HuffmanCodingEntry
T2=(HuffmanCodingEntry)(q.removeMin());T1.setParent(currentTa
bleSize);
T2.setParent(currentTableSize);
T2.setChildtype(true);
insertInTable(nextInternalValue,T1.getWeight()+T2.getWeight());
q.add(encodingTable[currentTableSize-1]);
nextInternalvalue++;
}}
catch(NoSuchElementException e) { //expected to happen at end
}}
```

- **Code to implement Step III of the Huffman Encoding algorithm :**

```
Private void writeEncodedFile()throwIoException{
BitInputStream f=null;
Try{
f=new BitInputStream(unencodedFileName);
encodedFile.writeInt(unencodedFilesize);
encodedFile.wirteInt(alphabetSize);
for(int i=0;i<currentTableSize;i++)
encodedFile.writeInt(encodingTable[i].encode());
// Read one character at a time,encode it, and write to file
int thisChar=0;
for(int j=0;j<unencodedFileSize;j++)
{
if((thisChar=f.readByte())==-1)
throw new IOException("Unexpected end of file reading character
"+ j +"in "+ unencodedFileName);
writeEncodedChar(thisChar,encodedFile);
}
encodedFile.flush();
}
catch(Exception e){
throw new IOException("Error encoding file: "+e.getMessage());
}
finally{
encodedFile.close();
if(f!=null)
f.close();}}
private void wirteEncodedChar(int thisChar,bitOutputStream bout)
throw IOException
{
int theEntry=invertedTable[thisChar]; // Find the position in the
table
int theParent=encodingTable[theEntry].getParent();
if(theParent==-1)
return;
writeEncodedChar(encodingTable[theParent].getValue(),bout);
bout.writeBit(encodingTable[theEntry].isRightChild());
}
```

- **Code to implement Huffman Decoding algorithm**

```
Byte huffman_decode_byte(pointer to binary tree node )
{
// First we have to check if we are in a fictive node or in leaf .
// In our case we decided that fictive nodes have a value
higher
// than any other symbol.If using byts,256.That's the way
// we know it.
If(node->value!=256)
{
//It's leaf,return  the symbol
return node ->value; }
else
{
//We are on a fictive node,we have to right to the
//left or the right node depending on the next bit.
// we'll return the value returned by the fuction
//called,which will be the decoded byte.
If(get_bit()==0)
{
//Go to the left
return (Huffman_decode_byte(node->left_node));
}
else
{
//Go to the right
return(huffman_decode_byte(node->right_node));
}}
```

2.8. SCREEN SHOTS

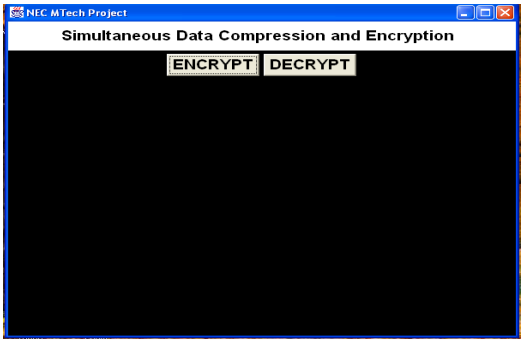

Fig.7.The above screen shows the main screen of encryption and decryption button.

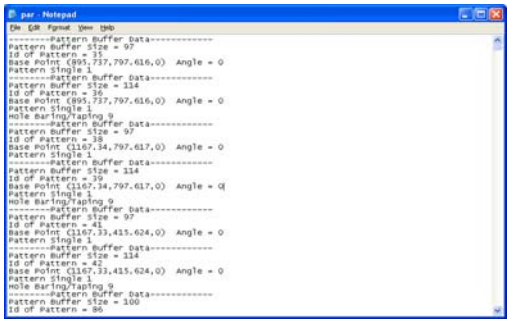

Fig 8 . This screen shows the content of the text file before the compression and encryption process.

www.manaraa.com

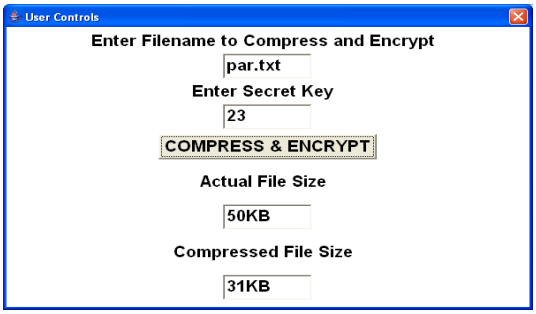

Fig 9. This screen shows the file after compression is 31KB. After this process one compresses and encryption and encrypted file is generated with the extension of " .huf " .

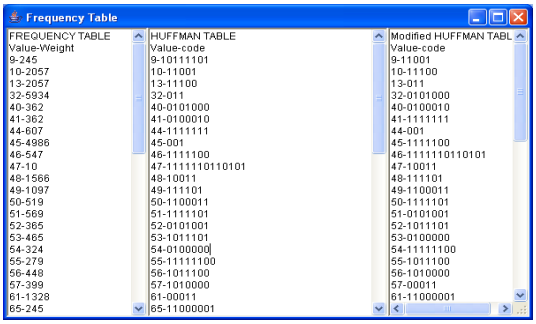

Fig 10 The screen shows the original Huffman table and the mapping Huffman table. Mapping Huffman table is generated after encryption.

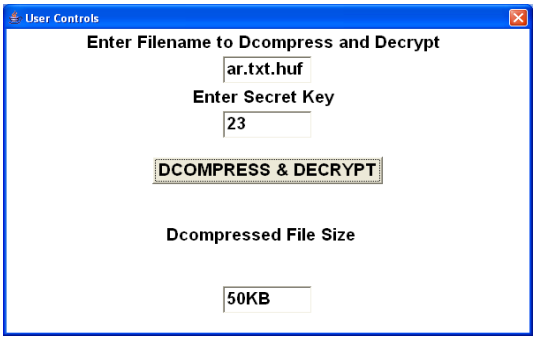

Fig 11. The screen shows the size of the file after decompression

Fig 12.This screen shows the content of the bmp file.

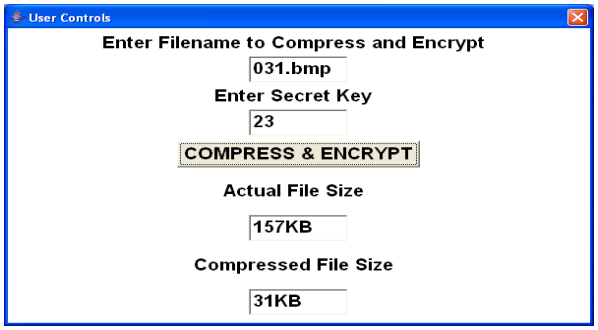

Fig 13. The size of the file after compression is 31KB. The screen shows the original Huffman table and the mapping Huffman table. Mapping Huffman table is generated after encryption.

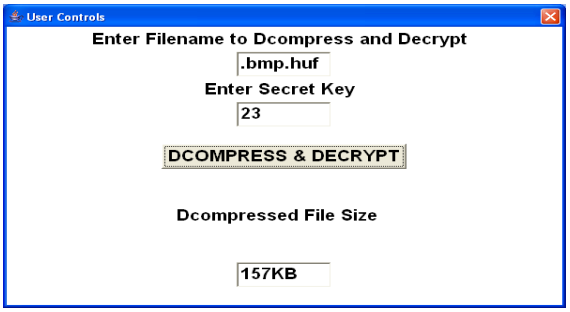

Fig 14 This is the decryption screen

CONCLUSION

Besides the obvious execution time advantage of combining the two processes of data compression and encryption, the encryption strengths of our methods are as good as any other encryption algorithms such as DES, triple DES, and RC5. In this approach the speed is very high because here we need to encrypt only the Huffman table rather encrypting the whole file which is to be transmitted. The encryption strength of the method of encrypting the Huffman table depends on the length of the encryption key. This approach is mainly developed to improve the speed and to provide more security. There may be enhancement in future to this approach, which can provide more efficiency. More over, there may be any attacks to this approach in future. So, enhancements to this approach may rectify those attacks.

REFERENCES AND BIBLIOGRAPHY

1. "Simultaneous Data Compression and Encryption" presented in SAM'03, by Chung-E Wang, Department of Computer Science California State University.
2. D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", in: Proc. IRE, 40, 9 (Sept.),1952, pp.
3. Andre Skupin "Use of Huffman trees to Encode Verbal Street Direction"
4. "Huffman Compression" Written for the PC-GPE and the World by Joe Koss (Shades)
5. "Basic Encryption "**by** Matt Recker
6. "Introduction to Data Compression" by Guy E. Blelloch, Computer Science Department, Carnegie Mellon University
7. *"The Huffman Compression Algorithm" by* Vimil Saju
8. "Computer Networks" - Tanenbaum, fourth edition 2002 PHI.
9. "Cryptography & Network Security" - Willim Stalling, third edition 2003.
10. "Software Engineering" - Pressman, fifth edition 2003 TMH.

www.manaraa.com