IMAGE ENCRYPTION USING RECURRSIVE HUFFMAN CODING AND DECODING USING HEAP STRUCTURE

SonaKhanna[1],SumanKumari[2] ,Taqdir[3]

[1]Student M.Tech, Computer Science, Guru Nanak Dev University RC, Gurdaspur, India [1,2]

[2]Assistant Professor, Computer Science, Guru Nanak Dev University RC, Gurdaspur, India [3]

ABSTRACT

The image compression is the mechanism by which size of the image is reduced. The compression techniques may be lossy or lossless in nature. The image compression will enhance the transmission process also. The image compression is achieved with the help of mechanism present within the image processing. The proposed system will use the concept of Huffman coding. The existing work focuses on the iterative approach. The iterative approach which is followed will use binary search mechanism in order to perform the encryption. The proposed work will enhance the clarity of the image in much less time than the existing approach.

INTRODUCTION

The image compression is the mechanism by which size of the image is reduced. The size of the image has to be reduced so that less bandwidth is consumed when data is being transmitted. The size will also affect the cost associated with the system. In the proposed scheme Huffman encoding is followed which is lossless form of image compression. The Huffman encoding using existing approach will use the sorting mechanism which will be complex in nature. In order to simplify the encoding and decoding process recursive Huffman coding is proposed. The coding and decoding process which is proposed has less complexity and hence is better as compared to the existing Huffman coding schemes. In order to perform simulation MATLAB is used. the Huffman code which is proposed is prefix code. The variable length code is produced using the Huffman encoding scheme. The time which is required in order to implement Huffman coding is depending upon the total number of inputs.

HUFFMAN CODING

The Huffman coding will produce the variable length code. The algorithm which is followed will use probability and frequency of occurrence in order to determine the code associated with the particular symbol. The symbols are presented as the input to the encoder. The encoded image is presented in the form a tree. The bottom up approach is followed in this case. The Huffman coding in the proposed model uses stacks. Stack is the last in first out system. The elements will be fetched out of the memory (stacks) and then conversion operation is performed. The proposed model is less complex as compared to the existing approach.

The technique works by creating a binary tree of nodes. These can be stored in a regular array, the size of which depends on the number of symbols, n. A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the symbol itself, the weight (frequency of appearance) of the symbol and optionally, a link to a parent node which makes it easy to read the code (in reverse) starting from a leaf node. Internal nodes contain symbol weight, links to two child nodes and the optional link to a parent node. As a common convention, bit '0' represents following the left child and bit '1' represents following the right child. A finished tree has up to n leaf nodes and $n - 1$ internal nodes. A Huffman tree that omits unused symbols produces the most optimal code lengths.

The process essentially begins with the leaf nodes containing the probabilities of the symbol they represent, then a new node whose children are the 2 nodes with smallest probability is created, such that the new node's probability is equal to the sum of the children's probability. With the previous 2 nodes merged into one node (thus not considering them anymore), and with the new node being now considered, the procedure is repeated until only one node remains, the Huffman tree.

The simplest construction algorithm uses a priority queue where the node with lowest probability is given highest priority:

1. Create a leaf node for each symbol and add it to the priority queue.
2. While there is more than one node in the queue:
    1. Remove the two nodes of highest priority (lowest probability) from the queue.
    2. Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.
    3. Add the new node to the queue.
3. The remaining node is the root node and the tree is complete.

THE FLOW OF IMAGE COMPRESSION SCHEME

The flow of image compression through the Huffman Coding will be described by using the stacks. The stack is the last in first out system. The image will be inputted to the encoder. The encoded image is then plotted. The encoded image is transferred toward the destination. The image will be passed through the decoder for decoding. The decoded image is then plotted.

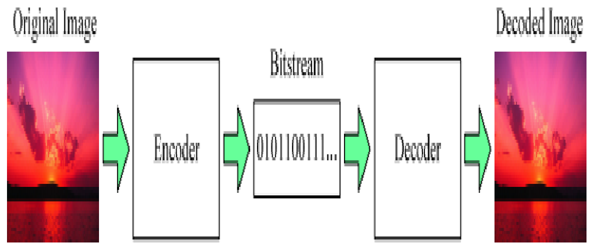

Fig. I Flow of Compression Algorithms

PROPOSED MODEL

The proposed model will be described with the help of the

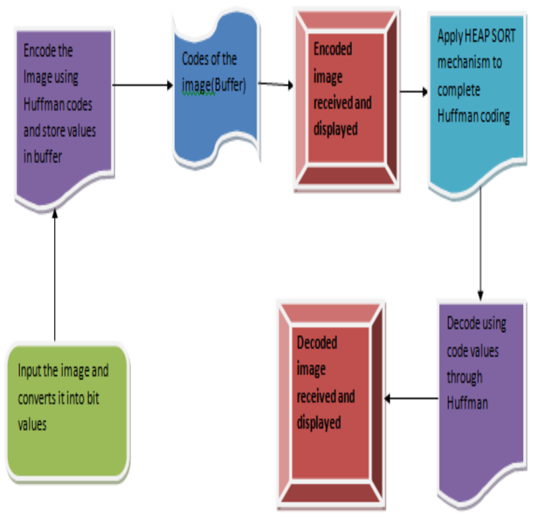
Encode the Image using Huffman codes and store values in buffer
Codes of the image(Buffer)
Encoded image received and displayed
Apply HEAP SORT mechanism to complete Huffman coding
Input the image and converts it into bit values
Decoded image received and displayed
Decode using code values through Huffman

figure as

Fig .II Model for iterative Huffman Encoding and Decoding

The image will be received and then passed through the encoder. The encoded image is received and then displayed. The heap sort mechanism is used to complete Huffman coding. The encoded image is passed through the decoder for performing decoding operation. The decoded image is then displayed using stacks. The recursive operation is used for the entire operation.

PROPOSED ALGORITHM

The proposed algorithm for performing the compression and then generating original image will be as follows

Algorithm is a sequence of steps which are expressed in such a way that desired result will be obtained. The algorithm describing recursive Huffman algorithm will be as follows

```
string HuffmanCode(const char& symbol, const
TreeNode<SymbolPriority>* huffman, string
&code)
{
  //Base case: you are in a leaf; if the leaf contains
the character
  //you are looking for then return the code
(third argument)
  //else return an empty string (which means
'wrong leaf').
  if(huffman->IsLeaf())
  {
    if(huffman->Value.Symbol == symbol)
      return code;
     else
       return "";
  }

  /*always start by going to the left and adding '0'
to the code
  (code + '0'); check the return value against the
empty string:
  if the return value is not empty then return it
without going
  to the right else return the result of going to the
right
  (do not forget to add '1' to the code).*/

  else
  {
    HuffmanCode(symbol,          huffman->Left,
code+'0');
    HuffmanCode(symbol,         huffman->Right,
code+'1');
  }
}
```

CONCLUSION AND FUTURE WORK

The proposed system is used in order to reduce the complexity of overall operation. The recursive approach will use stack and hence the speed of overall operation is low. The heap will be used rather than binary heap structure. The sorting is not required in this case. Hence large amount of data can be used in the proposed system. The bottom up approach is followed in this case. The existing system cannot implement post order traversal since stacks are not used. the proposed system still consumed more time so in the future the time consumption must be reduced using some encryption mechanism.

REFERENCES

[1] Abraham Lempel and Jacob Ziv. A universal algorithm forsequential data compression. IEEE Transactions on Information Theory, 23(3):337–343, May 1977.

[2] Abraham Lempel and Jacob Ziv. Compression of individual sequencesviavariable-ratecoding.IEEETransactionsonInformation Theory, 24(5):530–536, September 1978.

[3] Alistair Moffat and Andrew Turpin. On the implementation of minimumredundancy prefix codes.IEEE Transactions on Communications, 45(10):1200–1207, October 1997.

[4] Arun N. Netravali and Barry G. Haskell. Digital Pictures: Representation, Compression and Standards. Plenum Press, New York and London, second edition, 1995.

[5] Renato Pajarola and Jarek Rossignac. Compressed progressive meshes. IEEE Transactions on Visualization and Computer Graphics, 6(1):79–93, January-March 2000.

[6] Renato Pajarola and Jarek Rossignac. Squeeze: Fast and progressive decompression of triangle meshes. In Proceedings Computer Graphics International CGI 2000, pages 173–182. IEEE, Computer Society Press, Los Alamitos, California, 2000.

[7] Renato Pajarola and Peter Widmayer. Pattern matching in compressed raster images. In Third South American Workshop on String Processing WSP 1996, International Informatics Series 4, pages 228–242. Carleton University Press, 1996.

[8] Renato Pajarola and Peter Widmayer. Spatial indexing into compressed raster images: How to answer range queries without decompression.In Proc.Int. Workshop on Multi-Media Database Management Systems, pages 94–100. IEEE, Computer Society Press, Los Alamitos, California, 1996.

[9] Jorma Rissanen. A universal data compression system. IEEE Transactions on Information Theory, 29(5):656–664, September 1983.

[10] David Salomon. Data compression: the complete reference. Springer-Verlag, New York, 1998.

[11] Khalid Sayood. Introduction to data compression. Morgan Kaufmann Publishers, San Francisco, California, 1996.

[12] E. S. Schwartz and B. Kallick. Generating a canonical prefix encoding. Communications of the ACM, 7(3):166–169, 1964.

[13] Andrzej Sieminski. Fast decoding of the huffman codes. Information Processing Letters, 26(5):237–241, January 1988.

[14] James A. Storer, editor. Image and Text Compression. Kluwer Academic Publishers, Norwell, Massachusetts, 1992.

[15] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. ACM Transactions on Graphics, 17(2):84–115, 1998.

[16] Costa Touma andCraig Gotsman. Triangle Mesh Compression. In Proceedings Graphics Interface 98, pages 26–34, 1998.

[17] Terry A. Welch. A technique for high-performance data compression.IEEEComputer,pages8–19,June1984.

[18] IanH. Witten, Alistair Moffat, and TimothyC. Bell. Managing Gigabytes: Compressing and Indexing Documents and Images. Morgan Kaufmann Publishers, San Francisco, 1999.