International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 3, March - 2013

A Novel Approach of Lossless Image Compression using Hashing and Huffman Coding

Dr. T. Bhaskara Reddy [1] , Miss. Hema Suresh Yaragunti [2] ,Dr. S. Kiran [3] , Mrs. T. Anuradha [4]

1Associate Professor in the Dept of Computer Science & Technology , S.K.U ., Anantapur
2 Scholar in the Dept of Computer Science & Technology , S.K.U ., Anantapur
3 Assistant Professor in the Dept of Computers Applications ,Y.V. University, Kadapa.
4 Dept. of Computer Science , SPMVU, Thirupati .A.P

"***A complex idea can be conveyed in just single still image***". Storage and transmission of digital image has become more of a necessity than luxury these days, hence the importance of Image compression. Image data files commonly contain considerable amount of information that is redundant and irrelevant leading to more disk space for storage. Image compression [7][9] is minimizing the size in bytes of an image file without degrading the quality of the image to an unacceptable level. During a step called quantization, where part of compression occurs, the less frequencies [1] are discarded. This paper represent the lossless image compression on still image, which is based on Hashing and Huffman Coding technique to show the better compression.

Key words: Image compression, Hashing, Huffman Coding, Frequency Table, Encoder, Decoder, Quantizer

1. Introduction

Image compression [7][9] is technique of reducing the size, eliminating redundant or unnecessary information which is present in an image file. The compression techniques exploit inherent redundancy [1][12] and irrelevancy by transforming the image file into a smaller size, from which the original image file can be reconstructed exactly or approximately. The reduction in file size saves the memory space and allows faster transmission of images over a medium. It also reduces the time required for images to be sent over the Internet or downloaded from web pages.

1.1. Data Redundancy:

A commonly image contain redundant information i.e. because of neighboring pixels which are correlated and contain redundant information. The main objective of image compression [7][1] is redundancy[7][1] and irrelevancy reduction. It needs to represent an image by removing redundancies as much as possible, while keeping the resolution and visual quality of compressed image as close to the original image. Decompression [3][7] is the inverse processes of compression i.e. get back the original image from compressed image. Compression ratio is defined as the ratio of information units an original image and compressed image which is shown in Fig 1

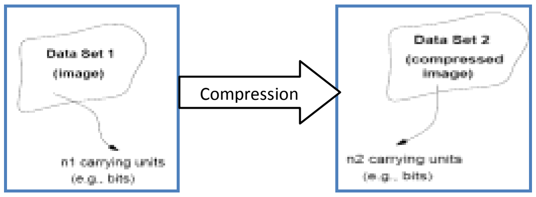
Data Set 1 (image)
n1 carrying units (e.g., bits)
Compression
Data Set 2 (compressed image)
n2 carrying units (e.g., bits)

Figure 1.1 Data Redundancy

Let n_1 and n_2 denote the number of information carrying units in two data sets that represent the same information. The relative redundancy [1] R_D is defined as: $R_D = 1 - \frac{1}{C_R}$

Where C_R commonly called the compression ratio[1], is $C_R = \frac{n_1}{n_2}$

If $n1 = n2$, $CR=1$ and $RD=0$ $\longrightarrow$ *no redundancy*

If $n1 >> n2$, $CR >>1$ and $RD >>0$ $\longrightarrow$ *high redundancy*

If $n1 << n2$, $CR<<1$ and $RD <<0$ $\longrightarrow$ *undesirable*

A compression ratio of 10 (10:1) means that the first data set has 10 information carrying units (say, bits) for every 1 unit in the second (compressed) data set. In image compression [1][7] three basic redundancies can be identified: Coding Redundancy, Interpixel Redundancy and Psychovisual Redundancy. In Coding redundancy some gray levels are more common than others. The gray levels with more frequency can be given code of smaller length to reduce the overall space. e.g. Huffman Coding [5][6]. In Inter-pixel redundancy, the value of any given pixel can be reasonably predicted from the value of its neighbors. The information carried by individual pixels is relatively small. This is also spatial redundancy, geometric redundancy or interframe redundancy e.g. Differential Coding, Run Length coding. In Psycho-visual Redundancy, as the human eye does not perceive all the details, less relative information is eliminated. The eye does not respond equally to all visual information. Information of less relative importance is psychovisually redundant. It can be eliminated without significantly impairing visual "quality". The elimination of psycho-visually redundant data results in loss of quantitative information; it is commonly referred as quantization. The Fig. 1.2 shows is an examples of simultaneous contrast. All the inner squares have the same intensity but the progressively look darker as the background becomes lighter.

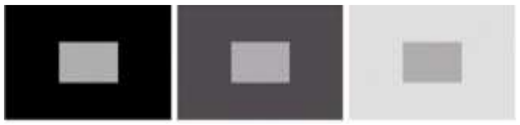

Fig 1.2: Simultaneous contrast .All the inner squares have the same intensity. But they appear progressively darker as the background becomes higher.

1.1. Image compression and Decompression.

The image compression [7][9] system is composed of two distinct functional components: encoder and decoder Fig. 1.3. Encoder performs compression while Decoder performs decompression [3][7]. Both operations can be performed in software, as in case of Web browsers and many commercial image editing programs, or in a combination of hardware and firmware, as in DVD players. A codec is a device which performs coding and decoding [9]. Input image f(x…) is fed into the encoder, which creates a compressed representation of input. It is stored for later use or transmitted for storage and use at a remote location. When the compressed image is given to decoder (Fig 1.3), is constructed output image f'(x...) is generated. In still image applications, the encoded input and decoder output are f(x,y)& f'(x,y) respectively. In video applications, they are f(x, y, t) & f'(x, y, t) where't' is time. If both functions are equal then the system is called lossless and error free, lossy otherwise.

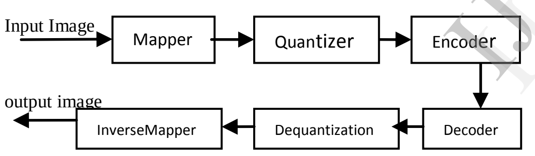

Figure 1.3 Block Diagram for Image compression

Encoding Process: Encoder is used to remove the redundancies through a series of three independent operations. *Mapper:* It transforms f(x…) into a format designed to reduce spatial and temporal redundancies. It is reversible. It may /may not reduce the amount of data to represent image. E.g. Run length coding. In video applications, mapper uses previous frames to remove temporal redundancies. *Quantizer [9]*: It keeps irrelevant information out of compressed representations. This operation is irreversible. It must be omitted whenever error free compression is desired. In video applications, bit rate of encoded output is often measured and used to adjust the operation of the quantizer [9] so that a predetermined average output is maintained. The visual quality of the output can vary from frame to frame as a function of image content. *Symbol Encoder*: Generates a fixed or variable length code to represent the quantizer [9] output and maps the output in accordance with the code. Shortest code words are assigned to the most frequently occurring quantizer output values, thus minimizing coding redundancy[1][2]. It is reversible. Upon its completion, the input image has been processed for the removal of all three redundancies. Decoder or Decoding Process [9]: Quantization [1][9] results in irreversible loss, an inverse quantizer block is not included in the decoder block.

1.3 Importance of Image Compression

The importance of image compression increases with advancing communication technology. Limited hardware and resources is also important in sending of datafield.The number of images compressed and decompressed daily is innumerable. Raw image can occupy a large amount of memory both in RAM and in storage. Compression reduces the storage space required by an Image and the bandwidth needed while streaming that image across a network. Image compression is important for web designers who want to create faster loading webpage's which in turn make website more accessible to others. In a medical field, image compression plays a key role as hospitals move towards filmless imaging and go completely digital . There are many applications which require image compression, such as multimedia, internet, satellite imaging, remote sensing etc.

1.4 Types of Compression: They are two types of image compression [7][3] algorithm: lossy and lossless

1.4.1 Lossy Compression

In this technique, size of an image is reduced, with loss of some data or information. The compressed image is similar to the original image but not identical. Lossy compression reduces file size by eliminating certain information, especially redundant information. When a file is decompressed, only part of the original information is present (but user may not notice it).Lossy compression is commonly used for photo graphs (JPEG-Joint Photographic Expert Group) and other complex still images on the web.

1.4.2 Lossless compression

In this technique, size of an image is reduced, without loss of any data or information. The compressed image is same as original image. With lossless compression, every single bit of data that was originally in the file, remains after the file is decompressed. All of the information is completely restored. Lossless compression is commonly used for GIF (Graphics Interchange Format) file.

2. Huffman Coding

It was developed by David A. Huffman while he was a PhD student at MIT. It was published in 1952 in the paper "A method for construction of minimum-Redundancy Codes"[1][2]. Huffman coding[5][12] is an entropy encoding algorithm which is used for lossless data compression to remove the redundancies. The term refers to the use of a variable length code table for encoding a source symbol (such as a character in file) where the variable length code table has been derived in a particular way based on estimated probability of occurrence for each possible value of source symbol. It is based on the frequency of occurrence of a data item i.e. pixels in an image. It yields smallest possible code symbol per source symbol.

2.1 Algorithm

The Huffman's approach is has two passes:-

a. Most frequently occurring symbols will have shorter code words than symbols that occur less frequently [1]. Create the series of reductions by ordering the probabilities.
b. The two symbols that occur least frequently will have the same length. Combine these lowest two probabilities. Sort the probabilities in descending order and repeat these steps till we get two probabilities.

2.1.1 Procedure Huffman Coding

```
//Read the pixels from an input image
Call getFrequencies (inputFile);      //Get the frequencies of each pixel
Procedure getFrequencies (inputFile)
//Collect the frequencies of symbols in the stream that we want to compress.
Call CanonicalCode (code, 257);    //Build into the tree with node and leaf
Call canonCode.toCodeTree ();      // Replace code tree with canonical one. For each symbol, the code value may change but the code length stays the same.
Call AdaptiveHuffmanCompress (); //compress the file using HCT
Call HuffmanEncoder ();       //To encode the data
Call CodeTree ()              //To reorder the data after adding two lowest frequencies
Call HuffmanDecoder();        //Call the decoder to get back
```

2.1.2 Algorithm for Huffman Coding

Frequency table // A table of symbol frequencies.

- Collect the frequencies of symbols in the stream that we want to compress.
    - Build a code tree that is statically optimal for the current frequencies.
    - This implementation correctly builds an optimal code tree for any legal number of symbols (2 to Integer.MAX_VALUE), with each symbol having a legal frequency (0 to Integer.MAX_VALUE). It is designed not to give error due to overflow.

CodeTree **//** A binary tree where each leaf codes a symbol, for representing Huffman codes

The main uses of a CodeTree:

.Read the 'root' field and walk through the tree to extract the desired information.

.Call getCode () to get the code for a symbol, provided that the symbol has a code. The path to a leaf node determines the leaf's symbol's code. Starting from the root, going to the left child represents a 0, and going to the right child represents a 1.

.Constraints: The tree must be complete, i.e. every leaf must have a symbol. No symbol occurs in two leaves. But not every symbol needs to be in the tree. The root must not be leaf node.

Huffman Code	Code tree:
0: Symbol A	`.`
10: Symbol B	`/ \`
110: Symbol C	`A   .`
111: Symbol D	`   / \`
	`  B   .`
	`     / \`
	`    C   D`

Table.2.1 Code Tree

Canonical Huffman Code

Code length 0 means no code. A canonical Huffman code only describes the code length of each symbol. The codes can be reconstructed from this information. In this implementation, symbols with lower code lengths, breaking ties by lower symbols, are assigned lexicographically lower codes as shown in Table 2.2.

Code lengths canonical code):	Huffman codes (generated from canonical code):
Symbol A: 1	Symbol A: 0
Symbol B: 3	Symbol B: 110
Symbol C: 0 (no code)	Symbol C: None
Symbol D: 2	Symbol D: 10
Symbol E: 3	Symbol E: 111

Table 2.2 Canonical Huffman Code

Huffman Encoder: The code tree can be changed after each symbol is encoded, as long as the encoder and decoder have the same code tree at the same time.

Huffman Compressor: Uses static Huffman coding[5][6] to compress an input file to an output file. Use Huffman Decompresser to decompress. Uses 257 symbols - 256 for byte values and 1 for EOF. The compressed file format contains the code length of each symbol under a canonical code, followed by the Huffman-coded data.

Huffman Decoder: The code tree can be changed after each symbol is decoded, as long as the encoder and decoder have the same code tree at the same time.

2.2PROCEDURE HUFFMANCODE GENERATION

```
//Read the pixels from an input image
Call getFrequencies (inputFile);
 //Get the frequencies of each pixel
Call CodeTree () //To reorder the data after adding two
//lowest frequencies
Call canonCode.toCodeTree (); // Replace code tree
with canonical one. For each symbol, the code value
may change but the code length stays the same.
Call AdaptiveHuffmanCompress ();
//compress the file using HCT
Call HuffmanEncoder ();   //To encode the data
Call HuffmanDecoder ();  //Call the decoder to get back
```

2.2.1 PROCEDURE NODE ( )

```
// A node in a code tree. This class has two & only two
subclasses: InternalNode, Leaf.
public abstract class Node
{
Node () {}//Package-private to prevent accidental
subclassing outside of this package
}
```

2.2.2 PROCEDURE LEAF()

```
// A leaf node in a code tree. It has a symbol value.
public final class Leaf extends Node
 {    public Leaf(int symbol)
  {
        if (symbol < 0)
                //Is a illegal symbol
        else
                //add the symbol to leaf  Assign
        symbol to the current Leaf
   } }
```

2.2.3 PROCEDURE FREQUENCYTABLE ( )

```
public FrequencyTable (int [] freqs)
{
   if (freqs == null)
   Not possible to create table
  if (freqs.length < 2)
   // Not possible to create a table it should contain
      at least 2 symbols.);
   Else
 frequencies = freqs. Clone ();   // Defensive copy
     for (int x: frequencies)
      {
          if (x < 0)
           Is a Negative frequency
          }
      }
public void increment (int symbol)
{
  if (symbol < 0 || symbol >= frequencies. Length)
          Then Symbol out of range
  If (frequencies [symbol] == Integer.MAX_VALUE)
          Arithmetic overflow
          else
          frequencies [symbol]++;
}
```

2.2.4 PROCEDURE CODETREE ( )

```
// Pad with zero-frequency symbols until queue has at
least 2 items
for (int i= 0; i < frequencies. Length && pqueue.size () < 2; i++)
{   if (i >= frequencies. length || frequencies[i] == 0)
          //add a node
pqueue.add (new Node Frequency (new Leaf (i), i, 0));
}
// Repeatedly tie together two nodes with the lowest
frequency
while (pqueue.size () > 1)
 {
     //remove the last two symbols from queue
    and store sum of that symbols into the pquee
    and store sum of that two symbols into the pqueue
//Remove nf1
 pqueue.remove ();
// Remove nf2
 pqueue.remove ();
//now add another two symbols
pqueue.add(new NodeWithFrequency( new Internal
Node(nf1.node, nf2.node),
//find the minimum two frequencies and find sum
Math.min(nf1.lowestSymbol,nf2.lowestSymbol),
nf1.frequency + nf2.frequency));
}
// Return the remaining node and store into code tree
Return new CodeTree((Internal Node) pqueue.remove
().node, frequencies. length);
private void buildCodeList (Node node, List<Integer>
prefix)
{
    if (node instanceofInternalNode)
      {
        //get the internal node
      InternalNodeinternalNode=(InternalNode)  node;
      //add prefix 0 to the leftchild node
      prefix. add (0);
      buildCodeList (internalNode.leftChild, prefix);
prefix.remove (prefix.size () - 1);
//add prefix 1 to the rightchild node
  prefix.add (1);
  buildCodeList(internalNode.rightChild, prefix);
  prefix.remove(prefix.size() - 1);
              }
}
```

2.2.5 PROCEDURE HUFFMANENCODER ( )

```
public final class HuffmanEncoder
{
private BitOutputStream output;
// The code tree can be changed after each symbol
encoded, as long as the encoder and decoder have the
same code tree at //the same time.
public HuffmanEncoder (BitOutputStream out) {
if (out == null)
Not possible to encode
else
//Assign encode value
output = out;
List<Integer> bits = codeTree.getCode (symbol);
for (int b: bits)
output. write (b);
}
public CodeTree toCodeTree ()
 {
//create the ArrayList of nodes
 List<Node> nodes = new ArrayList<Node>();
//add the leaves for    symbols in but symbols should
be in descending order
//move from max length code to min
for (int i = max (codeLengths); i >= 1; i--) {  // Descend
through positive code lengths
// Add leaves for symbols with code length i
for (int j = 0; j < codeLengths.length; j++) {
if (codeLengths[j] == i)
//add the node i.e. leaf
newNodes.add(new Leaf (j));
}
// Merge nodes from the previous deeper layer
for (int j = 0; j < nodes.size(); j += 2)
newNodes.add(new          InternalNode(nodes.get(j),
nodes.get(j + 1)));
nodes = newNodes;
//This canonical code does not represent a Huffman
code tree
}if (nodes.size() != 2)
//This canonical code does not represent a Huffman
code tree
if (nodes.size() % 2 != 0)
else
return new CodeTree (new InternalNode (nodes.get
(0), nodes.get(1)), codeLengths.length);
}
if (nodes.size() % 2 != 0)
//This canonical code does not represent a Huffman
code tree
}
if (nodes.size() != 2)
//This canonical code does not represent a Huffman
code tree
else
return new CodeTree (new InternalNode (nodes.get
(0), nodes.get(1)), codeLengths.length);
}
return new CodeTree (new InternalNode (nodes.get (0),
nodes.get(1)), codeLengths.length);
}
```

3. Hashing

Hashing [10] is the technique used for performing almost constant time search in case of insertion, deletion and search operation. Hashing[10][10] is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value. It is also used in many encryption algorithms. Hashing [10] is the process of running data through a hash function [14]. A hash function is a mapping between a set of input values and a set of integers, known as hash values. Hash Table [11] and Hash Map [11] are the data structures which uses hash function to generate key corresponding to the associated value.

```
PROCEDURE HASHING()
//read the input file converted into an array of pixels.
//store the array elements and repeated positions using HashTable
//Data Items
 hm        //hash map to hold pixels
img        //input image
int w      //width of the image
int h      //height of the image
 comparator // method of Collections class used to sort
Read the image and get its height, width
Int [][] data = new int[h][w];  //store the pixels into a matrix of w×h
 data[y][x] = raster.getSample(x, y, 0);
Convert the matrix into arraylist
myList.add (new Integer (data[x][y]));
//  put these into Hash Map
     Increment j till it reaches array list size
     Add the values into hm
//    to sort an ArrayList using comparator use,
       Sort the in reverseorder
```

In this,we have consider a image and converted the pixels into an Array ,Store these pixels using Hashing technique[10][1] ,sort the pixels and applied the Huffman coding[5][6] to get the better lossless compression. Fig 2 is explains it.

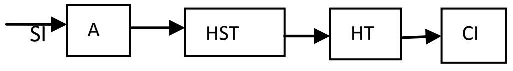

Figure 2:Block diagram of the proposed method

SI-Source Image
A-Pixel Array
HST-Hashing Technique
HT-Huffman Coding Technique
CI-Compressed Image

Huffman coding[5][6][12] is designed by merging the two lowest symbols which are in Frequency Table and this process is repeated until two probabilities are left and thus Code Tree is generated and Huffman codes are obtained from labeling of the code tree. The codes can be reconstructed by Canonical Huffman code. This is illustrated with example as shown below Table 2.8. Consider the sample image of size 6×6 matrix Table2.3, place these values into HashMap[11] using technique . And sort this matrix Table 2.4 ,find the number of occurrences of each pixel and create a frequency table in descending order.

Table2.3
Sample matrix 6x6

78	7E	78	80	B7	71
B1	71	B2	B1	44	57
B1	7E	80	80	7E	80
78	80	B7	B7	71	71
44	80	78	B2	7E	57
B7	44	B1	57	44	71

Table.2.4
Sorted matrix

44	44	44	44	57	57
57	71	71	71	71	71
78	78	78	78	80	80
80	80	80	80	7E	7E
7E	7E	B1	B1	B1	B1
B2	B2	B7	B7	B7	B7

Table 2.5
Frequency Table

S	F	P
80	6	0.166
71	5	0.14
B1	4	0.11
7E	4	0.11
78	4	0.11
B7	4	0.11
44	4	0.11
57	3	0.08
B2	2	0.056
Total	36	

Now reduce the source Table by combining the two lowest probabilities i.e. 0.06 and 0.08 ,place the result 0.14 in a table and arrange the probabilities in descending order, now combine the 0.11 and 0.11 to get 0.22 and so on . Repeat this until we reach two values i.e. 0.55 and 0.44.

S	F	P	1	2	3	4	5	6	7
80	6	0.16	0.16	**0.22**	0.22	**0.25**	**0.30**	**0.44**	**0.55**
71	5	0.14	0.14	0.16	**0.22**	0.22	0.25	0.30	0.44
B1	4	0.11	**0.14**	0.14	0.16	0.22	0.22	0.25	
7E	4	0.11	0.11	0.14	0.14	0.16	0.22		
78	4	0.11	0.11	0.11	0.14	0.14			
B7	4	0.11	0.11	0.11	0.11				
44	4	0.11	0.11	0.11					
57	3	0.08	0.11						
B2	2	0.06							

Table 2.6 Huffman Source Reduction

This table2.7 is showing all Huffman codes for all input sources.

S	80	71	B1	7E	78	B7	44	57	B2
K	000	001	011	110	111	100	101	0100	0101

Table 2.7

After reducing the sources, generate the Huffman coding for sources as shown in table2.6

S	F	P	Key														
80	6	0.16	000	0.16	000	**0.22**	10	0.22	10	**0.25**	01	**0.30**	00	**0.44**	1	**0.55**	0
71	5	0.14	001	0.14	001	0.16	000	**0.22**	11	0.22	10	0.25	01	*0.30*	00	0.44	1
B1	4	0.11	011	**0.14**	010	0.14	001	0.16	000	0.22	11	*0.22*	10	*0.25*	01		
7E	4	0.11	110	0.11	011	0.14	010	0.14	001	*0.16*	000	*0.22*	11				
78	4	0.11	111	0.11	110	0.11	011	*0.14*	010	*0.14*	001						
B7	4	0.11	100	0.11	111	*0.11*	110	*0.11*	011								
44	4	0.11	101	*0.11*	100	*0.11*	111										
57	3	*0.08*	0100	*0.11*	101												
B2	2	*0.06*	0101														

Table2.8 Huffman Code Generation

4.Conclusion:

Compression the images efficiently is one of the major problem in image applications. So we have tested the efficiency of image compression using Hash table and Huffman code technique. The Lossless algorithm is applied for image. This work may be extended the better compression rate than other compression Techniques. The performance of the proposed compression technique using hashing and human coding is performed on GIF, TIFF formats. We took only medical images where HST and HT are better. This technique can be applied on luminance and chrominance of color images for getting better compression.

5.Refrences:

[1] Dr.T.Bhaskar Reddy, S.Mahaboob Basha, Dr.B.Sathyanarayana and "Image Compression Using Binary Plane Technique" Library Progress, vol1.27, no: 1, June-2007, pg.no:59-64.

[2] D.A.Huffman, A method for the construction of Minimum-redundancy codes, Proc.IRE, vol.40, no.10, pp.1098-1101, 1952

[3] Jagadish H. Pujar, Lohit M. Kadlaskar (2010) "A new lossless method of image compression and decompression using Huffman coding techniques" Journal of Theoretical and Applied Information Technology

[4] C. Saravanan, R. Ponalagusamy (2010) "Lossless Grey-scale Image Compression using Source Symbols Reduction and Huffman Coding" International Journal of Image Processing (IJIP), Volume (3): Issue (5)

[5] Mamata Sharma,"Compression Using Huffman Coding",IJCSNS International Journal of Computer Science and Network Security ,VOL.10 No.5,May 2010,pp 133-141

[6] Sunil BhooshanShipra Sharma "An Efficient and Selective Image Compression Scheme using Huffman and Adaptive Interpolation" 2009 IEEE

[7] Rafael C.Gonzalez,Richard E.Woods "Digital Image Processing "Second edition Pearson Education,Printice Hall

[8] G.C Chang Y.D Lin (2010) "An Efficient Lossless ECG Compression Method Using Delta Coding and Optimal Selective Huffman Coding" IFMBE proceedings 2010, Volume 31, Part 6, 1327-1330, DOI: 10.1007/978-3-642-14515-5_338.

[9] The Scientist and Engineer's Guide to Digital Signal Processing by Steven W. Smith.Ph.D

[10] R. Pagh and F. F. Rodler, Cuckoo Hashing, in 9th Annual European Symposium on Algorithms, v.2161 of Lecture Notes in Computer Science, pp. 121-133, Springer-Verlag, 2001.

[11] Ulfar erlingsson,Mark Manasse,Frank Mcsherry "A cool and Practical Alternative to Traditional Hash Tables"

[12] http://www.Huffman coding-Wikipedia,the free encyclopedia.htm
http:/en.wikipedia.org/wiki/Huffman coding
http://en.wikipedia.org/wiki/Adaptive Huffman coding

[13] Hash table-Wikipedia,the free encyclopedia http:/en.wikipedia.org/wiki/Hash_table.htm

[14]Hash Functions:www.cs.hmc.edu/~geoff/classeshmc.cs070.200101/…/hashfuncs.htm

AUTHORS PROFILE

Dr.T.Bhaskara Reddy is an Associate Professor in the department of Computer Science and Technology at S.K University,Anantapur A.P. He holds the post of Deputy Director of Distance education at S.K.University and He also the CSE Co-coordinator of Engineering at S.K.University. He has completed his M.Sc and Ph.D in computer science from S.K.University. He has acquired M.Tech from Nagarjuna University. He has been continuously imparting his knowledge to several students from the last 17 years. He has published 47 National and International publications. He has completed major research project (UGC). Four Ph.D and Three M.Phil have been awarded under his guidance. His research interest are in the field of image Processing, computer networks, data mining and data ware house. E-Mail:bhaskarreddy _sku@yahoo.co.in

Miss. Hema Suresh Yaraguntiis research scholar in the department of Computer Science Technology at S.K.University. She acquired M.Sc in Computer Science from Karnataka University Dharwad. She has 5 years of experience in teaching and 2 years of experience in software field. Her research interest is in the field of Image Processing.
E-Mail:hema.asadianil@gmail.com

Dr.S.Kiran is an Assistant Professor in the department of Computer Science and Technology at Yogivenama University , Kadapa, A.P. He has completed his M.Sc and Ph.D in computer science from S.K.University. He has acquired M.Tech from Nagarjuna University. He has been continuously imparting his knowledge to several students from the last 5 years. He has published 4 National and International publications.. His research interests are in the field of image Processing, computer networks, data mining and data ware house.
E-Mail:kirans123@gmail.com