A Linear Representation of Huffman Tree

Ishita Goel, Suneeta Agarwal, Rajesh Prasad

Abstract— **In this paper, we present one efficient simple string representation for Huffman code tree while applying Huffman compression algorithm. This space efficient representation can be used at the time of coding as well as decoding. Instead of frequency table it can be attached with the coded file which saves the space and hence transmission time. We also present algorithms for coding and decoding directly from this string representation of Huffman tree.**

Keywords—**Huffman tree, pre traversal of a tree, coding, decoding, strictly binary tree.**

I. Introduction

Huffman coding is known to be one of the most leading methods in various applications related to the text compression[2]. It is a semi static compression technique. In the first phase it calculates the frequencies of the symbols of the text. Then based on these frequencies a binary tree called Huffman tree is constructed. Codes for different symbols are generated from this tree. When a text has been coded by Huffman algorithm then later to decode it, one again needs either the frequency table or Huffman tree .Each of these requires sufficient space. We have proposed a new representation of this Huffman tree in a linear form which takes very little space to store and with which coding and decoding both can be done very conveniently. More over this representation can be taken as a secret KEY also for security purpose of the text. Memory required to store this Key is 2n-1 bytes only, where n is the number of different symbols used in the text file to code. No other information is required to decode the file. In section two of this paper we explain the formation of linear representation of Huffman tree. In section three we present an algorithm for generating codes of various symbols using this key. In Section 4 we present an algorithm to decode the coded text file directly from this key. In section 5 we write another algorithm for decoding which is faster than the earlier decoding algorithm, but takes little extra space. Section 6 concludes the work done.

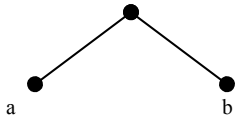
Ishita Goel is with Department of Computer Science & Engineering, United Institute of Technology, Allahabad, INDIA (Email: i.goel13@gmail.com)

Suneeta Agarwal is with Department of Computer Science & Engineering, Motilal Nehru National Institute of Technology Allahabad-211004, INDIA (Email: suneeta@mnnit.ac.in)

Rajesh Prasad is with Department of Computer Science & Engineering, Motilal Nehru National Institute of Technology Allahabad-211004, INDIA (Email: rajesh_ucer@yahoo.com)

II. KEY: A Linear Representation of Huffman tree

To code a text file by Huffman coding, a table of symbols arranged in ascending order of frequencies is used [1,3,5,8]. At any step of this coding scheme, two symbols of least frequencies say a and b are extracted from the table and combined into one (new variable say) making it a root with a and b as left and right children respectively .New variable is assigned frequency as the sum of the frequencies of variables a and b. is inserted in the frequency table at the appropriate place. This is continued till there is only one variable left. A partial Huffman tree and its Preorder traversal is shown in fig. 1.



Preorder traversal * a b
Here * is used as a label for the root of this tree instead of .

Figure 1. Partial Huffman Tree

Similarly, preorder traversal of partial Huffman tree shown in fig. 2 can completely be written as: * * * * e g * h c * f d * b a where a star is used for each internal node.

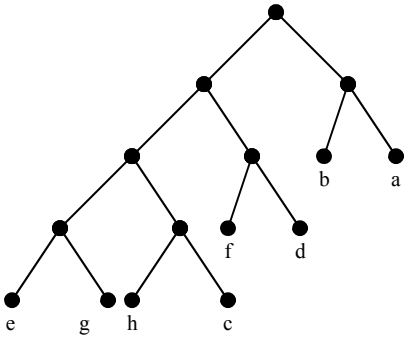


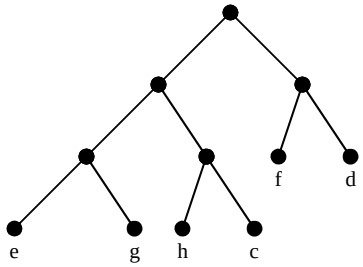
Figure 2. Huffman Tree

This preorder traversal is named "KEY" of the tree. Although Huffman tree is constructed in steps there is no need to traverse the Huffman tree again and again. At any stage preorder traversal of the new tree (Key) can be obtained by writing a star in the beginning followed by the two traversals of left and right sub trees one after the other.

It is easy to check that Huffman tree can be reconstructed from this traversal alone.

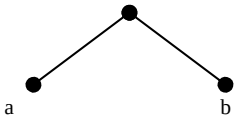
ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Consider the following two sub-trees shown in fig. 3 and fig. 4



Key: * * * eg * hc * fd

Figure 3. Sub-tree 1



Key: * b a

Figure 4. Sub-tree 2

When these two sub trees are combined the new key would be:
* * * * e g * h c * f d * b a

Thus we can attach this string "KEY" (instead of Frequency table or Huffman tree) along with the coded file .A receiver can decode the coded file with this KEY alone. Size of this string "KEY" is 2n-1 where n is number of different symbols used in the file [4,7].

Alternate Representation of KEY

Before sending this KEY we need to ensure that * is not present as a symbol any where in the text file. So an alternate better approach could be used:
KEY: * * * * e g * h c * f d * b a
this key can also be written as:
eeeeeghhcff dbb a
Here each star of key is replaced by the very first text symbol present to the right.

III. Algorithm for Generating Codes from the KEY

Codes of all the text symbols can directly be obtained from the KEY as follows:

1. Initialize the code as null.
2. Read the first symbol of the KEY. Append a zero to the code and move to the next symbol of KEY.
3. Compare the present symbol of key with its preceding symbol
4. if (equal)
5. then append a zero to the present code and move to the next symbol of key.
6. go to step 4.
7. if (different or End of Key)
8. then From cut a zero from right end of the present code and assign the resulting code to the preceding symbol. Increment the value of code by one. Delete all succeeding zeros (if any). Add a single zero at the right end. Move to the next symbol in the key .If End of Key cut a zero from the right end and assign to the last Symbol stop. Otherwise go to step 4.

Thus traversing the key: eeeeeghhcffdbba once, codes of all the text symbols are generated as
e: 0000, g: 0001, h: 0010, c: 0011, f: 010, d:011, b: 10, a: 11

IV. Algorithm for Decoding using the KEY directly (without expanding the KEY)

This algorithm is suitable if there is no extra space available as in hand held devices or small number of symbols in the original file or we need to decode only a small portion of the coded file**).**
Note: From the construction of the key it is known that a key symbol corresponds to a text symbol if it is different from its succeeding the key or it is the last symbol of key.

1. Initialize a variable 'count'= 0. Start from the first symbol of the KEY
2. Read the next symbol of the code and repeat line 3-14
3. if (Symbol ="0")
4. then skip the present symbol of the key.
5. if (the present symbol is a real symbol)
6. then stop, this corresponds to the code So write the symbol in the decoded file and go to the beginning of the KEY put 'count' =0. go to the beginning of step 5
7. if (not a real symbol**)**
8. then go to step 5.
9. if (Symbol = "1")
10. then repeat
11. if (the symbol of the KEY is **not a real symbol**)
12. then increment the variable 'count' by one, move to the next symbol of KEY and also go to step 10
13. else decrement the variable 'count' by one, go to the next symbol of the KEY.
14. until 'count' becomes zero or KEY ends
15. if (count = 0)
16. then check the present symbol of the KEY
17. if(**"EOK" or "Real symbol")**
18. then stop, present symbol is the one corresponding to the code read so far. So write it in the decoded file and go to the beginning of the KEY, initialize 'count' = 0. goto step 2.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

19. else goto step 2.

V. Algorithm for faster Decoding

Here the KEY is first converted into an array 'Key Array' (an expanded form of Huffman tree)

Algorithm for generating array

Use a variable index and initialize it to zero. Assign this value as index to each text symbol. Read the first symbol of the key.

1. Update the index of the first symbol of the key as 1
2. Compare the present symbol with its successor in the KEY (without moving)
3. if (**"same")**
4. then push the' index' in a stack, double the 'index' and move to the next symbol in the KEY, update the value of index to it only if it is higher than its previous index. If it is not end of key, go to step 2 otherwise exit.
5. else pop from the stack say let it be "i" set index=2i +1, move to the next symbol of the KEY, and assign this index to the symbol only if this value is higher than its previous index. If it is not end of key, go to step 2 otherwise exit.

Size of the Key array would be the highest index value. Each text symbol will be placed in this array at its index value. All other cells of this array would be marked as null.

Thus we have an array "Key Array" having cells **either** blank or a symbol of the original text file.

Algorithm for Decoding the File from the Key array

1. Start with an index 'i'= 1
2. Read the new symbol of the coded file, if End of file stops
3. if (Symbol read is "0")
4. then $i \leftarrow 2\times i$
5. **case 1**: A[i]=blank { go to step 2}
6. **case 2**: A[i]= a symbol {write it in the decoded file and go to step 1}
7. if (Symbol read is "1")
8. then $i \leftarrow 2\times i+1$
9. **case 3:** A[i]= a symbol.{write the symbol in the decoded file and go to step 1}
10. **case 4:** A[i] = blank {go to step 2}

VI. Conclusions

The main contribution of this paper is a linear representation of Huffman tree using which coding and decoding both are possible through this key. Moreover there is no need to construct the Huffman tree separately, as the Key can be determined directly while combining the keys of its components.

References

[1] Reza Hashermain, "Direct Huffman Coding and Decoding using the table of code lengths", Procedings of international conference on Tnformation Technology: computers and communications(ITCC'03). pp 237-241.

[2] S. Roman, "Coding and Information Theory", New York: Springer Verlag, 1992.

[3] R. Hashemain, "Memory Efficient and High Speed Search Huffman Coding" IEEE Transactions on Communications,Vol, No.10, October 1995, pp. 2576-2581.

[4] R. Hashemain, "Reduced Code Transmission and High Speed Reconstruction of Huffman Tables" Communications, Computer and Signal Processing, Vol 1, 26- 28 Aug, 2001, pp. 180-183.

[5] K. L. Chung, "Efficient Huffman Decoding," Information Processing Letters 61, pp 97-99,1997.

[6] H. C. Chen, Y. L, Wang, and Y. F. Lan, "A memory Efficient and Fast Huffman Decoding Algorithm," Information Processing Letters, 1999. pp. 119-122.

[7] R. Hashemain, "Condensed Table of Huffman coding, A New Approach to EfficientDecoding," IEEE Transactions Communications, Vol 52, no. 1, Jan.2004, pp. .6-8.

[8] M. Aggarwal and A. Narayan, "Efficient Huffman Decoding," Proc International Conference on Image Processing, 2000, pp936-939.

[9] P. C. Wang, Y. R. Yang, C. L. Lee and H. Y. Chang, "A Memory Efficient Huffman Decoding Algorithm, "Advanced Information Networking and Applications, Vol 2, 28-30, March 2005, pp. 475-479.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)