An Efficient Decoding Technique for Huffman Codes

Rezaul Alam Chowdhury and M. Kaykobad
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka-1000, Bangladesh, email: shaikat,audwit@bdonline.com
Irwin King
Department of Computer Science and Engineering
The Chinese University of Hong Kong, email: king@cse.cuhk.edu.hk

Abstract

We present a new data structure for Huffman coding in which in addition to sending symbols in order of their appearance in the Huffman tree one needs to send codes of all circular leaf nodes (nodes with two adjacent external nodes), the number of which is always bounded above by half the number of symbols. We decode the text by using the memory efficient data structure proposed by Chen et al (IPL vol. 69 (1999) pp. 119-122).

1. Introduction

Since the discovery of Huffman encoding scheme [4] in 1952 Huffman codes have been widely used in efficient storing of data, image and video [2]. Huffman coding has been subjected to numerous investigations in the past 50 years. Schack [7] considered distribution of the length of a typical Huffman codeword, while Katona and Nemetz [5] studied connection between the self-information of a source letter and its codeword length in a Huffman code. Traditional Huffman scheme collects frequency of occurrence of different symbols to be coded in the first pass, constructs a tree based upon which symbols receive codes. Then in the second pass symbols are coded and sent to the receiver. However, along with codes corresponding to symbols Huffman tree is also sent. There is a single pass version of Huffman coding called *dynamic Huffman coding* [8]. In the later case symbols are coded according to the frequencies of symbols so far sent, and the tree is updated by both sender and receiver using the same algorithm. In this paper we consider efficient decoding of the so-called *static Huffman code*.

Hashemian [3] presented an algorithm to speed up the search process for a symbol in a Huffman tree and reduce the memory size. He proposed a tree-clustering algorithm to avoid sparsity of Huffman trees. Finding the optimal solution of this clustering problem is still open. Later, Chung [2] gave a data structure requiring memory size of only $2n$ - 3 to represent the Huffman tree, where n is the number of symbols. Chen *et al.* [1] improved the data structure further to reduce memory requirement to $\lceil 3n/2 \rceil + \lceil (n/2)\log n \rceil + 1$. In our scheme in addition to sending information on symbols we send information on all *circular leaf nodes* (nodes with two adjacent external nodes), that uniquely determines a Huffman tree. Our memory requirement is better than any existing data structure since number of circular nodes is bounded above by $\lceil n/2 \rceil$. So in the worst case our memory requirement is $\lceil 3n/2 \rceil$. In fact, the number of circular leaf nodes of a Huffman tree is 1 more than the number of nodes with 2 internal son nodes. Such a saving in the representation of Huffman trees is very important in the context of repeated Huffman coding [6], where ultimate compression ratio depends upon the efficiency of representation of the Huffman tree. In our case decoding of codes is based upon Chen's data structure for storing the Huffman tree. While Chen *et al.* [1] addressed the problem of efficient decoding of a Huffman code, we address the problem of decoding streams of bits not knowing the

length of bit patterns that correspond to a symbol. For clarity of presentation we first describe the idea of Chen *et al.* [1] and their data structure.

Let T be a Huffman tree, which contains n symbols. The symbols correspond to leaves of T and are labeled from left to right as $s_0, s_1, \ldots, s_{n-1}$. The root is said to be at level 0. The level of any other node is 1 more than the level of its father. The largest level is the *height* h of the Huffman tree. The *weight* of a symbol is defined to be 2^{h-l}, where h is the height of the tree and l is the level of a node corresponding to a symbol. Let w_i be the weight of symbol s_i, $i = 0, \ldots, n - 1$. Let, f_i denote cumulative weight up to weight w_i. Then $f_0 = w_0$, $f_i = f_{i-1} + w_i$, for $i = 1, 2, \ldots, n - 1$.

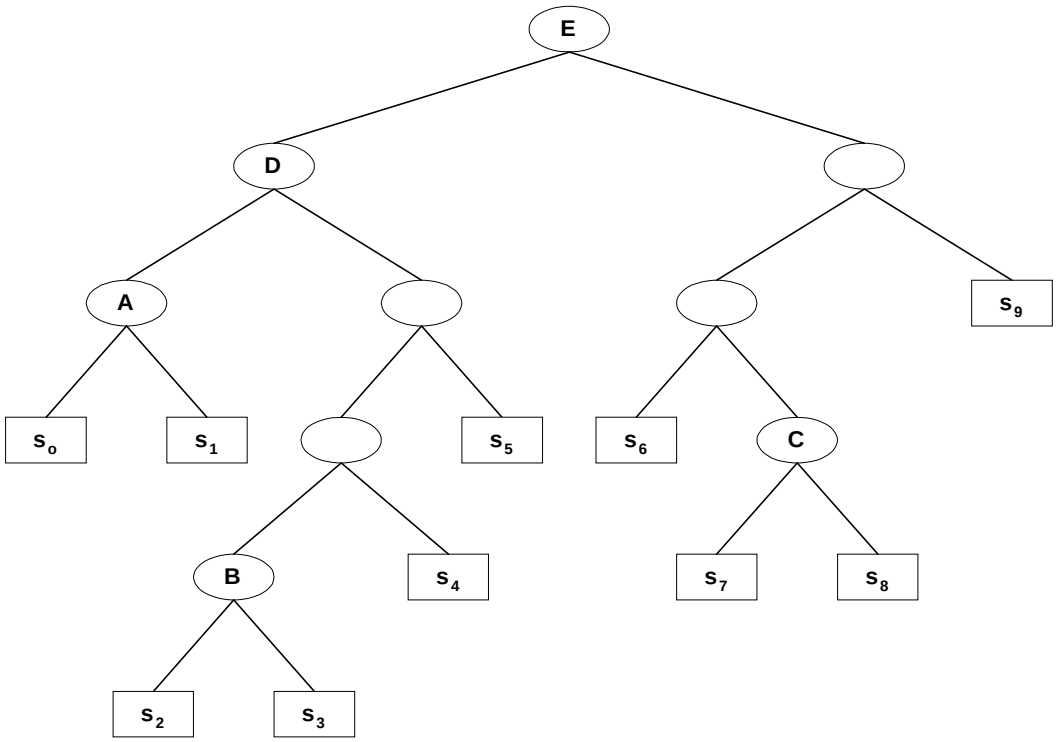

Fig. 1. An example of a Huffman tree

Table 1. The values of w_i *and* f_i

i	0	1	2	3	4	5	6	7	8	9
w_i	4	4	1	1	2	4	4	2	2	8
f_i	4	8	9	10	12	16	20	22	24	32

While Chen *et al.* [1] give an algorithm for decoding a codeword, and give the correct symbol as output or say that the given codeword does not correspond to any symbol, we address the problem of decoding the bit streams sent by the sender according to the Huffman tree that is available with the receiver in the data structure mentioned above. The above data structure requires an overhead of sending n symbols and n integers corresponding to them. This overhead can also be further reduced by the following.

If T is a Huffman tree of height h, then T', the *truncated* Huffman tree, is obtained by removing all leaves (square nodes) and will have height h - 1. T' uniquely determines the

Huffman tree T since symbols correspond to left and right son nodes of the circular leaf nodes and the other sons of other single-son circular nodes. Then Huffman codes, representing circular leaf nodes of T' uniquely determines T. If there are m circular leaf nodes, m codewords corresponding to them, and n symbols in order of appearance in the tree will suffice to represent the Huffman tree.

Consider the example in Figure 1. Let the leaf circular nodes of Figure 1 be denoted by A, B, and C respectively in order. Then the corresponding Huffman codes are 00, 0100 and 101. However, each of these codewords can be represented uniquely by integers whose binary representations are these codewords with 1 as prefix. The corresponding integers for A, B and C are respectively 4 for 100, 20 for 10100 and 13 for 1101. Along with s_0, s_1, …, s_9 if 4, 20 and 13 are sent as codes for the circular leaf nodes A, B and C, then the receiver can obtain Huffman codes 00, 0100 and 101 by deleting the attached prefix. These bit patterns will enable the receiver to construct all paths to the circular leaf nodes, and thus reconstruct the Huffman tree. So to reconstruct Huffman tree T, we have the following theorem.

Theorem: Let *S* be the array of symbols in order of appearance in the Huffman tree *T* and *D* be the set of integer codes, corresponding to circular leaf nodes, sent to the receiver. Then the receiver can uniquely reconstruct the Huffman tree *T*.

Proof: Let d_i be the integer corresponding to a circular leaf node i. By deleting the appropriate prefix from the binary representation of d_i, one can reconstruct the whole path leading to that node uniquely since bit 0 will take to the left and 1 to the right, and there is no ambiguity. Since each internal node of the Huffman tree will appear on the paths of some of these circular leaf nodes, each internal node of the Huffman tree will appear in at least one of these paths. Now external nodes will appear either as left or right sons of the circular leaf nodes, or as one of the sons of circular nodes on the path to the circular leaf nodes. This will result in the construction of a unique Huffman tree T since paths constructed for each d_i are unique. Furthermore, the set of symbols S will label the external nodes in an orderly fashion with appropriate symbols. ***QED***

Once the tree can be constructed all the values of Table 1 can be calculated, and decoding can be successfully accomplished by using binary search algorithm in the array of cumulative weights.

This gives us the possibility of reducing the overhead further since the number of circular leaf nodes cannot be more than half of the number of symbols. The number m of circular leaf nodes of any truncated Huffman tree is 1 more than the number nodes with 2 internal son nodes. Huffman codes are much more efficient than block codes when corresponding Huffman trees are sparse. In a sparse truncated Huffman tree the number of nodes with 2 internal son nodes can be much less than the total number of internal nodes, which in turn is 1 less than the number of external nodes corresponding to symbols. In the truncated Huffman tree, corresponding to Fig. 1, there are two nodes with 2 internal son nodes, namely nodes labeled D and E. So in representing the tree we need $O(n\log n)$ bits for the symbols and $O(h)$ bits for each circular leaf node, number of which is ($\leq \frac{1}{2} n$) bounded above by $O(n)$.

2. A Fast Decoding Technique

Chen *et al*. [1] give weight to a leaf v equal to the number of leaves in a complete binary tree under the node v. Every leaf node is assigned a number equal to cumulative weights of all the

leaves appearing before it and including itself. So this gives us an opportunity to apply binary search on these cumulative weights to obtain the symbol, bit pattern corresponding to which has been extracted from the bit stream. At any stage our algorithm takes h bits from the stream. Since a prefix of this string must contain a codeword we will always be able to decode that codeword using the following algorithm. In the algorithm we assume that input bit stream does not contain any error.

Algorithm Decode

Input: The values f_i, $i = 0, \ldots, n - 1$, of a Huffman tree T with height h and a bit stream b_j, $j = 1, \ldots, N$.
Output: The text symbols c_k, $k = 1, \ldots, M$, corresponding to the input bit stream.

// j is the pointer to last bit already decoded, k is the pointer to the last decoded symbol, *index* is the location where search for $d + 1$ fails. Otherwise, it is the location at which $d + 1$ has been found. q_i is a stream of i 1 bits. '#' is the concatenation operator. //

$j \leftarrow 0,\ k \leftarrow 0$
$f_{-1} \leftarrow 0$
while $j < N$ **do**
  if $(h < N - j)$ **then**
    $d \leftarrow b[j + 1 : j + h]$
  else
    $d \leftarrow b[j + 1 : N]$ # $q_{h - N + j}$
  endif
  binsearch(f, $d + 1$, *index*)
  $k \leftarrow k + 1$
  $c_k \leftarrow f_{index}$
  $j \leftarrow j + h - \log_2(f_{index} - f_{index-1})$
enddo

End of Algorithm Decode

In decoding the very last symbol of the text there may not be h bits available in the stream in which case d is obtained by appending the bit stream with sufficient number of 1 bits. It is obvious that in decoding a bit pattern we need to make a binary search among n symbols that requires $O(\log n)$ comparisons. If there are m codewords, our algorithm will run in $O(m \log n)$ time. It may be mentioned here that in order to improve repeated application of Huffman coding it is important to have the overhead of representing Huffman tree as efficiently as possible. So such efficient representation of Huffman header as proposed in this manuscript gives rise to the opportunity of applying repeated Huffman coding in cases where further compression is desirable.

Acknowledgement

Authors express their sincere thanks to anonymous referees for pointing out errors in the manuscript and for suggesting improvement of its presentation.

References

[1] Hong-Chung Chen, Yue-Li Wang and Yu-Feng Lan, A memory-efficient and fast Huffman decoding algorithm, *Information Processing Letters*, 69:119-122, 1999.

[2] K. L. Chung, Efficient Huffman decoding, *Information Processing Letters*, 61:97-99, 1997.

[3] R. Hashemian, Memory efficient and high-speed search Huffman coding, *IEEE Trans. Comm.*, 43(10): 2576-2581, 1995.

[4] D. A. Huffman, A method for construction of minimum redundancy codes, *Proc. IRE*, 40:1098-1101, 1952.

[5] G. O. H. Katona and T. O. H. Nemetz, Huffman Codes and Self-Information, *IEEE Transactions on Information Theory*, 22(3):337-340, 1978.

[6] J. N. Mandal, An approach towards development of efficient data compression algorithms and correction techniques, Ph.D. Thesis, Jadavpur University, India, 2000.

[7] R. Schack, The length of a typical Huffman codeword, *IEEE Transactions on Information Theory*, 40(4): 1246-1247, 1994.

[8] J. S. Vitter, Design and analysis of dynamic Huffman codes, *Journal of the ACM*, 34(4):825-845, 1987.