ELSEVIER

Information Processing Letters 61 (1997) 97-99

Information Processing Letters

Efficient Huffman decoding

Kuo-Liang Chung [1]

Department of Information Management, National Taiwan Institute of Technology, No. 43, Section 4, Keelung Road, Taipei, Taiwan 10672, ROC

Received 30 July 1996
Communicated by W.M. Turski

Abstract

We first present a memory-efficient array data structure to represent the Huffman tree. We then present a fast Huffman decoding algorithm. © 1997 Elsevier Science B.V.

Keywords: Data structures; Decoding algorithms; Huffman code

1. Introduction

Since Huffman discovered the Huffman encoding scheme [6] in 1952, Huffman code has been widely used in data, image, and video compression [1]. For example, the Huffman encoding is used to compress the result of a quantization stage in JPEG [7]. The simplest data structure used in a Huffman decoding scheme is the Huffman tree. The array data structure [6,8] has been used to implement the corresponding complete binary tree for the Huffman tree. The major disadvantage is the memory cost spent on storing such a complete binary tree by using an array. Suppose the height of the Huffman tree is t. The size of the array required in [6,8] is $O(2^t)$.

Consider the sparsity in the Huffman tree due to one-side growth of the tree. Using a novel array data structure, Hashemian [4] presented an efficient decoding algorithm consisting of an ordering and clustering scheme in order to alleviate the effect of sparsity in the Huffman tree and support quick search time in the look-up tables. However, how to partition the Huffman tree into many smaller clusters such that the memory required is minimum is still an open problem. Basically, the memory requirement in [4] is ranged from $O(n)$ to $O(2^t)$, where $2n-1$ denotes the number of nodes in the Huffman tree. Based on the modified S-tree [2], Chung and Lin [3] presented an array data structure with size $5n-4$ to represent the Huffman tree.

We first present a memory-efficient array data structure to represent the Huffman tree. The memory required in the proposed data structure is $3n-2$. Then we present a fast Huffman decoding algorithm based on the proposed data structure. Furthermore, the memory size can be reduced from $3n-2$ to $2n-3$.

2. The data structure

We take an example to demonstrate our proposed data structure for Huffman coding. Consider the source

[1] Email: klchung@cs.ntit.edu.tw. This research was supported in part by the National Science Council of R.O.C. under contract NSC86-2213-E011-010.

PII S0020-0190(96)00204-9

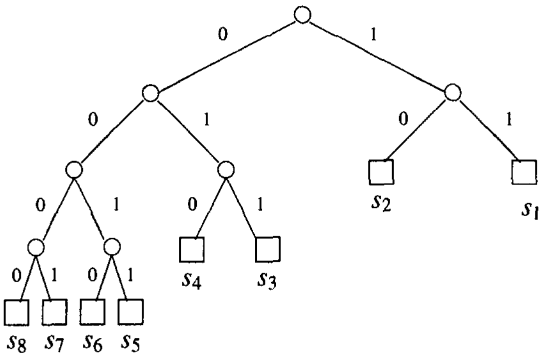

Fig. 1. An example of a Huffman tree.

symbols $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$ with frequencies $W = \{9,7,3,3,1,1,1,1\}$, respectively. Based on the Huffman encoding method, the corresponding Huffman tree is shown in Fig. 1, where each leaf node is corresponding to a source symbol.

Our data structure for representing the above Huffman tree is obtained by using the following two steps:

Step 1: We traverse the Huffman tree in preorder [5]. For each left edge (branch), we record the number of edges and leaf nodes in the subtree of that edge, then add one to each of these values. For convenience, these update values are called the "jump values" of these traversed edges. For example, the left edge of the root has the jump value 17 $(= 10 + 6 + 1)$.

Step 2: We traverse the Huffman tree in preorder again. At each time, we emit the "jump value" when a left edge is encountered, or a "1" when a right edge is encountered; we emit the source symbol when a leaf node is encountered. After traversing the Huffman tree, the sequence of these ordered values is saved in the array, namely, H_array.

Let $2n - 1$ denote the number of nodes in the Huffman tree. The memory required in the data structure H_array is $3n - 2$. According to the above procedure, the data structure for Fig. 1 is:

H_array: $17, 11, 5, 2, s_8, 1, s_7, 1, 2, s_6,$
$1, s_5, 1, 2, s_4, 1, s_3, 1, 2, s_2, 1, s_1$

3. The decoding algorithm

We follow the same example to demonstrate the basic concept of our Huffman decoding algorithm based on the above array H_array. Two variables, code_ptr and array_ptr, are used to point to current positions in the Huffman code (represented by array Huf_array) and H_array, respectively.

Consider the Huffman code 011. Initially, code_ptr and array_ptr point to the first element of Huf_array and H_array, respectively. Since Huf_array[1] = 0, code_ptr and array_ptr are increased by 1. At this time, Huf_array[2] = 1 and H_array[2] = 11. Then array_ptr is increased by 11, i.e., array_ptr = 2 + 11 = 13 and we have H_array[13] = Huf_array[2] = 1. Next, code_ptr is increased by 1. Since Huf_array[3] = 1, we proceed to the fourteenth element (= 13 + 1) of H_array and it follows that H_array[14] = 2. Therefore, array_ptr is increased by 2 and we have H_array[16] = 1. Since we have scanned the Huffman code, the decoding process is terminated. The Huffman code 011 is decoded to the symbol s_3 which is pointed by array_ptr + 1 (= 16 + 1) in H_array.

Our Huffman decoding algorithm is shown in Fig. 2.

The complexity of the above algorithm depends on the traversed path in the corresponding Huffman tree and takes $O(t)$ time, where t denotes the height of the Huffman tree.

4. Discussions and conclusions

The significance of the Huffman decoding is its popular use in data, image, and video compression. The main contribution of this paper is that we have presented a memory-efficient array data structure to store the Huffman tree and have designed the related Huffman decoding algorithm.

It is observed that in the array H_array, the value "1" is always following a source symbol. In fact, the value "1" can be removed in the array H_array and the memory size can be reduced from $3n-2$ to $2n-3$; the derived Huffman decoding algorithm still works well if we modify the above Huffman decoding algorithm slightly.

```
code_ptr:=1
array_ptr:=1
while(code_ptr <= len) /* 'len' denotes the length of Huffman code */
           /* for example, the length of Huffman code 011 is 3 */
do begin
If Huf_array[code_ptr]=0
  then begin
     code_ptr:=code_ptr+1
     array_ptr:=array_ptr+1
     end
  else begin
     code_ptr:=code_ptr+1
     array_ptr:=array_ptr+H_array[array_ptr]+1
   If Huf_array[code_ptr]<>$ /* '<>' denotes the symbol 'not equal' */
                  /* '$' is the symbol for end of Huffman code */
      then begin
         If Huf_array[code_ptr]=1
           then begin
              code_ptr:=code_ptr+1
              array_ptr:=array_ptr+H_array[array_ptr]+1
              end
           else begin
              code_ptr:=code_ptr+1
              array_ptr:=array_ptr+1
              end
          end
     end
  end
output H_array[array_ptr]
```

Fig. 2.

Acknowledgments

The author appreciates the anonymous referees and Professor W.M. Turski for their help in processing this paper.

References

[1] T.C. Bell, J.G. Cleary and I.H. Witten, *Text Compression* (Prentice Hall, Englewood Cliffs, NJ, 1990).

[2] K.L. Chung and C.J. Wu, A fast search algorithm on modified S-trees, *Pattern Recognition Lett.* 16 (1995) 1159-1164.

[3] K.L. Chung and Y.K. Lin, A novel memory-efficient and fast Huffman decoding algorithm, Research Rept., Dept. of Information Management, National Taiwan Institute of Technology, 1996.

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

[5] E. Horowitz, S. Sahni and S. Andersonfreed, *Fundamentals of Data Structures in C* (New York, 1993) 201.

[6] A. Huffman, A method for the construction of minimum redundancy codes, in: *Proc. IRE* 40 (1952) 1098-1101.

[7] W.B. Pennebaker and J.L. Mitchell, *JPEG: Still Image Data Compression Standard* (New York, 1993).

[8] S. Roman, *Coding and Information Theory* (Springer, New York, 1992).