ELSEVIER

Theoretical Computer Science 246 (2000) 227–238

Theoretical Computer Science

www.elsevier.com/locate/tcs

A space-efficient Huffman decoding algorithm and its parallelism

Yih-Kai Lin, Kuo-Liang Chung *,[1]

Department of Information Management and Institute of Information Engineering, National Taiwan University of Science and Technology, No. 43, Section 4, Keelung Road, Taipei, Taiwan 10672, Republic of China

Received May 1998; revised October 1998
Communicated by M. Nivat

Abstract

This paper first transforms the Huffman tree into a single-side growing Huffman tree, then presents a memory-efficient data structure to represent the single-side growing Huffman tree, which requires $(n+d)\lceil \log_2 n \rceil$-bits memory space, where n is the number of source symbols and d is the depth of the Huffman tree. Based on the proposed data structure, we present an $O(d)$-time Huffman decoding algorithm. Using the same example, the memory required in our decoding algorithm is much less than that of [3]. We finally modify our proposed data structure to design an $O(1)$-time parallel Huffman decoding algorithm on a concurrent read exclusive write parallel random-access machine (CREW PRAM) using d processors.

Keywords: Data structure; Decoding algorithm; Huffman code, Parallel algorithm; Single-side growing Huffman tree

1. Introduction

Since D.A. Huffman invented the Huffman encoding scheme in 1952, the Huffman code has been widely used in text, image, and video compression [1]. For example, it is used to compress the result of quantization stage in JPEG [6]. The simplest data structure used in the Huffman decoding is the Huffman tree. Array data structure [4, 7] has been used to implement the corresponding complete binary tree for the Huffman tree. However, the sparsity in the Huffman tree causes a huge waste of memory space

* Corresponding author.
E-mail address: klchung@cs.ntust.edu.tw (K.-L. Chung)

[1] Supported by the National Science Council of ROC under contracts NSC88-2213-E011-005 and NSC88-2213-E011-006.

0304-3975/00/$ - see front matter
PII: S0304-3975(99)00080-8

for array implementation [4, 7] which needs $O(2^d\lceil\log_2 n\rceil)$-bits space, where n is the number of source symbols and d is the depth of the Huffman tree.

Based on the single-side growing Huffman tree, Hashemian [3] presented an $O(d)$-time decoding algorithm consisting of ordering and clustering scheme in order to alleviate the effect of sparsity due to single-side growth in the single-side growing Huffman tree and to support quick search in the look-up array. The memory space required in [3] ranged from $O((n+d)\lceil\log_2 n\rceil)$-bits to $O(2^d\lceil\log_2 n\rceil)$-bits.

This paper first transforms the Huffman tree into a single-side growing Huffman tree, then presents a memory-efficient data structure to represent the single-side growing Huffman tree, which requires $(n+d)\lceil\log_2 n\rceil$-bits memory space. Based on the proposed data structure, we present an $O(d)$-time Huffman decoding algorithm. Using the same example, the memory required in our decoding algorithm is much less than that of [3]. We finally modify our data structure to design an $O(1)$-time parallel Huffman decoding algorithm on a CREW PRAM model [2] using d processors.

2. Data structure

Consider the source symbols $\{s_1, s_2, \ldots, s_n\}$ with frequencies $\{w_1, w_2, \ldots, w_n\}$ for $w_1 \geqslant w_2 \geqslant \cdots \geqslant w_n$, where the symbol s_i has frequency w_i. Using the Huffman's algorithm [4], a Huffman tree is obtained. Then the codeword c_i for $1 \leqslant i \leqslant n$, which is a binary string, for symbol s_i can be determined by traversing the path from the root to the leaf node associated with the symbol s_i, where the left edge is corresponding to '0' and the right edge is corresponding to '1'. Let the level of the root be zero and the level of the other node is equal to summing up its parent's level and one. Codeword length l_i for s_i can be known as the level of s_i. Then the weighted external path length $\sum_{i=1}^{n} w_i l_i$ is minimum. For example, the Huffman tree corresponding to the source symbols $\{s_1, s_2, \ldots, s_8\}$ with the frequencies $\{14, 13, 5, 3, 3, 2, 1, 1\}$ is shown in Fig. 1, and we have the two ordered sets $\langle c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\rangle = \langle 11, 10, 010, 001, 000, 0111, 01101, 01100\rangle$ and $\langle l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8\rangle = \langle 2, 2, 3, 3, 3, 4, 5, 5\rangle$.

We now want to construct a single-side growing Huffman tree from the given set $\{l_1, l_2, \ldots, l_n\}$ while preserving the same weighted external path length as the original Huffman tree (see Fig. 1). Instead of using the description [8], a formula for this construction is presented. Our proposed formula helps the derivations of some interesting properties. The single-side growing Huffman tree of Fig. 1 is shown in Fig. 2 whose construction will be described later.

From the set of codeword lengths $\{l_1, l_2, \ldots, l_n\}$ in the original Huffman tree, we present a codeword-assignment scheme to generate a new set of codewords for source symbols. Based on the proposed codeword-assignment scheme, a single-side growing Huffman tree is constructed. The first symbol s_1 in the single-side growing Huffman tree is assigned to codeword $c'_1 = \underbrace{11\ldots1}_{l_1}$. The next symbol s_2 is assigned to codeword $c'_2 = (c'_1 \times 2^{l_2 - l_1}) - 1$. The new codeword c'_2 is obtained by shifting the codeword c'_1

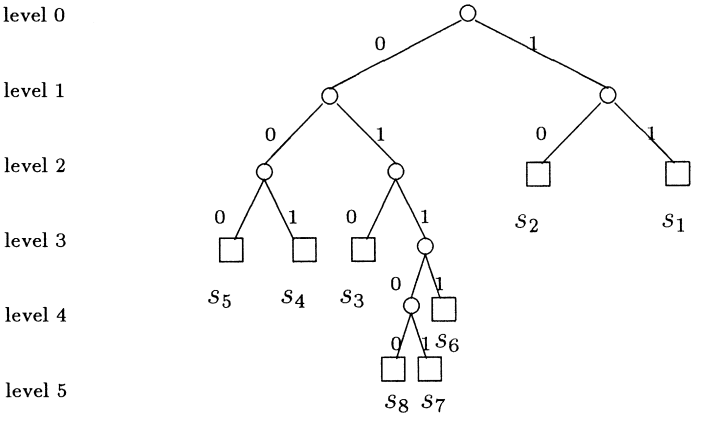


Fig. 1. An example of the Huffman tree.

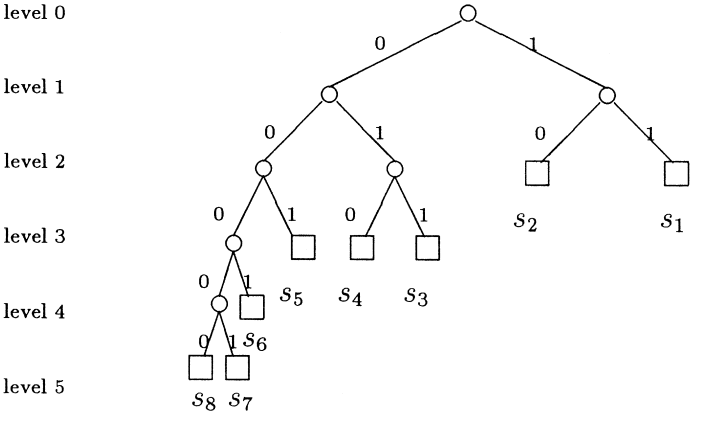


Fig. 2. The single-side growing Huffman tree of Fig. 1.

$(l_2 - l_1)$ bits to the left and appending $(l_2 - l_1)$ 0's to the tail, then subtracting from one. In general, the ith symbol s_i for $2 \leqslant i \leqslant n$ is assigned to codeword $c'_i = (c'_{i-1} \times 2^{l_i - l_{i-1}}) - 1$, where l_i and l_{i-1} are the codeword lengths for s_i and s_{i-1}, respectively, and summing up the length of c'_{i-1} and $(l_i - l_{i-1})$ is equal to the length of c'_i. Note that $c'_n = \underbrace{00\ldots0}_{l_n}$.

From the preceding example, we have $\langle l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8 \rangle = \langle 2, 2, 3, 3, 3, 4, 5, 5 \rangle$, so s_1 is assigned to codeword $c'_1 = \underbrace{11}_{2}$ based on our codeword-assignment scheme. The remaining symbols are assigned to codewords as shown below:

$$c'_2 = (11 \times 2^{2-2}) - 1 = 10,$$
$$c'_3 = (10 \times 2^{3-2}) - 1 = 011,$$
$$c'_4 = (011 \times 2^{3-3}) - 1 = 010,$$

$$c'_5 = (010 \times 2^{3-3}) - 1 = 001,$$
$$c'_6 = (001 \times 2^{4-3}) - 1 = 0001,$$
$$c'_7 = (0001 \times 2^{5-4}) - 1 = 00001,$$
$$c'_8 = (00001 \times 2^{5-5}) - 1 = 00000.$$

It is observed that the generated new set of codewords using our codeword-assignment scheme are consistent with the set of codewords in Fig. 2.

The following lemma guarantees that for the same symbol, i.e., s_3, the length of the codeword in the original Huffman tree is (see Fig. 1) equal to the one in the single-side growing Huffman tree (see Fig. 2). It implies that for the two Huffman trees, the weighted external path length is the same.

Lemma 1. *If a leaf node associated with the symbol s_i in the original Huffman tree has codeword-length l_i, the corresponding codeword c'_i obtained by using the above codeword-assignment scheme has the same length l_i.*

Proof. We prove it by induction. From $c'_1 = \underbrace{11\ldots1}_{l_1}$, the basis is satisfied since the codeword-length of s_1, i.e., the length of c_1, is equal to l_1. Assume that the length of c'_{i-1} for $3 \leqslant i \leqslant n$ is equal to l_{i-1} is true. We do the induction part. If $l_i = l_{i-1}$, then $c'_i = c'_{i-1} \times 2^0 - 1 = c'_{i-1} - 1$ and the length of c'_i is equal to $l_{i-1} = l_i$ because $c'_{i-1} > 0$ for $3 \leqslant i \leqslant n$; if $l_i \neq l_{i-1}$, then $c'_i = c'_{i-1} \times 2^{l_i - l_{i-1}} - 1$, where the length of c'_i is enforced to l_i by adding $(l_i - l_{i-1})$ 0's. So, the length of c'_i is equal to l_i $(= l_{i-1} + (l_i - l_{i-1}))$. □

From the preceding codeword-assignment scheme and Lemma 1, we have the following result immediately.

Theorem 1. *For each level in the constructed single-side growing Huffman tree, all the internal nodes are to the left side of all the leaf nodes at the same level.*

Although some data structures for representing the single-side growing Huffman tree have been presented in [3, 8, 5], our codeword-assignment scheme for constructing the single-side growing Huffman tree brings out the design of the space-efficient Huffman decoding algorithm using only $(n+d)\lceil \log_2 n \rceil$-bits memory space, where d is the depth of the single-side growing Huffman tree; d is also the depth of the original Huffman tree. The memory requirement in our proposed method is the least when compared to the previous methods [3, 8, 5]. We now want to assign logical addresses for storing those symbols $\{s_i,\ 1 \leqslant i \leqslant n\}$ in the single-side growing Huffman tree.

Let f_k be the number of leaf nodes at level k, each leaf node with codeword-length k; let $\mathscr{I}_k$ be the number of the internal nodes at level k, where $\mathscr{I}_k$ satisfies the recurrence relation: $\mathscr{I}_0 = 1$ and $\mathscr{I}_k = \mathscr{I}_{k-1} \times 2 - f_k$. In the remainder of this section, we discuss how to use an array called the symbol table to store the s_i's for $1 \leqslant i \leqslant n$, which will be used in the decoding phase.

The logical address, a_i, for storing s_i in the symbol table is given by the following address-assignment scheme:

$$a_i = c'_i - \mathscr{I}_{l_i} + \sum_{k=2}^{l_i} f_{k-1} \quad \text{for } 1 \leqslant i \leqslant n.$$

Lemma 2. *At the same level, if the leaf node p in the single-side growing Huffman tree is to the left side of the leaf node q, the logical address of p is smaller than that of q.*

Proof. Let s_i and s_{i+1}, $1 \leqslant i < n$, be at the same level, i.e., $l_i = l_{i+1}$. We have that $\mathscr{I}_{l_i} = \mathscr{I}_{l_{i+1}}$ and $\sum_{k=2}^{l_i} f_{k-1} = \sum_{k=2}^{l_{i+1}} f_{k-1}$. In addition, we see that the ordered sequence $\langle c'_{e+f_{l_i}-1}, c'_{e+f_{l_i}-2}, \ldots, c'_e \rangle$ at level l_i, where c'_e is the largest codeword at level l_i, is a strictly increasing sequence. Therefore, from the above address-assignment scheme, we have $a_{i+1} > a_i$ for $e \leqslant i < e + f_{l_i} - 1$. By using the transitive property, we complete the proof. □

Following the same notation $c'_{e+f_{l_i}-1}$ used in Lemma 2, Fig. 2 shows that $\mathscr{I}_3 = 1$ and $\mathscr{I}_3 - c'_5 = 1 - 1 = 0$. In general, we have the following lemma:

Lemma 3. *Subtracting $\mathscr{I}_k$ from c'_{e+f_k-1} at level k, $0 \leqslant k \leqslant d$, is equal to zero.*

Proof. We prove it by induction. The basis is satisfied since we have $\mathscr{I}_{l_1} - c'_{1+f_{l_1}-1} = (2^{l_1} - f_{l_1}) - (\underbrace{11\ldots1}_{l_1} - f_{l_1} + 1) = 0$ when $k = l_1$. Assume Lemma 3 is true for $k = i$, $l_1 < i < d$. We want to induce that Lemma 3 is also true for $k = i + j$, $j > 0$, where $i + j$ is the smallest level whose f_{i+j} is larger than 0. From the definition of $\mathscr{I}$, we have $\mathscr{I}_{i+j} = \mathscr{I}_i \times 2^j - f_{i+j}$. In addition, from the codeword assignment, we have $c'_{e+f_{i+j}-1} = c'_{e+f_i-1} \times 2^j - 1 - f_{i+j} + 1 = c'_{e+f_i-1} \times 2^j - f_{i+j}$. From the hypothesis, it follows that $\mathscr{I}_i = c'_{e+f_i-1}$. We then have $\mathscr{I}_{i+j} = c'_{e+f_i-1} \times 2^j - f_{i+j} = c'_{e+f_{i+j}-1}$. We complete the proof. □

Lemma 4. *If the level of the leaf node s_q in the single-side Huffman tree is larger than the level of the leaf node s_p, the logical address of s_p is smaller than that of s_q, where $0 \leqslant l_p < l_q \leqslant d$.*

Proof. From the proof in Lemma 3, it is clear that at any level l_p whose f_{l_p} is larger than 0 for $0 \leqslant l_p < d$ and the value of $(c'_p - \mathscr{I}_{l_p})$ ranges from 0 to $f_{l_p} - 1$. Let c'_p be the codeword of one leaf node at level l_p and $c'_{q'}$ be the codeword of one leaf node at level $l_{q'} > l_p$, where $l_{q'}$ is the smallest level which has $f_{l_{q'}} > 0$, it follows that $a_{q'} > a_p$ because $a_{q'} - a_p = (c'_{q'} - \mathscr{I}_{l_{q'}}) - (c'_p - \mathscr{I}_{l_p}) + f_{l_p} > 0$. By the transitive property, we complete the proof. □

The range of the logical address is shown in the following theorem:

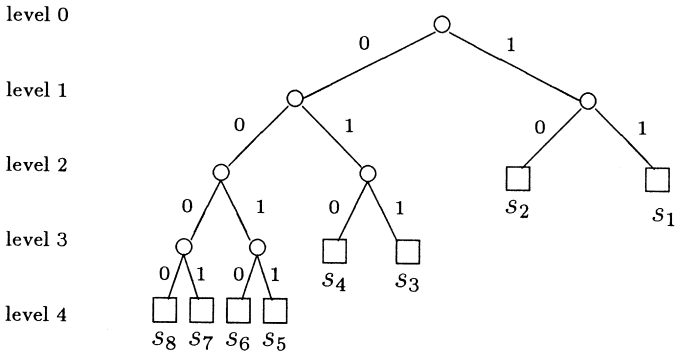


Fig. 3. The constructed single-side growing Huffman tree.

Theorem 2. *Let a_i be the logical address of s_i, then we have $0 \leqslant a_i \leqslant n-1$ for $1 \leqslant i \leqslant n$.*

Proof. From the logical address-assignment and the proof in Lemma 3 we have $a_1 = 0$. By Lemmas 2 and 4, it is clear that $0 \leqslant a_i$ for $1 \leqslant i \leqslant n$. On the other hand, from the logical address-assignment and the proof in Lemma 4, we have $a_i < n-1$ for $1 \leqslant i \leqslant n$.

Consider a new example. The given set of source symbols is $\langle s_1, s_2, \ldots, s_8 \rangle$ with the frequencies $\langle 9, 7, 3, 3, 1, 1, 1, 1 \rangle$ and the lengths $\langle l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8 \rangle = \langle 2, 2, 3, 3, 4, 4, 4, 4 \rangle$, respectively. Using our preceding codeword-assignment scheme, we have $\langle c'_1, c'_2, c'_3, c'_4, c'_5, c'_6, c'_7, c'_8 \rangle = \langle 11, 10, 011, 010, 0011, 0010, 0001, 0000 \rangle$ and the single-side growing Huffman tree is shown in Fig. 3. In addition, it follows that $\langle f_1, f_2, f_3, f_4 \rangle = \langle 0, 2, 2, 4 \rangle$ and $\langle \mathscr{I}_0, \mathscr{I}_1, \mathscr{I}_2, \mathscr{I}_3 \rangle = \langle 1, 2, 2, 2 \rangle$. Further, using our address-assignment scheme, the logical address a_i, $1 \leqslant i \leqslant n$, for s_i is shown below:

$$
\begin{aligned}
a_1 &= 3 - (2 \times 2 - 2) + 0 = 1,\\
a_2 &= 2 - (2 \times 2 - 2) + 0 = 0,\\
a_3 &= 3 - (2 \times 2 - 2) + 2 = 3,\\
a_4 &= 2 - (2 \times 2 - 2) + 2 = 2,\\
a_5 &= 3 - (2 \times 2 - 4) + 4 = 7,\\
a_6 &= 2 - (2 \times 2 - 4) + 4 = 6,\\
a_7 &= 1 - (2 \times 2 - 4) + 4 = 5,\\
a_8 &= 0 - (2 \times 2 - 4) + 4 = 4.
\end{aligned}
$$

Thus, the corresponding symbol table S for saving s_i of Fig. 3, $1 \leqslant i \leqslant n$, is shown in Fig. 4, where the logical address of s_i in S is denoted by a_i. For example, the logical address of $s_8(s_7)$ is $a_8 = 4(a_7 = 5)$.

The skip table is defined to be $\mathrm{F} = (f_1, f_2, \ldots, f_d)$, where d is the depth of the single-side growing Huffman tree and f_i for $1 \leqslant i \leqslant d$ has been defined. The corresponding

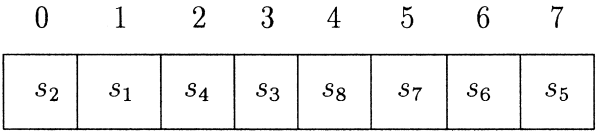


Fig. 4. The symbol table.

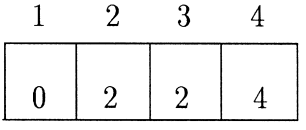


Fig. 5. The skip table.

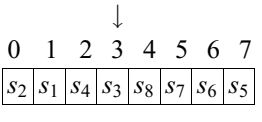
skip table of Fig. 3 is shown in Fig. 5. The symbol table and the skip table will be used in our Huffman decoding algorithm. Since the value of each entry in the skip table and the symbol table is dominated by the index of the source symbol, i.e., at most n, $\lceil \log_2 n \rceil$ bits are enough to save each entry. Here, in each entry of the symbol table, we use index i to represent s_i. In the case of a source with 256 symbols, 8 ($= \log_2 256$) bits are required to save each entry. Therefore, the memory space required in the two tables needs $(n+d)\log_2 n$ bits.

3. Decoding algorithm

In this section, we take the same example as shown in Fig. 3 to demonstrate our decoding algorithm.

Given an input code $H = h_0 h_1 \ldots h_r$, $h_i \in \{0,1\}, 0 \leqslant i \leqslant r$, to be decoded, let code_ptr denote the current position of H processed so far; prefix_H(t) be $h_0 h_1 h_2 \ldots h_{t-1}$ when code_ptr $= t-1$; cprefix_H(t) be $\sum_{i=0}^{t-1} h_i \times 2^{t-1-i}$. One variable, namely, symbol_ptr, is used to point to the current position in the symbol table.

Consider an input code H = 011. The decoding procedure starts with code_ptr = 1. At this time, we have prefix_H(1) = 0, cprefix_H(1) = 0, and $\mathscr{I}_1 = 2$. Since cprefix_(1) $< \mathscr{I}_1$ and F(code_ptr) = 0, we continue the decoding process and the value of code_ptr is increased by one, i.e., code_ptr = 2. We then have prefix_H(2) = 01, cprefix_H(2) = 1, and $\mathscr{I}_2 = 2$. Since cprefix_H(2) = 1 $< \mathscr{I}_2$ and F(code_ptr) = F(2) = 2, we continue the decoding process. Next, the value of code_ptr becomes 3 and the value of symbol_ptr becomes 2. Now we have cprefix_H(3) = 3 $> \mathscr{I}_3 = 2$, so the decoded symbol equals S(symbol_ptr + cprefix_H(3) − $\mathscr{I}_3$) = S(3) = s_3 and is shown below.



Following the preceding simulation example, the formal decoding algorithm based on the proposed data structure is listed below.

Algorithm
Input:
Huffman[ ] : array of binary values /* input code */
symbol[ ] : array of integers /* symbol table */
skip[ ] : array of integers /* skip table */
Output:
the source symbol corresponding to the input code

```
begin
  code_ptr := 1 /* point to the first position in the given code */
  symbol_ptr := 0 /* point to the first position in the symbol table */
  internal_node := 1 /* indicate the number of internal nodes at the current level */
  cprefix_H: = 0
  while (Huffman[code_ptr] = 0 or Huffman[code_ptr] = 1)
    do begin
      internal_node := internal_node × 2 − skip[code_ptr]
      cprefix_H := cprefix_H × 2 + Huffman[code_ptr]
      if cprefix_H ⩾ internal_node then /* arrive at a leaf node */
        return symbol[symbol_ptr+cprefix_H − internal_node]
      end
      symbol_ptr := symbol_ptr+skip[code_ptr] /* skip symbol*/
      code_ptr := code_ptr+1 /* go to the next level */
    end
end
```

Since the time complexity of the above decoding algorithm is dependent on the depth of the single-side growing Huffman tree, we have the result.

Theorem 3. *The Huffman decoding algorithm can be performed in* $O(d)$ *time using* $(n+d)\lceil \log_2 n \rceil$*-bits memory space.*

Under the same time bound $O(d)$, we now compare the memory space required in our decoding method to the Hashemian's method [3] by using the same example in [3]. Consider the single-side growing Huffman tree [3] as shown in Fig. 6. The look-up array needs 122 bytes by using Hashemian's method. The readers are referred to the detailed array-construction [3]. Based on our proposed data structure described in Section 2, the symbol table and the skip table are shown below

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
01	00	07	06	05	04	03	02	08	0b	0a	09	0d	0c	12	11	10	0f	0e	15	14	13	1a	19	18	17	16	1d	1c	1b	1f	1e

The symbol table.

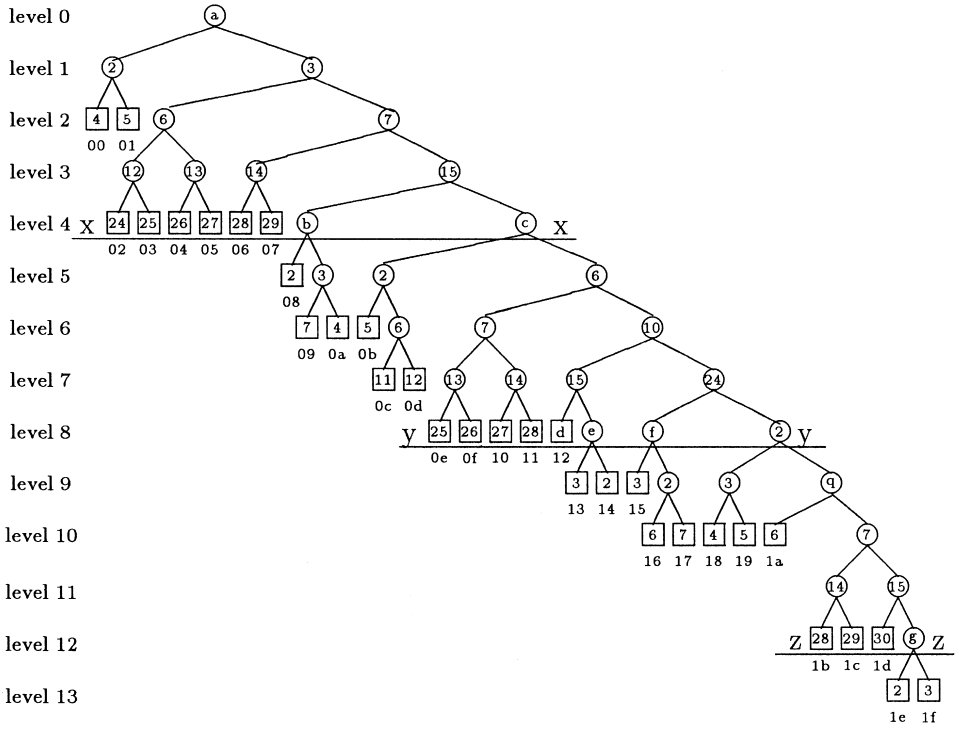


Fig. 6. A 13-level single-side growing Huffman tree.

1	2	3	4	5	6	7	8	9	10	11	12	13
0	2	0	6	1	3	2	5	3	5	0	3	2

The skip table.

Totally, our data structure takes 45 bytes. It is obvious that the memory required in our method is much less than that of [3]. Theoretically, our data structure needs $(n+d)\lceil\log_2 n\rceil$-bits memory while the memory required in [3] ranges from $(n+d)\lceil\log_2 n\rceil$-bits to $O(2^d\lceil\log_2 n\rceil)$-bits. On the other hand, the memory required in this paper is the lower bound required by Hashemian's method.

4. O(1)-time parallel decoding algorithm

In this section, we first modify the data structure described in Section 2 slightly. Then an O(1)-time parallel Huffman decoding algorithm for determining a source symbol each time is presented on a CREW PRAM [2] model using d processors.

Since the decoding algorithm described in Section 3 traverses the path in the single-side growing Huffman tree logically, the number of internal nodes at level i can be computed by $\mathscr{I}_i = \mathscr{I}_{i-1}\times 2 - f_i$ for $1 \leqslant i \leqslant d$. In our parallel decoding algorithm, we need keep $\mathscr{I}_i$ for $1 \leqslant i \leqslant d$, so an internal table $\mathrm{I} = (\mathscr{I}_1, \mathscr{I}_2, \ldots, \mathscr{I}_d)$ is built. The

1	2	3	4
2	2	2	0

Fig. 7. The internal table.

1	2	3	4
0	0	2	4

Fig. 8. The prefix-skip table.

corresponding internal table of Fig. 3 is shown in Fig. 7. In addition, we need a prefix-skip table $P=(0, f_1, f_1+f_2, \ldots, \sum_{i=1}^{d-1} f_i)$, which can be computed from the skip table $F=(f_1, f_2, \ldots, f_d)$. The corresponding prefix-skip table of Fig. 3 is shown in Fig. 8. In fact, the two tables I and P can be obtained in the preprocessing step.

We now take the same example as shown in Fig. 3 to demonstrate how our parallel decoding algorithm works. Then a formal parallel algorithm is given.

Suppose we have a CREW PRAM with d processors, say $PE_1, PE_2, \ldots,$ and PE_d. One shared variable, say *base*, is used to point out the current position of the given Huffman code $H=h_1h_2\ldots h_r$ and the variable *base* can be accessed by each PE_i for $1\leqslant i\leqslant d$. Initially, $base=1$. For convenience, let $H_{base,d}=h_{base}h_{base+1}\ldots h_{base+d-1}$. For decoding the corresponding source symbol in $H_{base,d}$, PE_i, for $1\leqslant i\leqslant d$, compares $Val(h_{base}h_{base+1}\ldots h_{base+i-1})$ to $\mathscr{I}_i$, where $Val(h_{base}h_{base+1}\ldots h_{base+i-1})$ denotes the decimal value of $h_{base}h_{base+1}\ldots h_{base+i-1}$. If $Val(h_{base}h_{base+1}\ldots h_{base+i-1})$ is larger than or equal to $\mathscr{I}_i$, then PE_i outputs the source symbol $S[\sum_{j=1}^{i-1} f_j + Val(h_{base}h_{base+1}\ldots h_{base+i-1}) - \mathscr{I}_i]$. From the prefix property of Huffman code, at each time only one PE, say PE_i for $1\leqslant i\leqslant d$, does output one source symbol with respect to $H_{base,d}$. Then the value of base is changed into $base+i$. Computing $\sum_{i=1}^{j} f_i$ and $\mathscr{I}_i$ for $1\leqslant j\leqslant d-1$ in preprocessing stage is the main difference between the parallel decoding algorithm and sequential decoding algorithm. So, the proposed parallel decoding algorithm is correct as described in Sections 2 and 3.

Consider an input code $H=0111110011$. The decoding procedure starts with $base=1$. At this time, PE_3 has $Val(h_1h_2h_3)=Val(011)=3\geqslant 2=\mathscr{I}_3$, but the value of $Val(h_1h_2\ldots h_i)$ in the other PE_i for $1\leqslant i\neq 3\leqslant 4$ is smaller than $\mathscr{I}_i$. Then the symbol $S[2+3-2]=S[3]=s_3$ is output by PE_3, and the *base* is changed into $4=base+3$. Now, PE_2 has $Val(h_4h_5)=Val(11)=3\geqslant 2=\mathscr{I}_2$. So, the symbol $S[0+3-2]=S[1]=s_1$ is output by PE_2 and the *base* is changed into $6=base+2$. Continuing this way, PE_4 has $Val(h_6h_7h_8h_9)=Val(0011)=3\geqslant 1=\mathscr{I}_4$. Finally, the symbol $S[4+3-0]=S[7]=s_5$ is returned by PE_4 and the *base* is changed into $base+4=10$ by PE_4.

Following the preceding description, the formal parallel decoding algorithm and one theorem are listed below.

```
Algorithm
Input: The arrays H = (h_1, h_2, ..., h_r), P = (0, f_1, f_1+f_2, ..., sum_{i=1}^{d-1} f_i),
       I = (I_1, I_2, ..., I_d) and S = (s_1, s_2, ..., s_n) stored in the shared memory.
Output: the source symbols corresponding to H
begin
  base := 1
  while (base ⩽ r) do
    begin
      for each i, 1 ⩽ i ⩽ d − 1 do in parallel
        if Val(base,...,base +i − 1) ⩾ I[i]
          begin
            output S[P[i] + Val(base,...,base +i − 1) − I[i]]
            base := base + i
          end
    end
end
```

Fig. 9. The O(1)-time parallel decoding algorithm.

Theorem 4. *Given an input code, the Huffman decoding algorithm for determining one source symbol can be performed in* O(1) *time on a CREW PRAM with* O(d) *processors.*

5. Conclusions

The significance of the Huffman code is due to its popular use in image and data compression. The major contributions of this paper are two fold: first, we have presented a space-efficient Huffman decoding algorithm based on a newly proposed data structure; secondly, an O(1)-time parallel Huffman decoding algorithm on CREW PRAM has been developed. Our future research topics focus on (1) preprocessing the single-side growing Huffman tree in a more compact form as well as keeping the efficiency of the decoding process and (2) modifying the proposed data structure to handle the dynamic Huffman coding.

Acknowledgements

The authors are indebted to the reviewers and Prof. M. Nivat for making some valuable suggestions and corrections that lead to the improved version of the paper.

References

[1] T.C. Bell, J.G. Cleary, I.H. Witten, Text Compression, Prentice-Hall, Englewood Cliffs, NJ, 1990.
[2] S. Fortune, J. Wyllie, Parallelism in random access machines, in: Proc. ACM Symp. on Theory of Computing, 1978, pp. 114–118.
[3] R. Hashemian, Memory efficient and high-speed search Huffman coding, IEEE Trans. Commun. 43 (1995) 2576–2581.
[4] D.A. Huffman, A method for the construction of minimum redundancy codes, Proc. IRE 40 (1952) 1098–1101.
[5] A. Moffat, A. Turpin, On the implementation of minimum redundancy prefix codes, IEEE Trans. Commun. 45 (1997) 1200–1207.
[6] W.B. Pennebaker, J.L. Mitchell, JPEG: Still Image Data Compression Standard, Van Nostrand Reinhold, New York, 1993.
[7] S. Roman, Coding and Information Theory, Springer, New York, 1992.
[8] B.W.Y. Wei, T.H. Meng, A parallel decoder of programmable Huffman codes, IEEE Trans. Circuits Systems Video Technol. 5 (1995) 175–178.