Improved Adaptive Huffman Compression Algorithm

Satpreet Singh
CEO, Singh Soft
Manteca, United States
satpreetsingh@ymail.com

Harmandeep Singh
Assistant Professor
Department of Computer Science
Sri Guru Angad Dev College, Khadoor Sahib
gill.gurseerat@gmail.com

1. ABSTRACT

In information age, sending the data from one end to another end need lot of space as well as time. Data compression is a technique to compress the information source (e.g. a data file, a speech signal, an image, or a video signal) in possible few numbers of bits. One of the major factors that influence the Data Compression technique is the procedure to encode the source data and space required for encoded data. There are many data compressions methods which are used for data compression and out of which Huffman is mostly used for same. Huffman algorithms have two ranges static as well as adaptive. Static Huffman algorithm is a technique that encoded the data in two passes. In first pass it requires to calculate the frequency of each symbol and in second pass it constructs the Huffman tree. Adaptive Huffman algorithm is expanded on Huffman algorithm that constructs the Huffman tree but take more space than Static Huffman algorithm. This paper introduces a new data compression Algorithm which is based on Huffman coding. This algorithm not only reduces the number of pass but also reduce the storage space in compare to adaptive Huffman algorithm and comparable to static.

2. KEYWORDS: Data Compression, Static Huffman, Adaptive Huffman, Algorithm.

3. INTRODUCTION

Compression Technique is a standard that squeeze source data, to take less space for storing the same data. The only threat to compress the data is loss of valuable data there are many algorithms existing in the information technology and every algorithm has its own advantages and disadvantages. Huffman is one of the techniques which are highly used by many solution provider companies now in these days. This technique has two principal algorithms static as well as adaptive. This paper is organized as follows

Section 4 illustrates the brief introduction of Static Huffman algorithm Section 5 defines the brief introduction of Adaptive Huffman algorithm Section 6 defines the design strategy of Improved adaptive Huffman algorithm and finally the conclusion are given in Section 7.

4. Static Huffman algorithm

The Static Huffman algorithm developed by David Huffman (1952) generates the encoded data in two passes that are as follows

1. Calculate the frequency of each different symbol present in the source data. After calculating frequencies construct the table of all frequency in decreasing order by sort it of each different symbol in decreasing order
2. Create Huffman tree by combining the least two symbols into one composite symbol

Example

Static Huffman Tree for encoding the Data "caaaddbddd "

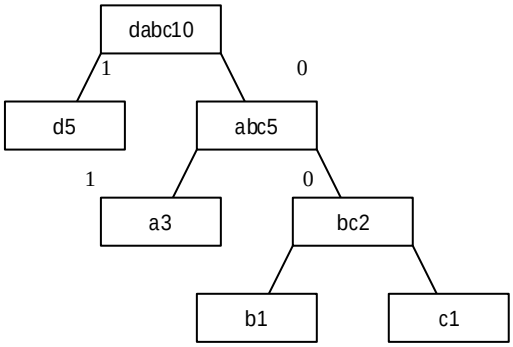
First pass:

Symbol	Frequency
a	3
b	1
c	1
d	5

Sort the Frequencies in decreasing order

Symbol	Frequency
d	5
a	3
b	1
c	1

Second pass;



Symbol	code
a	01
b	001
c	000
d	1

000	01	01	01	1
c	a	a	a	d
1	001	1	1	1
d	b	d	d	d

Results: Encoded data is seventeen bits long
For implementing Static Huffman Encoding require Two Passes.

5. Adaptive Huffman algorithm

Expanding on the static Huffman algorithm, Faller and Gallagher [Faller 1973; Gallagher 1978], and later Knuth [Knuth 1985] and Vitter [Vitter 1987], developed a way to perform Static Huffman algorithm as one pass that are as follows

1. Initially Adaptive Huffman algorithm generates a Huffman tree with all different symbols frequency count to one and took code for first symbol in the source data. For the second symbol it generates the second Huffman tree and took the code for second symbol (First and Second symbol may be same or different) and so on up to last bit (byte) of source data.

Concept

The basic concept behind an adaptive compression algorithm is very simple:

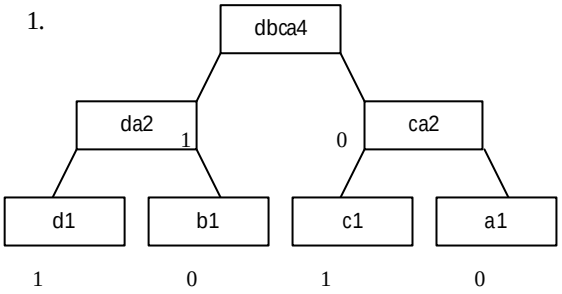
Initialize the model
  Repeat for each character
    {
      Encode character
      Update the model
    }

Decompression works the same way. As long as both sides have the same initialize and update model algorithms, they will have the same information.

Example

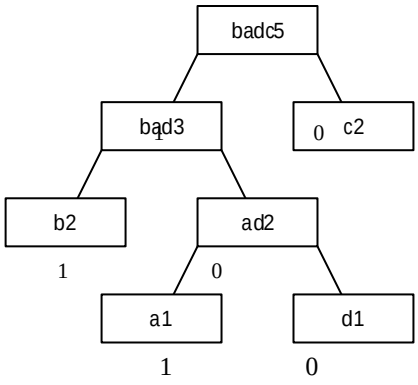
Adaptive Huffman Trees for encoding the Data " caaaddbddd"

1.



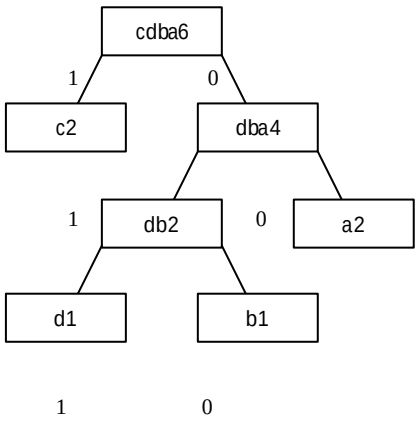
a-00 b-10 **c-01** d-11

2.



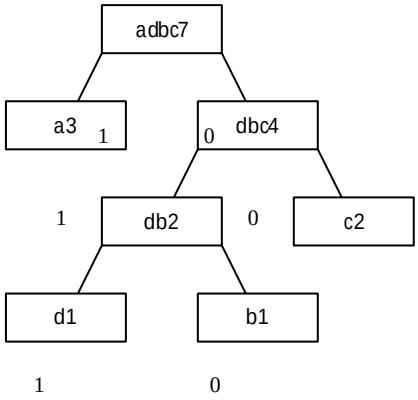
a-101 b-11 c-0 d-100

3.



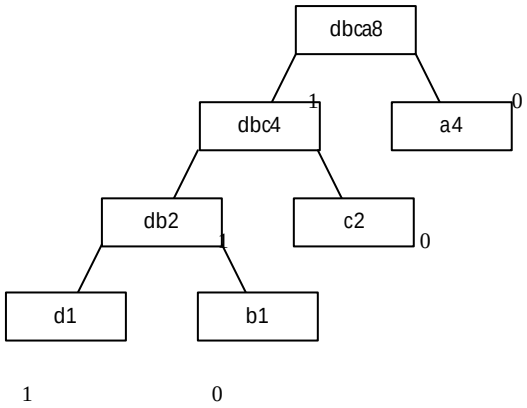
a-00 b-010 c-1 d-011

4.



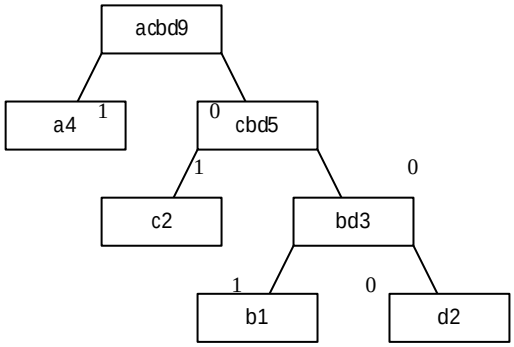
a-1 b-010 c-00 d-011

5.



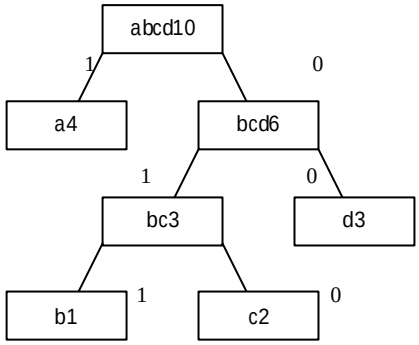
a-0 b-110 c-10 **d-111**

6.



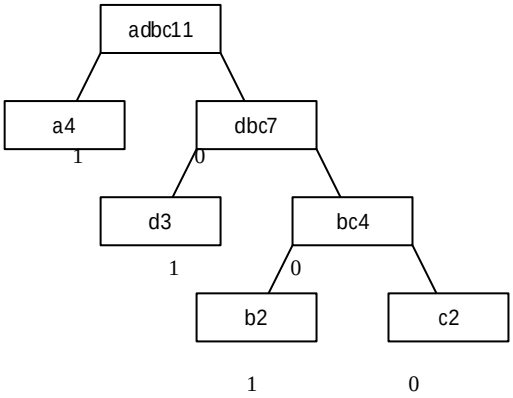
a-1 b-001 c-01 **d-000**

7.



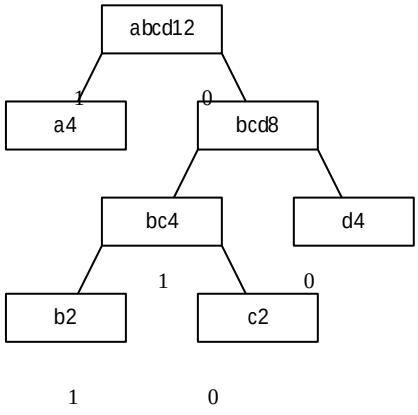
a-1 **b-011** c-010 d-00

8.



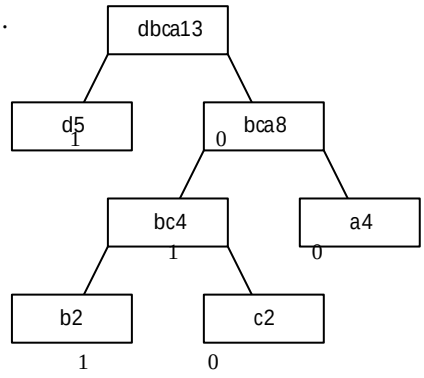
a-1 b-001 c-000 **d-01**

9.



a-1 b-011 c-010 **d-00**

10.



a-00 b-011 c-010 **d-1**

01	101	00	1	111	000
c	a	a	a	d	d

011	01	00	1
b	d	d	d

Result: Encoded data is twenty two bits long

For implementing Adaptive Huffman algorithm we must need to know in advance that how many different symbols are present in the source data. Adaptive Huffman Encoded data takes more space than static Huffman Encoded data. Adaptive Huffman Encoding generates a different tree for every next symbol (different or same). Every tree makes a different code for next symbol (even for same symbol) therefore it is must to remember all trees for decoding the data.

6. Improved Adaptive Huffman Algorithm

Static Huffman algorithm first scans all the source data and count frequency of each symbol. It then sorts the frequency table in decreasing order. In second pass it constructs the tree based on the pass one table. But in this method some time the source data is so lengthy it takes so much time to construct a table that is wastage of time as well as space required to store the table.

In Adaptive Huffman algorithm, while encoding the symbols, after each symbol is encoded we need to update the tree. Same is also done in decoding the code. That means there is some processing overburden involved. The encoded data by Adaptive Huffman algorithm requires more space than Static Huffman encoded data. The other major drawback of the Adaptive Huffman algorithm for encoding the data it must require in advance that how many different symbols are present in the source data. So it will first scan all the source data to determine that how many different symbols are present in the source data. The some other major drawbacks of adaptive Huffman algorithm are as follows:

a. Adaptive Huffman require more space to store the compressed data.
b. Adaptive Huffman must know in advance that how many different symbols are present in the data. So it will scan all the string before constructing the first tree.
c. It is very time consuming, it first construct the tree and than take the code for the symbol, for the next symbol it do the same (up to the last symbol).
d. In adaptive Huffman algorithm many of the different symbols having same code in the encoded data which creates a lot of confusing while decompressed the data.
e. In adaptive Huffman same symbol that occurs frequently has different code. So it can create confusion while decompressing the data.
f. Finally while decompressing the data we need all trees, for smaller data it is ok but for large data it needs a huge storage space.

Improved Adaptive Huffman algorithm which is based on existing Huffman algorithm having a one pass in compare to the existing static Huffman algorithms and requires less space for storing the encoded data as compare to adaptive Huffman algorithm. The purposed method with its algorithm is as follows:

1. Initially Improved adaptive Huffman algorithm generates strictly binary tree on reading first symbol in the source data. For the next symbol it generates a tree and so on up to last symbol of source data. On reading the last symbol it makes the final Huffman tree.

Advantages of Improved Adaptive Huffman over Adaptive Huffman are:

a. Improved adaptive Huffman requires less space to store the compressed data.
b. It saves the time because it does not need to scan the whole string for constructing the first tree. It also saves the time while constructing trees e.g. it needs only one symbol for constructing the first tree unlike in adaptive Huffman requires all different symbols to construct the tree.
c. In Improved adaptive Huffman one symbol (even occurs frequently) has a same code.
d. In improved adaptive Huffman, while constructing the next tree we do not need to remember the previous tree.
e. Finally while decompressing the data we need of only final tree.

Algorithm

1. Scan first Symbol and initialize their frequency to 1
2. Scan the next symbol from the source data

If any previous symbol = next symbol **then**

Increment the frequency of that

previous symbol

If any previous symbol

frequency < recently

incremented symbol frequency

then

Interchange both nodes

Else

Initialize their frequency to

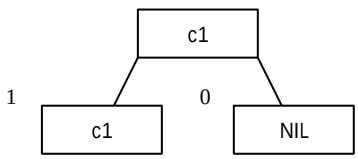
1

3. Create strictly binary tree with left and right node (Left or Right node can be NULL). And the root is the composite Symbols of left and right nodes. Assign 0 to Right node and 1to Left node.
4. Repeat step 2 to 4 till End of Source data.

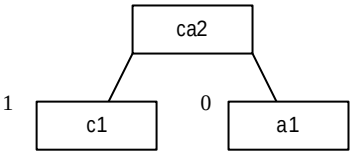
Example

New Huffman tree for encoding the data " **caaaddbddd** "

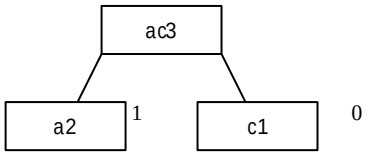
1.



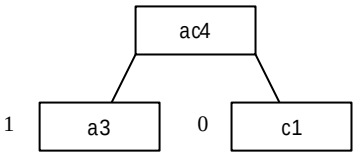
2.



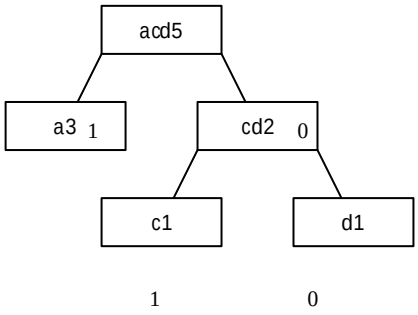
3.



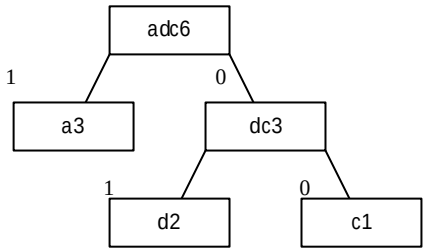
4.



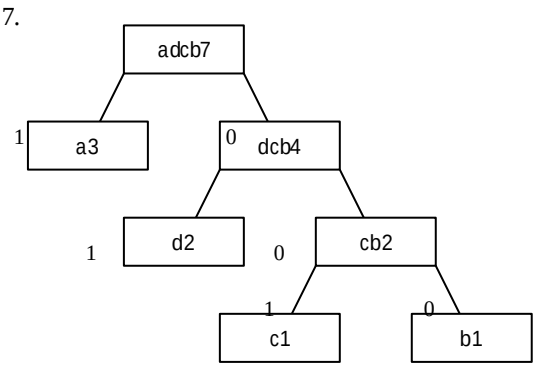
5.



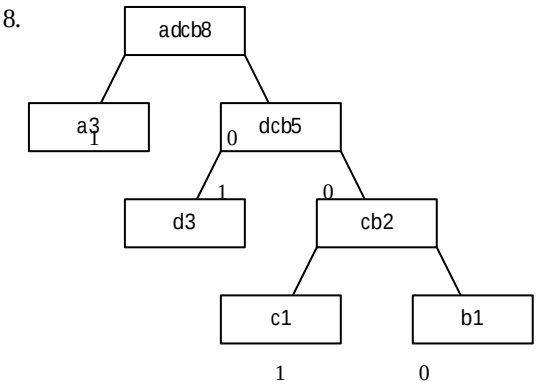
6.



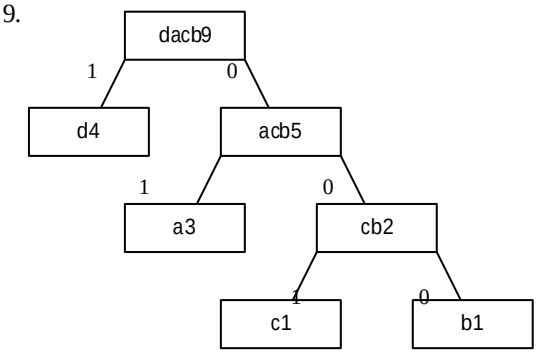
7.



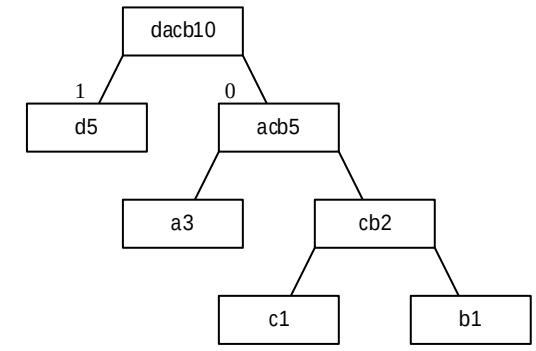
8.



9.



5. Final tree from where to construct the table for compression is as follows:



1 0

1 0

Symbol	code
a	01
b	000
c	001
d	1

001	01	01	01	1	1
c	a	a	a	d	d

000	1	1	1
b	d	d	d

Result: Encoded data is seventeen bits long

This method requires only one pass to encode source data and there is no need to scan each one of symbol before encoding. Encoded data needs less space as compare to Adaptive Huffman Encoding

7. Conclusion

Improved adaptive Huffman algorithm requires only single tree to compress the data instead of all tree required by Adaptive Huffman which takes lot of storage space. The Adaptive Huffman algorithm required to scan all the characters in a string at start to construct a tree where as Improved Adaptive Huffman start from first character which is time saving. It is further concluded that the encoded data of new Huffman algorithm is less than of adaptive and comparable to static.

8. References

[1]. D. A. Huffman, "A method for the construction of minimum-redundancy codes," Proc. IRE, vol. 40, pp. 1098-1101, 1952.

[2]. Knuth, D.E., “Dynamic Huffman Coding”, Journal of Algorithms 6,2(June), 1985.

[3]. Faller, N. “An adaptive system for data compression” In Record of the 7th Asilomar Conference on Circuits, Systems and

Computers (Pacific Grove, Calif., Nov.). Naval Postgraduate School, Monterey, Calif., pp. 593-597, 1973.

[4]. Gallager, R.G. "Variations on a theme by Huffman" IEEE Trans. Inf. Theory 24, 6 (Nov.), 668-674, 1978.

[5].Vitter, J.S. "Design and analysis of dynamic Huffman codes" J. ACM 34, 4 (Oct.), 825-845, 1987.

[6] Witten, I.H., Neal, R.M., and Cleary, J.G. "Arithmetic coding for data compression" Communications of the ACM, vol. 30, 520-540, 1987.

[7] Lelewer, D.A., Hirschberg D.S., "Data compression", ACM Computing Surveys 19,3 (Sept.): 261-266, 1987.

[8] Lempel, A., Ziv J., "A Universal Algorithm for Sequential Data Compression", IEEE Transaction on Information Theory 23,3 (May), 337-343, 1977.

[9] T.A Welch. "A technique for high-performance data compression" In IEEE Computer, 17:6:8–19, 1984.

[10] Cormack, G.V., and Horspool, R.N. "Algorithms for Adaptive Huffman Codes" Inform Process. Lett. 18, 3 (Mar.), 159-165, 1984.

[11] Moffat, A., Neal, R.M., and Witten, I.H. "Arithmetic coding revisited" ACM Transactions on Information Systems, vol. 16, 256-294, 1995.

[12] Ross Arnold and Tim Bell. "A corpus for the evaluation of lossless compression algorithms" In Data Compression Conference, pages 201{210. IEEE Computer Society Press, 1997.

[13] B. F. Varn, "Optimal variable length codes" Inform. Contr., vol. 19, pp. 289-301, 1971.

[14] A. Lempel and J. Ziv, "On the complexity of finite sequences," IEEE Trans. Inform. Theory, vol. IT-22, pp. 75-81, Jan. 1976.

[15] L. D. Davisson, "Universal noiseless coding," IEEE Trans. Inform. Theory, vol. IT-19, pp. 783-795, Nov. 1973.