Priority based LZW algorithm

Amit Jain
Department of Computer Engineering
NIT Kurukshetra ,India
amit014@gmail.com

Priyanka
Department of Computer Engineering
NIT Kurukshetra ,India
mannpammy@gmail.com

Vinay Goyal
Department of Computer Science
JIET,Jind,India
vinayeq@yahoo.com

***Abstract*— LZW is a popular and effective data compression algorithm for last many years. In this paper we proposed modified LZW on the basis of priority bit, termed as PLZW (Priority Based LZW). This replacement strategy helps us to avoid frequent flushing of the dictionary in LZW and increase in dictionary size.**

***Keywords*— Data Compression, LZW, Priority**

I. INTRODUCTION

Data compression Algorithms/techniques can be divided into two categories - Lossless data compression and Lossy data compression. In Lossless data compression, decompressed data is identical to the original uncompressed data. This type of compression scheme is adopted when storing software, text compression, spreadsheets, and word processing files. Lossy data compression is used, when a little loss of data/information is acceptable to user. Normally this technique is used for graphics images and digitized voice.

In their landmark papers in 1977 and 1978, Ziv and Lempel proposed two universal lossless data compression algorithms, which are called LZ77 and LZ78, respectively [1, 2]. Since then, many variants have been suggested such as Lempel-Ziv-Welch (LZW) [3]. Among the variants of the original LZ78, LZW designed by Welch in 1984, is perhaps the most famous and popular modification.LZW is a Directory-Based Lossless Data Compression Algorithm. Initially, all the alphabets are put into dictionary (0-256) . The LZW algorithm starts with a dictionary containing entries for each character in the alphabet. The algorithm scans the input matching it with entries in the dictionary. The matching is finished, whenever, we read from the input **a** string Y, not in the dictionary, such that Y=X.a, where X is a string already in the dictionary, "a" is a character and "." denotes the concatenation operation. The compression algorithm then sends the code for X (an index into the dictionary table) and inserts Y into the dictionary. The string Y is called a character extension of X. The encoding of the input continues from the character "a" that follows X. The decoder builds an identical dictionary to the one built by the encoder.

2. PROBLEMS WITH LZW ALGORITHM [5]

1. LZW algorithm works extremely well with repeated data streams or strings of English text, but if input data contain non repeated strings then dictionary get filled up frequently and dictionary is discarded very often, leading to degraded performance.
2. The limit imposed in the original LZW implementation by fact that once 4k dictionary is complete, no more strings can be added. Defining larger dictionary of course results in greater string capacity, but longer pointers reduce compression.
3. One puzzling thing about LZW is why the first 255 entries in 4k buffer are initialized to single character. There would be no point in setting pointers to single byte values. Since the pointers would be longer than byte values. 0-255 entries are reserved for standard character set. So initially when bytes appear in the encoded output, there is no compression rather single byte values are translated to 12-bit code size. If smaller text files containing large entries of the standard character set, then expansion of the data take place in spite of compression by assignment of 12 bit index to 8 bits data value and thus more bits are transferred over the communication channel. II.

3. PROPOSED SCHEME

Replacement strategy works good when we limited resources either in terms of time and space.We need a replacement strategy when dictionary filled completely and there is no space to add a new string in dictionary then we have following options:-

(1) Discard whole dictionary and start with a new one.
(2) Constantly monitor compression ratio, if it falls below a certain level, increase code size by one bit.
(3) Apply some replacement strategy to discard entries and provide space to the new one.

In this paper we use last strategy to further modify the LZW as first strategy is not very much practical and second strategy increases the dictionary size. So it is advantageous to replace old entry with new one. So our proposed scheme tries to remove less important dictionary entry at every stage of compression.

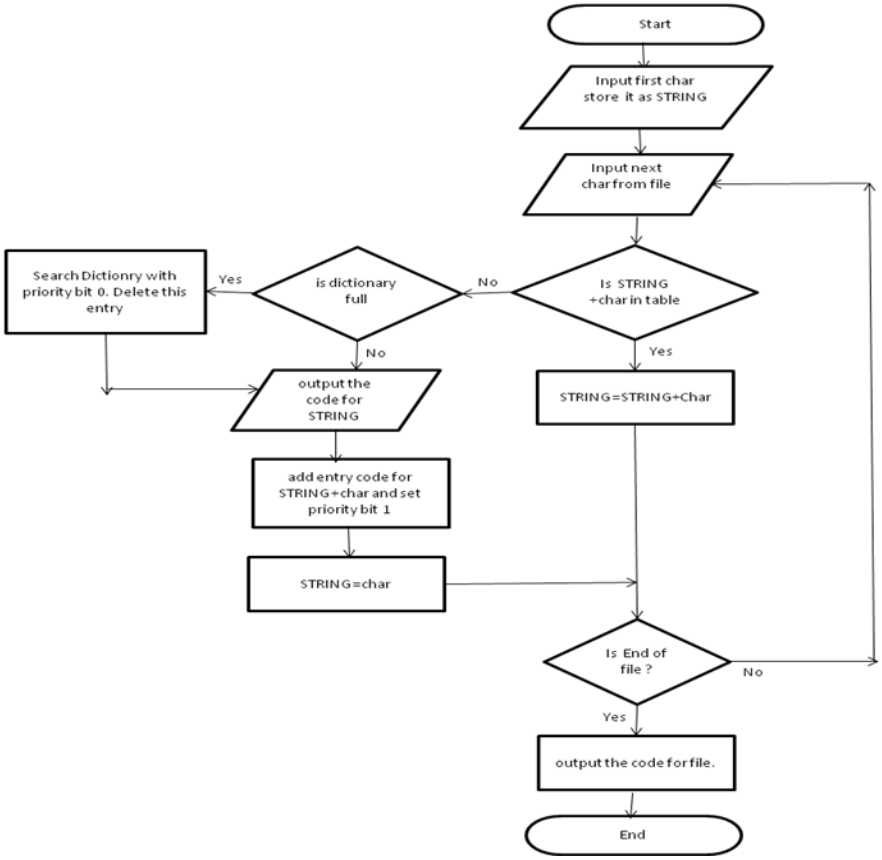

Fig 1

A. Replacement Based On priority

In this algorithm, we use a priority bit with dictionary entries. These bits can be either zero or one.Initially all the entry are given priority bits '0'. As a dictionary entry used more than one time it update priority bit from '0' to '1'.When our dictionary is full ,there is no space to add new entry ,we start scanning of dictionary from first entry. The entry in dictionary with priority bit '0' is deleted and new entry can be added to dictionary .The following steps will be taken to implement algorithm

Algorithm:

(1) *Input first character from file and store it as STRING*
(2) Input next character from file.
(3) Check whether this STRING+ character is in the dictionary. If yes then store it in STRING and go to step 2.If No then check if the dictionary is full.
(4) If dictionary is full then check dictionary for priority bit '0',delete this entry and output the code for string. Add entry of STRING +char in table and set priority '1'and store char in STRING.
(5) If dictionary is not full then output the code for string. Add entry of STRING +char in table and set priority '1'and store char in STRING.
(6) When EOF reach output the code for file.

The main benefit of this algorithm is that it removes less important entry from the dictionary .The flow chart is shown in figure 1

Advantages

The main advantage of using this algorithm is we need not to expand the size of dictionary. When dictionary filled completely we can replace less important entries with a new entries with better priorities. This helps us to restrict the dictionary size with in a limit and use less memory.

4. CONCLUSION

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

References

[1] J.Ziv, and A Lempel, "A universal algorithm for sequential data compression," IEEE Trans. on Information Theory, vol 23, pp. 337-343,May 1977.

[2] J. Ziv and A. Lempel, "Compression of individual sequences via variable length coding," IEEE Trans. Inf Theory, vol 24, pp. 530-536,1978.

[3] Terry Welch, "A technique for high performance data compression, "IEEE Computer, vol 17, pp. 8-19, June 1984.

[4] Dwane Phillips, "LZW Data Compression, " Circuit Cellar INK – The Computer Applications Journal, pp. 36-48, June/July 1992.

[5] Parvinder Singh, Sudhir Batra, and HR Sharma, "Evaluating the performance of message hidden in 1 st and 2nd bit plane", WSEAS Trans.on Information Science and Applications, issue 8, vol 2, pp. 1220-1227,August 2005

[6] R. N. Horspool, and G. V. Cormack, "Construction word-based text compression algorithms," in 2nd IEEE Data Compression Conference,Snowvird, 1992.

[7] Tong Lai Yu, "Data compression for PC software distribution,Software Practice and Experience, vol 26, pp. 1181-1195, November 1996.