LZW Compression

Ramana Kumar Kundella

Indiana State University

rkundella@sycamores.indstate.edu

December 13, 2014

Abstract

LZW is one of the well-known lossless compression methods. Since it has several remarkable features such that coding is simple, prior analysis on source is unnecessary, and the whole code table is not sent to its decoder, LZW is widely used in many applications. GIF, TIFF and PDF (compressing images) are good examples of them. In spite of its reputation on compression, however, few data hiding methods are directly applied to LZW itself. This may be due to few redundancies remained in losslessly compressed data: therefore there is not enough available room for data hiding. The existing methods are lossy, however, lossless approach is preferred to the lossy one. In this paper, we propose the DH-LZW method that embeds data to source data in a lossless manner. Through this paper, modifiable elements of LZW for data hiding are introduced and a way to handle them is proposed. Experiments show very promising results.

1 Introduction

Lempel Ziv Welch (LZW) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published by Welch in 1984 as an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978. The algorithm is simple to implement, and has the potential for very high throughput in hardware implementations.[1] It was the algorithm of the widely used Unix file compression utility compress, and is used in the GIF image format.

- Uncompressed data can take up a lot of space, which is not good for limited hard drive space and internet download speeds.
- While hardware gets better and cheaper, algorithms to reduce data size also helps technology evolve.
- One minute of uncompressed hd video can be over 1GB. How can we fit a two-hour filmon a 25 GB bluray dics? .

The Lempel Ziv Welch algorithm is one of many algorithms used for compression. It is typically used to compress certain image files, unix compress command, among other uses. It is lossless, meaning no data is lost when compressing. The idea: rely on reoccuring patterns to save data space.

For Example Consider ASCII code. Typically, every character is stored with 8 binary bits, allowing up to 256 unique symbols for the data. This algorithm tries to extend this library to 9-12bits per character. The new unique symbols are made up of combinations of symbols that occurred previously in the string. Does not always compress well, especially with short, diverse strings. But is good for compressing redundant data, and does not have to save the new dictionary with the data: this method can both compress and uncompress data.

1.1 Our Results

We try to extend the library taking ASCII code as example achieving both compression and decompression to any set of input file to either character or number correspondingly.

1.2 Organization

In Section ?? explains how the encoding part of code works. In Section ?? explains decoding code of the algorithm. In Section ?? explain the algorithm, encoding and decoding with examples. In Section ?? we discuss future work on this project.

2 Encoding Code

http://cs.indstate.edu/ rkundella/encoding.java

```
import java.util.*;
import java.io.*;
public class HelloWorld  {

         public static void main(String []args) throws Exception {
             int HVALUE = 27,l=0,m=0,z=0,g=0,i,j=0,k,c=0,p=2;;
         String st;
         //char []

         String sCurrentLine;
         BufferedReader br;
         br = new BufferedReader(new FileReader("D:\\testing.txt"));

         while ((st = br.readLine()) != null) {
           System.out.println(st);

         word[] w = new word[20];
         k=j+1;

         for(j=0;j<20;j+=(p-1))                      //If Given Sting is RamananaRamanakumarkukund
         {
             w[j] = new word();                     //reading values to objects of word.
             w[j].s = st.substring(j,j+2);             //reads 2 values w[o].s = Ra, w[1].s = am, .
         }


         p=2;

         do
         {
```

```
        for(j=2;j<20;j+=(p-1))
         {

             for(i=0;i<j;i++)        // checks for repetition in the assigned dictionary
             {                         //if you see Ra,am,ma,an,na,an,na* which is repetition

             if((w[i].s).equals(w[j].s)) // condition succeeds if repetition occurs
               {

                 w[j].s = st.substring(j+g,j+(p+1+g)); // assigns new value for repetition
               w[j].value = HVALUE;                       //means na is repeated so it gives naR n
               m = j+1;
               l=j+p+g;

               for(z=(j+1);z<20;z++)
               {
            w[z].s=st.substring(l,l+2);  // assigns new value to the right of repetition
                                    // so after giving naR our next value should start from R not
              l++;
               }

               g++;
             }
             }
             }
         p++;
          }while(p<5);

          for(j=0;j<20;j++)
         {
             System.out.println("");
           System.out.print(w[j].s);
             System.out.print(" ");
              w[j].value=HVALUE;              // assigns numerical value to objects
            System.out.println(w[j].value);
             HVALUE++;
         }

}
}
}
 class word{
    String s = new String();
    int value;
    }
```

2.1 Decoding Code

http://cs.indstate.edu/ rkundella/decoding.java

```
import java.util.*;
import java.io.*;
public class HelloWorld  {


 public static void main(String []args)  throws Exception{
            int HVALUE = 27,l=0,m=0,z=0,g=0,i=0,j=0,k,c=0,p=2,alt=0;
        String st;

  //       File file = new File("D:\\testing2.txt");
     //    file.createNewFile();



        BufferedReader br;
        BufferedWriter writer;
        br = new BufferedReader(new FileReader("D:\\testing.txt"));
        writer = new BufferedWriter(new FileWriter("D:\\testing2.txt"));

        while ((st = br.readLine()) != null) {
          System.out.println(st);

        word[] w = new word[20];
        k=j+1;

        for(j=0;j<20;j+=(p-1))
        {
            w[j] = new word();                    //reading values to objects of word.
            w[j].s = st.substring(j,j+2);
        }


        p=2;

        do
        {

       for(j=2;j<20;j+=(p-1))
        {

            for(i=0;i<j;i++)        // checks for repetition in the assigned dictionary
            {

            if((w[i].s).equals(w[j].s)) // condition succeeds if repetition occurs
              {

                w[j].s = st.substring(j+g,j+(p+1+g)); // assigns new value for repetition
              w[j].value = HVALUE;
              m = j+1;
              l=j+p+g;
```

```
      for(z=(j+1);z<20;z++)
      {
   w[z].s=st.substring(l,l+2); // assigns new value to the right of repetition

     l++;
      }



      g++;
    }
    }
    }
p++;
 }while(p<5);






 for(j=0;j<20;j++)
{
    System.out.println("");
  System.out.print(w[j].s);
    System.out.print(" ");
     w[j].value=HVALUE;             // assigns numerical value to objects
   System.out.println(w[j].value);
   writer.write(w[j].s+" "+w[j].value);
   writer.newLine();

    HVALUE++;
}
 writer.write(w[0].s.substring(0,1));
 for(i=0;i<20;i++)
 {

writer.write(w[i].s.substring(1,w[i].s.length()));
System.out.print(w[i].s.substring(1,w[i].s.length()));
alt++;
if(alt==2)
{
alt = 0;
}

 }

}
           writer.close();
```

```
}
}
 class word{
    String s = new String();
    int value;
    }
```

3 The Algorithm

The scenario described by Welch's 1984 paper[1] encodes sequences of 8-bit data as fixed-length 12-bit codes. The codes from 0 to 255 represent 1-character sequences consisting of the corresponding 8-bit character, and the codes 256 through 4095 are created in a dictionary for sequences encountered in the data as it is encoded. At each stage in compression, input bytes are gathered into a sequence until the next character would make a sequence for which there is no code yet in the dictionary. The code for the sequence (without that character) is added to the output, and a new code (for the sequence with that character) is added to the dictionary.

The idea was quickly adapted to other situations. In an image based on a color table, for example, the natural character alphabet is the set of color table indexes, and in the 1980s, many images had small color tables (on the order of 16 colors). For such a reduced alphabet, the full 12-bit codes yielded poor compression unless the image was large, so the idea of a variable-width code was introduced: codes typically start one bit wider than the symbols being encoded, and as each code size is used up, the code width increases by 1 bit, up to some prescribed maximum (typically 12 bits).

Further refinements include reserving a code to indicate that the code table should be cleared (a "clear code", typically the first value immediately after the values for the individual alphabet characters), and a code to indicate the end of data (a "stop code", typically one greater than the clear code). The clear code allows the table to be reinitialized after it fills up, which lets the encoding adapt to changing patterns in the input data. Smart encoders can monitor the compression efficiency and clear the table whenever the existing table no longer matches the input well.

Since the codes are added in a manner determined by the data, the decoder mimics building the table as it sees the resulting codes. It is critical that the encoder and decoder agree on which variety of LZW is being used: the size of the alphabet, the maximum code width, whether variable-width encoding is being used, the initial code size, whether to use the clear and stop codes (and what values they have). Most formats that employ LZW build this information into the format specification or provide explicit fields for them in a compression header for the data.

Encoding:

A high level view of the encoding algorithm is shown here:

Initialize the dictionary to contain all strings of length one. Find the longest string W in the dictionary that matches the current input. Emit the dictionary index for W to output and remove W from the input. Add W followed by the next symbol in the input to the dictionary. Go to Step 2.

A dictionary is initialized to contain the single-character strings corresponding to all the possible input characters (and nothing else except the clear and stop codes if they're being used). The algorithm works by scanning through the input string for successively longer substrings until it finds one that is not in the dictionary. When such a string is found, the index for the string without the last character (i.e., the longest substring that is in the dictionary) is retrieved from the dictionary and sent to output, and the new string (including the last character) is added to the dictionary with the next available code. The last input character is then used as the next starting point to scan for substrings.

In this way, successively longer strings are registered in the dictionary and made available for subsequent encoding as single output values. The algorithm works best on data with repeated patterns, so the initial parts of a message will see little compression. As the message grows, however, the compression ratio tends asymptotically to the maximum.

Decoding:

The decoding algorithm works by reading a value from the encoded input and outputting the corresponding string from the initialized dictionary. In order to rebuild the dictionary in the same way as it was built during encoding, it also obtains the next value from the input and adds to the dictionary the concatenation of the current string and the first character of the string obtained by decoding the next input value, or the first character of the string just output if the next value can not be decoded (If the next value is unknown to the decoder, then it must be the value that will be added to the dictionary this iteration, and so its first character must be the same as the first character of the current string being sent to decoded output). The decoder then proceeds to the next input value (which was already read in as the "next value" in the previous pass) and repeats the process until there is no more input, at which point the final input value is decoded without any more additions to the dictionary.

In this way the decoder builds up a dictionary which is identical to that used by the encoder, and uses it to decode subsequent input values. Thus the full dictionary does not need be sent with the encoded data; just the initial dictionary containing the single-character strings is sufficient (and is typically defined beforehand within the encoder and decoder rather than being explicitly sent with the encoded data.)

4 My Example

```
COMPRESSION:
I want to compress the following string:
                               thisisthe
h104                 i104                    h104                                       hi257
i105                  s115                   i105                                        is258
s115                  i105                   s115                                       si259
i105                  s115                   isis in the dictionary,check if ist is.
is258                 t116                   is258                                      ist260
t116                  h104                    th is in dictionary,check if the is.
th256                e101                     th256                                     the261
e101                   --                          e101
             116   104    105     115     258     256     101


  thisisthe
     116     104      105      115        258       256        101

Now our symbols are made up of 9 bits each instead of 8.
So this is 7*9= 63 bits long, instead of original 72 bits.
Thats 87.5% of what it used to be.


DECOMPRESSION:
```

```
To uncompress, we do need to know how many bits were used in the compression, the one new informat
We can read in the characters and build the new dictionary the same way as we did for compressing.

I want to decompress the following string:
           116  104  105  115  258  256  101

Current                    Next                       Output                          Add to Dicti
 104                        105                        104                             104 105(257)
 105                        115                        105                             105 115(258)
  115                         258                        115                         115 105 115(259
  258                        256                       105 115                       105 115 116(260
  256                      101                       116 104                          116 104 101(26

116  104  105  115  105  115  116  104  101
Or  it is thisisthe
```

5 Example:

http://en.wikipedia.org/wiki/LempelZivWelchExample verbatin

6 Our Contribution

Everyone should be aware there is compression happening/needed everywhere maybe in filming or to limit hard drive space so it helps speeding up the system/internet or any other scenario. Even though the hardware is improving and getting cheaper everyday, still we need some compression algorithms for technology to evolve

References

LZW Compression
http://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch
Example
https://www.youtube.com/watch?v=j2HSd3HCpDs
Substrings
http://en.wikipedia.org/wiki/Substring

7 Future Work

My future work on this project, I would like to work on my code for working on string with only one repeated values(like aaaaaaaaaaaaaaaaaaaaaaaa) and would like to crate every attribute as shown n my example.

Acknowledgements

This paper work is written with the help of information gathered while working on the project through internet. The data presented, examples discussed are solely responsible of the author.