

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**April 2018**

**LZ77 Compression**

**A Project Report**

**Under the Guidance of,**

**Prof. A.Srivani**

**By**

**17BCE2014-Samath Nair**

**17BCE0267 -Sameeran Bandishti**

**17BCE0914-Aryan Jain**

DECLARATION BY THE CANDIDATE

We hereby declare that the project report entitled **“LZ77 Compression”** submitted by us to VIT University, Vellore in partial fulfilment of the requirement for the award of the degree of **B. Tech (CSE)** is a record of J- component of project work carried out by us under the guidance of **Prof. A.Srivani**. We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : VIT University, Vellore

Date : 6th April, 2018

Signature of the faculty Signature of the Candidates

TABLE OF CONTENTS

1.Introduction

1.1Abstract

1.2Background

2.Overview and planning

2.1Proposed work

2.2Hardware requirement

2.3Software requirement

3. Literature survey and review

3.1Literature summary

4.Methodology

4.1 Methodology used

4.2 Applications

5. System Implementation

5.1 Code

5.2 Results and discussion

6. Conclusion

6.1 Conclusion

7. References

1. Introduction

1.1 Abstract

Data Compression is defined as the science and art of the representation of information in a crisply condensed form. . For decades, Data compression is considered as critical technologies for the ongoing digital multimedia revolution .There are variety of data compression algorithms which are available to compress files of different formats. This paper provides a survey of different basic lossless data compression algorithms such as LZ77 and LZ78.

1.2 Background

Original method of sliding window was designed in 1977 by Abrahamem Lempel and Jacob Ziv. It forms a basis for LZ variations.

2. Overview and Planning

2.1 Proposed Work

Compress a fixed string using LZ77 Compression and then transfer it from one Arduino to another using Serial communication. Calculate the transfer times for both compressed and uncompressed strings.

2.2 Hardware Requirements

Arduino chip X 2

Male to Male Jumper Wires X 3

Arduino Cables. X 2

Laptops X 2

2.3 Software Requirements

C++

Arduino IDE

3. Literature Survey and Review

3.1 Literature Summary

LZ77 algorithms achieve compression by replacing repeated occurrences of data with references to a single copy of that data existing earlier in the uncompressed data stream. A match is encoded by a pair of numbers called a length-distance pair, which is equivalent to the statement "each of the next length characters is equal to the characters exactly distance characters behind it in the uncompressed stream".

To spot matches, the encoder must keep track of some amount of the most recent data, such as the last 2 kB, 4 kB, or 32 kB. The structure in which this data is held is called a sliding window, which is why LZ77 is sometimes called [sliding-window compression](https://en.wikipedia.org/w/index.php?title=Sliding-window_compression&action=edit&redlink=1). The encoder needs to keep this data to look for matches, and the decoder needs to keep this data to interpret the matches the encoder refers to. The larger the sliding window is, the longer back the encoder may search for creating references.

It is not only acceptable but frequently useful to allow length-distance pairs to specify a length that actually exceeds the distance. As a copy command, this is puzzling: "Go back fourcharacters and copy ten characters from that position into the current position". How can ten characters be copied over when only four of them are actually in the buffer? Tackling one byte at a time, there is no problem serving this request, because as a byte is copied over, it may be fed again as input to the copy command. When the copy-from position makes it to the initial destination position, it is consequently fed data that was pasted from the beginning of the copy-from position. The operation is thus equivalent to the statement "copy the data you were given and repetitively paste it until it fits". As this type of pair repeats a single copy of data multiple times, it can be used to incorporate a flexible and easy form of [run-length encoding](https://en.wikipedia.org/wiki/Run-length_encoding).

4. Methodology

4.1 Method Used

This method uses window divided to search buffer and look-ahead buffer. Size of the search buffer is usually 8 192 bits and size of the look-ahead buffer about 10 to 20 bits. In demonstration both parameters can be set.

The algorithm can be described as follows. First the longest prefix of a look-ahead buffer that starts in search buffer is found. This prefix is encoded as triplet (i, j, X) where i is the distance of the begining of the found prefix from the end of the search buffer, j is the length of the found prefix and X is the first character after the prefix in look-ahead buffer. The following applet visualizes this algorithm. The number of bits written to the output depends on used encoding of numbers.

4.2 Applications

Lz77 is a lossless data compression algorithm.

LZ77 compression became the first widely used universal data compression method on computers. A large [English](https://en.wikipedia.org/wiki/English_language) text file can typically be compressed via LZW to about half its original size.

LZ77 was used in the public-domain program [compress](https://en.wikipedia.org/wiki/Compress), which became a more or less standard utility in [Unix](https://en.wikipedia.org/wiki/Unix) systems around 1986. It has since disappeared from many distributions, both because it infringed the LZW patent and because [gzip](https://en.wikipedia.org/wiki/Gzip" \o "Gzip) produced better compression ratios using the LZ77-based [DEFLATE](https://en.wikipedia.org/wiki/DEFLATE) algorithm, but as of 2008 at least FreeBSD includes both [compress](https://en.wikipedia.org/wiki/Compress) and [uncompress](https://en.wikipedia.org/wiki/Uncompress" \o "Uncompress) as a part of the distribution. Several other popular compression utilities also used LZW or closely related methods.

LZ77 became very widely used when it became part of the [GIF](https://en.wikipedia.org/wiki/Graphics_Interchange_Format) image format in 1987. It may also (optionally) be used in [TIFF](https://en.wikipedia.org/wiki/TIFF) and [PDF](https://en.wikipedia.org/wiki/PDF) files. (Although LZW is available in [Adobe Acrobat](https://en.wikipedia.org/wiki/Adobe_Acrobat_Reader) software, Acrobat by default uses [DEFLATE](https://en.wikipedia.org/wiki/DEFLATE) for most text and color-table-based image data in PDF files.)

5. System Implementation

5.1 Code

1. Encoder

#include<string.h>

using namespace std;

#include<SoftwareSerial.h>

SoftwareSerial myser(10,11);

int repeat=0;

void setup() {

// put your setup code here, to run once:

Serial.begin(9600);

myser.begin(9600);

}

void loop() {

char str[13]="abcdeabcdea";

char sb[10]={'\0'},lb[10]={'\0'};

int i,j,c=0,k=0,m,l,len,flag;

int atpos[50];

int numrep[50];

char rep[10];

int count=0;

if(k==0&& repeat==0)

{

Serial.println("Input 1 for Without Compression\nInput 2 for With Compression");

}

while(!Serial.available())

{

}

k=Serial.parseInt();

if(k==2)

{

char sb[10]={'\0'},lb[10]={'\0'};

int i,j,c=0,k=0,m,l,len,flag;

Serial.println("Given string is: ");

Serial.println(str);

for(i=0;i<5;i++)

{

lb[i]=str[i];

}

for(i=0;i<11;i++)

{

str[i]=str[i+5];

}

for(m=0;m<6;m++)

{

i=0;

j=0;

flag=0;

len=strlen(sb);

Serial.print("\nlength of sb : ");

Serial.print(len);

while(flag==0 && i<len)

{

if(sb[i]==lb[0])

{

flag=1;

for(l=0;l<len;l++)

{

if(sb[l]==lb[0])

{

if(sb[l+1]!=lb[1] && sb[i+1]==lb[1])

{

c=i;

}

else

{

c=l;

}

}

}

if(c>0)

{

for(l=c;l<len;l++)

{

if(sb[l]==lb[j])

j++;

}

}

else

{

for(l=0;l<strlen(lb);l++)

{

if(sb[l]==lb[j])

j++;

}

}

}

else

{

i++;

}

}

Serial.print("\n<");

Serial.print(c);

atpos[count]=c;

Serial.print(",");

Serial.print(j);

numrep[count]=j;

Serial.print(",");

Serial.print(lb[j]);

rep[count]=lb[j];

count++;

Serial.println(">");

if(k==7)

{

for(i=0;i<7;i++)

{

if(strlen(lb)>j)

{

sb[i]=sb[j+i+1];

}

else

sb[i]=sb[j+i];

}

if(strlen(lb)>j)

{

for(i=0;i<j+1;i++)

{

sb[k-j-1]=lb[i];

k++;

}

}

else

{

for(i=0;i<j+1;i++)

{

sb[k-j]=lb[i];

k++;

}

}

}

else

{

for(i=0;i<j+1;i++)

{

sb[k]=lb[i];

k++;

}

}

for(i=0;i<(4-j);i++)

{

lb[i]=lb[i+1+j];

}

for(l=0;l<j+1;l++)

{

lb[i]=str[l];

i++;

}

if(str!='\0')

{

for(l=0;l<13;l++)

{

str[l]=str[l+1+j];

}

for(l=13;l>(13-j);l--)

{

str[i]='\0';

}

}

Serial.print("\n In sb: ");

Serial.println(sb);

Serial.print("\t\t In lb: ");

Serial.println(lb);

Serial.print("\t\t In str: ");

Serial.println(str);

}

Serial.println("This is the Final Compressed String");

for(i=0;i<count-1;i++)

{

Serial.print("<");

Serial.print(atpos[i]);

Serial.print(",");

Serial.print(numrep[i]);

Serial.print(",");

Serial.print(rep[i]);

Serial.print(">");

}

Serial.print("<");

Serial.print(atpos[count-1]);

Serial.print(",");

Serial.print(numrep[count-1]);

Serial.print(",");

Serial.print(rep[count-1]);

Serial.println(">");

delay(5000);

for(i=0;i<count;i++)

{

myser.print(atpos[i]);

myser.print(numrep[i]);

myser.print(rep[i]);

}

repeat=1;

}

else

if(k==1)

{

myser.print("abcdeabcdea");

}

}

1. Receiver

int led=13;

char inChar;

byte index = 0;

void setup() {

// put your setup code here, to run once:

pinMode(led,OUTPUT);

Serial.begin(57600);

}

void loop() {

// put your main code here, to run repeatedly:

char s[20];

long int tstart;

long int tend;

while(Serial.available() > 0)

{

if(index < 19)

{

inChar = Serial.read();

s[index] = inChar;

index++;

s[index] = '\0';

}

}

if(s!="")

{

Serial.println("Received String");

Serial.println(s);

Serial.println("Time Required");

Serial.println(tend-tstart);

}

}

1. The C++ Implementation

#include<iostream.h>

#include<conio.h>

#include<string.h>

using namespace std;

int main()

{

char str[13]="abracadabra";

char sb[10]={'\0'},lb[10]={'\0'};

int i,j,c=0,k=0,m,l,len,flag;

strcpy(str,"abracadabra");

cout<<"Given string is: "<<str;

for(i=0;i<5;i++)

{

lb[i]=str[i]; // store first 5 characters in look ahead buffer

}

for(i=0;i<11;i++)

{

str[i]=str[i+5]; // move remaining characters to the front of the string

}

for(m=0;m<6;m++)

{

i=0;

j=0;

flag=0;

len=strlen(sb);

cout<<"\nlength of sb : "<<len;

while(flag==0 && i<len)

{

if(sb[i]==lb[0]) // checking for matching characters

{

flag=1;

for(l=0;l<len;l++)

{

if(sb[l]==lb[0]) // checking for matching characters - finding first character

{

if(sb[l+1]!=lb[1] && sb[i+1]==lb[1])

{

c=i;

}

else

{

c=l;

}

}

}

if(c>0)

{

for(l=c;l<len;l++) // checking for matching characters - checking how many of them are repeated

{

if(sb[l]==lb[j])

j++;

}

}

else

{

for(l=0;l<strlen(lb);l++)

{

if(sb[l]==lb[j])

j++;

}

}

}

else

{

i++;

}

}

cout<<"\n<"<<c<<","<<j<<","<<lb[j]<<">"; // printing resultant compression

if(k==7) // transferring to search buffer - max size is 7 (k = currently filled)

{

for(i=0;i<7;i++)

{

if(strlen(lb)>j)

{

sb[i]=sb[j+i+1]; // delete the oldest few characters from the search buffer

}

else

sb[i]=sb[j+i];

}

if(strlen(lb)>j)

{

for(i=0;i<j+1;i++)

{

sb[k-j-1]=lb[i]; // clear out last j spaces to put in j more from look ahead buffer

k++;

}

}

else

{

for(i=0;i<j+1;i++)

{

sb[k-j]=lb[i];

k++;

}

}

}

else

{

for(i=0;i<j+1;i++)

{

sb[k]=lb[i]; // adding to search buffer

k++;

}

}

for(i=0;i<(4-j);i++)

{

lb[i]=lb[i+1+j]; // j is the position of last matched character - removing that many elements from look ahead buffer

}

for(l=0;l<j+1;l++)

{

lb[i]=str[l]; // adding more elements from the string to look ahead buffer

i++;

}

if(str!='\0') // when string has not become empty - bring remaining characters to the front

{

for(l=0;l<13;l++)

{

str[l]=str[l+1+j];

}

for(l=13;l>(13-j);l--)

{

str[i]='\0';

}

}

cout<<"\n In sb: "<<sb;

cout<<"\t\t In lb: "<<lb;

cout<<"\t\t In str: "<<str;

}

return 0;

}

5.2 Results and discussion

It is clear that the time taken to transfer the compressed string is roughly 1000 microseconds lesser than that to transfer the uncompressed one. This is partly because the transfer time of numbers is less than that of characters. Though this is not properly implementable using serial communication, the difference is still appreciable.

Output:



The first string is the uncompressed form and it takes 1860 microseconds.

The 2 following string 0 is a buffer input.

The following 5 strings that amount to a total of 644 microseconds are the compressed form.

6. Conclusion

6.1 Conclusion

The transfer rate of the string is greater after the string is compressed.

7. References

<http://www.stringology.org/DataCompression/lz77/index_en.html>

<https://msdn.microsoft.com/en-us/library/ee916854.aspx>

<https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch#Uses>

<https://www.arduino.cc/>