LZW Compression

Outline

- Theory part of LZW compression/decompression
- About assignment 2 and some details

Introduction

- What is LZW?
  - Lossless compression method
  - Lempel-Ziv-Welch
  - Based on LZ78

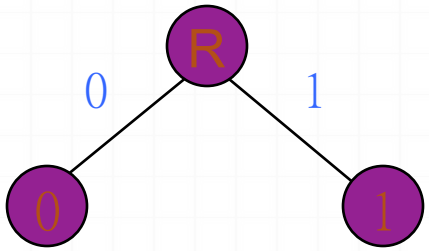
Basic idea

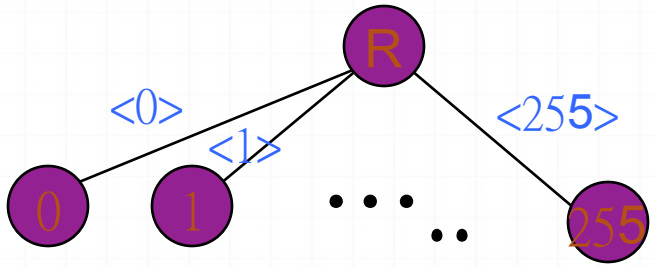
- Assume repetition of phase usually occurs
- Use a code to represent one phase
- Build a dictionary of phases that we met
  - If a phase is found in dictionary, use the code
  - If not found, add it to dictionary and give it a code
- Very high compression ratio if lots of repetition

Skip

Algorithm

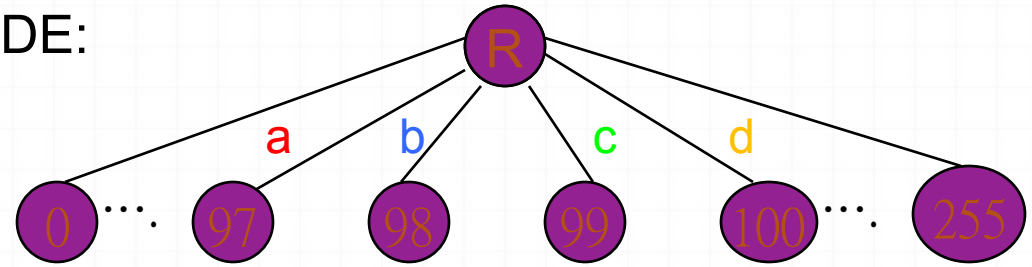
- But how to handle byte steam?
- The algorithm is similar to lecture note!
- However, each node have maximum number of 256 child nodes.





Tree Structure Example

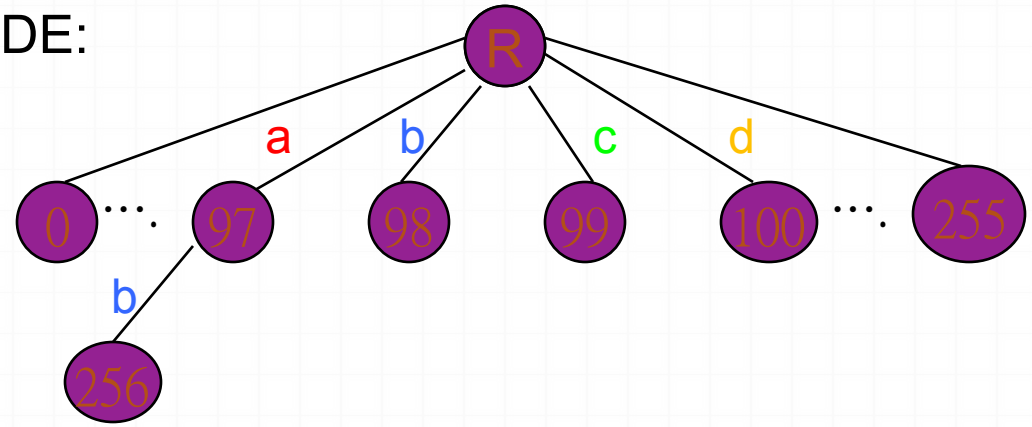
ENCODER SIDE:

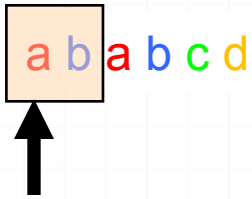


a b a b c d

Tree Structure Example

ENCODER SIDE:

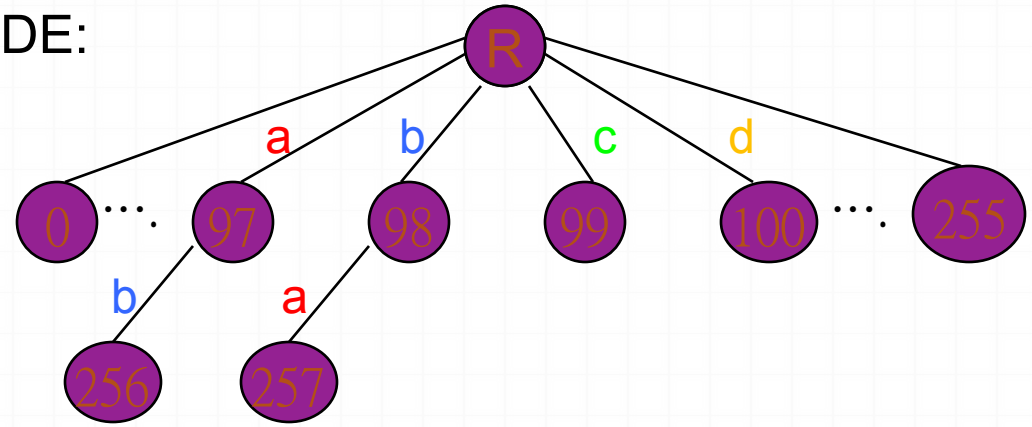


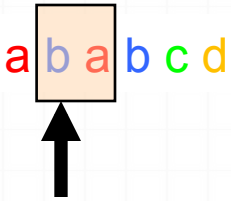


output: 97,

Tree Structure Example

ENCODER SIDE:

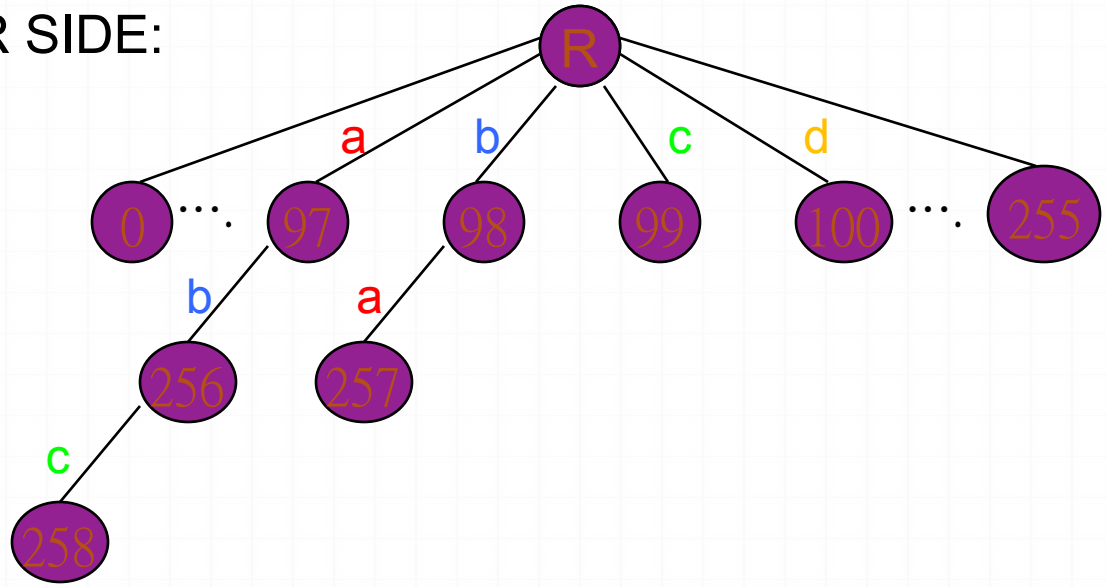


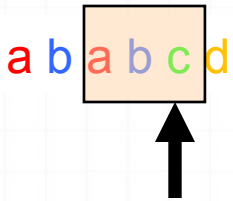


output: 97,98

Tree Structure Example

ENCODER SIDE:

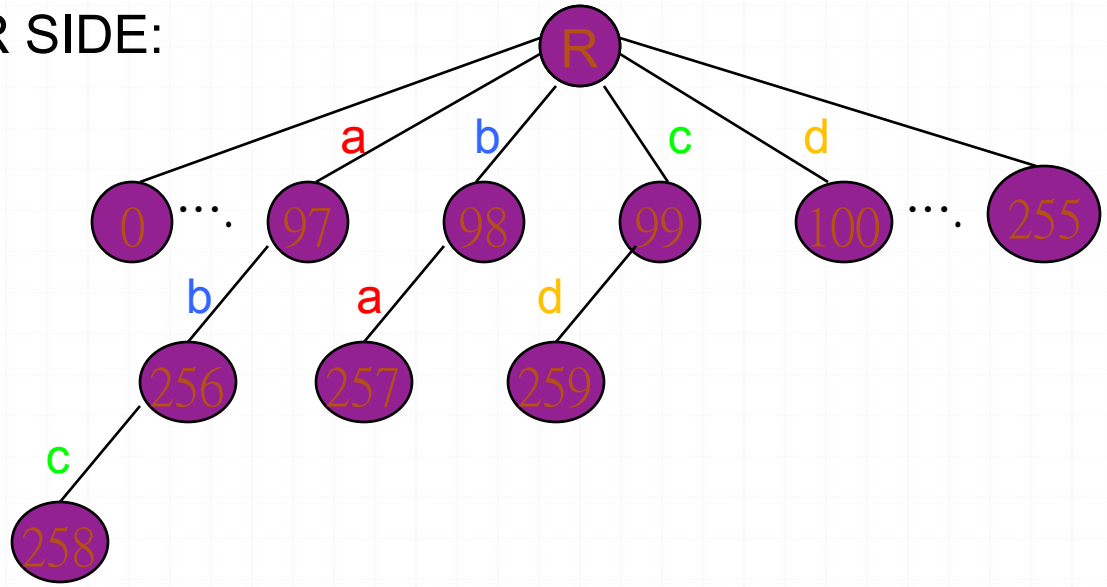


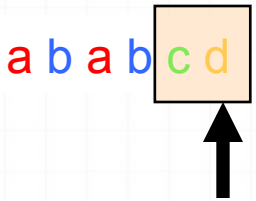


output: 97,98,256

Tree Structure Example

ENCODER SIDE:

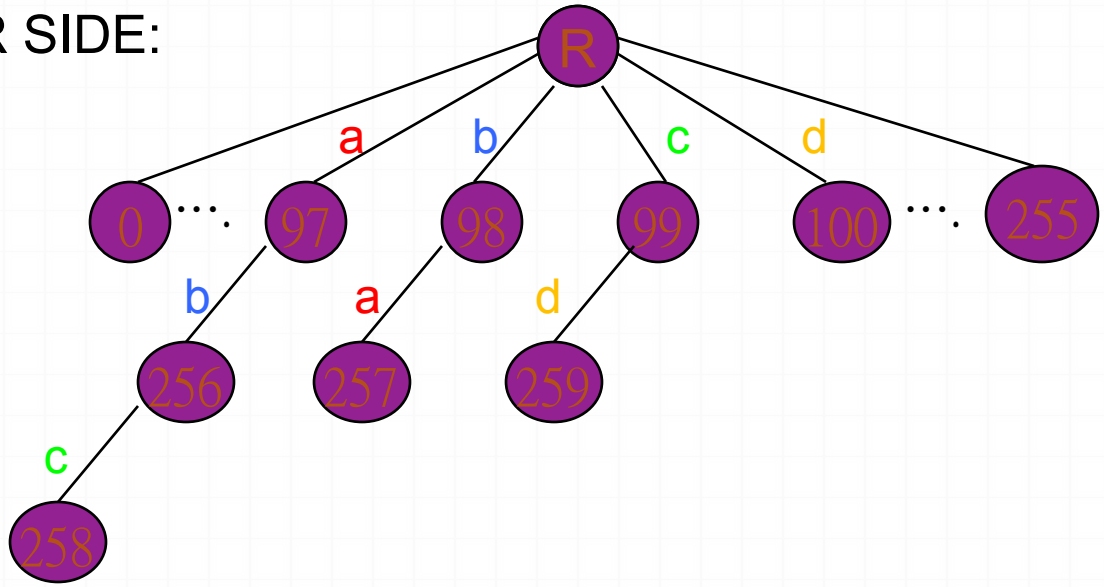




output: 97,98,256,99

Tree Structure Example

ENCODER SIDE:



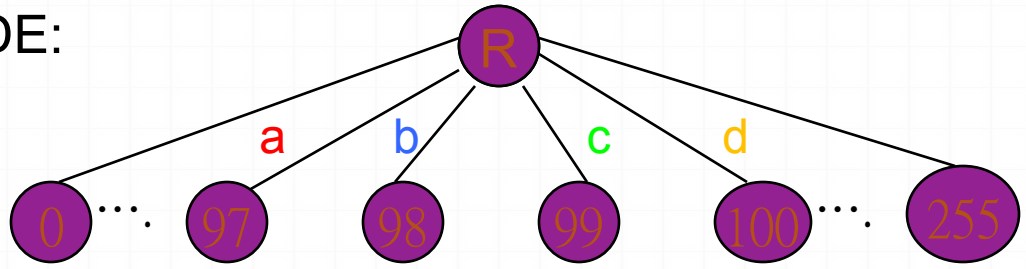
a b a b c d

output: 97,98,256,99,100

Encode complete!

Tree Structure Example

DECODE SIDE:

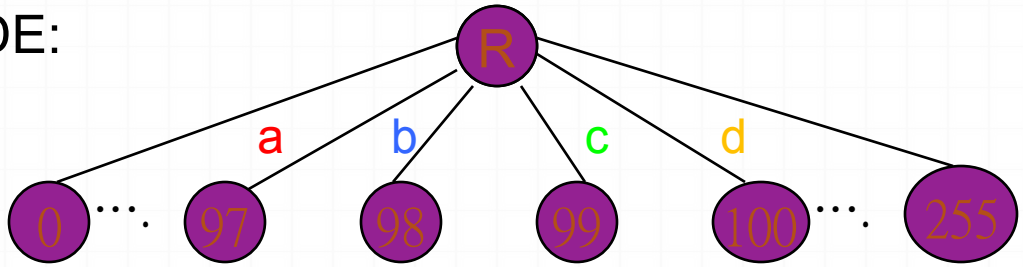


input: 97,98,256,99,100

output:

Tree Structure Example

DECODE SIDE:

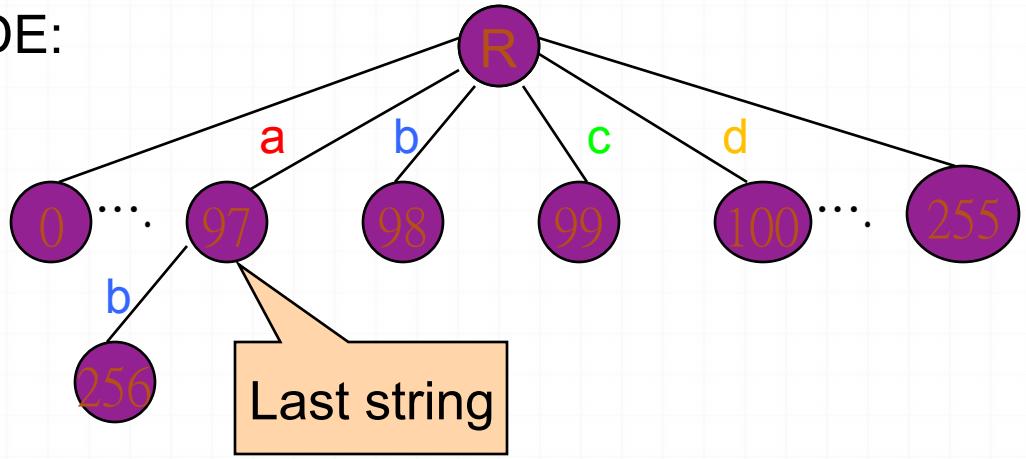


input: 97,98,256,99,100

output: a

Tree Structure Example

DECODE SIDE:



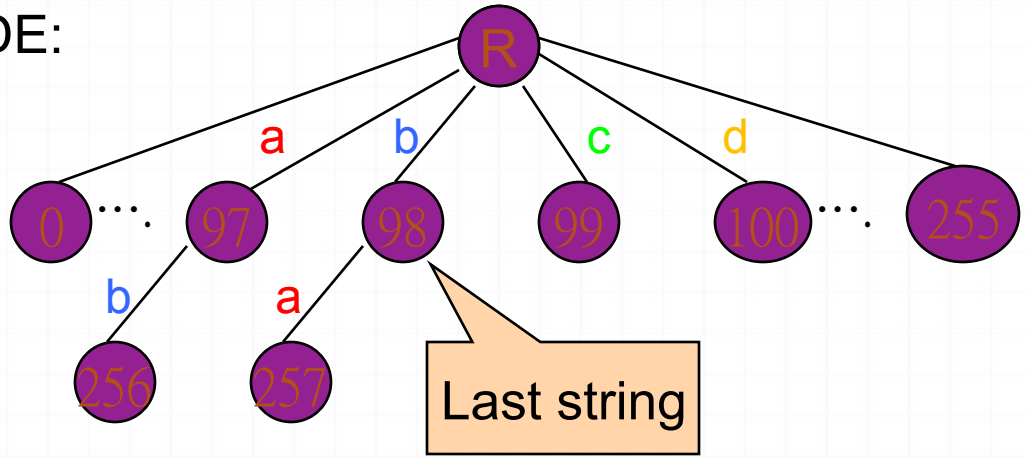
input: 97,98,256,99,100

output: a b

Last string = a

Tree Structure Example

DECODE SIDE:



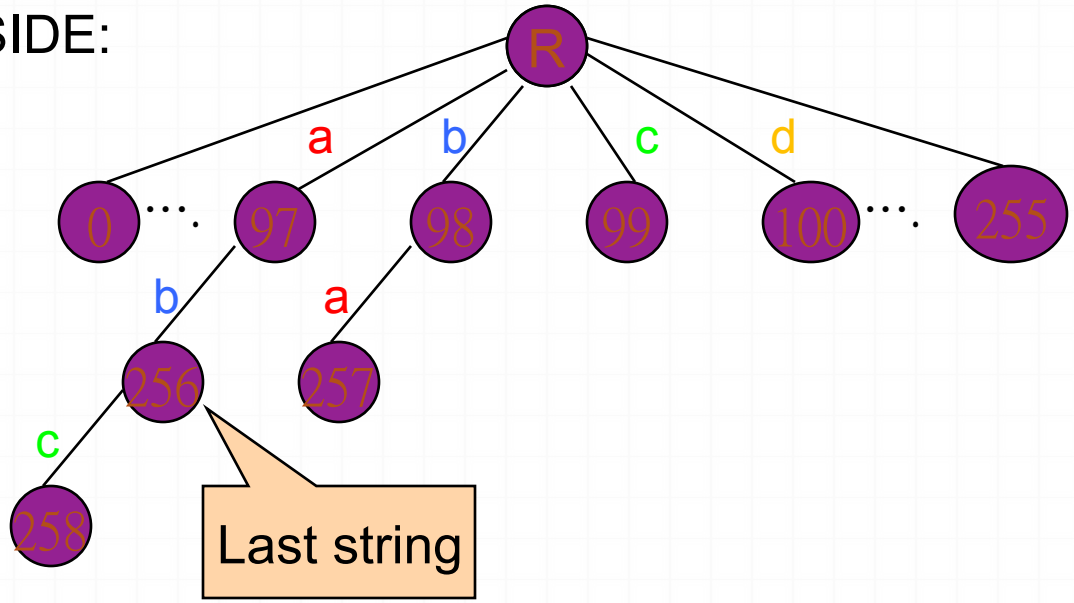
input: 97,98,256,99,100

output: a b a b

Last string = b

Tree Structure Example

DECODE SIDE:



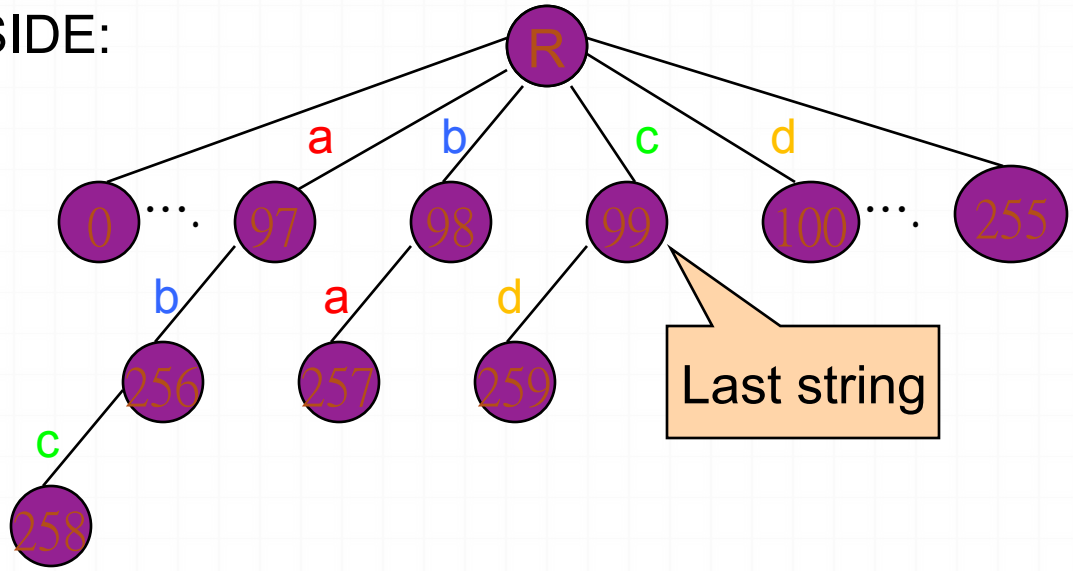
input: 97,98,256,99,100

output: a b a b c

Last string = a b

Tree Structure Example

DECODE SIDE:



input: 97,98,256,99,100

output: a b a b c d

Last string = c

Algorithm

- Now let's see a table structure example

Compression

IN: a b c c c d c c d

Prefix	Char.	Search	Code Saved
NUL L			

OUT:

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255
:	:

Compression

IN: a b c c c d c c d

Prefix	Char.	Search	Code Saved
NUL	‘a’	“a”	97
“a”	‘b’	“ab”	

OUT: 97

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255
256	**“ab**
	”

Compression

IN: a b c c c d c c d

Prefix	Char.	Search	Code Saved
NUL	‘b’	“b”	98
‘b’	‘c’	“bc”	

OUT: 97, 98

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255
256	“ab”
257	**“bc”**

Compression

IN: a b c c c d c c d

Prefix	Char.	Search	Code Saved
NUL	‘c’	“c”	99
‘c’	‘c’	“cc”	

OUT: 97, 98, 99

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255
256	“ab”
257	“bc”
258	“cc”

Compression

IN: a b c c c d c c d

Prefix	Char.	Search	Code Saved
NUL	‘c’	“c”	99
‘c’	‘c’	“cc”	258
‘cc’	‘d’	“ccd”	

OUT: 97, 98, 99, 258

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255
256	“ab”
257	“bc”
258	“cc”
259	**“ccd”**

Compression

IN: a b c c c d c c d

Prefix	Char.	Search	Code Saved
NUL	‘d’	“d”	100
‘d’	‘c’	“dc”	

OUT: 97, 98, 99, 258, 100

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255
256	“ab”
257	“bc”
258	“cc”
259	“ccd”
260	**“dc”**

Compression

IN: a b c c c d c c d

Prefix	Char.	Search	Code Saved
NUL	‘c’	“c”	99
‘c’	‘c’	“cc”	258
‘cc’	‘d’	“ccd”	259

OUT: 97, 98, 99, 258, 100, 259

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255
256	“ab”
257	“bc”
258	“cc”
259	“ccd”
260	“dc”

Reminder

1. Initialize the dictionary, and set P to an empty string
2. While there are still characters to read,
    a. Get the next character, C, from the input text
    b. Search for the string <P, C> in the dictionary
    c. if found,
        i. Append C to the end of the string P
    d. if not found,
        i. Output the code that corresponds to P
        ii. Add to the dictionary the new entry <P, C>
        iii. P := C
3. Output the code that corresponds to P

Reminder

- N-bit code → 2^N-1 phases
- First 0-255 entries are all the single characters
- Last phase 2^N-1 is reserved
  - Denote "End-Of-File"
  - Must be written as the last code of one file

Decompression

- Reverse of the compression process
- Look up the phase according to the code that we read
- Update the dictionary on-the-fly

Decompression

IN: 97, 98, 99, 258, 100, 259

cW	pW	C	dict(pW)+C
97			

OUT: a

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255

Decompression

IN: 97, 98, 99, 258, 100, 259

cW	pW	C	dict(pW)+C
98	97	'b'	"ab"

OUT: a b

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255
256	"ab"

Decompression

IN: 97, 98, 99, 258, 100, 259

cW	pW	C	dict(pW)+C
99	98	'c	"bc"

OUT: a b c

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255
256	"ab"
257	"bc"

Decompression

IN: 97, 98, 99, 258, 100, 259

cW	pW	C	dict(pW)+C
258	99	'c	"cc"

exception

OUT: a b c c c

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255
256	"ab"
257	"bc"
258	"cc"

Decompression

IN: 97, 98, 99, 258, 100, 259

cW	pW	C	dict(pW)+C
100	258	'd'	"ccd"

OUT: a b c c c d

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255
256	"ab"
257	"bc"
258	"cc"
259	"ccd"

Decompression

IN: 97, 98, 99, 258, 100, 259

cW	pW	C	dict(pW)+C
259	100	'c	"dc"

OUT: a b c c c d c c d

CODE	Entry
0	NUL
:	
97	a
98	b
99	c
100	d
:	:
255	ASCII-255
256	"ab"
257	"bc"
258	"cc"
259	"ccd"
260	"dc"

Reminder

1. Initialize the dictionary,
2. Get the first code, cW
3. Find cW in dictionary and output the string dictionary(cW)
4. While there are still codes to read,
    a. pW := cW
    b. Get the next code, cW, and find it in the dictionary
    c. if found,
        i. Output the string dictionary(cW)
        ii. Let C be the first character of dictionary(cW)
        iii. P := dictionary(pW)
        iv. Add the string <P, C> to the dictionary
    d. if not found, **(exception)**
        i. P := dictionary(pW)
        ii. Let C be the first character of P
        iii. Output the string <P, C>
        iv. Add the string <P, C> to the dictionary

Reminder

- Exception that a code from the lzw file refers to an entry not present yet
- Decompression “lags” behind compression
  - in compression, after producing the first code, the dictionary now contains the 256-th entry
  - in decompression, the 256-th entry is added after reading the second code

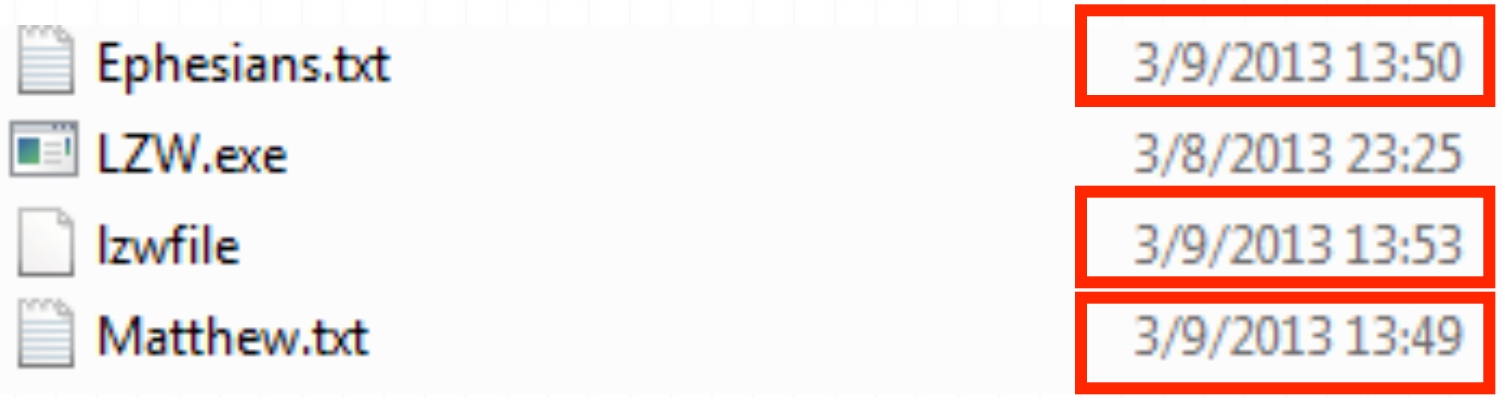
What have to do in Ass2 ?

- Compress and Decompress a list of files to a single archive
- In command line:
  - For compression :
    - >lzw –c <lzw filename> < list of file names>
  - For decompression:
    - >lzw –d <lzw filename>

What have to do in Ass2 ?

- Compression

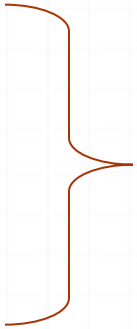
```
LZW.exe -c lzwfile Ephesians.txt Matthew.txt
```



File format

- LZW file format

<filename1>\n
<filename2>\n
<filename3>\n
….
\n



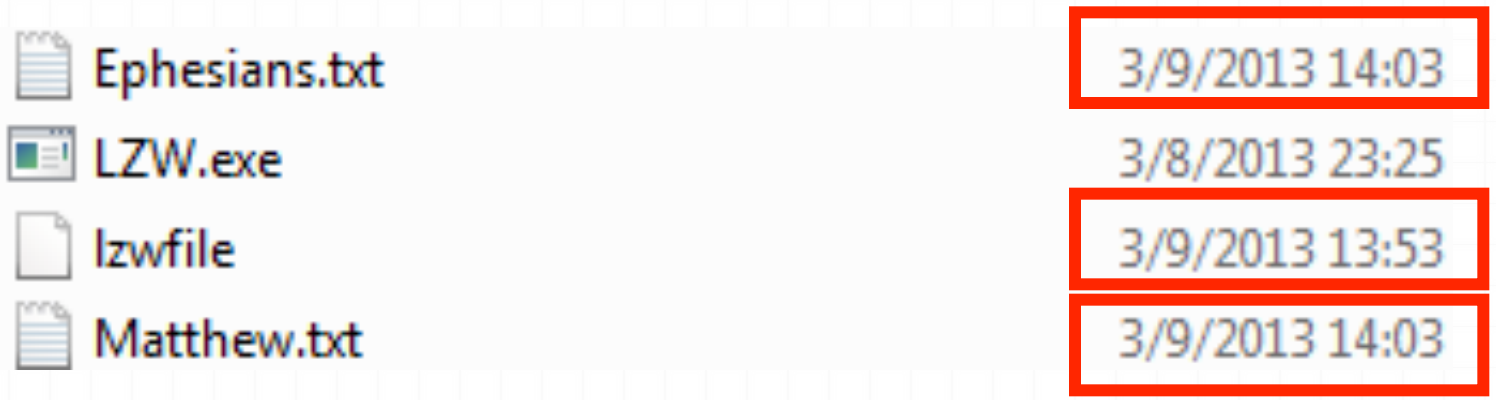
Header

<Compressed file1><end-of-file>
<Compressed file2><end-of-file>
<Compressed file3><end-of-file>
….

What have to do in Ass2 ?

0 Decompression

```
LZW.exe -d lzwfile
```



Details of Implementation

- Full dictionary
  - Code size = 12 in this assignment
  - At most 4096 entries
  - Create a new dictionary in compression
  - Similar steps in decompression

Discussion

- File end
  - Denote 2^N-1 as EOF code of one file
  - Output this EOF defined by us when finishing the compression of one file
  - Stop or open another file when detects the EOF in decompression

Discussion

- File end
  - Please do not make yourselves confused with “2^N-1 as EOF code” and EOF returned from reading file.
  - “2^N-1 as EOF code” is for separating the files from one to one. EOF, means that there is no any character to be read.

Discussion

- You are provided with sketch file. And six functions should be paid attention to.
  - write_code(lzw_file, code, code_size)

    This function is for writing the code to file. It is not necessary to modify this function.

Discussion

- Be careful on using the write_code() routine, since the write_code() routine stores things in a buffer before actual writing

- Remember to flush the buffer at last!
  - write_code(lzw_file, 0, 12);

Discussion

0 read_code(lzw_file, code_size)

This function is for reading the code from file. It is not necessary to modify this function as well.

Discussion

- readfileheader(lzw_file, outputfile, numberoffile)

  This function is for reading the information of header from file. It is not necessary to modify this function.

Discussion

0 writefileheader(lzw_file, outputfile, numberoffile)

This function is for writing the information of header to file. It is not necessary to modify this function as well.

Discussion

- compress(inputfile, lzw_file)

  This function is for compression. You are required to implement this function.

Discussion

- decompress(lzw_file, ouputfile)

  This function is for decompression. You are required to implement this function.

Reminder

- You are provided with a sample program
  - Your output should be the same as the ones from sample program.
  - “fc /w /c file1 file2” for file comparison in Windows Command Line.

Reminder

- Try to speed up look-up process
  - Tree structure
  - Hashing function
  - There are bonus for advanced data structure accelerating the speed of compression and decompression.
  - Reference page: http://en.wikipedia.org/wiki/LZW