Correction From Tuesdays' Lecture - Checking the CRC

```
                1 0 1 0 1 1 0
          ┌─────────────────────
1 0 1 1   │ 1 0 0 1 0 0 1 0 1 0
        - 1 0 1 1
          ─────────
          0 1 0 0
            1 0 0 0
          - 1 0 1 1
            ─────────
            0 1 1 1 0
          - 1 0 1 1
            ─────────
              1 0 1 1
            - 1 0 1 1
              ─────────
                  0 0 0
```

- Remainder = 0
- Message is Correct
- The quotient is discarded; only the fact that the remainder is zero or non-zero is important

Data Compression

- Introduction
- Huffman
- LZW
- Burrows Wheeler Transform

Compression Techniques

- Variable length codes
  - Huffman, or similar variable length codes
  - Assign short codes to more frequent symbols
  - Needs a coding dictionary
  - Not the most effective
- Run-length coding
  - Many files have lots of repeating characters, for example faxes
  - 00000000000000111111100000000111 may be encoded as 14,0,7,1,8,0,3,1
- Codebooks
  - Assign codes to frequent words and then run as an extension to VL codes
  - Excellent in some contexts but inflexible

Compression Techniques

- Dictionary-based
  - Use redundancy in text to replace already known letter sequences or *phrases* by dictionary indices
  - These compressors learn from the text and adapt to it
- Relative or Differential coding
  - Used for image data
  - Sends the changes between frames
- Statistical-based
  - Assign each symbol a probability
  - Encode the more probable symbol using few bits

Huffman Encoding

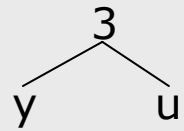
- Obtain a symbol frequency count for the message
- Build the Huffman tree
- Encode the message
- If the decoder does not have access to the same code-set, this will need to be transmitted with the message

Huffman Encoding - Example

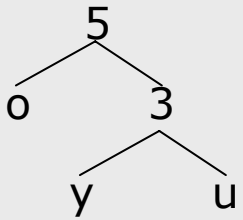
- Assume a message has the following frequencies
  - {a=4,e=12,i=8,o=2,u=2,y=1}
- Assume every symbol is a single leaf node
- Build a binary tree by continually adding nodes using the two lowest frequencies
- The first node will consist of *u,y*

Huffman Tree - Example

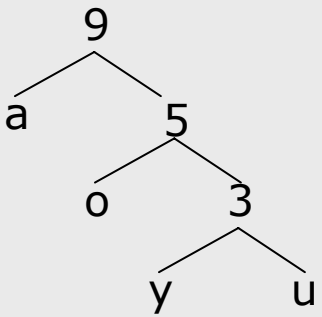
1. {a=4,e=12,i=8,o=2,u=2,y=1}



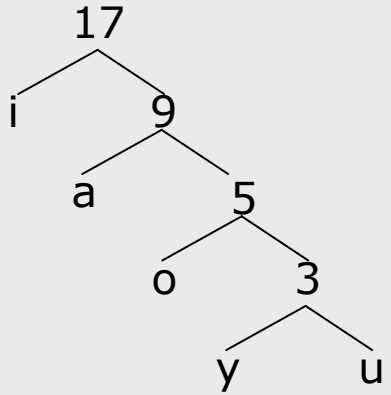
2. {a=4,e=12,i=8,o=2,uy=3}



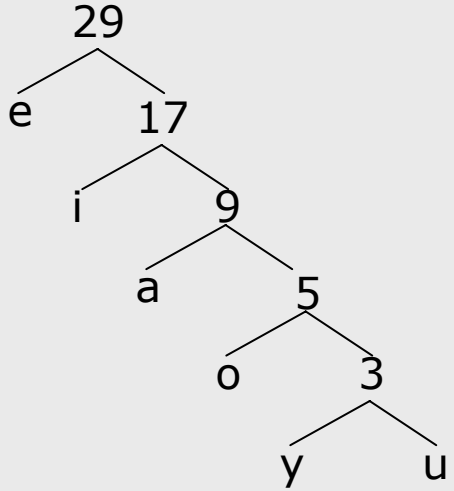
3. {a=4,e=12,i=8,ouy=5}



4. {e=12,i=8,aouy=9}



5. {e=12, iaouy=17}



6. {a=110,e=0,i=10, o=1110,u=11111, y=11110}

Huffman Tree

- More frequent symbols have shorter codes
- The decoder needs to have access to this code set

LZW Compression

- LZW was proposed by Welch as a variant of a compressor originally proposed by Ziv & Lempel, and is the compressor normally used in data communications
- Uses a dictionary of known phrases or letter sequences
  - The dictionary is initialized with all possible character codes, e.g. 0…255
- Compressor accepts input symbols as long as the most recent symbols correspond to known phrases in the dictionary

LZW Compression

- As soon as the matching fails, the compressor emits the dictionary index of the known phrase and then creates a new phrase consisting of the known phrase followed by the character which ‘failed’
- Continues building a new test string for phrase comparisons
- The dictionary is initialized to all values, 0…255

Example of LZW Compression

- Encoder input is "the_theme_that_they_heard"
- Assume an empty dictionary

the_theme_that_they_heard

Pass	Input	Test string	Emit	Make entry	Comment
1	t	t			
2					
3					
4					
5					
6					
7					
8					
9					
10					

Dictionary

Index	Contents
0	256
…	ASCII
255	codes
256	NA
257	NA
258	NA
259	NA
260	NA
261	NA
262	NA

Example of LZW Compression

- Encoder input is "the_theme_that_they_heard"

the_theme_that_they_heard

Pass	Input	Test string	Emit	Make entry	Comment
1	t	t			t exists
2	h	th	t	th	
3	e	he	h	he	
4	_	e_	e	e_	
5	t	_t	_	_t	
6	h	th			the exists – continue
7	e	the	th(256)	the	
8	m	em	e	em	
9	e	me	m	me	
10	_	e_			e_exists

Dictionary

Index	Contents
0	256
…	ASCII
255	codes
256	th
257	he
258	e_
259	_t
260	the
261	em
262	me

Example of LZW Compression

the_theme_that_they_heard

Pass	Input	Test string	Emit	Make Entry	Comment
11	t	e_t	e_(258)	e_t	
12	h	th			th exists
13	a	tha	th(256)	tha	
14	t	at	a	at	
15	_	t_	t	t_	
16	t	_t			_t exists
17	h	_th	_t(259)	_th	
18	e	he			he exists
19	y	hey	he(257)	hey	
20	_	y_	y	y_	

Dictionary

Index	contents
263	e_t
264	tha
265	at
266	t_
267	_th
268	hey
269	y_

Example of LZW Compression

the_theme_that_they_heard

Pass	Input	Test string	Emit	Make Entry	Comment
21	h	_h	_	_h	
22	e	he			he exists
23	a	hea	he(257)	hea	
24	r	ar	a	ar	
25	d	rd	r	rd	

Dictionary

Index	Contents
270	_h
271	hea
272	ar
273	rd

LZW Decoding

- The basic steps are
  - Remember the previous phrase
  - Receive the current index and get its current phrase
  - Emit the current phrase
  - Create a new dictionary entry of the previous phrase followed by the first symbol of the current phrase

Example of LZW Decompression

the_theme_that_they_heard				Dictionary	
Pass	Receive codeword	Make entry	deliver	Index	Contents
1	t		t	0	256
2	h	th	h	…	ASCII
3	e	he	e	255	codes
4	_	e_	_	256	th
5	(256) th	_t	th	257	he
6	e	the	e	258	e_
7	m	em	m	259	_t
8	(258) e_	me	e_	260	the
9	(256) th	e_t	th	261	em
10	a	tha	a	262	me
11	t	at	t	263	e_t
12	(259) _t	t_	_t	264	tha

Summary of VL and dictionary-based Codes

- Huffman maps fixed length symbols to variable length codes
  - Can be made adaptive
  - Optimal when symbol frequencies are powers of two
- LZW is a dictionary-based compression method
  - Builds a growing dictionary of phrases which are mapped to a sequentially increasing code-set

BZIP2 Compressor (Burrows-Wheeler Transform)

- Write out all cyclic permutations of the input as rows of a matrix
- Sort the rows into alphabetic order, this collects together similar contexts
- Transmit the *last symbol* of each row, and the *index of the first row*
  - Similar contexts produce similar symbols and often runs of the one symbol

BZIP2 Compressor (Burrows-Wheeler Transform)

- Use a Move-To-Front algorithm to recode each symbol
  - Replace each by the number of different symbols seen since it was last encountered
  - This produces a skewed symbol distribution

Value	0	1	2	3	4	5	6	7
Frequency	66.8%	9.0%	4.0%	2.9%	2.3%	1.8%	1.6%	1.4%

- Use a Huffman encoder on the MTF output

BWT Example

- Get all permutations
- Sort them
- Transmit the last column along with the index of the original string to the MTF

String
takaka
akakat
kakata
akatak
kataka
atakak

Row	String
1	**akakat**
2	**akatak**
3	**atakak**
4	**kakata**
5	**kataka**
6	**takaka**

6, tkkaaa

Reversing BWT

- We have the last column, and therefore can build up the first column of the matrix
- Sorting the pairs reveals the second column
- The transmitted index reveals the original string

Row	String
1	**a????t**
2	**a????k**
3	**a????k**
4	**k????a**
5	**k????a**
6	**t????a**

Row	String
1	**ak???t**
2	**ak???k**
3	**ak???k**
4	**ka???a**
5	**ka???a**
6	**ta???a**

MTF Example

- This output is then transmitted to the Huffman encoder
- Is easily reversed
- Benefits become apparent when using a longer string
  - *aaaaaaaaakkkkkkkkkkkkkkk*
  - *0000000001000000000000000*

MTF Example

- Initialize the MTF alphabet 0..#A
- After encoding each symbol, 'push' it to the front of the MTF alphabet

Symbol	Code
A	0
K	1
t	2

Symbol	Before MTF			After MTF			Emitted Code
	0	1	2	0	1	2	
t	a	k	t	t	a	k	2
k	t	a	k	k	t	a	2
k	k	t	a	k	t	a	0
a	k	t	a	a	k	t	2
a	a	k	t	a	k	t	0
a	a	k	t	a	k	t	0

Compression Results

- LZW – 4 – 4.5 bits per character or 50 – 55% of the original text size
- BZIP 2 – 2.5 bits per character or 25 – 30 % of the original text size

Recommended Reading

- Understanding Data Communications and Networks
  - Pages 189 – 194 (Huffman Codes)
  - Liv-Zempel Codes (LZW)