Web Data Management

Data Compression and Search

Similarity Search

Overview

The foundations of

- how different compression tools work.
- how to manage a large amount of data on small devices.
- how to search gigabytes, terabytes or petabytes of data.
- how to perform full text search efficiently without added indexing.
- how to query data with similarity measurements efficiently.

Course Aims

As the amount of Web data increases, it is becoming vital to not only be able to search and retrieve this information quickly, but also to store it in a compact manner.

This is especially important for mobile devices which are becoming increasingly popular.

Without loss of generality, within this course, we assume Web data (excluding media content) will be in XML and its like (e.g., XHTML).

Course Aims

1. Introduce the concepts, theories, and algorithmic issues important to Web data compression and search.

2. Discuss similarity search techniques for flat strings and hierarchical data (for example, XML).

3. Selected methods will be presented, their effectiveness and efficiency will be discussed.

4. Filtering techniques to improve the efficiency will be introduced.

Course info

- Lectures: 14:00-16:25 (Mon)
  - Y 201
  - Weeks 1-12

Lecturer in charge

吕建华

Office: Room 230, Computer Building

Email: lujianhua@seu.edu.cn

Assumed knowledge

At the start of this course students should be able to:

- understand fundamental data structures.
- knowledge and tools of RDBMS and SQL
- produce correct programs in C/C++, i.e., compilation, running, testing, debugging, etc.
- appreciate use of abstraction in computing.
- produce readable code with clear documentation.

Final Mark

- Assignments (50%)
- Representations (30%)
  - in class
  - with/without preparations
- Final report (20%)
  - About Assignments/Representations

Assignments

- Programming assignments (except assigt1) are relatively challenging
- In addition to correctness, reasonable performance is required

Tentative course schedule

Week	Lecture
1	Course overview; basic information theory
2	Arithmetic coding, adaptive coding, dictionary coding
3	Adaptive Huffman, LZW; Overview of BWT
4	Pattern matching and regular expression
5	FM index, backward search, suffix array
6	Suffix array O(n), compressed BWT
7	XML overview
8	XML compression
9	Similarity search overview
10	String edit distance
11	q-gram distance
12	Trees, RDB, and tree edit distance

Now

- Others you should know…

- Data compression and search starts …

Learning outcomes

- have a good understanding of the fundamentals of text compression
- be introduced to advanced data compression techniques such as those based on Burrows Wheeler Transform
- have programming experience in Web data compression and optimization
- have a deep understanding of XML and selected XML processing and optimization techniques
- understand the advantages and disadvantages of data compression for Web search
- have a basic understanding of XML distributed query processing
- appreciate the past, present and future of data compression and Web data optimization

Questions to discuss

- What is data compression
- Why data compression
- Where

Compression

- Minimize amount of information to be stored / transmitted

- Transform a sequence of characters into a new bit sequence
  - same information content (for lossless)
  - as short as possible

Familiar tools

- Tools for
  - .Z
  - .zip
  - .gz
  - .bz2
  - …

A glimpse

raaabbccccdabbbbeee$

Run-length coding

- Run-length coding (encoding) is a very widely used and simple compression Technique
  - replace runs of symbols (possibly of length one) with pairs of (symbol, run-length)

RLE

raaabbccccdaaaaabbbbeeeeeed$

How?

ra3bbc4da5b4e6d$

Example: BWT

rabcabcababaabacabcabcabcababaa$

Example: BWT

rabcabcababaabacabcabcabcababaa$

aabbbbccacccrcbaaaaaaaaaaabbbbba$

Example: BWT+RLE

rabcabcababaabacabcabcabcababaa$

aabbbbccacccrcbaaaaaaaaaaabbbbba$

aab4ccac3rcba10b5a$

HTTP compression

HTTP/1.1 200 OK

Date: Mon, 23 May 2005 22:38:34 GMT

Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)

Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT

Etag: "3f80f-1b6-3e1cb03b"

Accept-Ranges: bytes

Content-Length: 438

Connection: close

Content-Type: text/html; charset=UTF-8

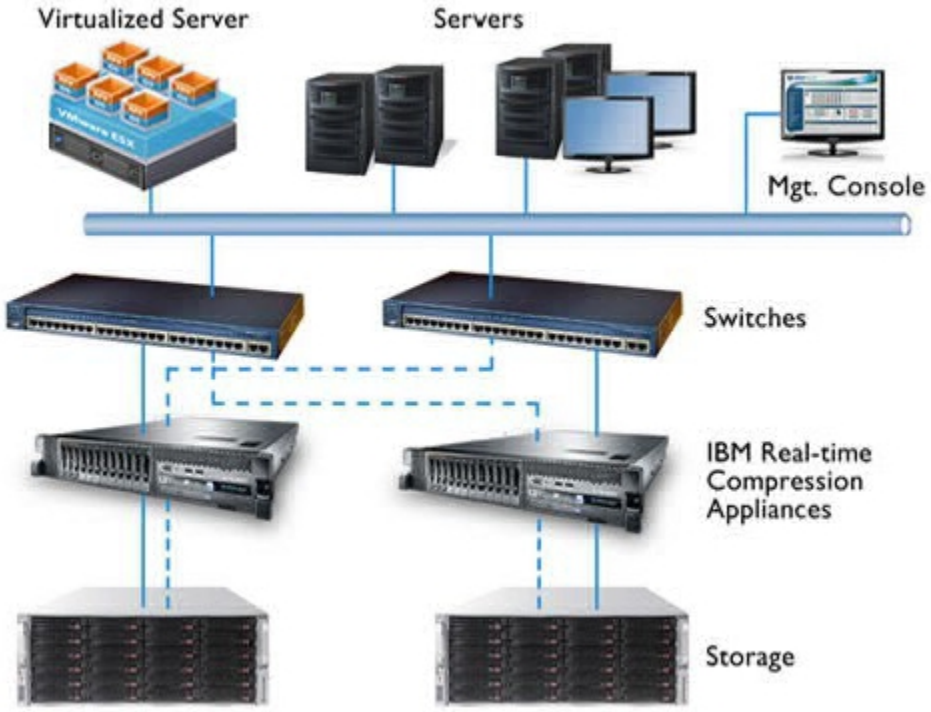
Content-Encoding: gzip

From: http://www.telerik.com/products/aspnet-ajax/compression.aspx

Storwize

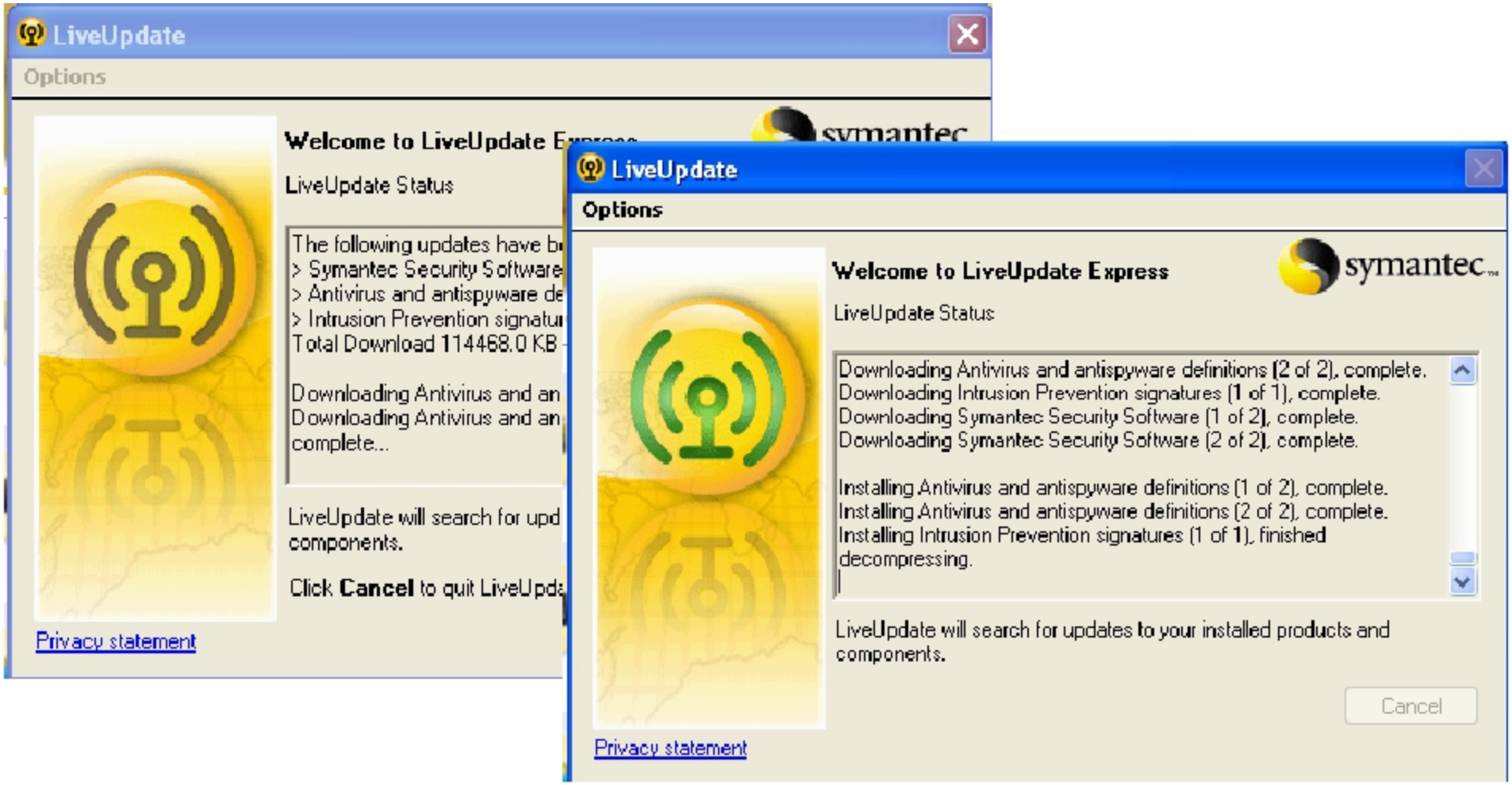
Better Storage Utilization
Reduces existing storage utilization up to 80%
No performance degradation
Lowers Capital and Operational Costs
Better Energy Efficiency
Less to store, power and cool

…



Source: from storwize.com

Anti-virus definitions & updates



Others

- Software updates

  e.g., Reg files, UI schemas / definitions
- Software configuration/database updates

  e.g., Virus database for anti-virus software
- Data streams

  e.g., RSS

```
wong:Desktop wong$ ls -l image.jpg
-rwx------@ 1 wong  staff  671172 11 Feb 17:32 image.jpg
wong:Desktop wong$ gzip image.jpg
wong:Desktop wong$ ls -l image.jpg.gz
-rwx------@ 1 wong  staff  424840 11 Feb 17:32 image.jpg.gz
wong:Desktop wong$ mv image.jpg.gz image.jz
wong:Desktop wong$ gzip image.jz
wong:Desktop wong$ ls -l image.jz.gz
-rwx------@ 1 wong  staff  424932 11 Feb 17:32 image.jz.gz
wong:Desktop wong$ mv image.jz.gz image.jzz
wong:Desktop wong$ gzip image.jzz
wong:Desktop wong$ ls -l image.jzz.gz
-rwx------@ 1 wong  staff  425018 11 Feb 17:32 image.jzz.gz
wong:Desktop wong$
```

Similarity measure

If two objects compress better together than separately, it means they share common patterns and are similar.

From: Li, M. et al., “The similarity metric”, IEEE Transactions on Information Theory, 50(12), 2004

Overview

- Compression refers to a process of coding that will effectively reduce the total number of bits needed to represent certain information.

- Information theory studies efficient coding algorithms
  – complexity, compression, likelihood of error

Compression

- There are two main categories
  - Lossless (Input message = Output message)
  - Lossy (Input message ≈ Output message)
    - Not necessarily reduce quality

Compression

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}$$

$$\text{Space Savings} = 1 - \frac{\text{Compressed Size}}{\text{Uncompressed Size}}$$

Example

- Compress a 10MB file to 2MB
- Compression ratio = 5 or 5:1
- Space savings = 0.8 or 80%

Terminology

- Coding (encoding) maps source messages from alphabet (S) into codewords (C)

- Source message (symbol) is basic unit into which a string is partitioned
  – can be a single letter or a string of letters

Terminology (Types)

- Block-block
  - source message and codeword: fixed length
  - e.g., ASCII
- Block-variable
  - source message: fixed; codeword: variable
  - e.g., Huffman coding
- Variable-block
  - source message: variable; codeword: fixed
  - e.g., LZW
- Variable-variable
  - source message and codeword: variable
  - e.g., Arithmetic coding

Terminology (Symmetry)

- Symmetric compression
  - requires same time for encoding and decoding
  - used for live mode applications (teleconference)

- Asymmetric compression
  - performed once when enough time is available
  - decompression performed frequently, must be fast
  - used for retrieval mode applications (e.g., an interactive CD-ROM)

Decodable

A code is

- distinct if each codeword can be distinguished from every other (mapping is one-to-one)
- uniquely decodable if every codeword is identifiable when immersed in a sequence of codewords

Example

- A: 1
- B: 10
- C: 11
- D: 101
- To encode ABCD: 11011101
- To decode 11011101: ?

Uniquely decodable

- Uniquely decodable is a prefix free code
  if no codeword is a proper prefix of any other

- For example {1, 100000, 00} is uniquely decodable, but is not a prefix code
  – consider the codeword {…1000000001…}

- Practical we prefer prefix code (why?)

Example

S	Code
a	00
b	01
c	10
d	110
e	111

Example

S	Code
a	00
b	01
c	10
d	110
e	111

0100010011011000

Example

S	Code
a	00
b	01
c	10
d	110
e	111

0100010011011000

babadda

Static codes

- Mapping is fixed before transmission
  - E.g., Huffman coding

- probabilities known in advance

Dynamic codes

- Mapping changes over time
  - i.e. adaptive coding
- Attempts to exploit locality of reference
  - periodic, frequent occurrences of messages
  - e.g., dynamic Huffman

Traditional evaluation criteria

- Algorithm complexity
  - running time
- Amount of compression
  - redundancy
  - compression ratio
- How to measure?

Measure of information

- Consider symbols $\underline{s_i}$ and the probability of occurrence of each symbol $p(s_i)$
- In case of fixed-length coding , smallest number of bits per symbol needed is
  - $L \geq \log_2(N)$ bits per symbol
  - Example: Message with 5 symbols need 3 bits ($L \geq \log_2 5$)

Variable length coding

- Also known as entropy coding
  - The number of bits used to code symbols in the alphabet is variable
  - E.g. Huffman coding, Arithmetic coding

Entropy

- What is the minimum number of bits per symbol?
- Answer: Shannon's result – theoretical minimum average number of bits per code word is known as Entropy (H)

$$\sum_{i=1}^{n} - p(s_i) \log_2 p(s_i)$$

Entropy example

- Alphabet S = {A, B}
  - p(A) = 0.4; p(B) = 0.6
- Compute Entropy (H)
  - $-0.4*\log_2 0.4 + -0.6*\log_2 0.6 = .97$ bits
- Maximum uncertainty (gives largest H)
  - occurs when all probabilities are equal

Example: ASCII

0	nul	1	soh	2	stx	3	etx	4	eot	5	enq	6	ack	7	bel
8	bs	9	ht	10	nl	11	vt	12	np	13	cr	14	so	15	si
16	dle	17	dc1	18	dc2	19	dc3	20	dc4	21	nak	22	syn	23	etb
24	can	25	em	26	sub	27	esc	28	fs	29	gs	30	rs	31	us
32	sp	33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	del

ASCII

- Example: SPACE is 32 or 00100000. ‘z’ is 122 or 01111010
- 256 symbols, assume same probability for each
- P(s) = 1/256
- Optimal length for each char is log 1/P(s) = log 256 = 8 bits

David A. Huffman

David Huffman is best known for the invention of Huffman code, a highly important compression scheme for lossless variable length encoding. It was the result of a term paper he wrote while a graduate student at the Massachusetts Institute of Technology (MIT)…

From: Wikipedia

Huffman coding algorithm

1. Take the two least probable symbols in the alphabet

   (longest code words, equal length, differing in last digit)

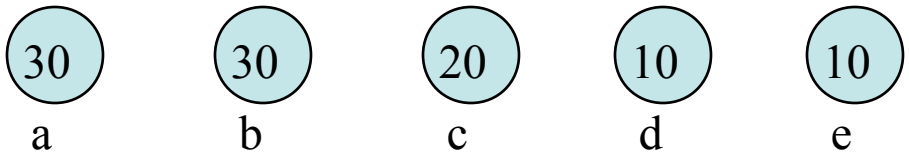
2. Combine these two symbols into a single symbol
3. Repeat

Example: Huffman coding

S	Freq
a	30
b	30
c	20
d	10
e	10

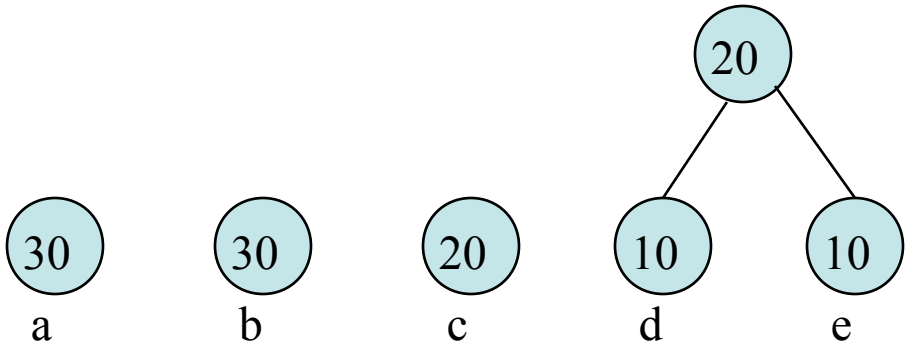
Example

S	Freq	Huffman
a	30	
b	30	
c	20	
d	10	
e	10	



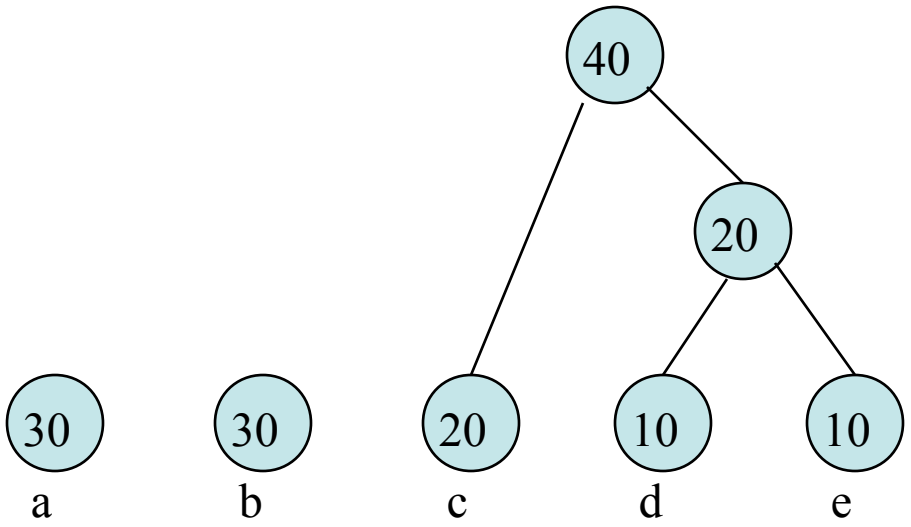
Example

S	Freq	Huffman
a	30	
b	30	
c	20	
d	10	
e	10	



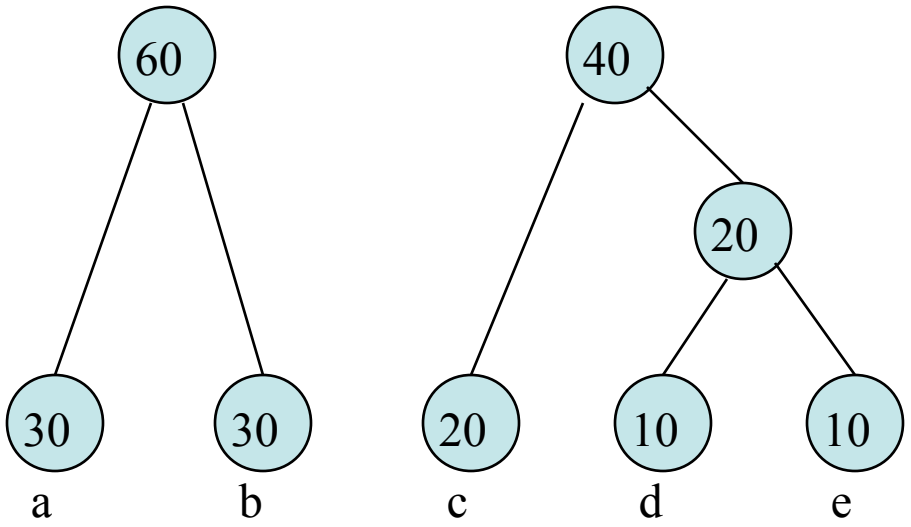
Example

S	Freq	Huffman
a	30	
b	30	
c	20	
d	10	
e	10	



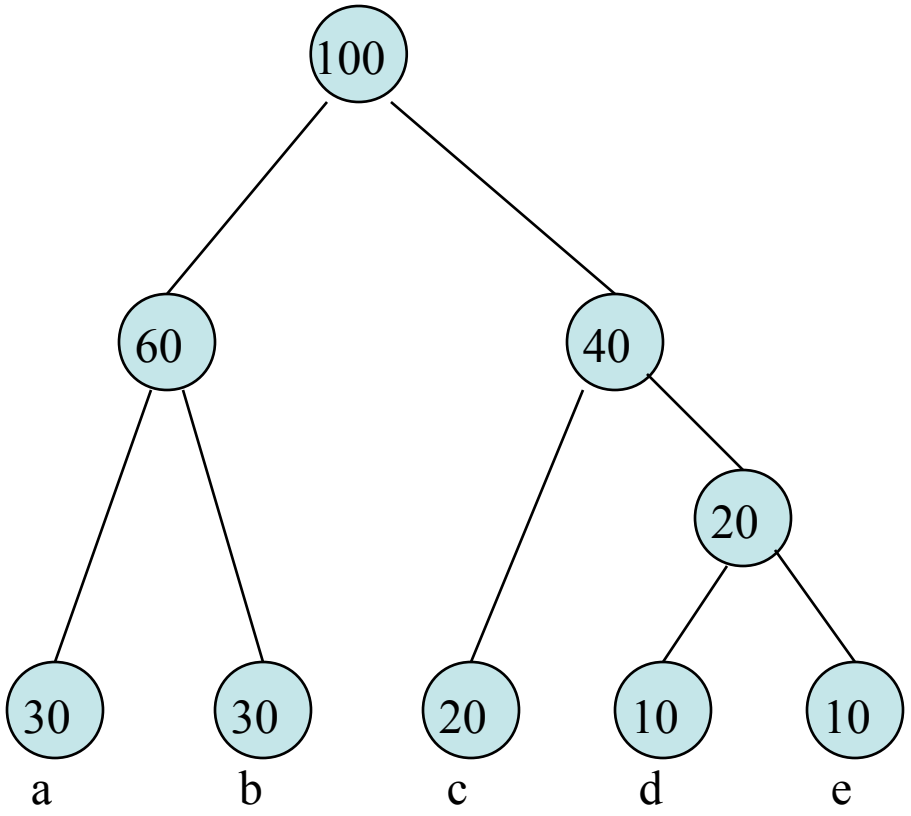
Example

S	Freq	Huffman
a	30	
b	30	
c	20	
d	10	
e	10	



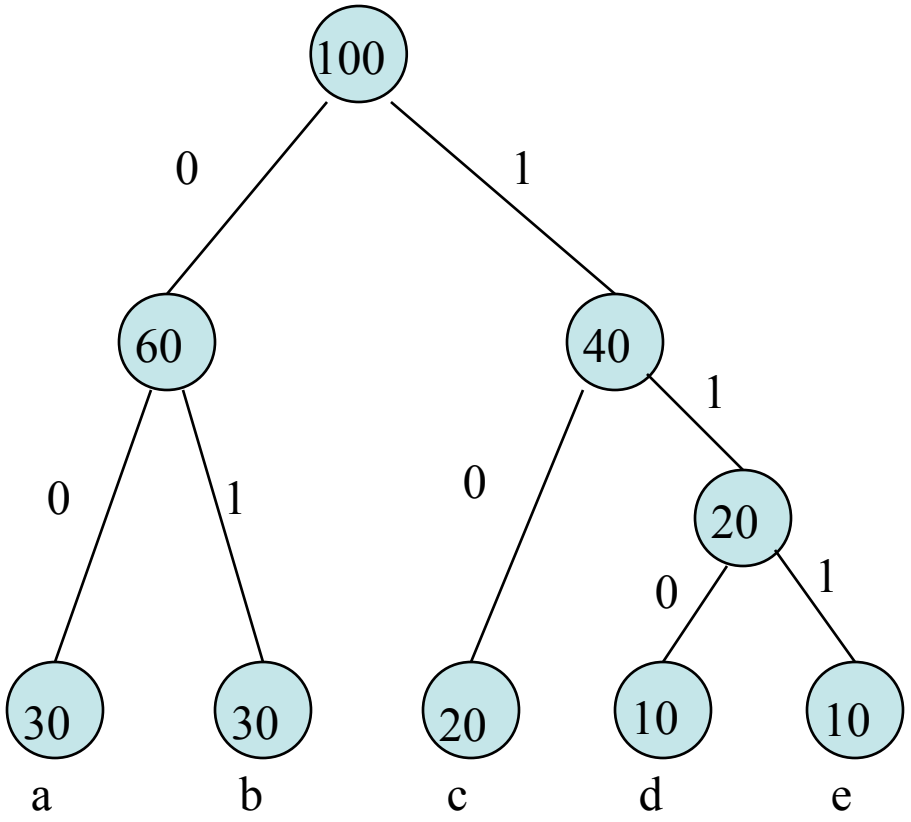
Example

S	Freq	Huffman
a	30	
b	30	
c	20	
d	10	
e	10	



Example

S	Freq	Huffman
a	30	00
b	30	01
c	20	10
d	10	110
e	10	111



Average length L

$= (30*2 + 30*2 + 20*2 + 10*3 + 10*3) / 100$

$= 220 / 100$

$= \underline{2.2}$

Average length L

$= (30*2 + 30*2 + 20*2 + 10*3 + 10*3) / 100$

$= 220 / 100$

$= \underline{2.2}$

Better than using fixed length 3 bits for 5 symbols.

Entropy

H = -0.3 * log 0.3 + -0.3 * log 0.3 + -0.2 * log 0.2 + -0.1 * log 0.1 + -0.1 * log 0.1

= -0.3*(-1.737) + -0.3*(-1.737) + -0.2 * (-2.322) + -0.1 * (-3.322) + -0.1 * (-3.322)

= 0.3 log 10/3 + 0.3 log 10/3 + 0.2 log 5 + 0.1 log 10 + 0.1 log 10

= 0.3*1.737 + 0.3*1.737 + 0.2* 2.322 + 0.1*3.322 + 0.1*3.322

= 2.17

Another example

- S={a, b, c, d} with freq {4, 2, 1, 1}
- $H = 4/8*\log_2 2 + 2/8*\log_2 4 + 1/8*\log_2 8 + 1/8*\log_2 8$
- $H = 1/2 + 1/2 + 3/8 + 3/8 = 1.75$
- a => 0 b => 10 c => 110 d => 111
- Message: {abcdabaa} => {0 10 110 111 0 10 0 0}
- Average length L = 14 bits / 8 chars = 1.75
- If equal probability, i.e. fixed length, need $\log_2 4 = 2$ bits

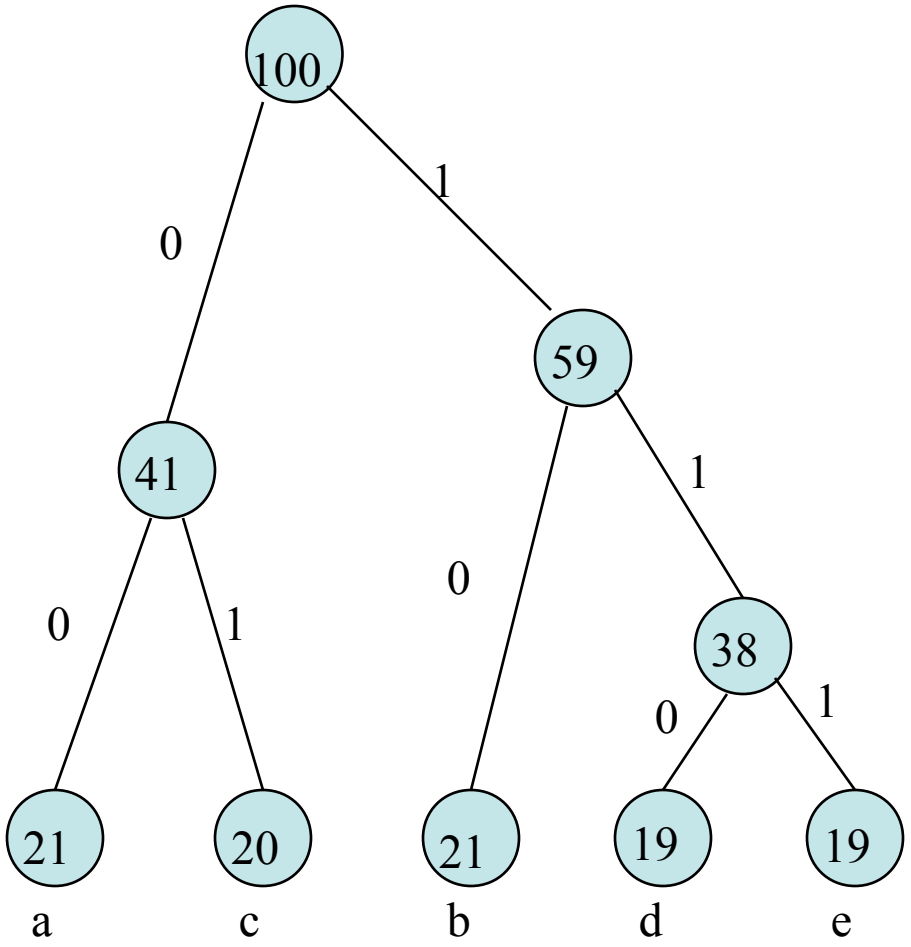
Huffman coding

S	Freq	Huffman
a	3021	
b	3021	
c	2020	
d	1019	
e	1019	

Total: 100

Huffman coding

S	Freq	Huffman
a	21	00
b	21	10
c	20	01
d	19	110
e	19	111



Huffman optimal?

H = 0.21 log 100/21 + 0.21 log 100/21 + 0.2 log 5 + 0.19 log 100/19 + 0.19 log 100/19

= 0.21*2.252 + 0.21*2.252 + 0.2* 2.322 + 0.19*2.396 + 0.19*2.396

= ~~2.32~~

L = (21*2 + 21*2 + 20*2 + 19*3 + 19*3)/100

= 2.38

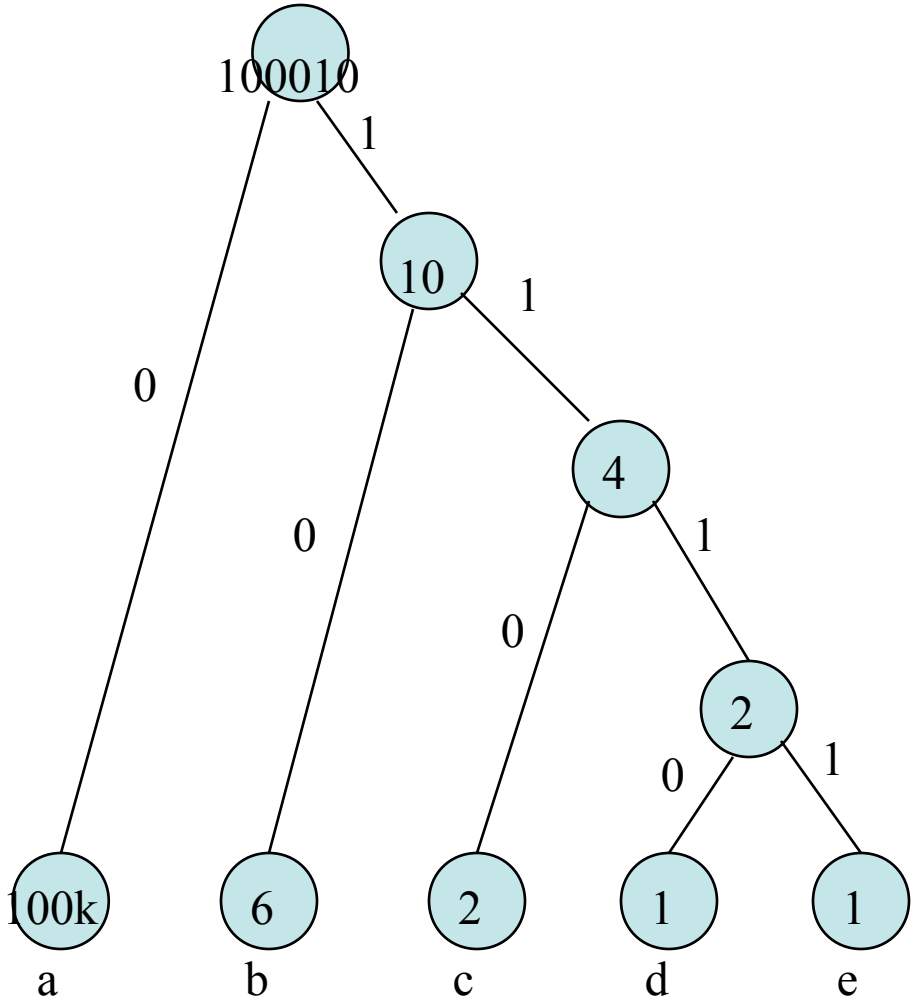
Huffman coding

S	Freq	Huffman
a	30100000	
b	306	
c	202	
d	101	
e	101	

Total: 100010

Huffman coding

S	Freq	Huffman
a	100000	0
b	6	10
c	2	110
d	1	1110
e	1	1111



Huffman optimal?

$H = 0.9999 \log 1.0001 + 0.00006 \log 16668.333 + \ldots + 1/100010 \log 100010$

≈ 0.00

$L = (100000*1 + \ldots)/100010$

≈ 1

Problems of Huffman coding

- Huffman codes have an integral # of bits.
  - E.g., log (3) = 1.585 while Huffman may need 2 bits
- Noticeable non-optimality when prob of a symbol is high.

=> Arithmetic coding

Arithmetic coding

Message to encode:

BILL GATES

Character	Probability
SPACE	1/10
A	1/10
B	1/10
E	1/10
G	1/10
I	1/10
L	2/10
S	1/10
T	1/10

Example extracted from February, 1991 issue of Dr. Dobb's Journal

Arithmetic coding

Character	Probability	Range
SPACE	1/10	0.00 - 0.10
A	1/10	0.10 - 0.20
B	1/10	0.20 - 0.30
E	1/10	0.30 - 0.40
G	1/10	0.40 - 0.50
I	1/10	0.50 - 0.60
L	2/10	0.60 - 0.80
S	1/10	0.80 - 0.90
T	1/10	0.90 - 1.00

Arithmetic coding algorithm

Set low to 0.0
Set high to 1.0
While there are still input symbols do
    get an input symbol
    code_range = high - low.
    high = low + range*high_range(symbol)
    low = low + range*low_range(symbol)
End of While
output low or a number within the range

Arithmetic coding

New Character	Low value	High Value
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
SPACE	0.25720	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168
T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

Example

Consider the second L as new char:

code_range = 0.258 – 0.256 = 0.002

high = 0.256 + 0.002*0.8 = 0.2576

low = 0.256 + 0.002*0.6 = 0.2572

Decoding algorithm

get encoded number

Do

find symbol whose range straddles the encoded number

output the symbol

range = symbol high value - symbol low value

subtract symbol low value from encoded number

divide encoded number by range

until no more symbols

Arithmetic coding

Encoded Number	Output Symbol	Low	High	Range
0.2572167752	B	0.2	0.3	0.1
0.572167752	I	0.5	0.6	0.1
0.72167752	L	0.6	0.8	0.2
0.6083876	L	0.6	0.8	0.2
0.041938	SPACE	0.0	0.1	0.1
0.41938	G	0.4	0.5	0.1
0.1938	A	0.2	0.3	0.1
0.938	T	0.9	1.0	0.1
0.38	E	0.3	0.4	0.1
0.8	S	0.8	0.9	0.1
0.0				

Example

At the first L, encoded number is 0.72167752.

output the first L

range = 0.8 – 0.6 = 0.2

encoded number = (0.72167752 – 0.6) / 0.2

= 0.6083876

Advantage of arithmetic coding

Assume: A 90% END 10%

To encode: AAAAAAA

New Character	Low value	High Value
	0.0	1.0
A	0.0	0.9
A	0.0	0.81
A	0.0	0.729
A	0.0	0.6561
A	0.0	0.59049
A	0.0	0.531441
A	0.0	0.4782969
END	0.43046721	0.4782969

Advantage of arithmetic coding

Assume: A 90% END 10%

To encode: AAAAAAA

New Character	Low value	High Value
	0.0	1.0
A	0.0	0.9
A	0.0	0.81
A	0.0	0.729
A	0.0	0.6561
A	0.0	0.59049
A	0.0	0.531441
A	0.0	0.4782969
END	0.43046721	0.4782969

e.g., 0.45

Lossless compression revisited

- Run-length coding
- Statistical methods
  - Huffman coding
  - Arithmetic coding
- Dictionary methods
  - Lempel Ziv algorithms

Static vs Adaptive

Dictionary coding

- Patterns: correlations between part of the data
- Idea: replace recurring patterns with references to dictionary
- LZ algorithms are adaptive:
  - Universal coding scheme
  - Single pass (dictionary created on the fly)
  - No need to transmit/store dictionary

LZ77 & LZ78

- LZ77: referring to previously processed data as dictionary
- LZ78: use an explicit dictionary

Lempel-Ziv-Welch (LZW) Algorithm

- Most popular modification to LZ78
- Very common, e.g., Unix compress, GIF87
- Read http://en.wikipedia.org/wiki/LZW regarding its patents
- Fixed-length references (12bit 4096 entries)
- Static after max entries reached

LZW Compression

```
w = NIL;
  while ( read a character k )
      {
        if wk exists in the dictionary
         w = wk;
        else
          add wk to the dictionary;
          output the code for w;
          w = k;
      }
```

Example

Input: ^WED^WE^WEE^WEB^WET

```
w=nil
while(read a character k){
   if wk exist in the dic
      w = wk;
   else
      add wk to the dic
      output the code for w
      w=k;
}
```

w	k	output	index	symbol
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE
E	^	E	261	E^
^	W			
^W	E			
^WE	E	260	262	^WEE
E	^			
E^	W	261	263	E^W
W	E			
WE	B	257	264	WEB
B	^	B	265	B^
^	W			
^W	E			
^WE	T	260	266	^WET
T	EOF	T		

LZW Compression

- Original LZW used dictionary with 4K entries, first 256 (0-255) are ASCII codes.
- In the above example, a 19 symbols reduced to 7 symbols & 5 code. Each code/symbol will need 8+ bits, say 9 bits.
- Reference: Terry A. Welch, "A Technique for High Performance Data Compression", IEEE Computer, Vol. 17, No. 6, 1984, pp. 8-19.

LZW Decompression

```
read a character k;
  output k;
  w = k;
  while ( read a character/code k )
      {
        entry = dictionary entry for k;
        output entry;
        add w + entry[0] to dictionary;
        w = entry;
      }
```

Example

Input: ^WED<256>E<260><261><257>B<260>T

```
read a char k;
output k;
w=k;
while(read a char/code k){
  entry = dic entry for k;
  output entry;
  add w+entry[0] to dic;
  w=entry;
}
```

w	k	output	index	symbol
	^	^		
^	W	W	256	^W
W	E	E	257	WE
E	D	D	258	ED
D	<256>	^W	259	D^
<256>	E	E	260	^WE
E	<260>	^WE	261	E^
<260>	<261>	E^	262	^WEE
<261>	<257>	WE	263	E^W
<257>	B	B	264	WEB
B	<260>	^WE	265	B^
<260>	T	T	266	^WET

LZW implementation

- Parsing fixed number of bits from input is easy
- Fast and efficient

More online readings

http://www.ics.uci.edu/~dan/pubs/DC-Sec1.html

http://marknelson.us/1991/02/01/arithmetic-coding-statistical-modeling-data-compression/

The end

We have covered:

- Course overview
- Prefix code, entropy
- Huffman code
- Arithmetic code
- LZW