Example 1: 80 days weather

☀ S

🌧 R

All sunny days except the last 16 days:

SSS…RRRSSSSSRRRRSSSS

Example 2: 80 days weather

☀ S ☁ C

🌧 R

All sunny days except the last 16 days:

SSS…RRRCSSSSRRRRCSSS

Run-length coding

- Run-length coding (encoding) is a very widely used and simple compression technique
  - does not assume a memoryless source
  - replace runs of symbols (possibly of length one) with pairs of (symbol, run-length)

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…}

- In practice, we prefer prefix code (why?)

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

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)

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

Huffman coding algorithm

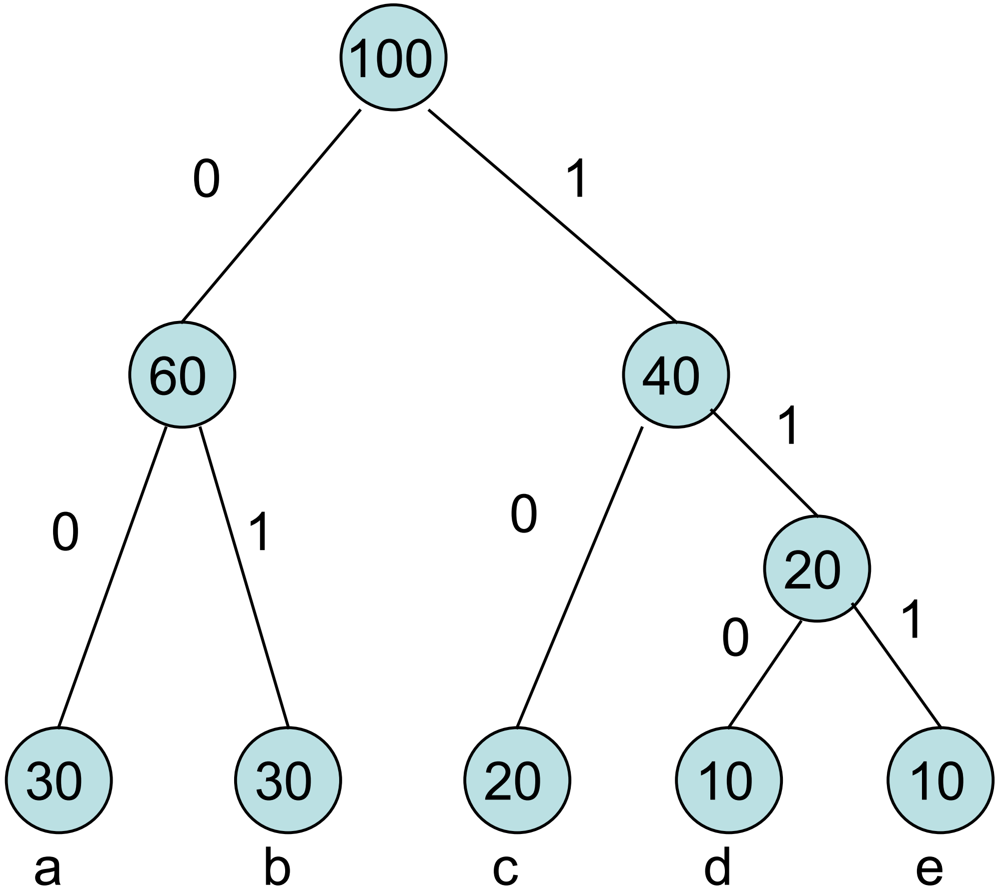
1. Take the two least probable symbols in the alphabet

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

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

Example

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



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

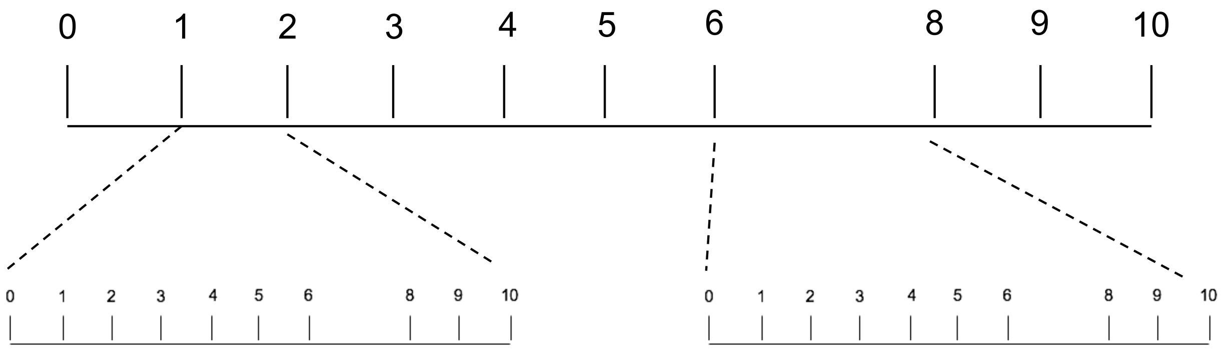
Problems of Huffman coding

- 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.

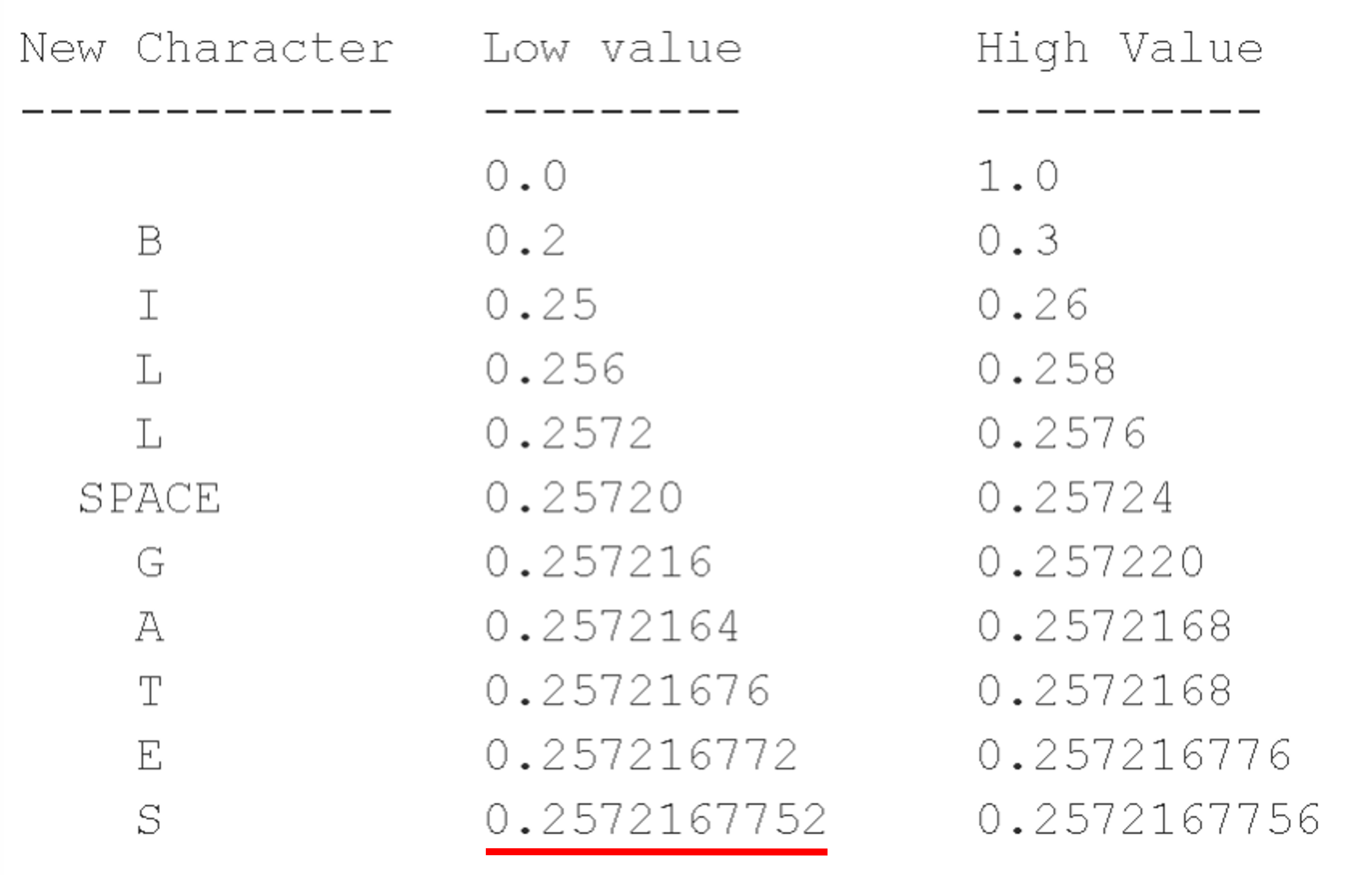
=> Arithmetic coding

Arithmetic coding

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



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

COMP9319 Web Data Compression and Search

LZW,

Adaptive Huffman

Dictionary coding

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

Lempel-Ziv-Welch (LZW) Algorithm

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

Problems of Huffman coding

Need statistics & static: e.g., single pass over the data just to collect stat & stat unchanged during encoding

To decode, the stat table need to be transmitted. Table size can be significant for small msg.

=> Adaptive compression e.g., adaptive huffman

Adaptive Huffman Coding (dummy)

Encoder

```
Reset the stat
Repeat for each input char
 (
   Encode char
   Update the stat
   Rebuild huffman tree
 )
```

Decoder

```
Reset the stat
Repeat for each input char
 (
   Decode char
   Update the stat
   Rebuild huffman tree
 )
```

This works but too slow!

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

Summarised schedule

0. Information Representation (today)

1. Compression

2. Search

3. Compression + Search on plain text

4. “Compression + Search” on Web text

5. Selected advanced topics (if time allows)

COMP9319 Web Data Compression and Search

Basic BWT

Basic BWT
(to be discussed more detailed next week)

Recall from Lecture 1’s RLE and BWT example

rabcabcababaabacabcabcabcababaa$

aabbbbccacccrcbaaaaaaaaaabbbbba$

aab4ccac3rcba10b5a$

A simple example

Input:

#BANANAS

All rotations

```
#BANANAS
S#BANANA
AS#BANAN
NAS#BANA
ANAS#BAN
NANAS#BA
ANANAS#B
BANANAS#
```

Sort the rows

```
#BANANAS
ANANAS#B
ANAS#BAN
AS#BANAN
BANANAS#
NANAS#BA
NAS#BANA
S#BANANA
```

Output

```
#BANANAS
ANANAS#B
ANAS#BAN
AS#BANAN
BANANAS#
NANAS#BA
NAS#BANA
S#BANANA
```

Exercise: you can try this example

rabcabcababaabacabcabcabcababaa$

aabbbbccacccrcbaaaaaaaaaabbbbba$

Now the inverse, for decoding…

Input:

S

B

N

N

#

A

A

A

First add

```
S
B
N
N
#
A
A
A
```

Then sort

```
#
A
A
A
B
N
N
S
```

Add again

```
S#
BA
NA
NA
#B
AN
AN
AS
```

Then sort

```
#B
AN
AN
AS
BA
NA
NA
S#
```

Then add

```
S#B
BAN
NAN
NAS
#BA
ANA
ANA
AS#
```

Then sort

```
#BA
ANA
ANA
AS#
BAN
NAN
NAS
S#B
```

Then add

```
S#BA
BANA
NANA
NAS#
#BAN
ANAN
ANAS
AS#B
```

Then sort

```
#BAN
ANAN
ANAS
AS#B
BANA
NANA
NAS#
S#BA
```

Then add

```
S#BAN
BANAN
NANAS
NAS#B
#BANA
ANANA
ANAS#
AS#BA
```

Then sort

```
#BANA
ANANA
ANAS#
AS#BA
BANAN
NANAS
NAS#B
S#BAN
```

Then add

```
S#BANA
BANANA
NANAS#
NAS#BA
#BANAN
ANANAS
ANAS#B
AS#BAN
```

Then sort

```
#BANAN
ANANAS
ANAS#B
AS#BAN
BANANA
NANAS#
NAS#BA
S#BANA
```

Then add

```
S#BANAN
BANANAS
NANAS#B
NAS#BAN
#BANANA
ANANAS#
ANAS#BA
AS#BANA
```

Then sort

```
#BANANA
ANANAS#
ANAS#BA
AS#BANA
BANANAS
NANAS#B
NAS#BAN
S#BANAN
```

Then add

```
S#BANANA
BANANAS#
NANAS#BA
NAS#BANA
#BANANAS
ANANAS#B
ANAS#BAN
AS#BANAN
```

Then sort (???)

```
#BANANAS
ANANAS#B
ANAS#BAN
AS#BANAN
BANANAS#
NANAS#BA
NAS#BANA
S#BANANA
```

Exercise: you can try this example

rabcabcababaabacabcabcabcababaa$

aabbbbccacccrcbaaaaaaaaaabbbba$