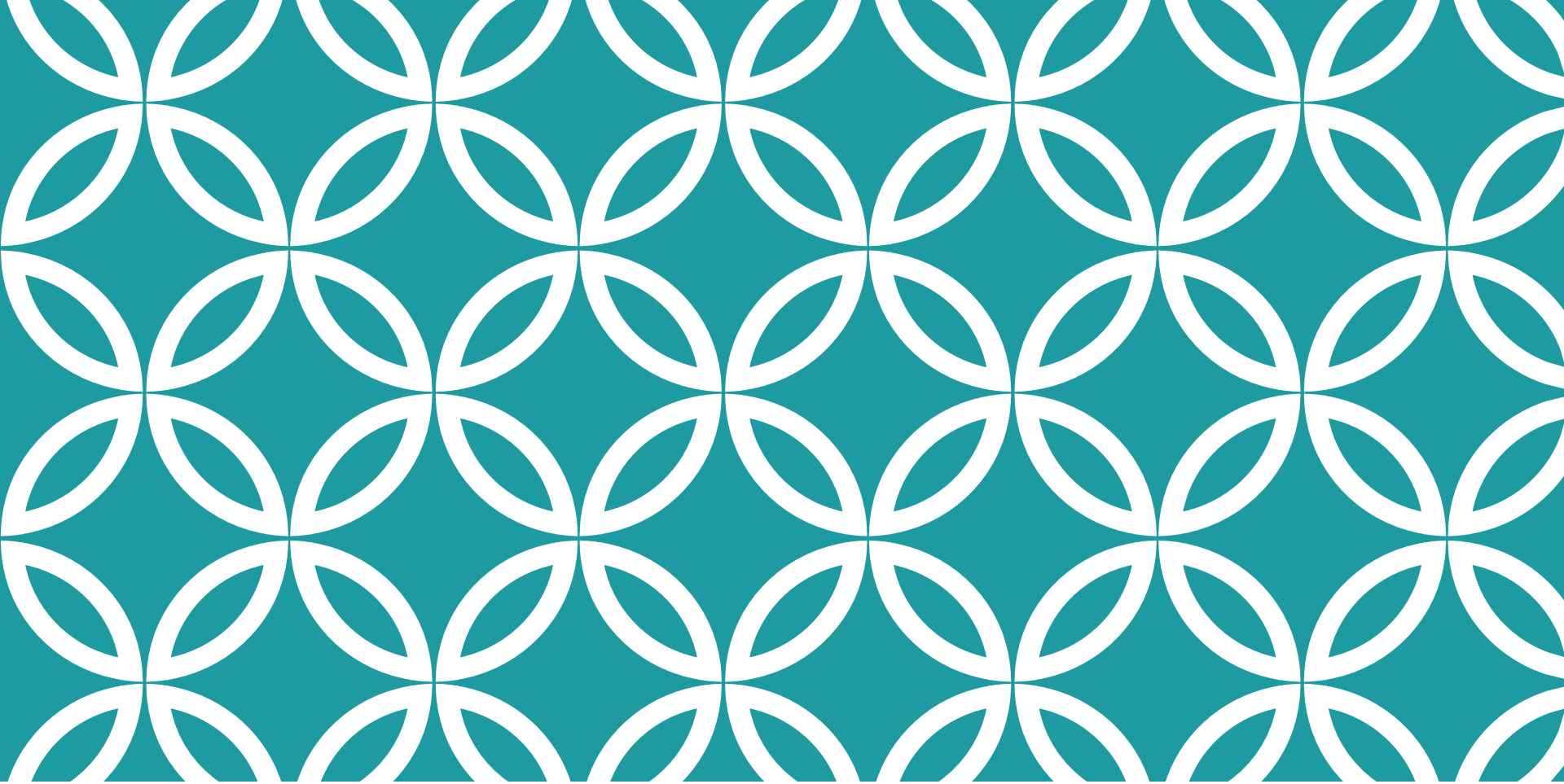
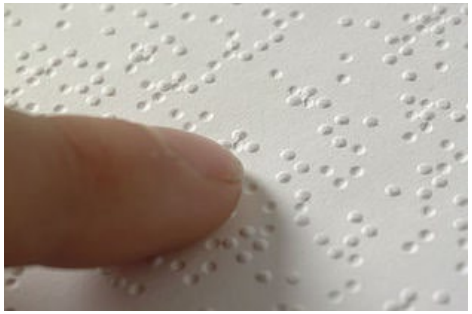


INTRODUCTION TO DATA COMPRESSION

CT102: Information Systems

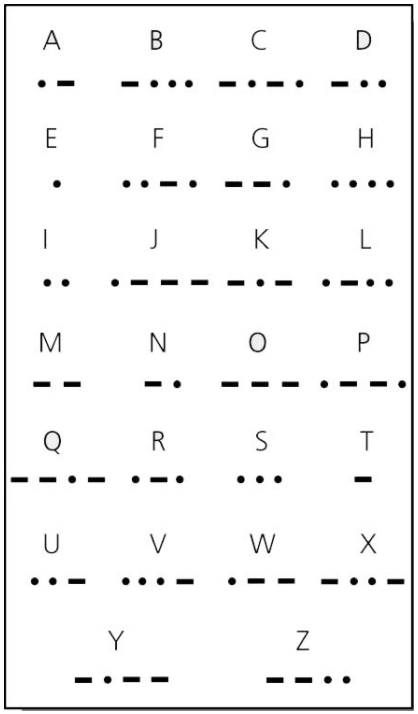
COMPRESSION



Reduces the space a file or message occupies

Specifically:

- encoding information using fewer bits than the original representation



WHY?

To save space when storing

To save time/bandwidth when transmitting

Still needed? ……

Moore's law:

- Number of transistors on a chip doubles every 18-24 months …

Parkinson's Law:

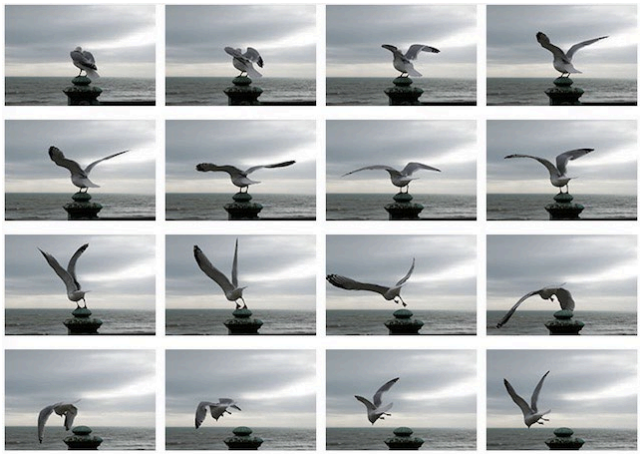
- Data expands to fill the space available for storage/transmission

HOW?

Many techniques are based on the fact that most files have redundancy. Examples:

- *Text:* ths sntnc cn b rd rthr qckly by mst ppl

- *Images:* large areas of same colour



- *Videos:* frames that are very similar to last

SOME EXAMPLES ….

Data: Gzip, Boa, Pkzip, Brotli

Images:

- **.gif** (graphics interchange format). Lossless
- **.jpg** (joint photographic experts group). Lossy. Full-colour or gray-scale digital images of "natural", real-world scenes
- **.png** … gif like … not as common as gif or jpeg

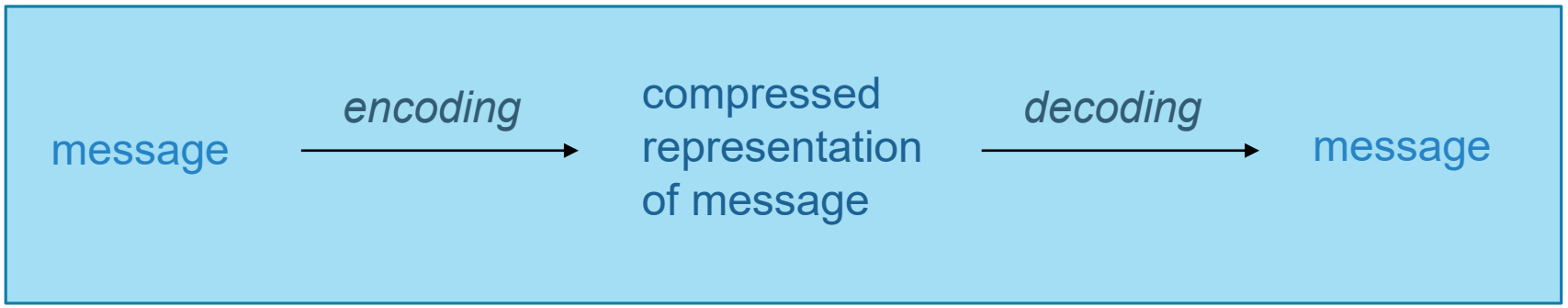
Sound: MP3

Video: MPEG, DivX, HDTV

TWO COMPONENTS:
Encoding and Decoding

Goal:

- The encoded message (compressed representation of the message) has fewer bits than the original message
- The decoded message is the same, or *approximately* the same as the original message



"A free lunch"? .... No!

To be used **must** undergo the opposite process of decompression

An example of a space-time complexity trade-off (common in computing):

- i.e., storage and transmission time gains versus execution (CPU) time for encoding and decoding

HIGH LEVEL CATEGORISATION OF COMPRESSION ALGORITHMS

- Lossy
- Lossless
- Hybrid

LOSSY COMPRESSION ALGORITHMS

Loose some information … in general not noticeable to human eye.

Cannot be used for text files or images that need to be closely analysed (e.g., medical images).

LOSSLESS COMPRESSION ALGORITHMS

No loss of information

Can be used for text files and any image files that need to be closely analysed

Is there a lossless algorithm that can compress all file types?

No …. need to assume some **bias** in the data (message) and exploit this bias to reduce the size of the file

QUALITY OF COMPRESSION APPROACHES

Lossless:

Time taken to encode

Time taken to decode (e.g., when streaming)

Compression ratio, e.g., 3.5MB Vs 50MB for mp3 song

Generality

Lossy:

Same as lossless but also need to judge how good the reconstructed message is

For text, we normally work with *fixed length encoding*

Fixed length encoding *u*ses same number of bits for each symbol

For example, ASCII:

Char	Decimal	Code
a	97	1100001
b	98	1100010
c	99	1100011
d	100	1100100
etc		

FOR EXAMPLE

A message is represented by:

1100010110000111100100

Char	Decimal	Code
a	97	1100001
b	98	1100010
c	99	1100011
d	100	1100100
etc		

What is message?

How many bits in message?

HOW TO KNOW THE LENGTH REQUIRED?

If N = number of different symbols

Then **lower($\log_2 N$)** is the length of code required

e.g., In genomic sequences have only 4 codons: a c t g

lower($\log_2 4$) = 2

So 2 bit code sufficient:

Say, a = 00, c = 01, t = 10, g = 11

EXAMPLE:

If a = 00, c = 01, t = 10, g = 11
in a fixed length 2-bit code, decode the following:

000110000100110010100

Note:

The decoder (us in this case) need to know code

VARIABLE LENGTH CODING

Variable length coding uses different length codes for different symbols

Example 1: Given the following variable length code:

a = 1

b = 01

c = 101

d = 011

Is it possible to decode this message?:

1011

PROBLEM?

Which is the correct decoding?

Solutions:

1. Add special stop/separator symbol
2. Choose codes which can always be uniquely decoded by choosing codes where no code is a prefix of another

SAME EXAMPLE AGAIN (4 SYMBOLS) BUT WITH DIFFERENT CODES

a = 1	b = 01
c = 000	d = 001

Is any code a prefix of another?

Decode:

- 1001
- 1011
- 0001
- 0111

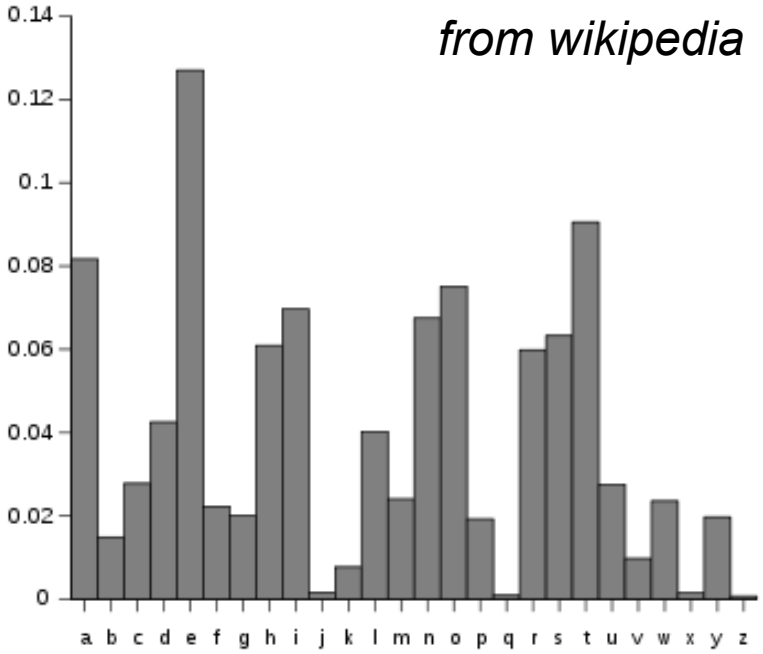
TEXT COMPRESSION

One approach to text compression uses *probability distribution of the symbols* in the message/file, i.e.,

- **More frequent symbols/words versus shorter symbols/words**

LETTER FREQUENCY IN ENGLISH?

- Mostly calculated based on *general human written text*
- May be differences if:
- file was generated by computer (e.g. server logs)
- file was created for a very specific context, e.g an essay on zebras or x-rays or Qatar



from wikipedia

LETTER FREQUENCY IN ENGLISH?

In order of most frequent:

e t a o i n s h r d l u c m f w y p v b g k j q x z

How frequent is *frequent*?

The top-12 letters comprise about 80% of the total usage:

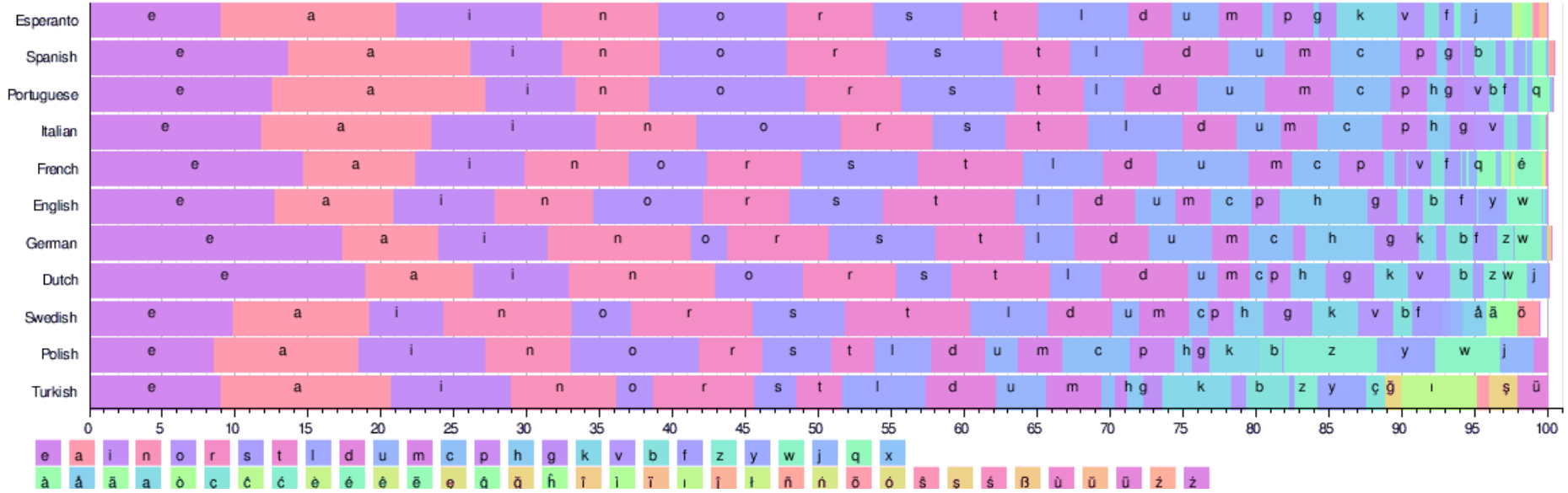
e t a o i n s h r d l u

The top-8 letters comprise about 65% of the total usage:

e t a o i n s h

LATIN LETTER FREQUENCY ACROSS SOME OTHER LANGUAGES

(http://en.wikipedia.org/wiki/Letter_frequency)



HUFFMAN CODING

- A **lossless** data compression technique
- Produces **optimal prefix codes** which are generated from a set of probabilities (based on frequency of occurrence) by the Huffman Coding Algorithm
- Guarantees **prefix property** – that is, no code is a prefix of any other code.
- Used as back-end of GZIP, JPEG, Brotli and many other utilities
- If good letter probabilities are available - and not **too costly** to obtain - then Huffman coding is a good compression technique and can achieve an average of 2.23 bits per symbol

OBTAINING LETTER PROBABILITIES?

1. Can use generic ones – derived for the language and domain:

e.g., English:

- e: 12.7
- t: 9.06
- a: 8.17
- o: 7.51
- i: 6.97

etc.

2. Can be the actual frequencies found in the text being compressed - this requires that a frequency table must be stored with the text (for decoding)

Letter	English
a	8.17%
b	1.49%
c	2.78%
d	4.25%
e	12.70%
f	2.23%
g	2.02%
h	6.09%
i	6.97%
j	0.15%
k	0.77%
l	4.03%
m	2.41%
n	6.75%
o	7.51%
p	1.93%
q	0.10%
r	5.99%
s	6.33%
t	9.06%
u	2.76%
v	0.98%
w	2.36%
x	0.15%
y	1.97%
z	0.07%

PREFIX CODE REPRESENTATION

The "trick" with Huffman coding is to represent the prefix codes using a **binary tree** where:

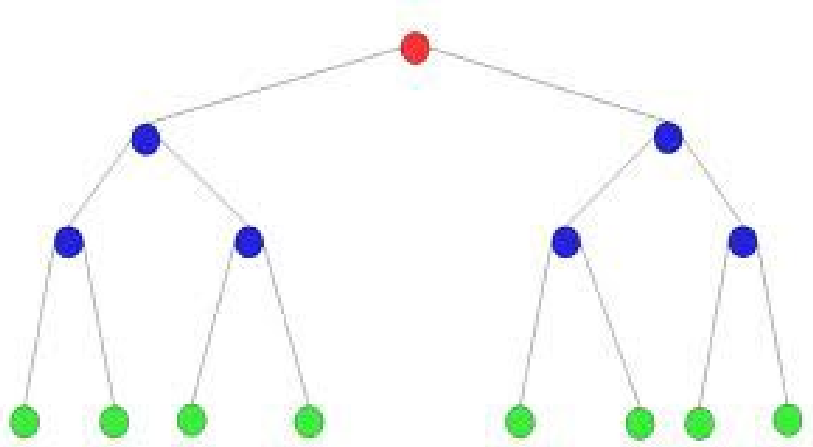
- each symbol is a leaf in the binary tree
- the code for each symbol is given by following a path from the root to the leaf and appending:
  - 0 for each left branch
  - 1 for each right branch

Note: by convention LHS is given 0 and RHS is given 1. But as long as encoder and decoder use the same labelling (are consistent) it does not matter.

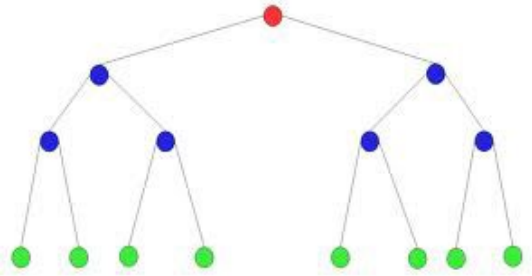
DEFINITION: BINARY TREE

A Binary tree consists of a set of non-linear nodes such that there is:

- One distinctive root node
- All other nodes are arranged such that each parent node can have at most 2 “child” nodes (a left and a right sub-node)



BINARY TREES

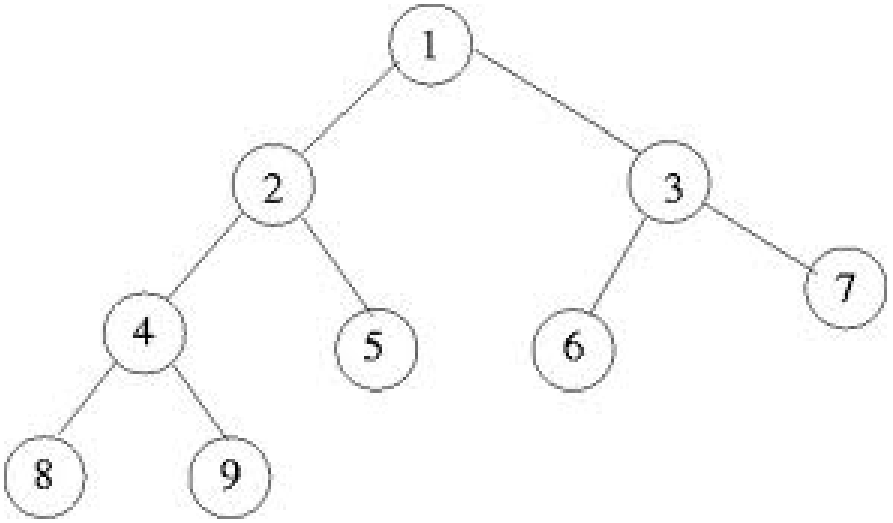


- The nodes with no child nodes (or sub-nodes) are often called ‘leaf nodes’
- The lines connecting the nodes are often called branches
- Paths are generally taken from the root node to the leaf nodes. At each stage can potentially choose to go left or right at a node and “follow” the branch to the next node.

EXAMPLES: List the paths
. . . list the nodes visited

From the root node (1) to:

- Node 8
- Node 6
- Node 5



CODE TREES (Weighted binary trees)

- Each leaf node represents a symbol
- Each branch has a "weight" associated with it (either 0 or 1)
- Left branches are weighted 0
- Right branches are weighted 1
- To find the code associated with the symbol:

  Start at root node and keep appending the weights from root to the symbol as you follow path from root to leaf node of interest

* This also gives the length of the code (the number of branches traversed).

EXAMPLE 1:

The code for symbol E is:

1111 (right,right,right,right)

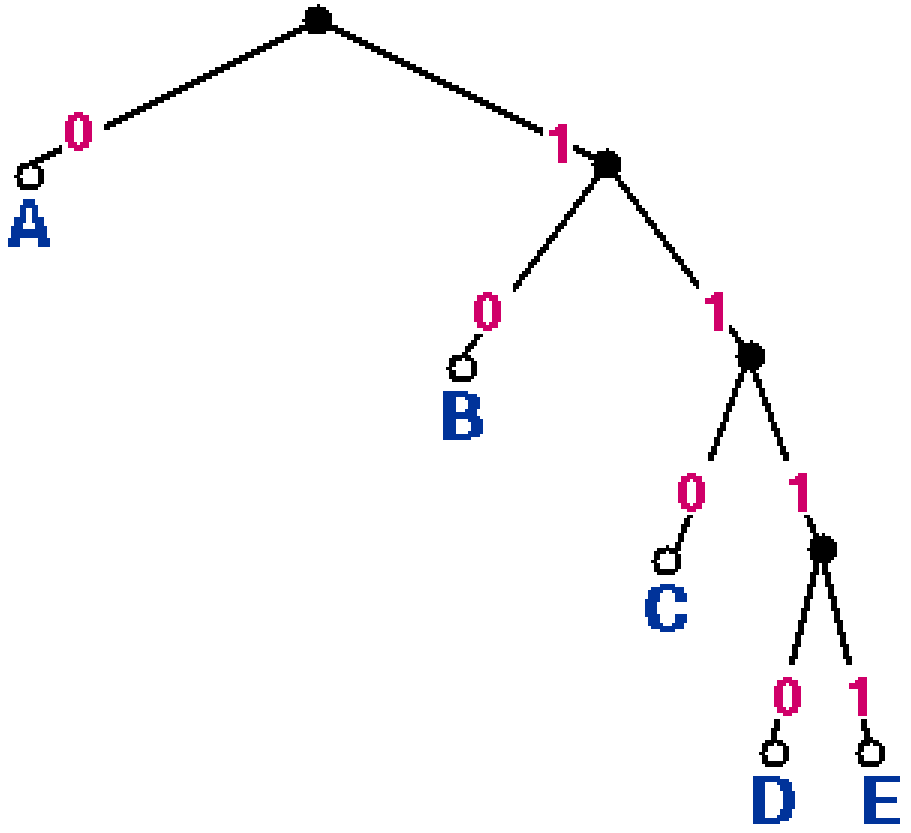
List the codes for each of the other symbols:

A

B

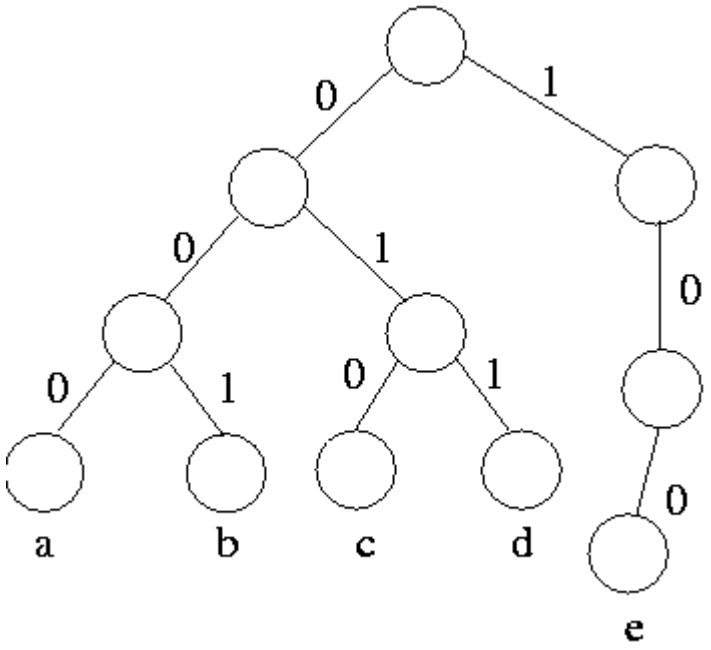
C

D



EXAMPLE 2:

What length (number of branches) are the codes for the symbols in the given code tree?



HUFFMAN COMPRESSION ALGORITHM

Input: Symbols (letters) and their probabilities (frequencies)

1. Create a trivial tree (node) for each letter

2. Assign a weight to each node – initially, the weight of each node is the frequency of the letter associated with that node

3. Sort (order) trees by weight (a priority queue), smallest to largest.

4. Decide on rule for ties: - will not affect code length but must be consistent with encoding/decoding stages. Our approach:

- If there is a tie with single node trees, order (alphabetically) by letter (symbol)
- If there is a tie otherwise, order by tree size (number of nodes in tree) - smallest to largest

5. while (more than one tree left in priority queue)

   {

   merge the two trees at the start of the priority queue (the "smallest") to create a new tree such that:

   - Root of tree has, as its weight, the summation of the weights of the sub-trees
   - the tree at the top of the queue is a **left sub-child** of root; the next tree is a right **sub-child** of root

   place new tree back in queue in correct place (in "sorted" order – i.e. so that smallest to largest is maintained in queue

   }

6. Label edges of final tree (left 0; right 1)

Output: Huffman code tree from which can read codes for each letter

EXAMPLES

3. Given the following letters and their frequency, construct a Huffman code tree:

t	a	e	h
10	5	15	3

4. By calculating the frequencies of each letter using just the message given, find optimal prefix codes, using Huffman compression, for each unique letter in the message:

this is mississippi

EXAMPLE 4: this is mississippi

Frequencies:

h	m	t	p	i	s
1	1	1	2	6	6

HUFFMAN DECOMPRESSION ALGORITHM

Input: letters and their frequencies and sequence of binary codes

Approach:

1. Build Huffman tree using **exact** same algorithm as was used for compression
2. For each encoded symbol, follow path from root node to leaf node, based on current number (1 or 0), until you reach symbol at leaf node.

Output: original message

EXAMPLE 5:
Decompress the messages

Given the probabilities of the following 5 symbols:

- P(a) = 0.12
- P(b) = 0.4
- P(c) = 0.15
- P(d) = 0.08
- P(e) = 0.25

What are the words represented by the following Huffman codes?

01010

01011111110

ARITHMETIC CODING

Arithmetic coding encodes a stream of symbols (rather than a single symbol) as a single floating point number, in the range from 0.0 to 1.0

Also an example of *lossless* data compression

Stream text encoding is more common now

ARITHMETIC CODING APPROACH

Input: message and symbols and their frequencies

General Approach: Work with intervals and sub-intervals where each interval represents a proportion relative to the probability of the occurrence of the message

Output: real number in range [0, 1.0)

ASIDE: RANGES ...

[0, 1] : 0 and 1 included in range

[0, 0.5): 0 included in range; 0.5 not

ARITHMETIC ENCODING ALGORITHM

1. Begin with interval = [0.0, 1.0)
2. Get all symbols and their probabilities of occurrence
3. Order the symbols – smallest to largest - smaller frequency first, alphabetically if there are ties (with the same frequency)
4. Place symbols from message in a queue

5. while (message) symbols left in queue {

- For current interval, divide the interval according to the probabilities of all symbols occurring and the order of these symbols (step 3), starting at lowest range of interval (e.g., 0.0)
- Let current message symbol = symbol at top of queue
- Find in which interval current message symbol lies, this becomes new interval and divide interval as before
- Get next message symbol

}

EXAMPLE 6:

Inputs:

- symbols are: a c r
- Probabilities are: P(r) = 0.2, P(a) = 0.4, P(c) = 0.4
- Message = car

Approach:

Order symbols according to probabilities first, alphabetically second:

- r a c

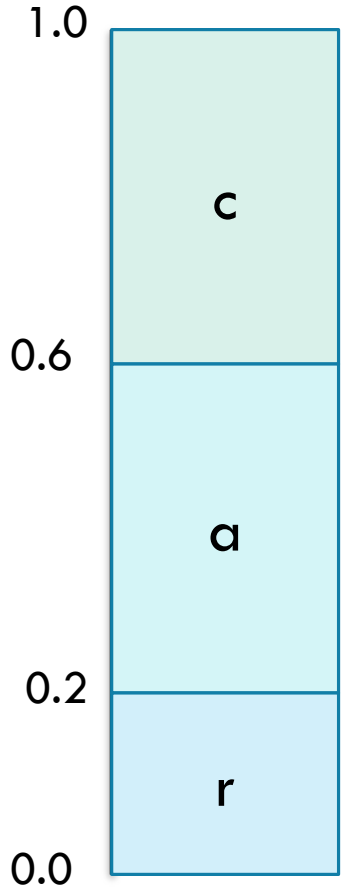
First iteration of loop:

- Assign each symbol to the an interval in range:
  - r : [0.0, 0.2)
  - a : [0.2, 0.6)
  - c : [0.6, 1.0)
- First symbol of message is c, so new interval is [0.6, 1.0)

MESSAGE: car

P(r) = 0.2, P(a) = 0.4, P(c) = 0.4

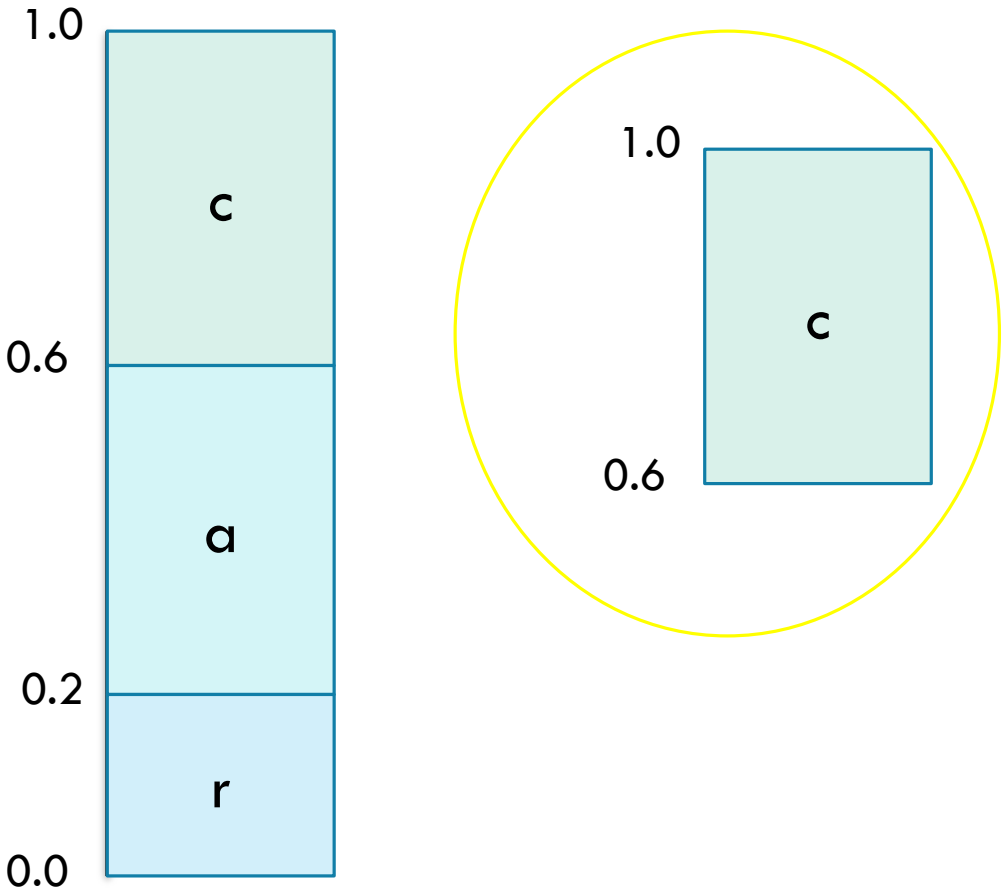
1st interval: 0.0 to 1.0



First iteration of loop:

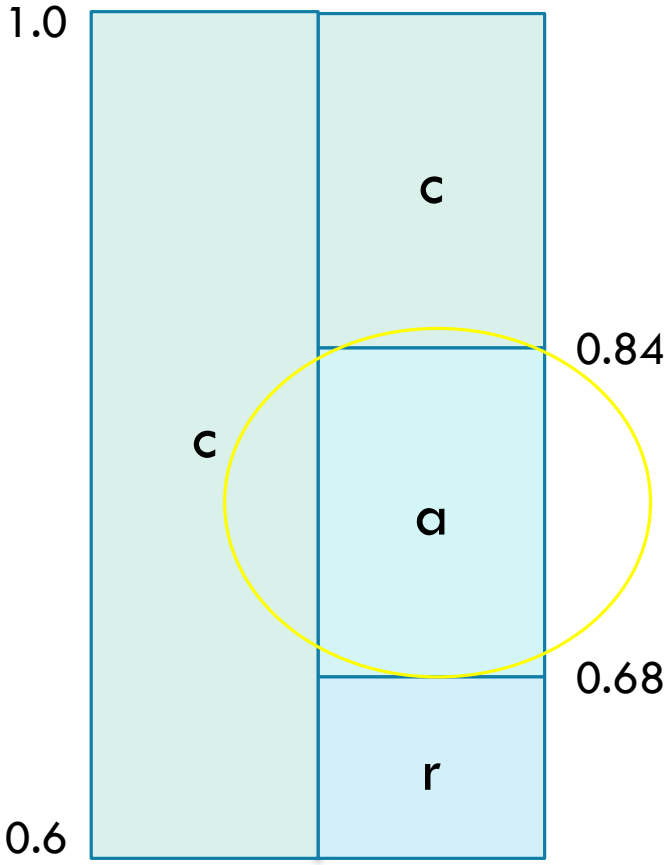
- Assign each symbol to the an interval in range:
  - r : P(r) = 0.2 … [0.0, 0.2)
  - a : P(a) = 0.4 … [0.2, 0.6)
  - c : P(c) = 0.4 … [0.6, 1.0)
- First symbol of message is c, so new interval is [0.6, 1.0)

1st symbol = "c"
Interval with "c": [0.6-1.0)



2nd symbol = "a"

P(r) = 0.2, P(a) = 0.4, P(c) = 0.4



r: .2*.4 = 0.08

So cr interval is represented by: [0.6, 0.6+.08) = [0.6, 0.68)

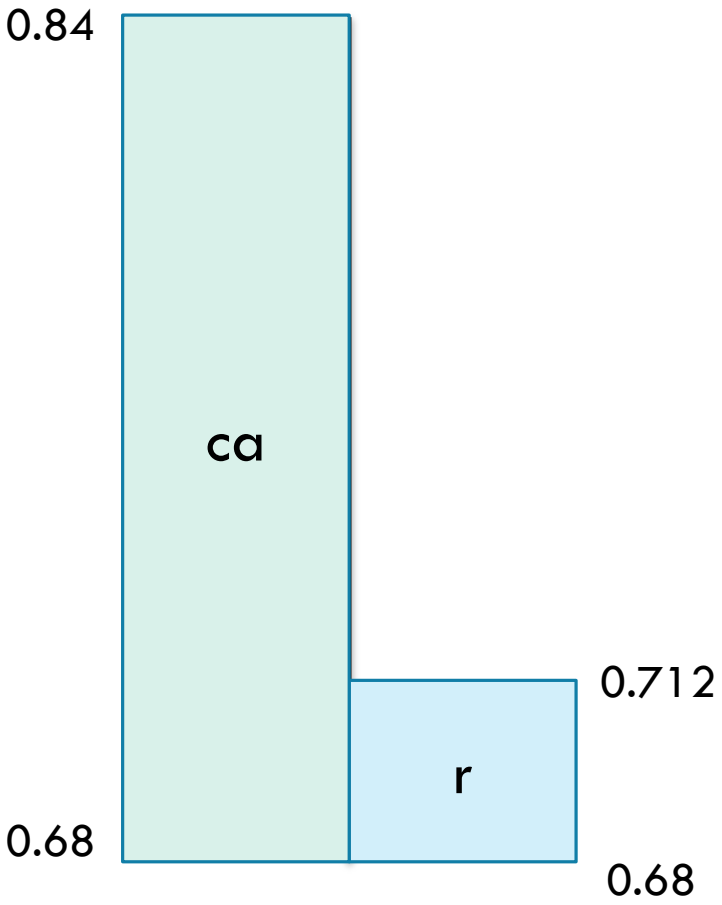
a: .4*.4 = 0.16

So ca interval is represented by: [0.68, 0.68+0.16) = [0.68, 0.84)

c: is remainder of the interval [0.84, 1.0)

3rd symbol = "r"

P(r) = 0.2, P(a) = 0.4, P(c) = 0.4



r: .2*.16 = 0.032

So car interval is represented by: [0.68 + .68+.032) = **[0.68, 0.712)**

Don't need to do any more work …

Interval with "car" : [0.68,0.712)

PRACTICAL CONCERNS …

The longer the text stream (sequences/message) to encode the more precise the interval to encode it becomes but machines have finite precision.

The solution is to limit the length of strings encoded at any one time … so that strings of a certain length are encoded where the maximum length is determined by the precision available.

INTERVAL SENT?

In reality rather than transmit the interval (2 numbers), a real-valued number (or rather binary representation of if) is transmitted instead

With this approach, for decoding, the length of the string is needed also (fixed length can be used so that it only has to be transmitted once)

e.g., for interval [**0.68**, **0.712**)

Some number around 0.71 can be transmitted, also knowing string is of length 3

What is 0.71… in binary?

EXAMPLE 7:

Input:

- symbols are: A, B, C
- probabilities are P(A) = 0.5, P(B) = 0.25, P(C) = 0.25
- Message is CAB

First iteration of loop:

- Assign each symbol to the an interval in range:
  - B : [0.0, 0.25)
  - C : [0.25, 0.5)
  - A : [0.5, 1.0)
- First message symbol is C, so new interval is ???
- etc.

ALGORITHM: DECODING

Input: binary number and symbol length

(assuming symbols, their order and frequencies known)

General Approach: Get real number, follow the same approach as for encoding but at each stage consider next current digit to find correct interval

Stop when at sub-interval of the correct length

Output: symbols from message

EXAMPLE 8:

Inputs:

- Symbols are a, c, r
- Symbols are ordered as: r a c based on the probabilities of: P(r) = 0.2, P(a) = 0.4, P(c) = 0.4
- Compressed message is represented by binary

0.01 and all messages are of length 3 (3 symbols)

- What is decompressed message?

SUMMARY: IMPORTANT TO KNOW

Compression: encoding and decoding

Variable length and fixed length codes

Prefix Codes

Two text approaches based on letter frequencies:

- Huffman Coding
- Arithmetic Coding
- Worked examples for encoding and decoding using both techniques