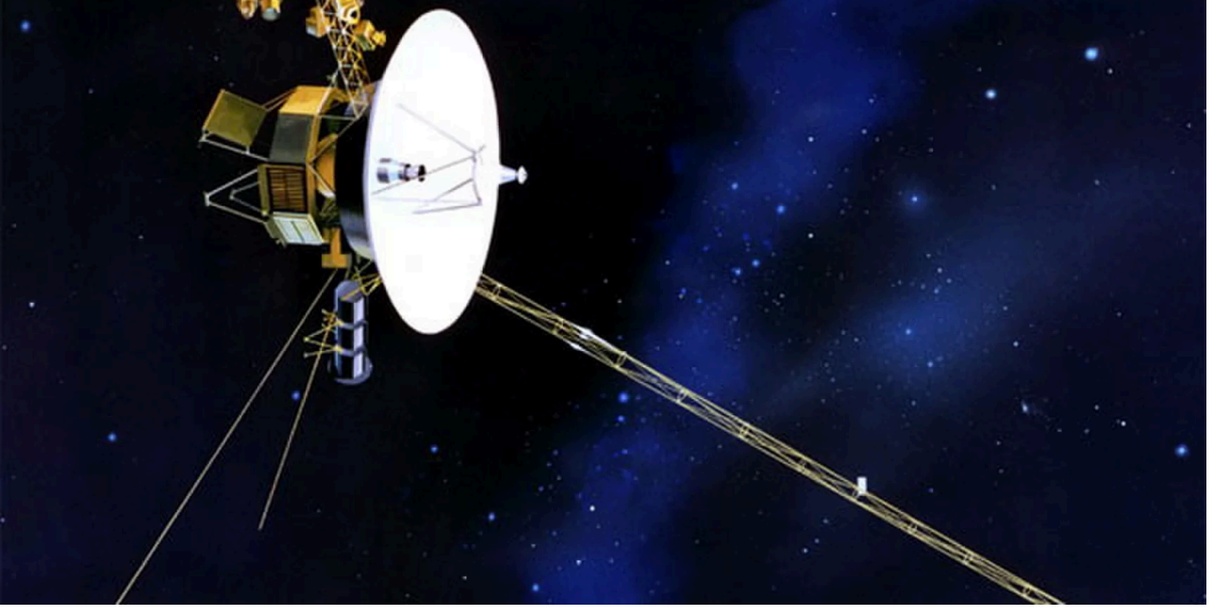
# Efficient Data Representation

**Big Ideas**
Checksums
Binary Trees
Hashing
Compression
Entropy

## Learning Outcomes

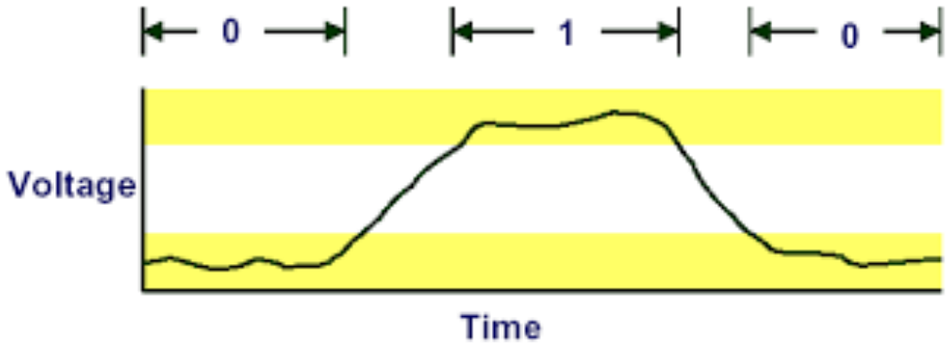
You should be able to:

- Explain the use of parity and checksums
- Explain the connection between hashing and checksums
- Explain the link between entropy and compression
- Calculate the information content of simple messages
- Construct a simple Huffman code



# Alien Alert System

000 Aliens contacted, RUN!
101 No aliens sighted
011 Unknown object sighted
110 All systems OK




How do we know we received the code correctly?

# ASCII parity bits

**ASCII character A: 1000001 (7 bits)**

Even parity: 010000001

Added a 0 to maintain balance of 1 bits.

**ASCII character C: 1000011 (7 bits)**

Even parity: 11000011

Added a 1 to balance the 1 bits

Ensures the byte has an even number of one bits.

# Error Detection - Parity

Spotting data transmission errors

Count number of ‘set’ bits

Even or Odd?

Set parity bit accordingly

Only useful for detecting errors

| 7 bits of data | (count of 1-bits) | 8 bits including parity | |
|---|---|---|---|
| | | even | odd |
| 0 | 0 | 0000000**0** | 0000000**1** |
| 1010001 | 3 | 1010001**1** | 1010001**0** |
| 1101001 | 4 | 1101001**0** | 1101001**1** |
| 1111111 | 7 | 1111111**1** | 1111111**0** |

## Odd numbers of errors

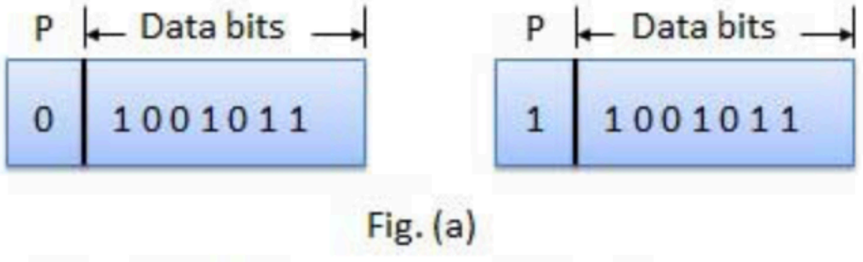



Fig. (a)

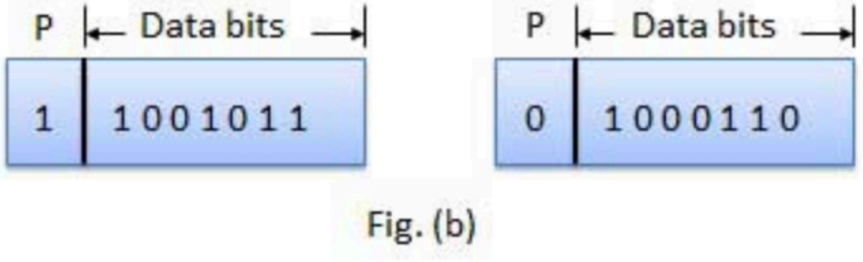
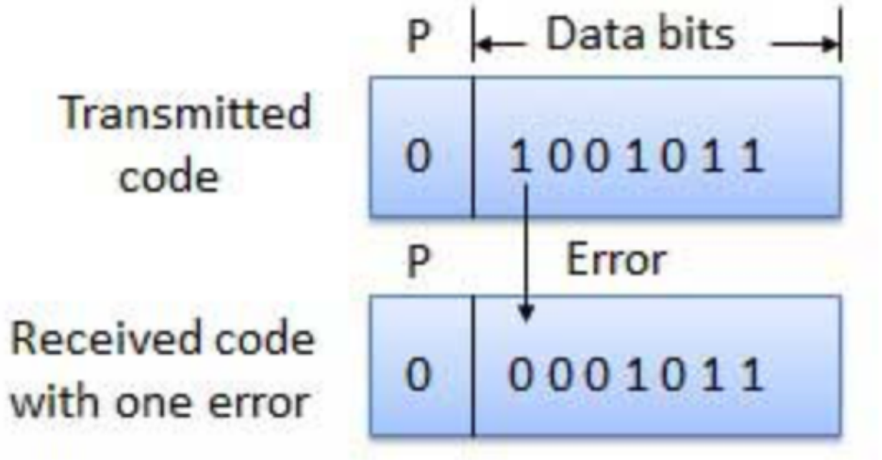



Fig. (b)



# Checksums

A **checksum** is a small-size datum from a block of digital data for the purpose of **detecting errors** which may have been introduced during its transmission or storage.
Checksum can be recalculated to check for errors

Unix cksum command

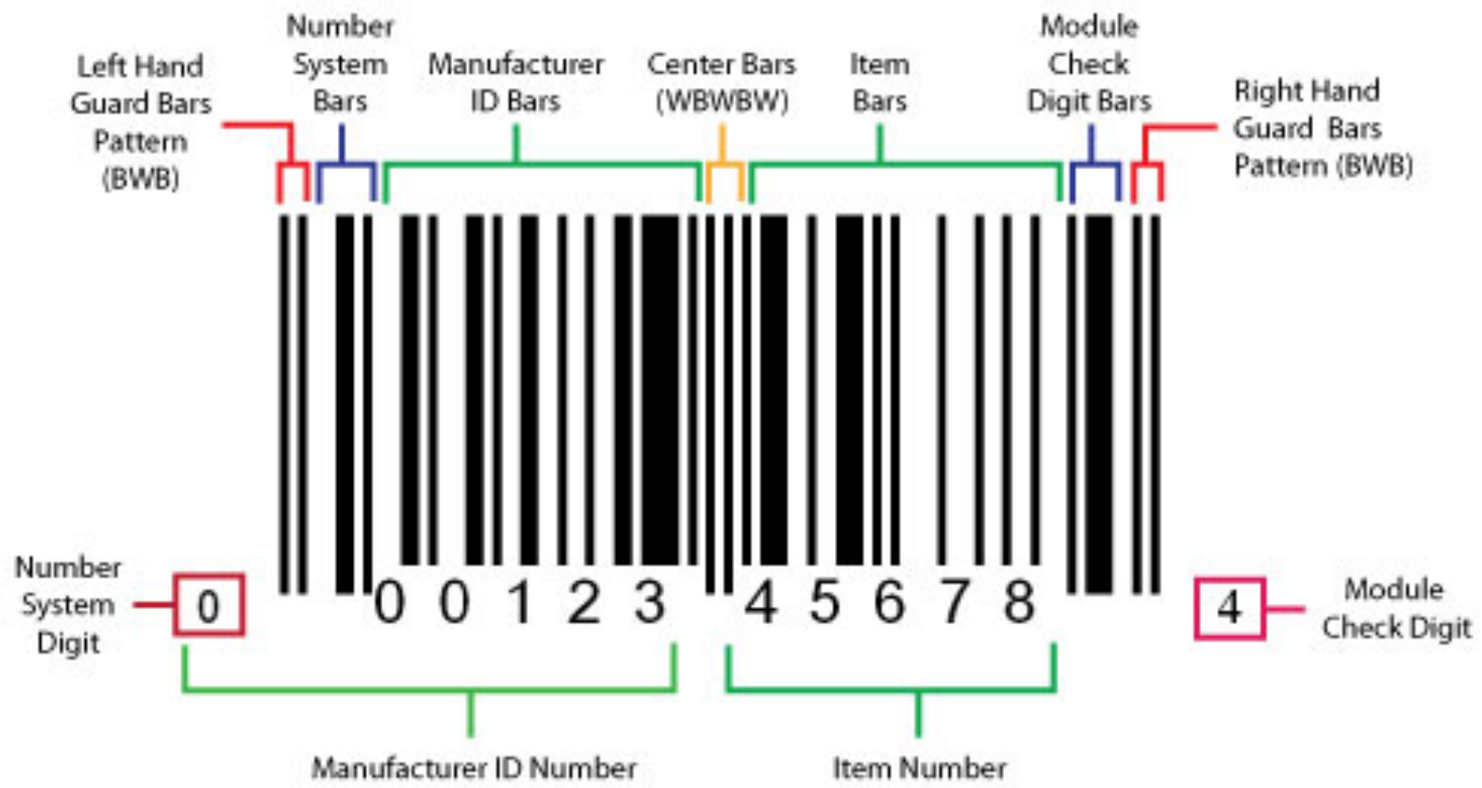
$ cksum  <filename>

```
MacBook-Air-2:workspace ladymead$ cksum
mysecretmessage.txt
2549616871 16 mysecretmessage.txt
```

Other unix commands

- pwd
- ls
- ls -la
- more <filename>
- grep <token> <file>
- cd
- cd ..
- man
- cat <filename>
- less <filename>

# Barcodes, Credit Cards, Bank Acc Numbers



# Checksums Python

## Hash Function

A function that maps data of arbitrary size to data of fixed size.
Hashes can be used as checksums
+ loads of other uses
MD5 https://en.wikipedia.org/wiki/MD5

**Hashing**

```
In [ ]: import hashlib
```

```
In [68]: Myhash = hashlib.md5(b'The quick brown fox jumped over the lazy dog.')
         Yourhash = hashlib.md5(b'The quick brown fox jumped over the lazy dog.')
         Hishash = hashlib.md5(b'the quick brown fox jumped over the lazy dog.')
         Hishash
```

```
Out[68]: <md5 HASH object @ 0x104383cb0>
```

```
In [69]: print ("Myhash   %s \nYourhash %s \nHishash  %s" %
                (Myhash.hexdigest(),Yourhash.hexdigest(),Hishash.hexdigest()))
```

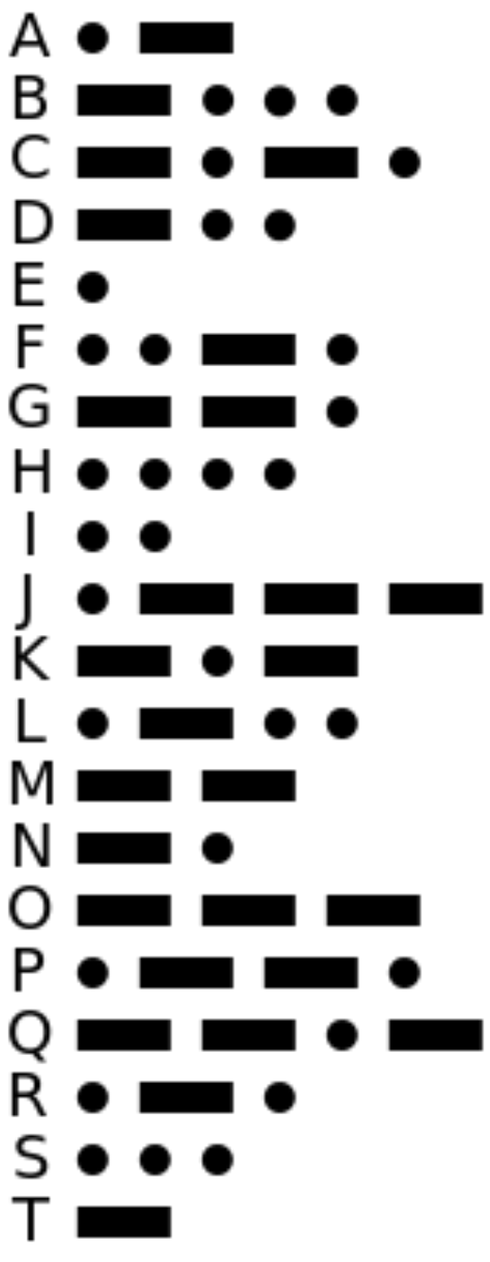
```
Myhash   5c6ffbdd40d9556b73a21e63c3e0e904
Yourhash 5c6ffbdd40d9556b73a21e63c3e0e904
Hishash  bb0fa6eff92c305f166803b6938dd33a
```

# Morse Code

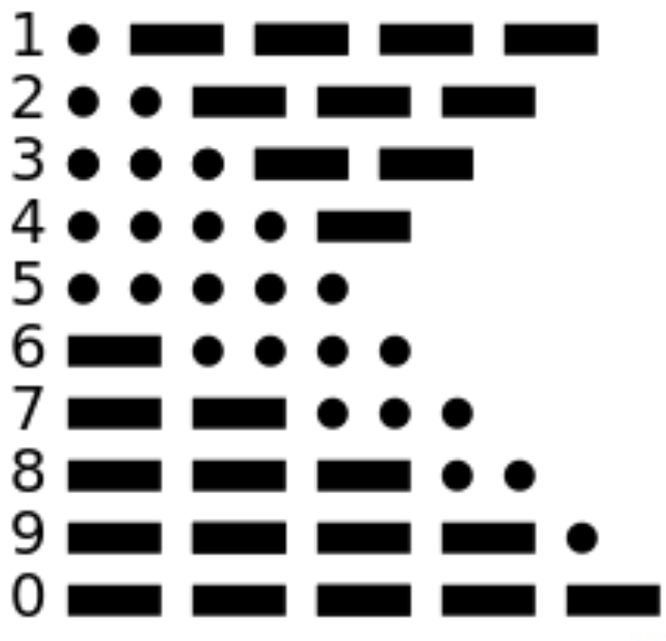
Transmit text by series of tones, lights or clicks.

## International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.



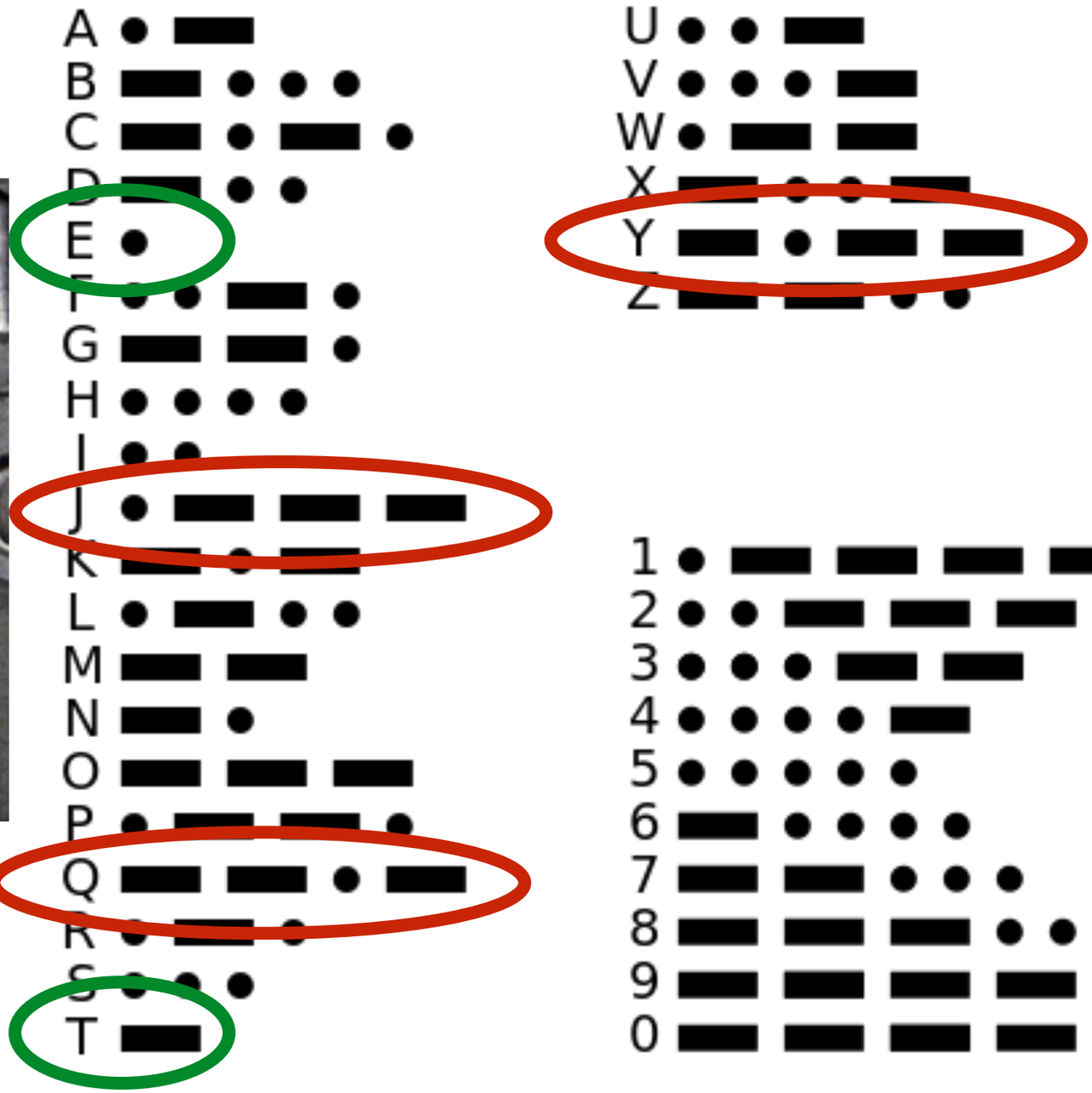




# Morse Code

## International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.



# Information Theory for Dummies

Consider a game where I toss a coin and throw a die and you have to give me €5 if I get  

Which event is more ‘interesting’ (provides more information)?

the coin or the die?

A fair coin toss or a weighted one?

A fair coin toss or double sided coin?

The source with the most potential for surprise has the most information

# Claude Shannon - Information Theory

The Bell System Technical Journal

*Vol. XXVII* *July, 1948* *No. 3*

## A Mathematical Theory of Communication

By C. E. SHANNON

### Introduction

THE recent development of various methods of modulation such as PCM and PPM which exchange bandwidth for signal-to-noise ratio has intensified the interest in a general theory of communication. A basis for such a theory is contained in the important papers of Nyquist[1] and Hartley[2] on this subject. In the present paper we will extend the theory to include a number of new factors, in particular the effect of noise in the channel, and the savings possible due to the statistical structure of the original message

**bit = 0 or 1**

**measure of information**

All of the books in the world contain no more information than is broadcast as video in a single large American city in a single year.
**Not all bits have equal value.**

**Carl Sagan**

Bit = Uncertainty divided by 2

# Redundancy in English text

"…randomising letters in the middle of words [has] little or no effect on the ability of skilled readers to understand the text. This is easy to denmtrasote. In a pubiltacion of New Scnieitst you could ramdinose all the letetrs, keipeng the first two and last two the same, and reibadailty would hadrly be aftcfeed. My ansaylis did not come to much beucase the thoery at the time was for shape and senqeuce retigcionon. Saberi's work sugsegts we may have some pofrweul palrlael prsooscers at work. The resaon for this is suerly that idnetiyfing coentnt by paarllel prseocsing speeds up regnicoiton. We only need the first and last two letetrs to spot chganes in meniang."

# Information Entropy

| | Raw data | Information? |
|---|---|---|
| Fair coin  | **1 bit** | **1 bit** |
| Double headed coin  | **1 bit** | **0 bit** |
| Fair dice  | **3 bits** | **3 bits** |
| Loaded dice  | **3 bits** | **0 bits** |

We can use the probabilities to reduce the number of bits needed —if the variable has a nonuniform distribution.

# Weather App

50%

50%

1 bit of information

* regardless of encoding

# Weather App

100%

0%

0 bit of
information

# Weather App

25%

25%

25%

25%

# Reducing uncertainty by 4

2 bits of information

# Weather App

75%

25%

-log2(0.25) = 2

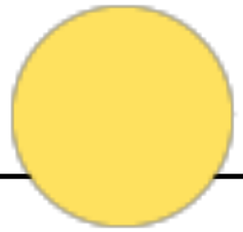
2 bits of information

# Weather App

75%

25%

-log2(.75) = 0.41 bits



0.41 bits of information

# Weather App

75%

25%

Average information every day

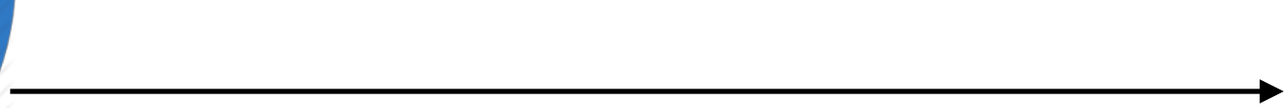
75% x 0.41
+ 25% x 2 = 0.81 bits

$$H = -\sum p(x) \log p(x)$$

0.81 bits of information on average

# Weather App: Encoding

| 01 | 10 | 11 | 00 |
| --- | --- | --- | --- |
|  |  |  |  |
| 75% | 15% | 5% | 5% |



# Weather App: Efficient Coding

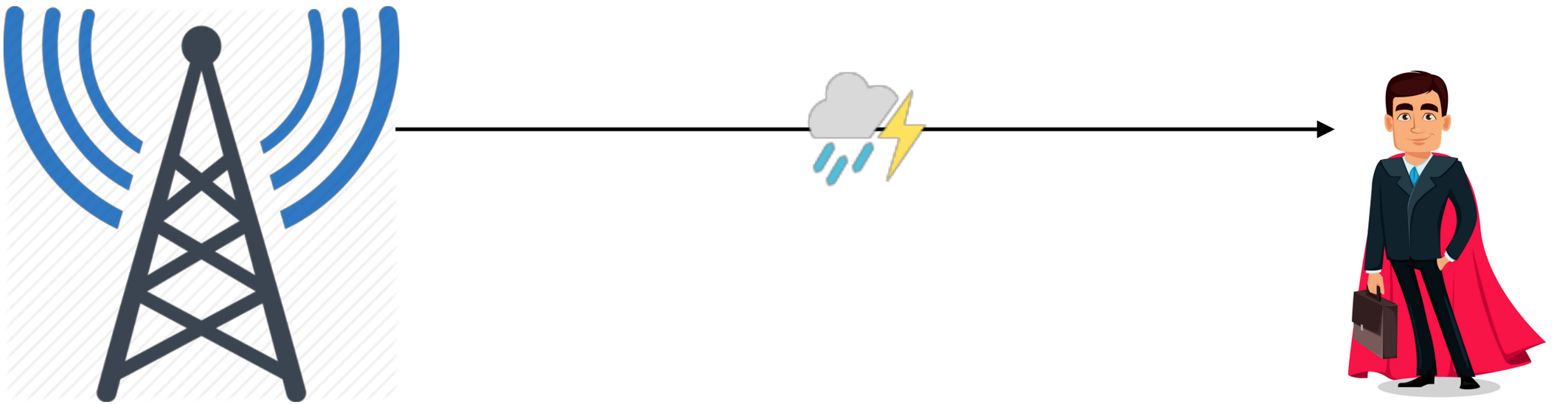
| 0 | 10 | 110 | 111 |
|---|---|---|---|
|  |  |  |  |
| 75% | 15% | 5% | 5% |

# Efficient coding accounts for the probabilities

# Weather App - Efficient Encoding

| 75% | 10% | 10% | 5% |
|---|---|---|---|
|  |  |  |  |
| 01 | 10 | 11 | 00 |

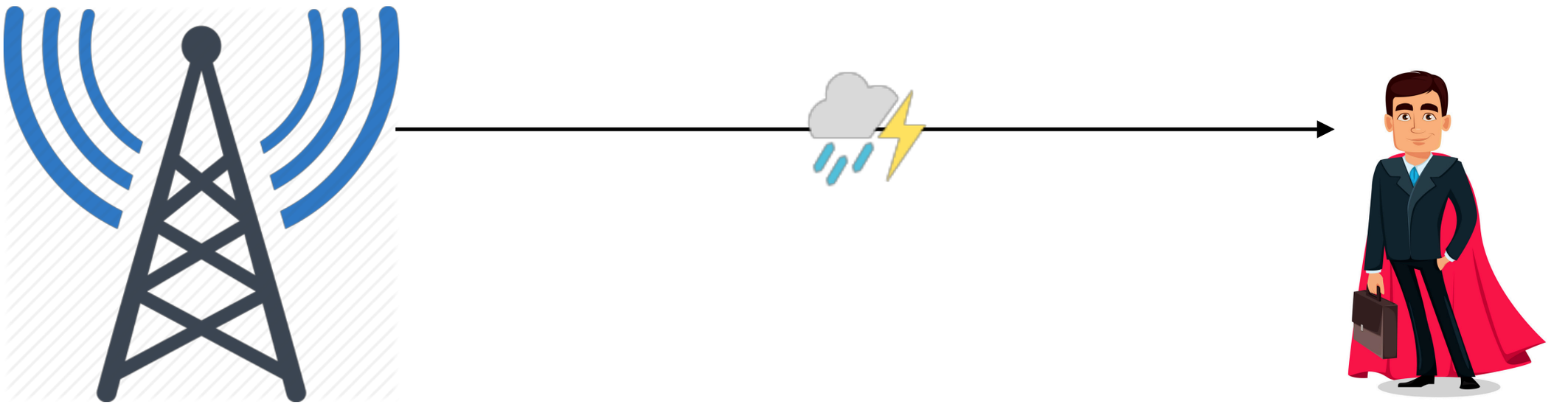
We can use the underlying probability of the events to create more efficient communication codes.



# Weather App - Efficient Encoding

| 75% | 13% | 6% | 6% |
|---|---|---|---|
|  |  |  |  |
| 0 | 10 | 110 | 111 |

We can use the underlying probability of the events to create more efficient communication codes.



# Information Theory

How many bits required to communicate:

the result of a coin toss?

the result of a throw of a die?

the throwing of a six?

**Compression**
Exploit runs of 0s to transmit in less than 1 bit.

# Entropy - 2 outcomes

$S$ is a sample of events

$p$ the proportion of positive examples in $S$

$q$ the proportion of negative examples

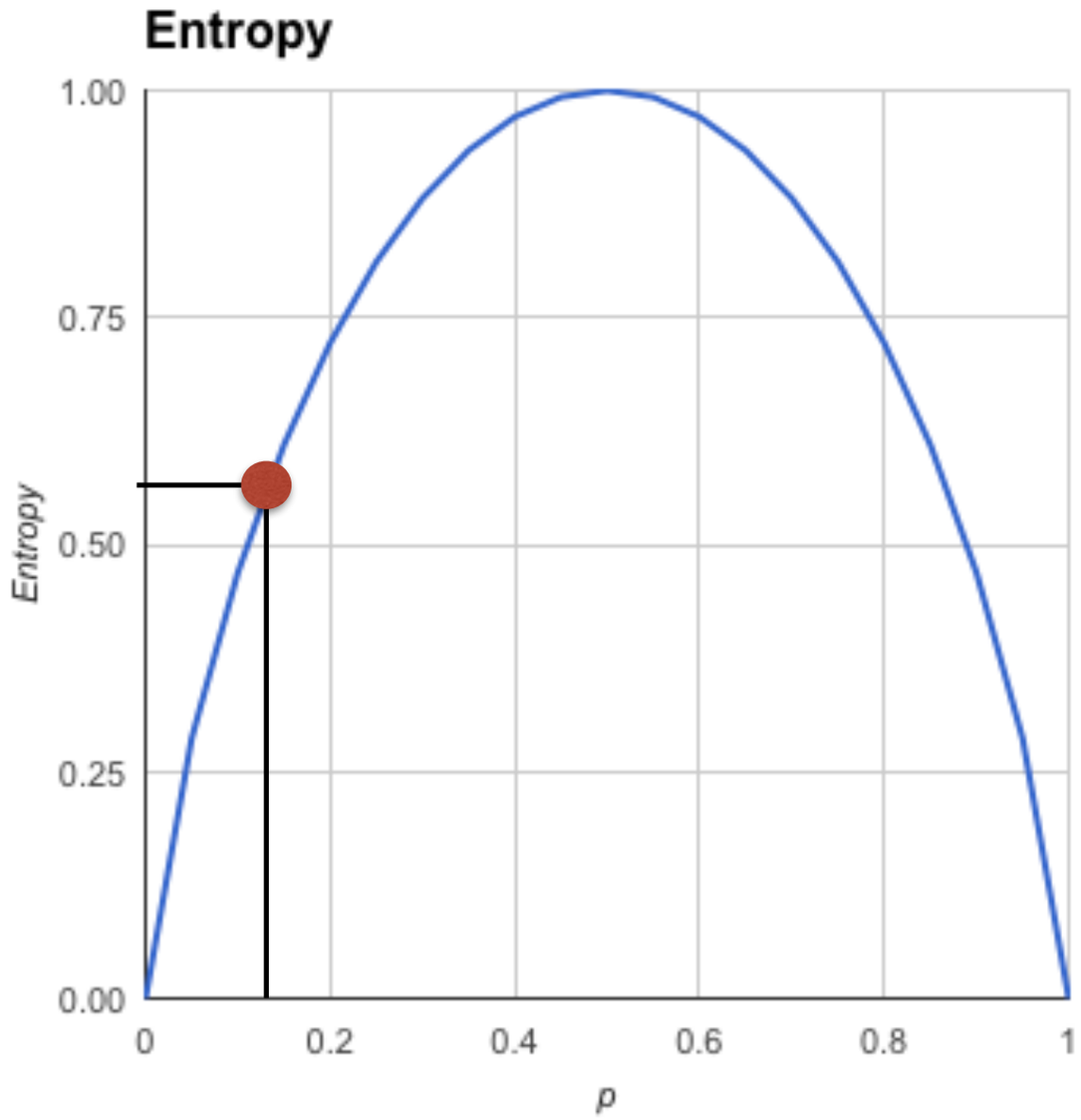
Entropy measures the impurity of $S$

$$Entropy(S) = -p\log_2(p) - q\log_2(q)$$

Examples:

Coin toss: p = q = 0.5: Entropy = 1

Throw six: p= 0.17: Entropy = 0.66

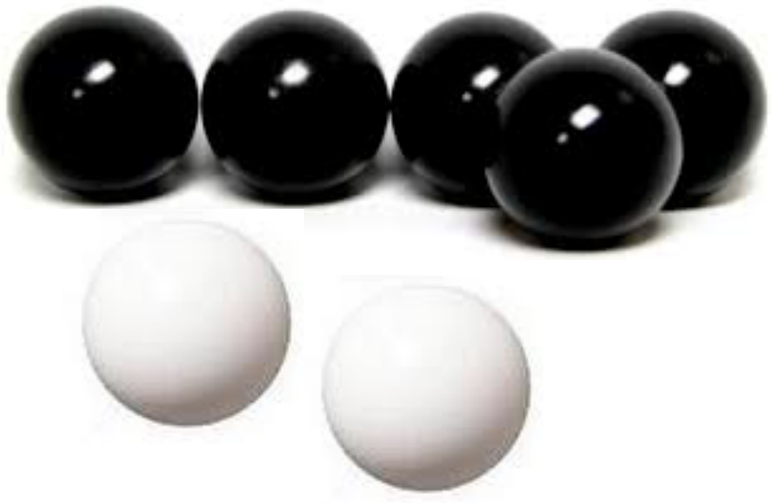



See Wikipedia page for an explanation of the entropy formula:

http://en.wikipedia.org/wiki/Entropy_(information_theory)

Entropy effectively bounds the performance of the strongest lossless compression possible

# Balls example



A bag contains 7 balls, 5 black and 2 white

Balls are selected at random and replaced

How many bits are required to report each colour in the sequence, e.g.

B B W B W W B B B W B B B…

Use the Entropy calculation spreadsheet on the Moodle page

What if one of the white balls is replaced with another black?

# Run-Length Encoding

Simple strategy for loss-less compression

Consider:

WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWW
WWWWWWWWWWWWBWWWWWWWWWWWWWW

Represent this as:

12W1B12W3B24W1B14W

Requires a probability imbalance to be effective

i.e lowish entropy

Predictability = compressible

# Compression

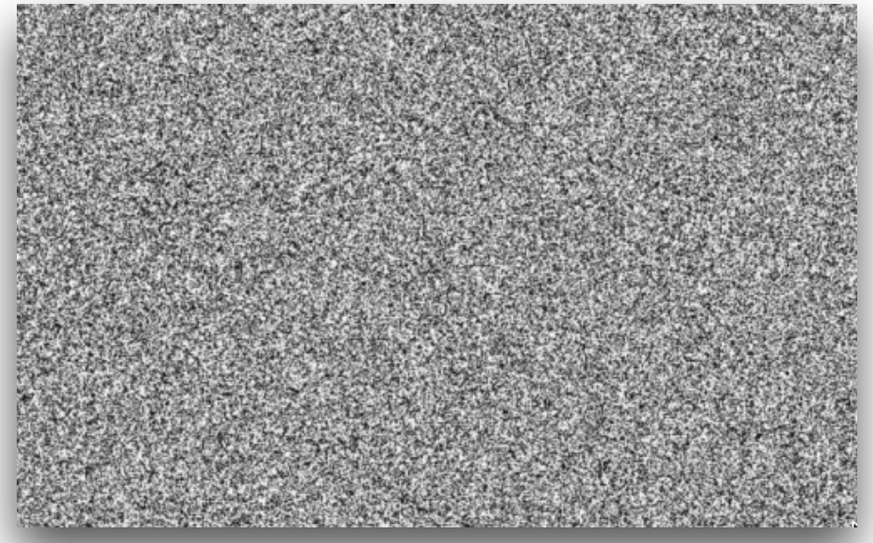
Encode information using fewer bits than original representation

Lossless: Zip

https://en.wikipedia.org/wiki/Zip_(file_format)

1 Bar.tiff

2 WhiteNoise.tiff



Which image will be more compressible (without loss)?

| | | |
|---|---|---|
| 1 Bar.tiff | Today, 10:56 | 6 KB |
| 1 Bar.tiff.zip | Today, 11:36 | 2 KB |
| 2 WhiteNoise.tiff | Today, 10:54 | 319 KB |
| 2 WhiteNoise.tiff.zip | Today, 11:36 | 317 KB |

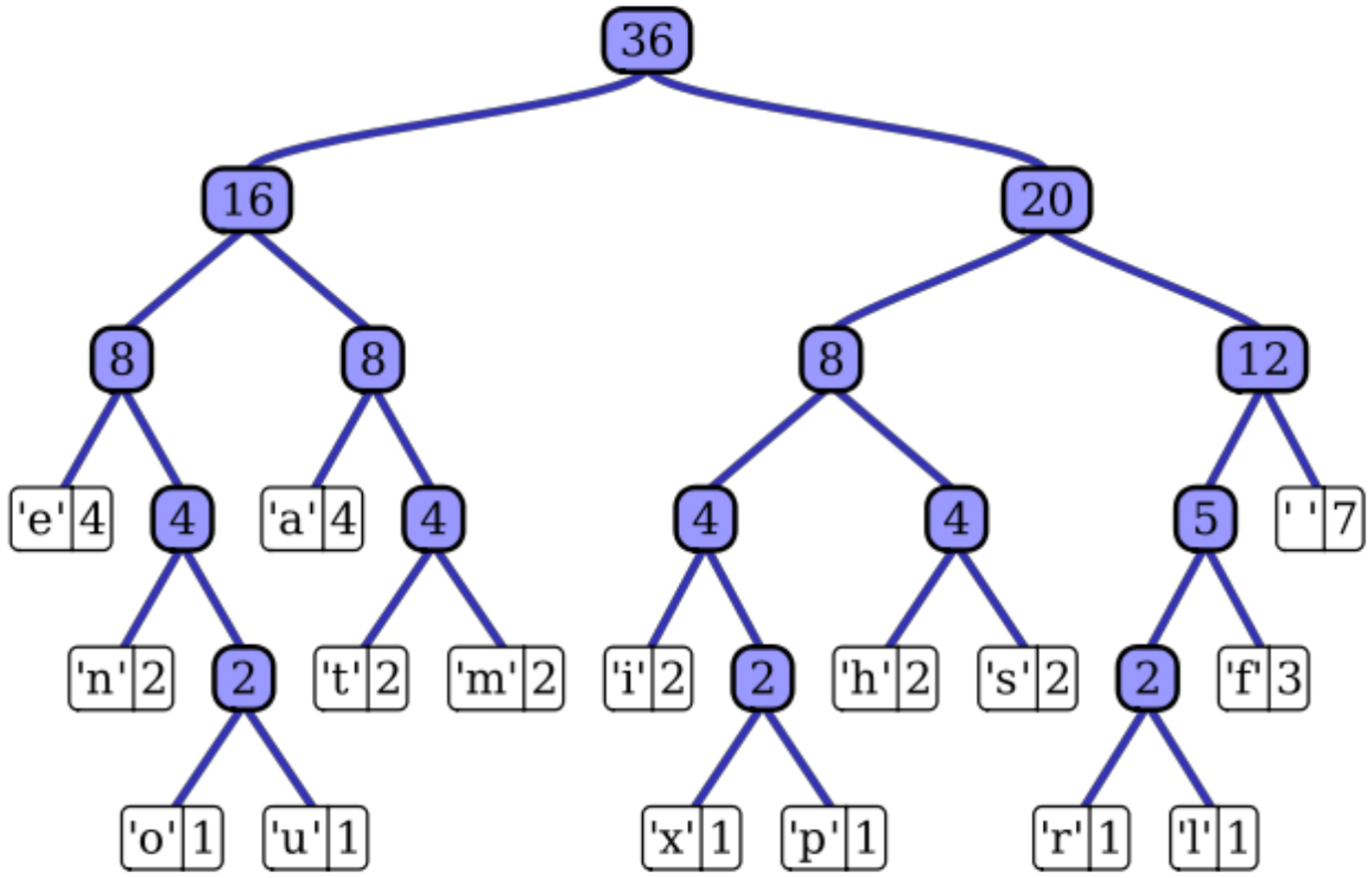
# Lossy Compression - jpg

# Compression

Lossy: JPEG, MPEG
Lossless: Zip
https://en.wikipedia.org/wiki/Huffman_coding

**Huffman Coding**
Sample text:

- ”this is an example of a huffman tree”




## Huffman Coding

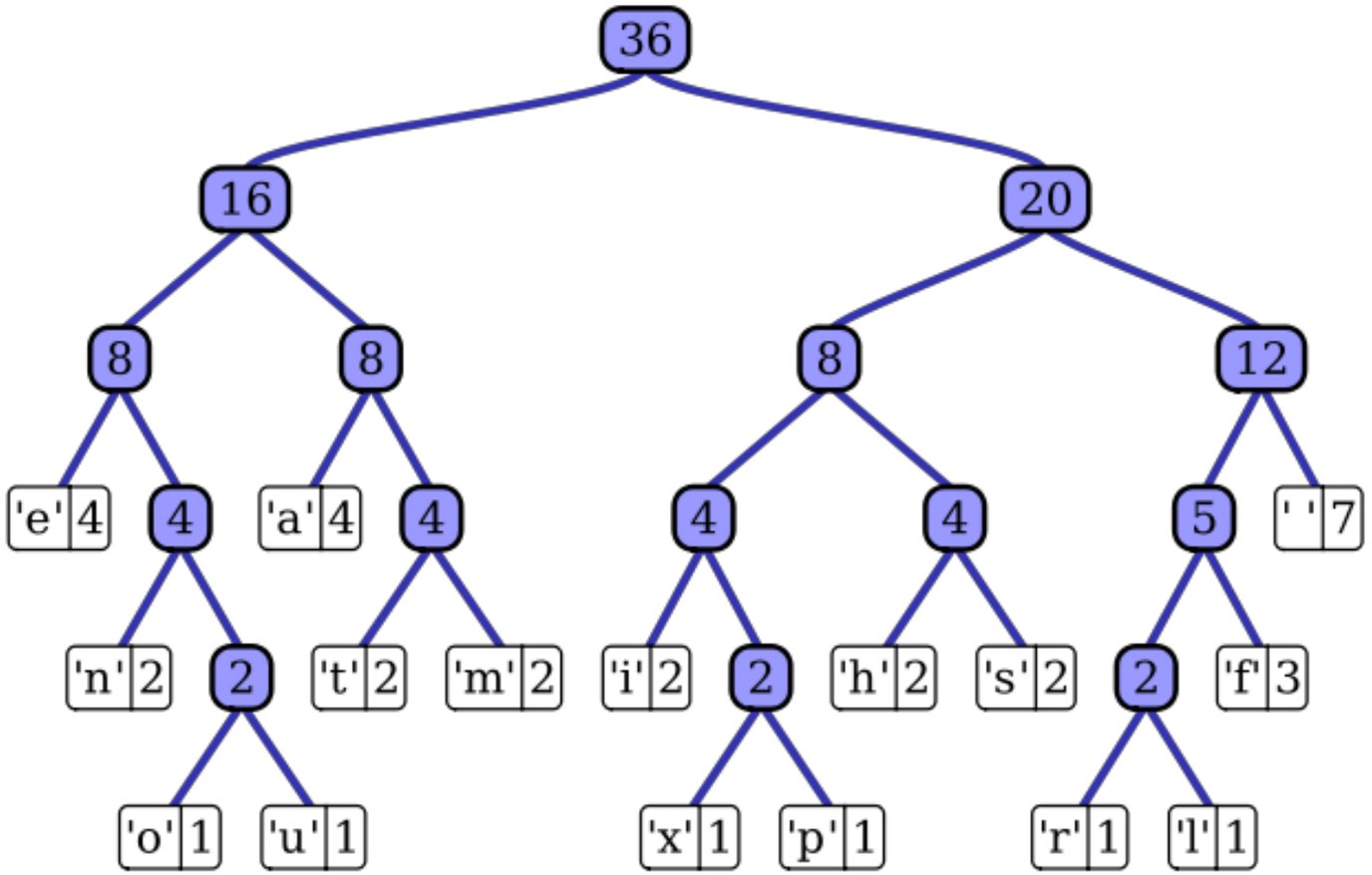
| Char | Freq | Code |
|---|---|---|
| space | 7 | 111 |
| a | 4 | 010 |
| e | 4 | 000 |
| f | 3 | 1101 |
| h | 2 | 1010 |
| i | 2 | 1000 |
| m | 2 | 0111 |
| n | 2 | 0010 |
| s | 2 | 1011 |
| t | 2 | 0110 |
| l | 1 | 11001 |
| o | 1 | 00110 |
| p | 1 | 10011 |
| r | 1 | 11000 |
| u | 1 | 00111 |
| x | 1 | 10010 |

# Compression

How does the tree work?

What is?

01110011000110001011110111001100110




## Huffman Coding

| Char | Freq | Code |
|---|---|---|
| space | 7 | 111 |
| a | 4 | 010 |
| e | 4 | 000 |
| f | 3 | 1101 |
| h | 2 | 1010 |
| i | 2 | 1000 |
| m | 2 | 0111 |
| n | 2 | 0010 |
| s | 2 | 1011 |
| t | 2 | 0110 |
| l | 1 | 11001 |
| o | 1 | 00110 |
| p | 1 | 10011 |
| r | 1 | 11000 |
| u | 1 | 00111 |
| x | 1 | 10010 |

# Building Huffman Trees

Low frequency chars should be far from the root
i.e. longer codes

1. Each character is assigned a leaf node

2.Select the two lowest frequency character nodes
1.Link with an internal node and sum their frequencies to get the frequency for the internal node.

3.Repeat from 2 till there is just one node

**Ties:**

Resolve ties randomly

Sometimes many equally good trees
One *n*-character code is as good as another *n*-character code

# Efficient Data Representation

**Big Ideas**
Checksums
Binary Trees
Hashing
Compression
Entropy

## Learning Outcomes

You should be able to:

- Explain the use of parity and checksums
- Explain the connection between hashing and checksums
- Explain the link between entropy and compression
- Calculate the information content of simple messages
- Construct a simple Huffman code