Topic 5 Fundamentals of Data Representation

Number types

Natural Numbers

Natural numbers are a set of numbers which are all whole numbers and either positive or zero. They are often used to count how many of something there are, for example, 2 dogs, 3 cats and 0 rabbits. The symbol for natural numbers is $\mathbb{N}$.

$\mathbb{N} = \{0,5,22,3,1\}$

Integer Numbers

Integer numbers are whole numbers which are positive, negative or zero. The symbol for integer numbers is $\mathbb{Z}$

$\mathbb{Z} = \{-1,-20,0,5,41,3,1\}$

Rational numbers

Rational numbers can be either a whole number or a number with a decimal point or fraction. They must be able to be written exactly as a fraction and can be positive, negative or zero. They are also known as quotients and are represented with the symbol $\mathbb{Q}$.

$\mathbb{Q} = \{½, 55, 0, 6.34234, ¼, 8\ ¾\}$

Irrational numbers

Irrational numbers cannot be written as a whole fraction and do not have a symbol to represent them.

$IrrationalNumbers = \{ \pi, \sqrt{2}, e, \sqrt{3}\}$

Real Numbers

Real numbers incline all the number types mentioned above and are represented using the symbol $\mathbb{R}$

Ordinal Numbers

Ordinal numbers are integers which show the position of numbers or objects in a sequence such as 1st, 3rd or 252nd. Ordinal numbers are used to index items in arrays starting with position 0.

Counting and Measuring

When counting objects, for example cars, books or files, natural numbers should be used as the result can only be a whole number. When measuring something, for example the area of a circle, the quantity may not come out to a whole number and as such real numbers should be used.

Number Bases

The same number can be shown or written in several different ways. Most often base 10, also called denary or decimal, is used but a number of different options exist. Any given number has the same value no matter how it is written, it will just be written differently.

Different number systems use a different number of digits in each character place. The more characters the number system uses, the less space taken up to represent a given number. As an example, the number 12 can take 2 characters to write in decimal (12), 4 in binary (1100) and only 1 in hexadecimal (C)

© 2025 Exam Papers Practice. All Rights Reserved

Decimal (base 10)

This is the number system most commonly used by people. It uses ten digits, 0 to 9 to represent numbers. Decimal numbers can be written with a subscript 10 to indicate they are in denary.

61_{10}

Binary (Base 2)

Binary uses two characters, 0 and 1. These can be easily processed by computer circuits as on (1) or off (0). A subscript 2 can be used to indicate that a number is written in binary.

0111011_{2}

Hexadecimal (Base 16)

Hexadecimal used 16 characters, 0-9 followed by A-F in uppercase to represent numbers. A subscript 16 can be used to indicate that a number is written in hexadecimal.

$A1_{16}$

Converting Between Number Bases

Converting Binary to Decimal

Draw your conversion table.

1) Write the binary number in the conversion table.
2) Add together all numbers with a 1 beneath them

128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
1	1	0	0	1	1	0	0

128 + 64 + 8 + 4 = 204 - 11001100 in binary is 204 in denary

Converting Decimal to Binary

1) Draw your conversion table.
2) Is the number higher than the first column in the table?
    a) If so, put a 1 in that column and work out the difference.
    b) If not, put a 0 in that column.
3) Repeat the step above with the difference.
4) Keep going until the difference is 0, put a 0 in any empty columns.

Read the number from the bottom row of the table

128	64	32	16	8	4	2	1
1	1	0	0	1	0	0	0

200 – 128 = 72
72 – 64 = 8
8 < 32
8 < 16
8 – 8 = 0

200 in denary is 11001000 in binary

EXAM PAPERS PRACTICE

Converting Decimal to Hexadecimal

1) Divide the denary number by 16 and write down both the answer and the remainder.
2) Divide the answer by 16 again. Write down both the answer and the remainder.
3) Keep going until you reach an answer of 0.
4) Read the remainders from bottom to top.
5) Convert each remainder to hex.

62 ÷ 16 = **3** R 14
3 ÷ 16 = **0** R 3
3 14
3E

62 in denary is 3E in hexadecimal

Converting Binary to Hexadecimal

1) Draw two separate conversion tables.
2) Write the binary number across both tables.
3) For each table, add up the numbers which have a 1 beneath them.
4) Convert each number to hexadecimal.

8	4	2	1		8	4	2	1
0	1	1	0		1	1	0	1
4 + 2 = 6 6					8 + 4 +1 = 13 D			

01101101 in binary is 6D in hexadecimal

Bits and Bytes

A bit is the smallest unit of information and can be either a 0 or 1. Computers represent this as either high or low current.

8 bits together are known as a byte and 4 bits, or half a byte are a nybble. Bits are written with a lowercase b whilst a byte used an uppercase B.

The number of bits assigned to a number limits how many values it can represent, more bits allow larger numbers to be shown. The formula to calculate this is 2^n where n is the number of bits. As an example, two bits can represent four numbers ($2^2 = 4$) whilst 10 bits can represent 1024 numbers ($2^{10} = 1024$)

Units

Binary prefixes and decimal prefixes are used to describe quantities of bytes. Binary prefixes go up in powers of two and decimal prefixes increase in powers of 10.
Decimal prefixes are widely used, for example 1000 ml is one litre, whereas binary prefixes are less widely used.

Binary		Decimal	
Prefix	**Value**	**Prefix**	**Value**
Kibi (Ki)	21^0 = 1024	Kilo (K)	10^3 = 1000
Mebi (Mi)	2^{20} = 1048576	Mega (M)	10^6 = 1000000
Gibi (Gi)	2^{30} = 1073741824	Giga (G)	10^9 = 1000000000
Tebi (Ti)	2^{40} ≈ 1.0995×10^{12}	Tera (T)	101^2 = 1×10^{12}

© 2022 Exam Papers Practice. All Rights Reserved

Signed and Unsigned Binary Numbers

Binary numbers can be signed or unsigned, but by looking at the number alone there is no way to know which it is. A computer has to be told whether a number is signed or unsigned. Unsigned binary numbers can only be positive but signed numbers can be negative or positive.

Unsigned Binary Arithmetic

Adding unsigned binary integers

Use the four rules below to add unsigned binary numbers.

1. 0 + 0 + 0 = 0
2. 0 + 0 + 1 = 1
3. 0 + 1 + 1 = 10
4. 1 + 1 + 1 = 11

Multiplying Unsigned Binary Integers

Write out one number as a guide. Underneath each 1 in the first number, write out the second number lining the least significant bit up with the number 1. Finally, perform binary arithmetic on the columns. The example below shows how this works:

			1	**0**	**1**	**1**
			1	**0**	**1**	**0**
		1	**0**	**1**	**0**	0
1	**0**	**1**	**0**	0	0	0
1	**1**	**0**	**1**	**1**	**1**	**0**

Write one of the two numbers out in columns.

Write the second number under every 1 in the 1st number, aligning the least significant but with the 1. Fill in any blanks on the right with 0

Perform binary addition on the columns.

Signed Binary With Two's Complement

AQA uses the two's complement coding scheme for binary which can represent both positive and negative numbers. Two's complement gives the most significant bit a negative value to show a negative number.

Subtracting Using Two's Complement

Computers work by adding numbers, so to perform a subtraction, computers will add negative numbers.

Example: Subtract 12 from 8

	-16	8	4	2	1
	0	**1**	**0**	**0**	**0**
+	**1**	**0**	**1**	**0**	**0**
	1	**1**	**1**	**0**	**0**

Range of Two's Complement Numbers

Two's complement signed binary numbers can include positive and negative numbers in a given range of bits. For example, with 4 bits the largest value possible is 7 and the smallest is -8.

© 2025 Exam Papers Practice. All Rights Reserved

Fractional Numbers in Binary

Binary can also represent numbers which have a fractional part to them. There are two ways of doing this, one uses fixed point whilst another uses floating point.

Fixed Point Binary

The fixed point approach places a specific number of bits before a binary point, with the rest behind the point. The columns before the point use standard binary values of 1,2,4,8,etc. whilst those after the point use the values 1/2, 1/4, 1/8, 1/16, etc.

This example uses 8 bits, split into 4 bits before the binary point and four bits after. This allows us to write the number 11.3125 in binary as 10110101.

8	4	2	1	.	1/2	1/4	1/8	1/16
1	**0**	**1**	**1**		**0**	**1**	**0**	**1**

8 + 2 + 1 + 1/4 + 1/16 = 11.3125

Floating Point Binary

Floating point binary is similar to scientific notation in that numbers are made up of a mantissa and an exponent. Scientific notation would write the number 3,100000 as 3.1×10^6. 3.1 is the mantissa and 6 is the exponent. This approach allocates a number of bits to the mantissa and the rest to the exponent.

0	**1**	**1**	**0**	**1**		**0**	**1**	**1**
Mantissa						**Exponent**		

The steps below show how to convert a floating point binary number to decimal.

0 1 1 0 1 0 1 1	
0 1 1 0 1 3	Convert the exponent to decimal
0. 1 1 0 1 3	Place a binary point between the first and second numbers
0 1 1 0. 1	
0 1 1 0. 1	Treat the mantissa as a fixed point binary number and convert to decimal. 4 + 2 + 1/2 = 6.5
8 4 2 1. 1/2	

© 2025 Exam Papers Practice. All Rights Reserved

These steps show how to convert from decimal to floating point binary.

Convert 14.625 to floating point binary.

8 4 2 1 . 1/2 1/4 1/8	Convert the number to fixed point binary
1 1 1 0 . 1 0 1	
16 8 4 2 1 . 1/2 1/4 1/8	If the number does not begin 01 for a positive number or 10 for a negative number add the required digit at the start
0 1 1 1 0 . 1 0 1	
0 . 1 1 1 0 1 0 1	Move the binary point to between the first two digits, making a note of how many places it has moved.
01110101 0100 = 14.625	The number of places the point was moved, in this case 4 or 0100 in binary.

Comparing Fixed and Floating Point

Floating point allows a greater range of numbers to be represented in the same number of bits because it can take advantage of an exponent which can be either positive or negative. The number of bits allocated to each side of the floating point number affect the range of numbers which can be represented. A large exponent allows for a large range of numbers but limits precision whilst a large mantissa allows for good precision but a small range.

The placement of the binary point in fixed point also affects the balance between range and precision. Placing the point close to the left gives good precision but limits how many numbers can be represented. Moving the point to the right allows more numbers to be represented but limits precision.

Normalisation Floating Point Numbers

Normalising floating point numbers provides the maximum available precision within a range of bits. It requires making sure the number starts with 01 if it is positive and 10 if negative. To normalise a floating point number:

1. Split the number into the mantissa and exponent.
2. Adjust the mantissa so that it begins 01 if positive or 10 if negative by moving the bits as required.
3. Reduce the exponent by the same number of bits as you moved in step 2.

Underflow and Overflow Errors

Underflow Errors

Underflow happens when a small number needs to be represented but there are enough bits available to do so. For example, 0.15625 could be written in fixed point binary using seven bits as 0000001, however, if only five bits were available it would be 00000.

Overflow

Overflow happens when a number is too large to be represented using the available number of bits and is particularly important when working with signed binary.

Rounding Errors

It is not possible to represent every single decimal number exactly in binary, in the same way that 1/3 cannot be wholly accurately repressed in decimal. This means that numbers at some point must be rounded and that fixed point and floating point numbers may not be 100% accurate.

Absolute and Relative Errors

Calculating the absolute and relative errors allow us to see how close a particular representation of a number is to the actual value.

Absolute Error Calculation

The absolute error is the actual amount by which a value is incorrect. It can be calculated by finding the difference between the given value and the actual value.

Relative Error Calculation

The relative error is a measure of the uncertainty in a given value when compared to the actual value. This is calculated using the formula $relative\ error = \frac{absloute\ error}{actual\ value}$. The result is a decimal and can be multiplied by 100 to give a percent.

Errors in Relation to Magnitude

The impact of an error is greater in larger numbers. As an example, an error of 0.5% when measuring 20cm would only be a difference of 0.5mm, whereas an error of 0.5% when measuring 5KM would be 25m.

Representing Characters

Character sets allow characters to be represented by a number, allowing computers to work with characters despite only being able to work with binary numbers. Each character is assigned a numeric character code which is unique to that character. Character codes can be written in either decimal or binary.

ASCII & Unicode

ASCII was introduced in 1963 and used 7 bits to represent 128 characters including basic symbols, the numbers 0-9 and the letters a-z in upper and lower case. Because ASCII is limited to 128 characters, it does not have the space to represent characters from languages such as Arabic or Hebrew which use a wide array of characters instead of the letters A-Z.

Unicode was introduced in 1991, and depending on the version, uses between 8 and 48 bits which allows it to include many more characters than ASCII. Unicode includes not only symbols, numbers and letters from many different languages such as Chinese and Greek, but also small pictorial icons known as emojis.

Error Checking

When computers share data, there is the possibility that parts of the data can be corrupt or lost, causing errors in the final data. To spot this, and prevent incorrect data being processed, a number of error checking and correction processes exist.

Parity Bits

Parity bits are single bits added into a transmission to check for errors. The sender calculates the value based on the data itself and attaches it to the data before transmission. The receiver receives the data and runs the same calculation, if the bits match, then the data is correct, if they are different then the data contains errors, and the receiver will ask for the data to be resent.

Even parity chooses a value of the parity bit to create an even number of 1s in the transmission. For example, 001001 would have a parity bit of 0 because there are already an even number of 1s. On the other hand 111011 would have a parity bit of 1 because there are an odd number of 1s.

Odd parity works in the same way, but instead uses the parity bit to create an odd number of 1s in the message. As an example, 001001 would have a parity bit of 1 because there are an even number (2) of 1s, and 111011 would have a parity bit of 0 because there are already an odd number (5) of 1s.

Data	Even Parity Set	Data Received	Parity Check
1011	**10111**	10111	No Error Detected
0000	**00000**	00100	Error Detected
1000	**10001**	11001	Error Detected
1001	**10010**	11110	No error detected

The example above shows this process in action. The last row shows the main issue with using parity bits. In this example an even number of bits have changed during transmission (1001 became 1111) but because there is still an even number of 1s, the parity check passes.

This shows that parity bits cannot detect errors where an even number of bits are changed during transmission.

Majority Voting

In Majority Voting each bit is transmitted more than once and when the data is received, the most commonly occurring value is taken to be correct. This means that majority voting not only detects errors but corrects them too. It also has the advantage over parity bits of being able to detect when multiple bits have changed.

The main disadvantage of majority voting is that it increases the amount of data which needs to be transmitted to share the message, which increases the time needed to send the message.

Checksums

Checksums work in a similar way to parity bits, by using the data to work out an additional value, which is then added to the message before it is transmitted. An algorithm is used to determine this value, and both the sender and receiver must use the same algorithm.

In this example, the modulo function, which returns the remainder after division, is used. The value of the checksum is worked out as 22 MOD 8, giving a checksum value of 6. This is then converted to binary and added to the end of the message before it is transmitted.

Data to send	22 = 10110
Calculate the Checksum	22 MOD 8 = 6 = 110
Data Sent	10110110

The recipient removes the checksum, applies the same algorithm and checks that the checksum matches. If it does, the data is valid. Otherwise, the recipient asks for the data to be retransmitted.

Check Digits

Check digits use the same process as checksums but add only a single digit. This means that only a small number of algorithms can be used and so makes it less efficient than checksums.

Analogue and Digital Data

Analogue data is continuous with no limits on the possible values, whilst digital data can only be one of a set number of values and can only change at set intervals. When drawing a graph showing the data signal, analogue data appears as a smooth wave, whilst digital data appears with rough steps between each value.

Converting Digital to Analogue

A DAC (Digital to Analogue Converter) is used to convert digital signals to analogue. It reads the bit pattern representing the signal then outputs an alternating, analogue signal. DACs are often used to convert digital audio into an analogue signal.

Converting Analogue to Digital

Sensors such as light sensors and microphones generate an analogue output, which must be converted to a digital signal for computers to read it. This process uses an ADC (Analogue to Digital Converter). It reads the analogue signal at set intervals, and outputs the value at that time.

This process is known as sampling, and the frequency at which samples are taken is measured in Hertz. One Hertz is one sample per second, 10 Hertz is 10 samples per second, etc. A greater number of samples gives a higher quality and more accurate representation of the signal.

Bitmap Graphics

Bitmap graphics break down images into pixels (short for picture element) which each has a binary value assigned to it. The resolution of a bitmap image can be measured in dots per square inch, where each dot is a pixel, or in the number of pixels, such as 5x5.

The value assigned to the pixel determines the colour of that pixel in the image. The higher the number of bits available to the pixel, the more colours can be used in the image. This is called colour depth. For example, if only two bits were available, each pixel could only be one of two colours. If 2 bits were available, 4 colours (2^2) could be used.

Clearly, the greater the resolution of the image and the greater the colour depth, the larger the image file will be. To calculate the storage requirements use this formula:

Width x Height x Colour Depth

So, for example, an image 5 pixels wide by 5 pixels height with 2 bit colour depth would be 5 x 5 x 2 = 50 bits.

In real life, there may also be additional metadata associated with the file which will increase the overall file size.

Vector Graphics

Vector graphics use basic geometric shapes such as circles, triangles and rectangles to build images. Each shape has properties such as dimensions, fill style and fill colour which are stored in a list.

Because vector graphics describe how to draw the image using shapes, they can be easily enlarged without losing quality. Enlarging a bitmap image makes each pixel larger, reducing the quality of the image and making it appear pixelated. This approach also means that vector graphics require less storage space, especially for larger images.

On the other hand, vector graphics are not well suited to store photographs or other detailed images which can't be built up from basic shapes.

Storing Sound Digitally

Computers store sound as a sequence of samples, each with a set value. The number of samples taken each second is measured in Hertz and is called the sampling rate. A higher sampling rate gives a higher quality sound but requires more storage space.

The number of bits available for each sample is called the sample resolution. A higher sample resolution needs more storage space but gives a higher quality sound.

The size of a sound file is calculated by this formula:

Duration(seconds) X SamplingRate(Hertz) x Sample Resolution

For example, a 30 second file with a 20 Hertz sample rate and 24 bit sample resolution would require 30 x 20 x 24 = 14,400 bits. We can divide this by 8 to give a value of 1800 bytes.

There may also be metadata such as a title or artist stored along with the file, which increases the overall file size.

<u>The Nyquist Theorem</u>

This states that the sampling rate of a digital audio file must be at least twice the frequency of the sound and that anything below this does not give an accurate representation of the sound.

MIDI (Musical Instrument Digital Interface)

MIDI allows electronic musical instruments to send data digitally to computers. Instead of analogue sound, MIDI stores sound as a series of event messages, which together form a series of instructions on how to recreate the music.

Each message relates to a single note played on a single instrument and contains the type of instrument along with the volume and duration of the note and if it should be sustained.

MIDI allows music to be easily manipulated, such as changing the instrument or editing certain notes, without loss of quality. MIDI files are usually much smaller than sampled audio files and are lossless meaning that no information is lost.

MIDI cannot be used for storing speech, and can often sound less realistic for certain instruments

Data Compression

Compressing files reduces their size, reducing the amount of storage space needed and allowing them to be more quickly sent over a network.

Lossy Compression

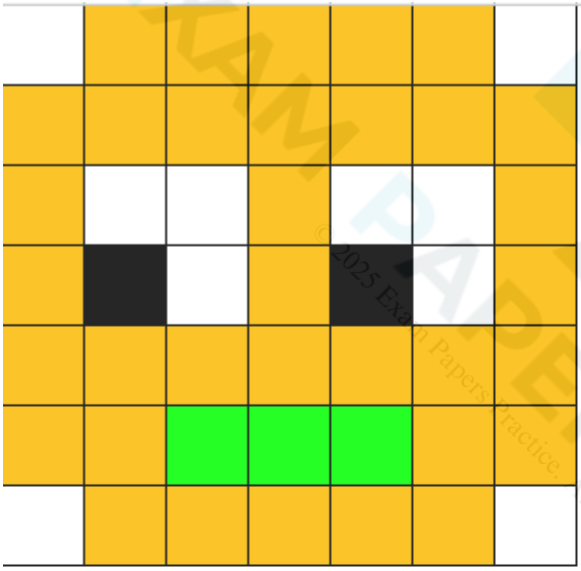
Lossy compression results in some information being lost in the compression process, meaning that the original file cannot be recreated. This technique is often used to compress images and sound, where certain data can be removed without a noticeable impact on quality.

Lossless Compression

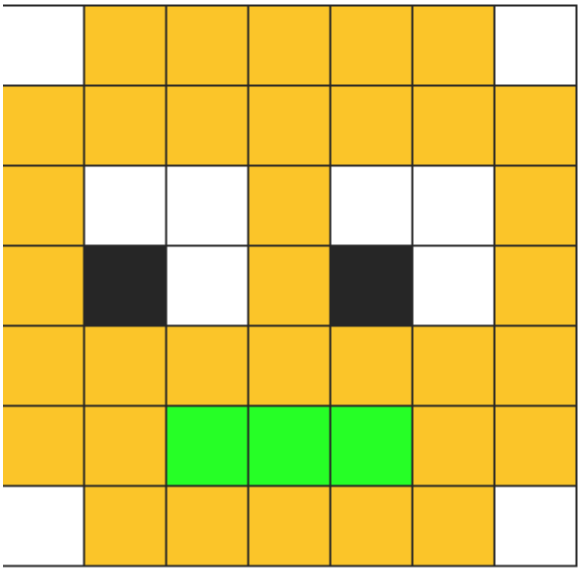
Lossless compression reduces the file size without losing any data, meaning the original file can be recreated if needed. Run Length Encoding (RLE) and Dictionary Based are two common methods of lossless compression.

RLE

This reduces the size of a file by removing repeated information, replacing it with a single occurrence and the number of times the data is to be repeated.



00 11 11 11 11 11 00
11 11 11 11 11 11 11
11 00 00 11 00 00 11
11 01 00 11 01 00 11
11 11 11 11 11 11 11
11 11 12 12 12 11 11
11 11 11 11 11 11 11



00 11x5 00
11x7
11 00x2 11 00x2 11
11 01 00 11 01 00 11
11x7
11x2 12x3 11x2
00 11x5 00

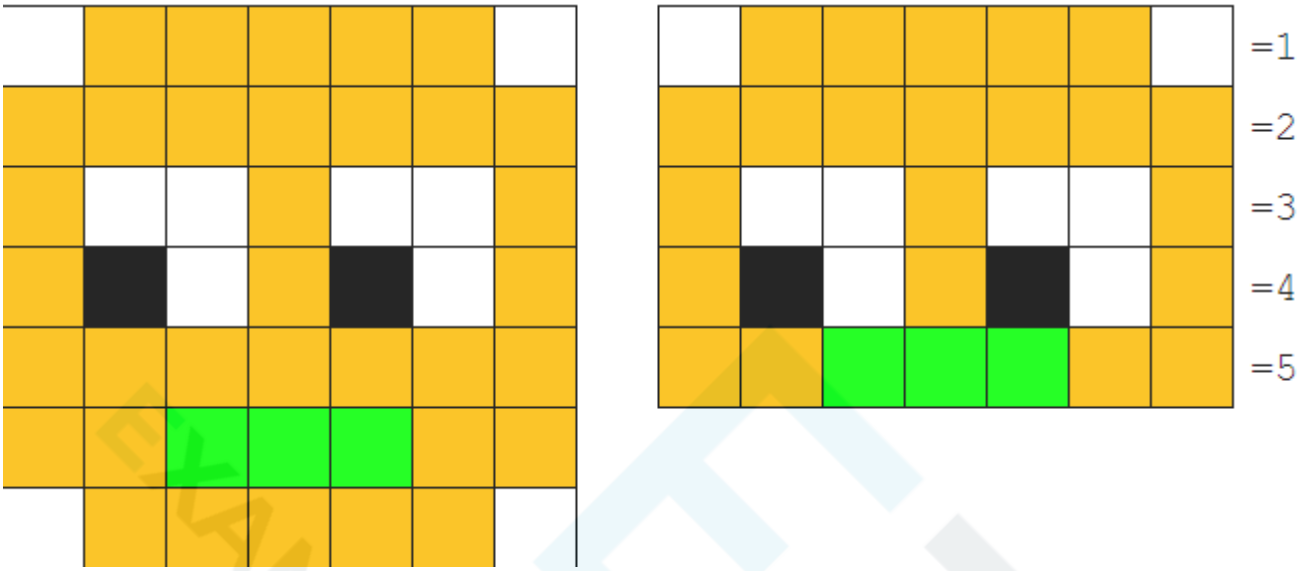
This image shows how a bitmap image could be compressed with RLE. Where repeating pixels occur in a line, they have been replaced with a single value followed by a count, reducing the amount of space

EXAM PAPERS PRACTICE
© 2025 Exam Papers Practice. All Rights Reserved

needed to store the file. Notice that line four has no repeating pixels and therefore can't be compressed, and so this method works best on files with a large amount of repeated data.

Dictionary Based

This approach uses a dictionary with repeated data which is appended to the file.



This example shows how the same picture could be written as 1234251 along with the dictionary shown above, reducing the size of the file.

Encryption

Encrypting data can be thought of as scrambling it, meaning it can't be read or understood. This helps to keep the data secure whilst it is being stored or transmitted. Unencrypted information is called plain text, and encrypted data is called ciphertext. The sender uses an encryption method and encryption key to encrypt the data, and the recipient must know the key and the method used in order to decrypt the data.

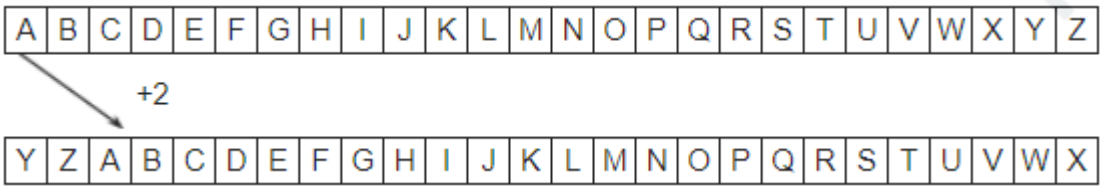
Caesar Ciphers

These encrypt data by replacing characters within the data, with each character always being replaced with the same character.

Caesar ciphers can be cracked. How often different characters occur within the ciphertext can give a clue as to which character it has been replaced with. Since E is the most often used character in English, it is fair to assume that whichever character is used most in the ciphertext has been replaced with E.

Shift Ciphers

These ciphers shift all letters by a certain amount, and this amount forms the key.



This example shows a shift cipher with a key of 2, so A becomes Y, B becomes Z and so on. Using this example, the word ENCRYPT would become CLAPWNR.

For more help, please visit www.exampaperspractice.co.uk

Substitution Ciphers

Substitution ciphers randomly replace letters and do not use a pattern.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	J	R	G	Z	E	W	O	V	N	T	C	L	S	A	K	H	U	Y	F	M	P	X	I	Q	D

This example uses a substitution cipher. Encrypting the word ENCRYPT with this cypher would give ZSGUQKF.

Vernam Ciphers

Vernam ciphers use a random key which must be as long as or longer than the plaintext which is being encrypted. Each key should only ever be used once and should not be reused. Ciphers which work like this are called one time pad ciphers.

Vernam Ciphers perform a number of steps to encrypt data:

1) Align the characters of the plaintext and key.
2) Convert each character to binary.
3) Apply an XOR operation to the two bit patterns
4) Convert the result back to a character.

When decrypting, the process is run in reverse order.

Because the chosen key is completely random, this cipher has been proven mathematically to be secure and uncrackable.

Plaintect	E	N	C	R	Y	P	T
Plaintext Binary	01000101	01001110	01000011	01010010	01011001	01010000	01010100
Key	P	R	B	Y	T	A	G
Key Binary	01110000	01110010	01100010	01111001	01110100	01100001	01100111
Plaintext Binary XOR Key Binary	11010100	11110000	10000100	10101100	10110100	11000100	110011
Ciphertext	Ô	ð		¬	´	Ä	3

Computational Security

With the exception of the Vernam Cipher, all ciphers can in theory be cracked. However, current computer processing limits means that it would take so long (hundreds of years in some cases) to crack that it is not practical to do so. This is known as relying on computational security.

© 2015 Exam Papers Practice. All Rights Reserved