Compression, Encryption and Hashing

**Introduction**

There are a range of issues relating to the exchange of data on a network. Firstly, if file sizes are too big, bandwidth can fast become used up resulting in a slower online experience. Secondly, security can pose a big problem when sending sensitive data across a network as it could be intercepted and read. The following workbook will look at measures that can be implemented to improve the online experience of users.

**Compression**

Compression software reduces the size of files. There are various reasons why the size of a file might need to be reduced:

* Less storage space required
* Faster download times – improving online experience
* Faster steaming speeds of video/audio files

There are two ways in which compression software might reduce the size of a file.

***Lossy Compression***

This is when unrequired data is removed from a file. MP3s are an example of this where sound quality may reduce but not to a point which is noticeable by the listener.

***Lossless Compression***

This is when data is temporarily removed from the file, but added back (rebuilt) when the file is to be used again. Zip files are an example of this. They will need to be unzipped (extracted) to be useable again.

**Lossy Compression**

As already stated, this is when unrequired data is removed from a file.

Regarding MP3s, certain frequencies that are not noticeable are removed reducing the data that the file contains but not giving any ‘noticeable’ drop in sound quality.

When chatting online or via mobile phone networks, lossy compression is used to ensure that only a small amount of bandwidth is used and although sound quality reduces, it doesn’t affect the ability to understand the other person.

Images are often compressed using lossy compression techniques. Details of the image may be lost but not impairing the overall quality. Lossy compression is especially important on websites where page load speeds can be seriously affected with image files that are large in size.

Lossy compression results in a much smaller file, compared to the lossless compression method.

**Lossless Compression**

Lossless compression although removes data temporarily, will later recreate the file exactly as it was.

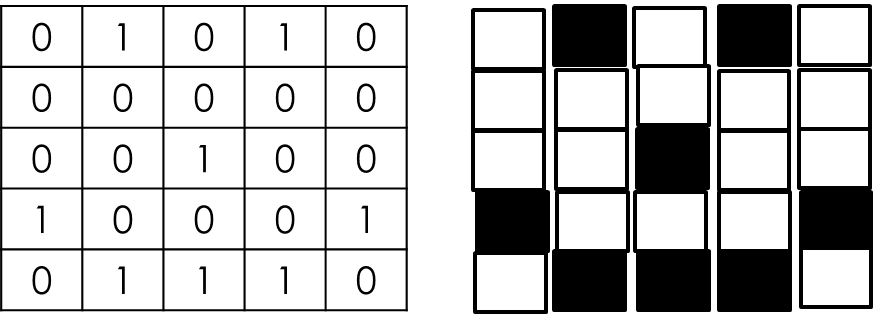
This type of compression looks at the sequencing of the data stored within the file and constructs an algorithm that can later reconstruct the file by reproducing the sequence of data.

This type of compression doesn’t remove as much data at lossy compression, but does allow the size to reduce and still retain the file’s data.

**Run Length Encoding (lossless example)**

Run length encoding is a simple form of lossless compression and demonstrates the nature of how lossless compression seeks to find patterns / sequences of data that can be reconstructed at a later date.

Consider the following terrible attempt at a 1 bit image of a smiley:



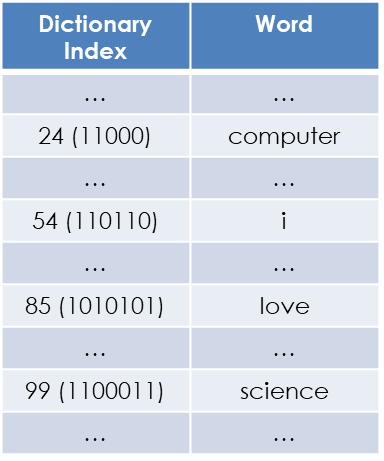
If we take the 2nd row for example, the data would ordinarily be 5 bits in length - 00000.

However, run length encoding would focus on the pattern of 5 white pixels in a row and so would store the data (whilst the file is compressed) as 5W or 101 0, which uses 1 fewer bits.

This is the principle of Run Length Encoding. As images increase in size, generally the patterns of similar shades increase in length and so the data required to store the pattern gets much smaller in relation to the actual file’s data.

**Dictionary Coding (lossless example)**

Dictionary coding is another form of lossless compression which looks to replace the file’s data with a reference to what the data is.

Imagine a dictionary which contains all of the words in the English language. Imagine this dictionary was contained in a table with every word having a unique indexed location.

Consider the following string, 23 bytes in size: “I love Computer Science”

With dictionary coding, these words could be replaced with their reference (index location) in the dictionary. So the data would be stored (whilst being compressed) as 11000 110110 1010101 1100011 which is just 25 bits. Which is far smaller. Yes, the dictionary would have to accompany the compressed file, however, for large files, the size of the dictionary doesn’t have a massive impact on the size of the compressed file.

**Encryption**

Encryption is where data is scrambled before being sent across a network so that it is unreadable if intercepted.

To encrypt data, an encryption key is used which will convert ‘plain text’ into ‘cipher text’. An encryption key is an algorithm which will systematically alter each piece of data in a file. For example, a key may convert each letter in a text file to the next letter in the alphabet:

e.g. ‘hello’ becomes ‘ifmmp’

For the ‘cipher text’ to be converted back to ‘plain text’, either the same key (or another in special cases) is required by the recipient to reverse the encryption.

For your exam you need to have an understanding of both ‘Symmetric’ and ‘Asymmetric’ encryption.

***Symmetric – private key***

With symmetric encryption the same key is used to encrypt and decrypt a file. This means that the key also has to be passed from the sender to the recipient, which if intercepted could be used to reveal the data and therefore pose a security risk. This form of encryption is therefore risky.

***Asymmetric – public key***

Asymmetric encryption differs in that two separate keys are used in the encrypt/decrypt process. One is used to encrypt the file and another is used to decrypt it. With asymmetric encryption, the encryption key is made public and is used by people who wish to send you data securely. You (the recipient) will own a separate private key which can be used to decrypt the file. This is a far more secure form of encryption as the decryption key cannot be intercepted as it doesn’t get sent anywhere.

***Digital Signatures***

The only real possible issue with asymmetric encryption is that you could receive a file from someone, which has been encrypted with your public key, but because you can’t check its contents, you could decrypt it to find a malicious file.

To ensure that you only receive legitimate files, encrypted with your public key, from acceptable sources, the public key can also generate a digital code (digital signature) which verifies the files content and the sender. This way you can be confident that digitally signed files contain acceptable data.

Hashing

Hashing is the process of applying an algorithm an a piece of data, in order to calculate a numerical value.



Hashing can be a really good way to encrypt data that cannot be decrypted. Why would you want to irreversibly encrypt data I hear you say?!

Consider a password. No one else should ever know a password and it certainly shouldn’t be stored on a computer system in the same form as it was inputted – a hacker could steal the password and use it to gain access to a user’s account(s). Hash algorithms are used to convert an inputted password into a numerical value (which cannot be reversed engineered), which is then stored on file. When the user types their password in for account access, the hash algorithm is preformed again on the inputted data. The numerical value produced is compared with the one stored, and access will be granted if the values match.

***Keywords / Key Terms:***

**Compression** - Reducing the size of a file.

**Encryption** - Scrambling the data of a file so that it becomes unreadable.

**Hashing** - Performing an algorithm on inputted data to produce a numerical output – which cannot be reverse engineered.