OCR Computer Science A Level

1.3.1 Compression, Encryption and Hashing

Advanced Notes

Specification:

1.3.1 a)

- Lossy vs Lossless compression

1.3.1 b)

- Run length encoding and dictionary coding for lossless compression

1.3.1 c)

- Symmetric and asymmetric encryption

1.3.1 d)

- Different uses of hashing

Compression

Compression is the process used to reduce the storage space required by a file, meaning you can store more files with the same amount of storage space. Compression is particularly important for sharing files over networks or the Internet. The larger a file, the longer it takes to transfer and so compressing files increases the number of files that can be transferred in a given time.

Apps like Google Photos compress files so that they can quickly be searched for and downloaded. Downloading a compressed file over the Internet is faster than downloading the full version of the file.

Lossy vs Lossless Compression

There are two categories of compression: lossy and lossless. As the name suggests, lossy compression reduces the size of a file while also removing some of its information. This could result in a more pixelated image or less clear audio recording. On the other hand, lossless compression reduces the size of a file without losing any information.

When using lossless compression, the original file can be recovered from the compressed version. Something which is not possible when using lossy compression which reduces the size of the file by completely disregarding some information. For example, audio files can be compressed lossily by removing the very high or very low frequencies which are least noticeable to the ear. There's no way to go from the lossy version of the recording back to the full version as there's no record of what the high and low frequencies were.

Run Length Encoding

Sometimes called RLE, run length encoding is a method of lossless compression in which repeated values are removed and replaced with one occurrence of the data followed by the number of times it should be repeated.

For example, the string AAAAAABBBBBCCC could be represented as A6B5C3.

In order to work well, run length encoding relies on consecutive pieces of data being the same - if there's little repetition, run length encoding doesn't offer a great reduction in file size.

Dictionary Encoding

Dictionary encoding is another example of a method of lossless compression. Frequently occurring pieces of data are replaced with an index and compressed data is stored alongside a dictionary which matches the frequently occurring data to an index. The original data can then be restored using the dictionary.

Take for example, the following passage.

> *We shall go on to the end.*
> *We shall fight in France,*
> *we shall fight on the seas and oceans,*
> *we shall fight with growing confidence and growing strength in the air,*
> *we shall defend our island, whatever the cost may be.*

Frequently occurring phrases include “We shall”, “fight”, “the”, “on” and “in”. Placing these phrases in the dictionary below and replacing any occurrence with the phrase’s index, the size of the passage is substantially reduced.

Index	Phrase
1	We shall
2	fight
3	the
4	on
5	in
6	and

1 go on to 3 end.
1 2 5 France,
1 2 on 3 seas 6 oceans,
1 2 with growing confidence 6 growing strength 5 3 air,
1 defend our island, whatever 3 cost may be.

It’s important to remember that data compressed using dictionary compression must be transferred alongside its dictionary. Without a dictionary, the data cannot be used.

Encryption

Encryption is used to keep data secure when it's being transmitted. There are a variety of different methods which can be used to scramble data before it's transmitted and then decipher it once it arrives at its destination. We're going to look at symmetric and asymmetric encryption.

Symmetric Encryption

In symmetric encryption, both the sender and receiver share the same private key, which they distribute to each other in a process called a key exchange. This key is used for both encrypting and decrypting data.

It's important that the private key is kept secret. If the key is intercepted during the key exchange then any communications sent can be intercepted and decrypted using the key. Asymmetric encryption gets around this issue.

Asymmetric Encryption

When sending information using asymmetric encryption, two keys are used: one public and a second, private, key. The public key can be published anywhere, free for the world to see, while the private key must be kept secret. Together, these keys are known as a key pair and are mathematically related to one another.

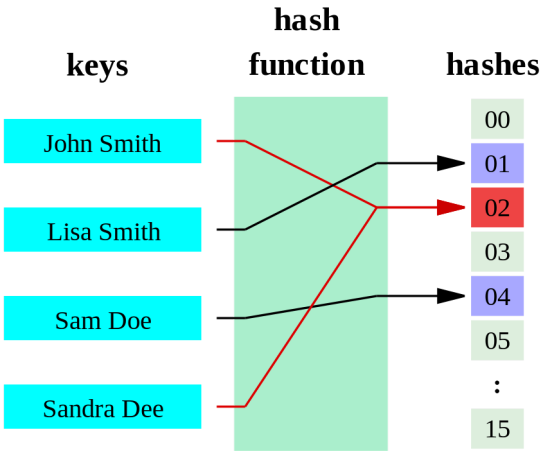
In contrast to symmetric encryption, a single key cannot be used to both encrypt and decrypt communication. Instead, messages encrypted with the recipient's public key can only be decrypted with the recipient's private key, which should only be in the possession of the recipient.

If someone wants to send you a message, they must first find your public key. There are a variety of websites which can do this for you. The message is then encrypted with your public key meaning that only you can decrypt it.

If you want to prove that a message has been sent by you, you can encrypt it using your private key. This means that anyone can decrypt it (as your public key is available to anyone) and by doing so, can guarantee that you encrypted the message, as only you have access to the private key. This forms the basis of a system called digital signatures.

Hashing

Hashing is the name given to a process in which an input (called a key) is turned into a fixed size value (called a hash). There are a vast number of algorithms, called hash functions, which do this.



Unlike encryption, the output of a hash function can't be reversed to form the key. This quality makes hashing useful for storing passwords. A password entered by a user can be hashed and checked against the key to see if it is correct, but a successful hacker would only gain access to the keys, which can't be reversed to gain the passwords.

Synoptic Link

Hashing is used in a type of data structure called **Hash tables**

Hash tables are covered in **1.4.2 Data Structures**

Another use of hashing is hash tables. A hash table is a data structure which holds key-value pairs. Formed from a bucket array and a hash function, hash tables can be used to lookup data in an array in constant time. When data needs to be inserted, it is used as the key for the hash function and stored in the bucket corresponding to the hash.

Hash tables are used extensively in situations where a lot of data needs to be stored with constant access times. For example, in caches and databases.

If two pieces of data (keys) produce the same hash, a collision is said to have occurred. For example, in the image above the keys `John` and `Sandra` both hash to `02`. There are a variety of methods to overcome collisions, including storing items together in a list under the hash value, or using a second hash function to generate a new hash.

A good hash function should have a low chance of collision and should be quick to calculate. Furthermore, a hash function should provide an output which is smaller than the input it was provided, otherwise searching for the hash could take longer than simply searching for the key.