

Syllabus Content:

3.1 Data representation

3.1.2 File organisation and access

- file organisation: serial, sequential (using a key field) and random (using record key)
- file access: sequential access (serial & sequential files) – direct access (sequential & random files)
- select an appropriate method of file organisation and file access for a given problem

File Organisation and Access:

hir.com

Key terms

Serial file organisation – a method of file organisation in which records of data are physically stored in a file, one after another, in the order they were added to the file.

Sequential file organisation – a method of file organisation in which records of data are physically stored in a file, one after another, in a given order.

Random file organisation – a method of file organisation in which records of data are physically stored in a file in any available position; the location of any record in the file is found by using a hashing algorithm on the key field of a record.

Hashing algorithm (file access) – a mathematical formula used to perform a calculation on the key field of the record; the result of the calculation gives the address where the record should be found.

File access – the method used to physically find a record in the file.

Sequential access – a method of file access in which records are searched one after another from the physical start of the file until the required record is found.

Direct access – a method of file access in which a record can be physically found in a file without physically reading other records.

Serial file organisation:

Serial file organisation is the simplest file organisation method. In serial files, records are entered in the order of their creation. As such, the file is unordered, and is at best in chronological order.

| | | | | | |
|------------------|------------------|------------------|------------------|------------------|--------------------|
| Transaction 1 | Transaction 2 | Transaction 3 | Transaction 4 | Transaction 5 | and so on |
|------------------|------------------|------------------|------------------|------------------|--------------------|

Start of File

Serial files are primarily used as **transaction files** in which the transactions are recorded in the order that they occur for example storing customer meter reading for electricity or gas before they are used to process and send bills.

Serial file organization method physically stores records of data in a file, one after another. New records are appended to the end of the file.

Sequential file organisation:

The **Sequential file organization** method physically stores records of data in a file, one after another, in a given order.

The **key difference between a sequential file and a serial file** is that it is ordered in a logical sequence based on a **key field**. This key is usually the primary key, though secondary keys may be used as well. Sequential files are therefore files that are sorted based on some key values.

| | | | | | | |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-----------|
| Customer 1 record | Customer 2 record | Customer 3 record | Customer 4 record | Customer 7 record | Customer 8 record | and so on |
| Start of file | | | | | | |

New records must be inserted in file in correct place e.g. **Customer5 Record** added after **Customer4 Record**

| | | | | | | | |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-----------|
| Customer 1 record | Customer 2 record | Customer 3 record | Customer 4 record | Customer 5 record | Customer 7 record | Customer 8 record | and so on |
| Start of file | | | | | | | |

Sequential files are primarily used in applications where there is a **high file hit rate**. Hit rate is a measure of the proportion of the records that is accessed in a single run of the application. Therefore, sequential files are ideal for **master files** and **batch processing** applications such as payroll systems or electricity/gas bill generation in which almost all records are processed in a single run of the application.

Random file organisation:

In random file organisation, records are stored in random order within the file. Though there is no sequencing to the placement of the records, there is however, a **pre-defined relationship** between the **key of the record** and its **location within the file**.

| | | | | | | |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-----------|
| Customer 8 record | Customer 2 record | Customer 4 record | Customer 7 record | Customer 3 record | Customer 1 record | and so on |
| Start of file | | | | | | |

In other words, the value of the record key is mapped by an established function to the address within the file where it resides. Therefore, any record within the file can be directly accessed through the **mapping function** in roughly the same amount of time. The location of the record within the file therefore is not a factor in the access time



of the record. As such, random files are also known in some literature as direct access files.

To create and maintain a random file, a mapping function must be established between the record key and the address where the record is held. If **M** is the **mapping function**, then

$$M(\text{value of record key}) = \text{address of record}$$

File Access:

There are different methods files are accessed. These methods physically find records in a file. The methods are

-  Sequential file access
-  Direct file access

Sequential Access:

The sequential access method searches record one after another from the physical start of file till the record is found. This method is used for serial and sequential files.

For a **serial file**, if a particular record is being searched for, every record needs to be checked until that record is found or the whole file has been searched and that record has not been found. Any new records are appended to the end of the file.

For a **sequential file**, if a particular record is being searched for, every record needs to be checked until the record is found or **the key field of the current record being checked is greater than the key field of the record being searched for**. The rest of the file does not need to be searched as the records are sorted on ascending key field values.

Any new records to be stored are inserted in the correct place in the file. For example, if the record for **Customer 6** was requested, each record would be read from the file until **Customer 7** was reached. Then it would be assumed that the record for **Customer 6** was not stored in the file.





| | | | | | | | |
|------------|------------|------------|------------|------------|------------|------------|-------|
| Customer 1 | Customer 2 | Customer 3 | Customer 4 | Customer 5 | Customer 7 | Customer 8 | |
|------------|------------|------------|------------|------------|------------|------------|-------|


Customer 6 record not found

Sequential access is efficient when every record in the file needs to be processed, for example, a monthly billing or payroll system. These files have a high hit rate during the processing as nearly every record is used when the program is run.

Direct access

The direct access method can physically find a record in a file without other records being physically read.

-  Both sequential and random files can use direct access. This allows specific records to be found more quickly than using sequential access.
-  Direct access is required when an individual record from a file needs to be processed.
-  For example, when a single customer record needs to be updated when the customer's phone number is changed.
-  Here, the file being processed has a low hit rate as only one of the records in the file is used.

For a sequential file, an index of all the key fields is kept and used to look up the address of the file location where a given record is stored. For large files, searching the index takes less time than searching the whole file.

For a random access file, a hashing algorithm is used on the key field to calculate the address of the file location where a given record is stored.



Hashing algorithms

Hashing Algorithms:

There are various mapping techniques. Some involve using the **key field** to directly map to the location with the file, while others refer to some **lookup table** for the location. However, the more common method is to employ a **hash function** to derive the address.

Hashing is the process of transforming the key value of a record to yield an address location where the record is stored. In some literatures, it is also known as Key-to-Address Transformation, Address-Calculation, Scatter Storage, or Randomization.

A hash function generates the record address by performing some simple operations on the key or parts of the key. A good hashing function should be

-  Quick to calculate
-  Cover the complete range of the address space.



Give an even distribution



Not generate addresses that tend to cluster within a few locations, thus resulting in frequent collisions.

Here is an example of a simple hashing algorithm:

If a file has space for **2000** records and the key field can take any values between **1** and **9999**, then the hashing algorithm could use the **remainder** when the value of key field is divided by **2000**, together with the start address of the file and the size of the space allocated to each record.

In the simplest case, where the start address is **0** and each record is stored in one location.

To store a record identified by a key field with value **3024**, the hashing algorithm would give address **1024** as the location to store the record.

| Key field | Remainder | Address |
|-----------|-----------|-----------------------|
| 3024 | 1024 | $1024 = 0 + 1 * 1024$ |

Unfortunately, storing another record with a key field 5024 would result in trying to use the same file location and a collision would occur.

| Key field | Same remainder | Same address |
|-----------|----------------|-----------------------|
| 5024 | 1024 | $1024 = 0 + 1 * 1024$ |

This often happens with hashing algorithms for direct access to records in a file.

There are two ways of dealing with this:

1. An open hash where the record is stored in the next free space.
2. A closed hash where an overflow area is set up and the record is stored in the next free space in the overflow area.

When reading a record from a file using direct access, the address of the location to read from is calculated using the hashing algorithm and the key field of the record stored there is read. But, before using that record, the key field must be checked against the original key field to ensure that they match. If the key fields do not match, then the following records need to be read until a match is found (open hash) or the overflow area needs to be searched for a match (closed hash).

References

- Book: AS and A-level Computer Science by Hodder Education
- AS and A level Book by Sylvia Langfield and Dave Duddell
- <http://www.bbc.co.uk/education/guides/zifgixs/revision/1>
- https://www.tutorialspoint.com/vb.net/vb.net_constants.htm
- https://www.cs.drexel.edu/~introcs/F2K/lectures/5_Scientific/overflow.html
- <https://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Data/underflow.html>

Computer Science (9608) at www.majidtahir.com