• Chapter 5: Managing Files of Records

→ Goal

- extend the file structure concepts
  1. search keys and canonical forms
  2. sequential search
  3. direct access
  4. file access and file organization

*5.1 Record Access

↳ it is convenient to identify the record with a key based on the record's contents.

↳ we must define a standard form for keys, along with associated rules and procedures for converting keys into this standard form

↳ a standard form is often called canonical form for the key.

- Canonical means "conforming to the rule"
- canonical form is the single representation

EX: key composed solely of upper case letters and have no extra blanks at the end

Ames → AMES

ames → AMES

↳ it is desirable to have distinct keys, or keys

↳ We should prevent confusion as new records are added to the file

- When the user enters a new record, we form a unique canonical key for the record and then search the file for the key.
- this concern about uniqueness applies only to primary keys
- a primary key, by definition, is the key that is used to identify a record uniquely.
- it is also possible to search on secondary keys. Typically, secondary keys do not uniquely identify a record.

↳ In general primary keys should be dataless and unchanging

5.1.2 A sequential search

↳ simple solution :

- reads through the file, record by record, looking for a particular key
- program (as exercise)
- simple baseline, can be used to measure the improvement of "smarter" searches

↳ Performance measurement

* we usually use the number of comparisons required for the search
* but ... the cost of comparison in memory is so small compared with the cost of disk access
* Instead, we count low-level read calls
  - Read call requires a seek

↳ So, suppose we have a file with a thousand records and we want to use a sequential search to find Al Smith's record.

- how many reads are necessary ?

→ if is the first : one read

→ if the last : thousand reads

→ average search : 500 calls needed

→ if we double the file size, we double the costs / times

↳ So, the amount of work required for a sequential search is directly proportional to the number of records in the file

- average $n/2$ comparisons
- $O(n)$ because the time it takes is proportional to n.

↳ When Sequential Search is good?

- sequential search is just too expensive for most serious retrieval situations
- two advantages
  - simple to program
  - requires the simplest of file structures

- Situation where it is reasonable:
  - ASCII files in which you are searching for some pattern (grep)
  - file with few records (for example, ten records)
  - files that hardly ever need to be searched
  - files in which you want all records with a second secondary key value, where a large number of matches is expected

5.1.3 Unix tools

- most common file structure in Unix is:
  → ASCII file with new-line character as the record delimiter and, when it is possible, white space as the field delimiter
  (white-space/new-line structure)

- cat (show file content) % cat myfile
- wc (word count) % wc myfile
- grep (check if a text file has a certain word or character string in it)

grep = generalized regular expression

grep searches sequentially through a file for a pattern
it return all the lines in the file that contain the pattern

* (idea: implement grep function) *

% grep <pattern> myfile

5.1.4 Direct Access

↳ the most radical alternative to searching sequentially through a file is a retrieval mechanism know as direct access

- we can seek directly to the begining of the record and read it
  - → sequential search O(n)
  - → direct access O(1)
- using the byte address of the record as the record reference

↳ the major problem: knowing where the begining of the required record is

- Sometimes this is kept in a separate index file
- relative record number (RRN)
  ↳ we see the file as a collection of records rather than a collection of bytes

- the first record in a file has RRN 0, the next RRN 1, and so forth.
- to support direct access by RRN, we need to work with records of fixed, known lenght

↳ If the records are all the same lenght, we can use a record's RRN to calculate the byte offset of the start of the record relative to the start of the file

EX:

- record with an RRN of 546
- file has fixed-lenght record size of 128 bytes per record

So, byte offset:

byte offset = 546 × 128 = 69.888

or generally:

byte offset = $n \times r$

n: RRN
r: record size

* 5.2 More About Records

5.2.1 Choosing Record Structure and Record Length

- Suppose we are building a file of sales transactions that contain the following information about each transation:
  - six digit account number of the purchaser
  - six digits for the date field
  - five-character stock number for the item purchased
  - three-digit field for quantity
  - ten-position field for total cost

- all of them fixed-lenght fields (sum 30 bytes)
- try to work with power of 2 (f.e, 32 bytes)
- use fixed-lenght fields with fixed-lenght records or variable-lenght fields and fixed-lenght records

Fixed-lenght Fields
→ simplicity

Variable-lenght Fields
→ more efficient use of space

- we can combine them
  - some fields fixed
  - some fields variable

5.2.2 Header Record

→ it is often necessary or useful to keep track of some general information about a file to assist in future use of file

→ a header record is often placed at the beginning of the file to hold this kind of information

→ can keep: count of the number of records in the file.
the lenght of data records
the date and time of the most recent update
the name of the file
...

→ it has a different structure than data (Fig 5.2)

→ Header records are a widely used tool

* Summary

- one higher level of organization, in which records are grouped into blocks, is also often imposed on files
- Sometimes we identify individual records by their relative record number (RRN) in a file

- It is also common to identify them by a key whose value is based on some of the record's content
- Key values must be converted to a canonical form (primary keys), unique and unambiguous
- sequential search (poor for long files)
- wc and grep (Unix) use sequential search
- byte offset opens possibility of accessing the record directly by RRN, rather than sequentially
- a header record, stored at the beginning of the file is a useful tool for storing general information about the file