Binary Searching, Keysorting and Indexing

Last Time : Reclaiming Space in Files

Today : Binary Searching (Chapter 6.3.1 - 6.3.3), Keysorting (Chapter 6.4) and Introduction to Indexing (Chapter 7.1-7.3)

Binary Searching (Chapters 6.3.1 - 6.3.3)

Let us consider fixed-length records that must be searched by a **key value**.

If we knew the RRN of the record identified by this key value, we could jump directly to the record (using "seek").

In practice, we do not have this information and we must search for the record containing this key value.

If the file is not sorted by the key value we may have to look at every possible record before we find the desired record.

An alternative to this is to maintain the file **sorted by key value** and use **binary searching**.

A binary search algorithm using C++ :

```
class FixedRecordFile{
   public:
     int NumRecs();
     int ReadByRRN(RecType & record, int RRN);
};

class KeyType {
   public:
     int operator==(KeyType &);
     int operator<(KeyType &);
};
```

```
class RecType {
   public:
     KeyType key();

};

int BinarySearch(FixedRecordFile & file, RecType & obj, KeyType & key) {
   int low=0; int high=file.NumRecs() -1;
   while (low <= high) {
     int guess = (high + low)/2;
     file.ReadByRRN(obj,guess);
     if (obj.key() == key) return 1;
     if (obj.key() > key)
       high = guess - 1;
     else
       low = guess + 1;
   }
   return 0;  //did not find key
}
```

Note: the above algorithm corrected some mistakes from the textbook.

Binary Search versus Sequential Search :

Binary Search : $O(\log_2 n)$
Sequential Search : $O(n)$

If file size is doubled, sequential search time is doubled, while binary search time increases by 1.

In assignment#2, binary search must be employed in searching for the record containing a given student number.

For this purpose, ReadbyRRN must be implemented.
You can use method `seekg`.
To rewrite the password after correct position is found, move the put pointer by using method `seekp`.

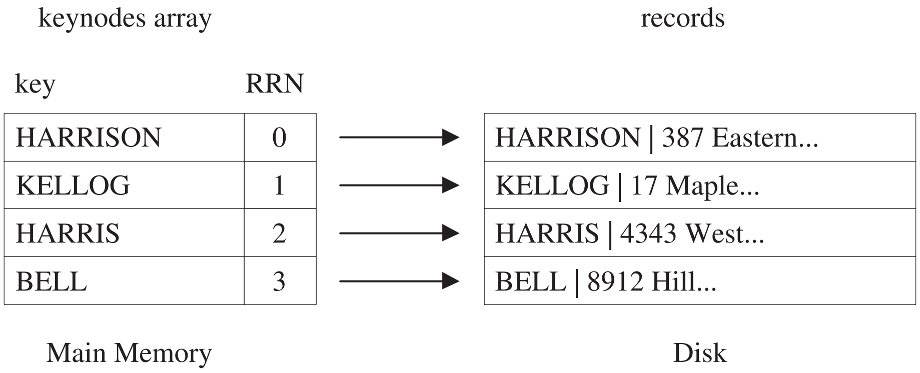
Keysorting (Section 6.4)

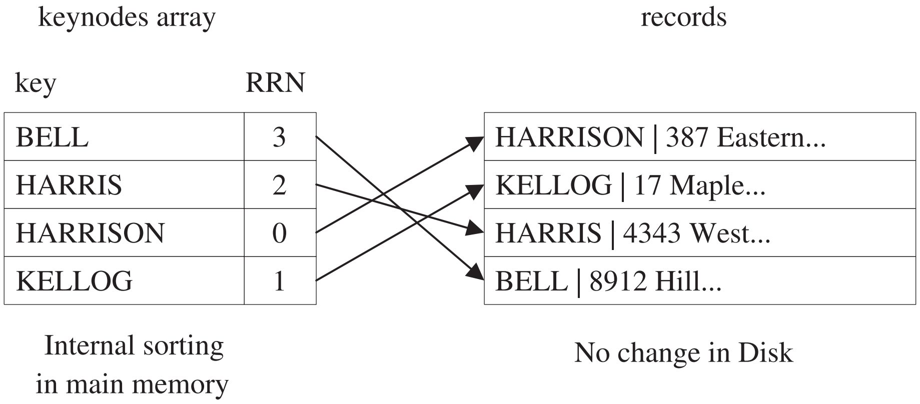
Suppose a file needs to be sorted, but it is too big to fit into main memory.

To sort the file, we only need the **keys**. Suppose that all the keys fit into main memory.

Idea:

- Bring the keys to main memory plus corresponding RRN
- Do internal sorting of keys
- Rewrite the file in sorted order





keynodes array

BELL	3
HARRIS	2
HARRISON	0
KELLOG	1

records

BELL | 8912 Hill...
HARRIS | 4343 West...
HARRISON | 387 Eastern...
KELLOG | 17 Maple...

create new sorted file to replace previous

How much effort we must do (in terms of disk accesses ?)

- Read file sequentially once
- Go through each record in random order (seek)
- Write each record once (sequentially)

Why bother to write the file back?

Use keynode array to create an **index file** instead.

index file

BELL	3
HARRIS	2
HARRISON	0
KELLOG	1

records

HARRISON | 387 Eastern...
KELLOG | 17 Maple...
HARRIS | 4343 West...
BELL | 8912 Hill...

Leave file unchanged

This is called INDEXING !!

Pinned Records

Remember that in order to support deletions we used `AVAIL LIST`, a list of available space.

The `AVAIL LIST` contains info on the physical information of records.
In such a file a record is said to be **pinned**.

Using an **index file** for sorting leaves the `AVAIL LIST` and positions of records unchanged. This is convenient.

Indexing (Chapter 7.1-7.3)

- Simple indexes use simple arrays.
- An index lets us **impose order on a file** without rearranging the file.
- Indexes provide **multiple access paths** to a file - **multiple indexes** (library catalog providing search for author, book and title).
- An index can provide keyed access to variable-length record files.

A Simple Index for Entry-Sequenced File

Records (Variable Length)

Address of Record	
17	LON | 2312 | Symphony N.S | ...
62	RCA | 2626 | Quartet in C sharp | ...
117	WAR | 23699 | Adagio | ...
152	ANG | 3795 | Violin Concerto | ...

Primary key = company label + record ID (LABEL ID).

- Index is sorted (main memory).
- Records appear in file in the order they entered.

Index :

key	Reference field
ANG3795	152
LON2312	17
RCA2626	62
WAR23699	117

How to search for a recording with given LABEL ID ?

"Retrieve recording" operation :

- Binary search (in main memory) in the index : find LABEL ID, which leads us to the reference field.
- Seek for record in position given by the reference field.

Two issues to be addressed :

- How to make a persistent index (i.e. how to store the index into a file when it is not in main memory).
- How to guarantee that the index is an accurate reflection of the contents of the file. (This is tricky when there are lots of additions, deletions and updates.)