8 BINARY SEARCHING, KEYSORTING & INDEXING

Content

- Binary Searching
- Keysorting
- Introduction to Indexing

Binary Searching

- 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 (by using fseek function)
- 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**

Binary Search Algorithm in C++


```
template <typename KeyType,typename RecordType>
bool BinarySearch(FILE *file,RecordType &rec, KeyType &key){
   int low=0,high=getFileLength(file)/sizeof(RecordType)-1 ;
   int guess ;
   while (low<=high){
       guess = (high+low)/2 ;
       readRecord(file,rec,guess) ;
       if (Equal (rec.key(),key)) return true ;
       if (Greater (rec.key(),key)) high = guess-1 ;
       else low = guess+1 ;
   }
   return false;
}
```


TBook


```
typedef struct TBook {
    char author[16] ;
    char title[24] ;
    char isbn[10] ;
    char *key(){return isbn;}
} SBook ;
```


Equal()


```
template <typename KeyType>
bool Equal(KeyType key1,KeyType key2){
    if (key1==key2) return true ;
    return false ;
}

bool Equal(char *key1,char *key2){
    return (strcmp(key1,key2)==0) ;
}
```

readRecord

```
template <typename RecordType>
void readRecord(FILE *file,RecordType &rec,int rnn){
    fseek(hFile,rnn*sizeof(RecordType),SEEK_SET) ;
    fread(&rec,sizeof(RecordType),1,hFile) ;
}
```


int main()

```
int main(int argc, char* argv[]) {
   FILE *hFile ;
   ...
   TBook book ;
   BinarySearch(hFile,book, "Da Vinci Code") ;
   cout << book.author ;
   return 0;
}
```

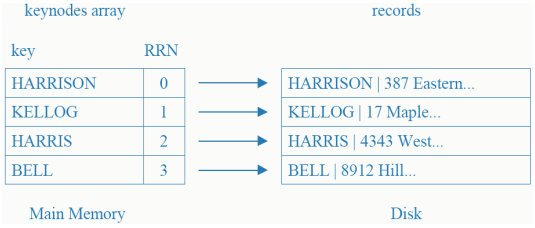

Binary Search *vs.* Sequential Search

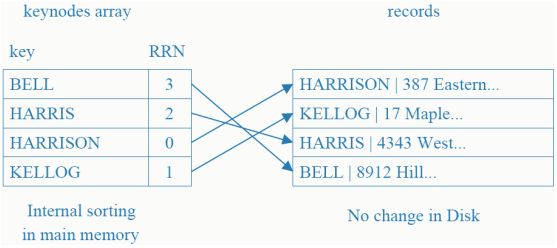
- Sequential Search: O(n)
- Binary Search: O($\log_2 n$)
- If file size is doubled, sequential search time is doubled, while binary search time increases by 1

Keysorting

- 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

Example





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?

- 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 records
- The **AVAIL LIST** contains info on the physical information of records. In such a file, a record is said to be **pinned**
- If we use an **index file** for sorting, the **AVAIL LIST** and positions of records remain unchanged.
- This is a good news ☺

Introduction to Indexing

- 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** (like 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

index:

key	reference field
ANG3795	152
LON2312	17
RCA2626	62
WAR23699	117

Index

- Index is sorted (main memory)
- Records appear in file in the order they entered
- How to search for a recording with given LABEL ID?
  - Binary search (in main memory) in the index: find LABEL ID, which leads us to the referenced field
  - Seek for record in position given by the reference field

Some Issues

- 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