CMPUT291 Project 2 Design Report

By

Zhaorui (Teppie) CHEN

Zhengyang (Nicholas) LI

Jiaxuan YUE

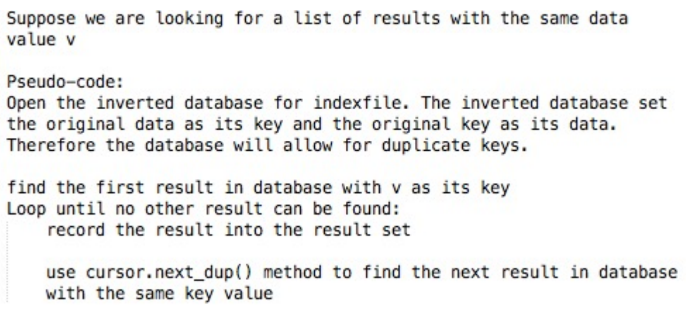
Introduction

This report introduces the design of a database which uses various file structures including B-Tree, Hash Table, and Index File, with different search methods. In the report the speed of each structure-method combination is listed. The reason why Index File improves searching speed is briefly discussed.

Method

As it's mentioned in introduction, the script has three different file structures. The user will need to declare which structure he or she would use to build the database by typing btree, hash, or index as command line arguments. The implementation of B-Tree and Hash is almost the same. In key search method, the program asks for a valid key input and directly search the corresponding value by Berkeley DB get() method. In data search method, the program asks for a valid data input and compare it with all values in data until it is matched. In range search method, the program asks for a valid range of characters, traverse the whole database and put all key-value pair in between the given range into a list.

The implementation of Index File is different from the previous ones in data search method and range search method. In this structure we build two hash table structure databases, one exactly the same as previous one (here we call it *original database*), while the other one is built based on values (here we call the data *value_as_key*, and the new database *second database*) and accepts duplicate keys. For key search method, the program can directly retrieve the data using get() method in *original database* as well as the previous structures. For data search method, since the second data base is built based on values, we can retrieve the key much faster than the previous file structures. The program built up a cursor in the *second database* and the cursor will be moved to the first key the same as input *value_as_key* using cursor.set() method. Then the cursor will keeps moving to the next key which is the same as the input using next_dup() method and return all the result key-value pairs into a list. The pseudo code is shown below.



```
Suppose we are looking for a list of results with the same data
value v

Pseudo-code:
Open the inverted database for indexfile. The inverted database set
the original data as its key and the original key as its data.
Therefore the database will allow for duplicate keys.

find the first result in database with v as its key
Loop until no other result can be found:
    record the result into the result set

    use cursor.next_dup() method to find the next result in database
    with the same key value
```

Lastly for range search method, the program first built up a cursor in the *original database* and find out the smallest key (here we call it *SK*) greater than or equal to lower bound. If this *SK* is lower than the upper bound, it will be added to the result list and the cursor will find out the next key and repeat the method.

Result

We took five test cases for each kind of searching method. The test result is shown in the table below.

	TEST CASES	B-Tree	Hash Table	Index File
Retrieve By Key	teppie	54	41	35
	nicholas	52	41	41
	jiaxuan	56	43	37
	qwerty	54	53	52
	wasd	56	44	38
avg		54.4	44.4	40.6
Retrieve By Data (# of results)	chen	236111(2)	232951(2)	55(2)
	li	237077(1)	231287(1)	40(1)
	yue	237729(1)	233073(1)	49(1)
	abcdefghi	235551(1)	230928(1)	50(1)
	udlr	235185(1)	228297(1)	45(1)
avg		236330.6	231307.2	46.4
Retrieve By Range	o-t	134390	147768	27090
	h-o	174478	179427	30573
	a-y	91558	111134	111890
	t-y	102824	122529	23159
	n-t	135263	148094	27455
avg		127702.6	141790.4	44033.4

Table 1. Test result

Discussion

From the result table we could see the speed improvement of data search and range search in Index File is quite obvious. For data search in Index File, since we have a cursor on *second database*, the program can easily locate the first matched *value_as_key*, as fast as the get() method in key search. After the first *value_as_key* is found, the next_dup() method can efficiently get all duplicate keys if they are exist, which is because, we assume, the database with duplicate keys allowed somehow sorted all the keys and put the duplicate keys together, or make them related. Instead of traversing the whole database in B-Tree and Hash Table structure, this cursor method is getting all the wanted keys directly, just as efficient as the get() method in key search.

For range search method, we could see the speed is likely random in B-Tree and Hash Table, while it depends more on the size of range in Index File. From the table we could see the searching of largest range a-y in Index File takes even more time than B-Tree and Hash Table structure, while the rest with range size around 5-8 characters is having a similarly fast speed. In the first two methods, the program will traverse all keys in database and check if they are in range, therefore the speed is slow and random. In Index File the key is searched using something like an iterator, or, in other words, we are searching keys in a sorted list. The time is only cost by checking whether the next value is larger than upper bound or not. If it is, the search stops and time wasting by over bounded keys are markedly reduced.

Another reason is that the range search in Index File is faster is because cursor directly returns key-value pairs, while the B-Tree and Hash Table will have to retrieve values with keys after they are found. Therefore cursor is reducing time wasting in two ways which makes Index File far more efficient in range searching.