CSE3207 Project #2

Implementation of a Disk Based B$^+$-Tree

- ✓ Assignment Date: May 16, 2019
- ✓ Due Date: June 21, 2019
- ✓ TA: 진영화, jyh2378@naver.com
- ✓ Weight: 15%
- ✓ Please read this description carefully.

I. Environment

1. Programming Language: C/C^{++}

2. Input data type: integers (4 bytes each)

II. Assignment

1. Specifications

- Your B$^+$-tree should be stored in a single binary file, e.g., *"btree.bin"*

- Do not load a whole B$^+$-tree index in the main memory.

- A node is corresponding to a block. Thus, the size of each node exactly one block.

- Each block is identified by its BID (Block ID), which represented as also a 4-byte integer.

- BID starts from 1, and 0 indicates a NULL block.

- Physical offset of a block in the B$^+$-Tree binary file is calculated in the following way:

$$12 + ((\text{BID}-1) * \textit{BlockSize})$$

- *BlockSize* represents the physical size of a B$^+$-Tree node, e.g., 1024 bytes.

- The number of entries per node is calculated as follows:

$$(\textit{BlockSize} - 4) / 8$$

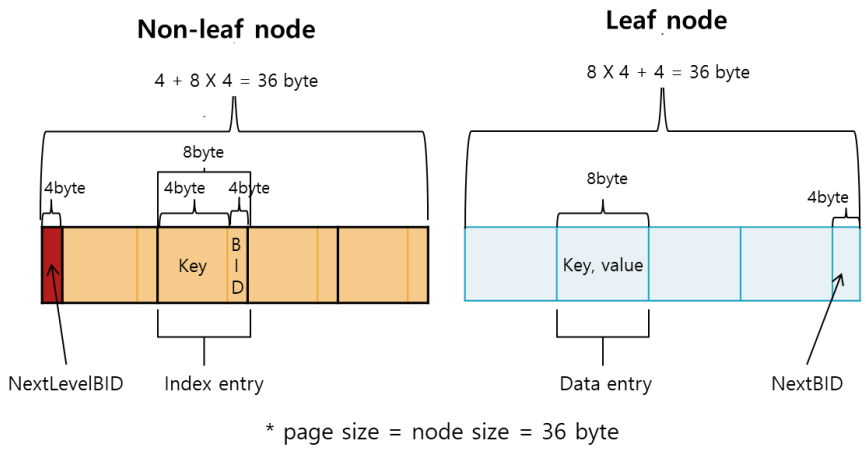

Figure 1. Details of nodes

2. Entry structure

A. Index entry (Non-leaf)

<Key, NextLevelBID> (8 bytes)

Key: an integer

NextLevelBID: a right child node's BID in the B^+-Tree binary file.

B. Data entry (Leaf)

<Key, Value> (8 bytes)

Key: an integer

Value: an integer

3. Node structure

A. Non-leaf node

<NextLevelBID, Index entry, Index entry, …>

NextLevelBID: a left child node's BID in the B^+-Tree binary file.

Index entry: as described above (II.2.A)

B. Leaf node

<Data entry, Data entry,…,Data entry, NextBID>

Date entry: as described above (II.2.B)

NextBID: the BID of the next leaf node in the B^+-Tree binary file.

4. B^+-Tree binary file structure

A. File header

<BlockSize, RootBID, Depth> (12 bytes)

BlockSize: the physical size of a B^+-Tree node, which represented as an integer.

RootBID: the root node's BID in the B^+-Tree binary file.

Depth: the depth of the B^+-Tree. By using this variable, we can check whether a node is leaf or not.

B. The rest part of the file stores all the nodes in the B^+-Tree.

4. Operations to be implemented

A. Insertion

B. Point (exact) search

C. Range search

D. Print B^+-Tree (print the root node and its child nodes only, i.e., top-2 levels only)

5. Example of a B+-Tree class:

```
class BTree {
public:
  Btree(const char *fileName, int blockSize);

  bool insert(int key, int rid);
  void print();
  int* search(int key); // point search
  int* search(int startRange, int endRange); // range search
};
// Test
int main (int argc, char* argv[])
{
  char command = argv[1][0];
  BTree myBtree = new BTree(any parameter);
  switch(command)
  {
  case 'c' :
  // create index file
        break;
  case 'i' :
  // insert records from [records data file], ex) records.txt
        break;
  case 's' :
  // search keys in [input file] and print results to [output file]
        break;
  case 'r' :
  // search keys in [input file] and print results to [output file]
        break;
  case 'p' :
  // print B+-Tree structure to [output file]
        break;
}
```

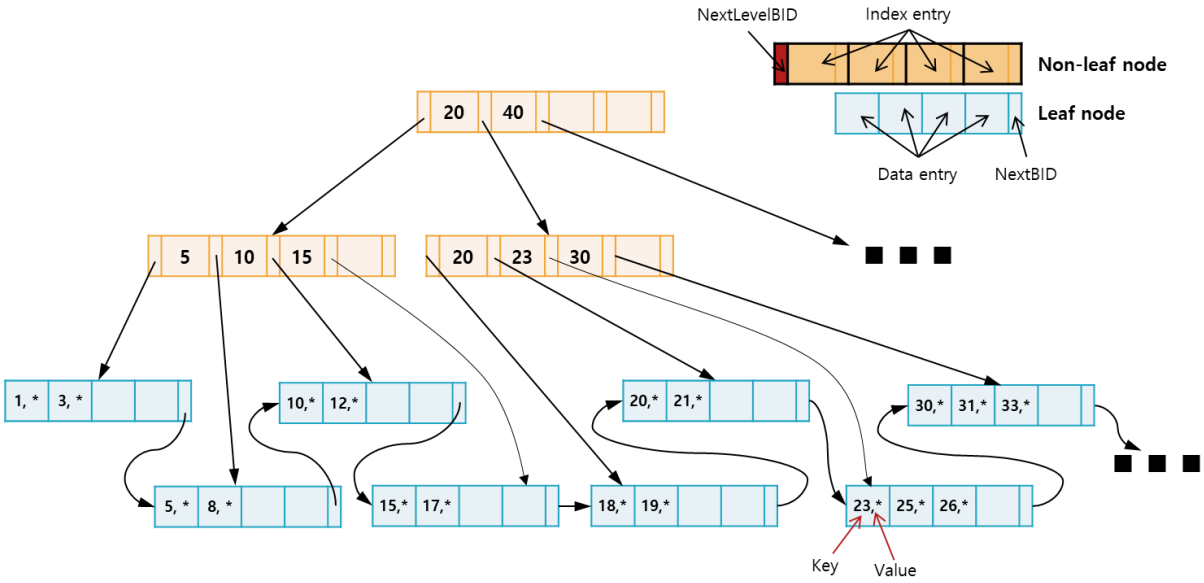

Figure 2. Example of a B+-Tree, where the block size sets to 36 bytes.

III. Command Interfaces and test

※ Please make sure to strictly follow the command interfaces as described in this section. Otherwise, you will not get any credit.

1. Index creation

btree.exe c [btree binary file] [block_size]

e.g., btree.exe c Btree.bin 1028

2. Insertion

btree.exe i [btree binary file] [records data text file]

e.g., btree.exe i Btree.bin records.txt

Included “record.txt” file has the following format:

```
Key, ID\n
Key, ID\n
Key, ID\n
…
```

3. Point (exact) search

btree.exe s [btree binary file] [input text file] [output text file]

e.g., btree.exe s Btree.bin search.txt result.txt

“search.txt” has the following format:

```
Key\n
Key\n
Key\n
…
```

The result of search operation, “result.txt” should have “Keys” in the following format:

```
Key, ID\n
Key, ID\n
Key, ID\n
…
```

4. Range search

btree.exe r [btree binary file] [input text file] [output text file]

e.g., btree.exe r Btree.bin range_search.txt range_result.txt

“range_search.txt” has the following format:

StartRange, EndRange \n	// 1st query
StartRange, EndRange \n	// 2nd query
StartRange, EndRange \n	// …
…	

The result of range search operation, "range_result.txt" should have "Keys" in the following format:

Key, ID \t Key, ID \t Key, ID …\n	// results of 1st query
Key, ID \t Key, ID …\n	// results of 2nd query
Key, ID \t Key, ID \t Key, ID \t Key, ID \t Key, ID …\n	// …
…	

5. Print B$^+$-Tree structure

btree.exe p [btree binary file] [output text file]

e.g., btree.exe p Btree.bin print.txt

"print.txt" should have the following format:

<Level> \n
Key, Key, Key … \n
<Level> \n
Key, Key, Key, Key, Key, Key … \n

Example of "print.txt" file

<0>
20, 40
<1>
5, 10, 15, 20, 23, 30, 40, 50

IV. Submission

1. Files to submit

1) A single source file (**btree.cpp or btree.c**) – ***please follow these names***

2) README.doc file containing the followings

- What you have implemented and what you have not
- Brief explanation of your implementation (Avoid any fancy designs and make it less than 1 page)
- How to compile and run
- Talk about your experience of doing this project
- Write your available contact information such as phone number (just in case)

3) Please submit the files in .zip format with the filename corresponding to your student id (e.g.

20191234.zip)

2. Where to submit:

I-Class website (learn.inha.ac.kr)

3. Deadline

1) **June 21, 2019 23:59:59**

2) Delay Penalty

- 20% penalty per day

V. Score

1) B+Tree: 80%

- search: 20%

- range search: 20%

- insert: 25%

- print: 15%

2) README.txt & comments: 20%

3) COPY: - 2100%