Module 3. B^+-Tree

Younghoon Kim
(nongaussian@hanyang.ac.kr)

Problem

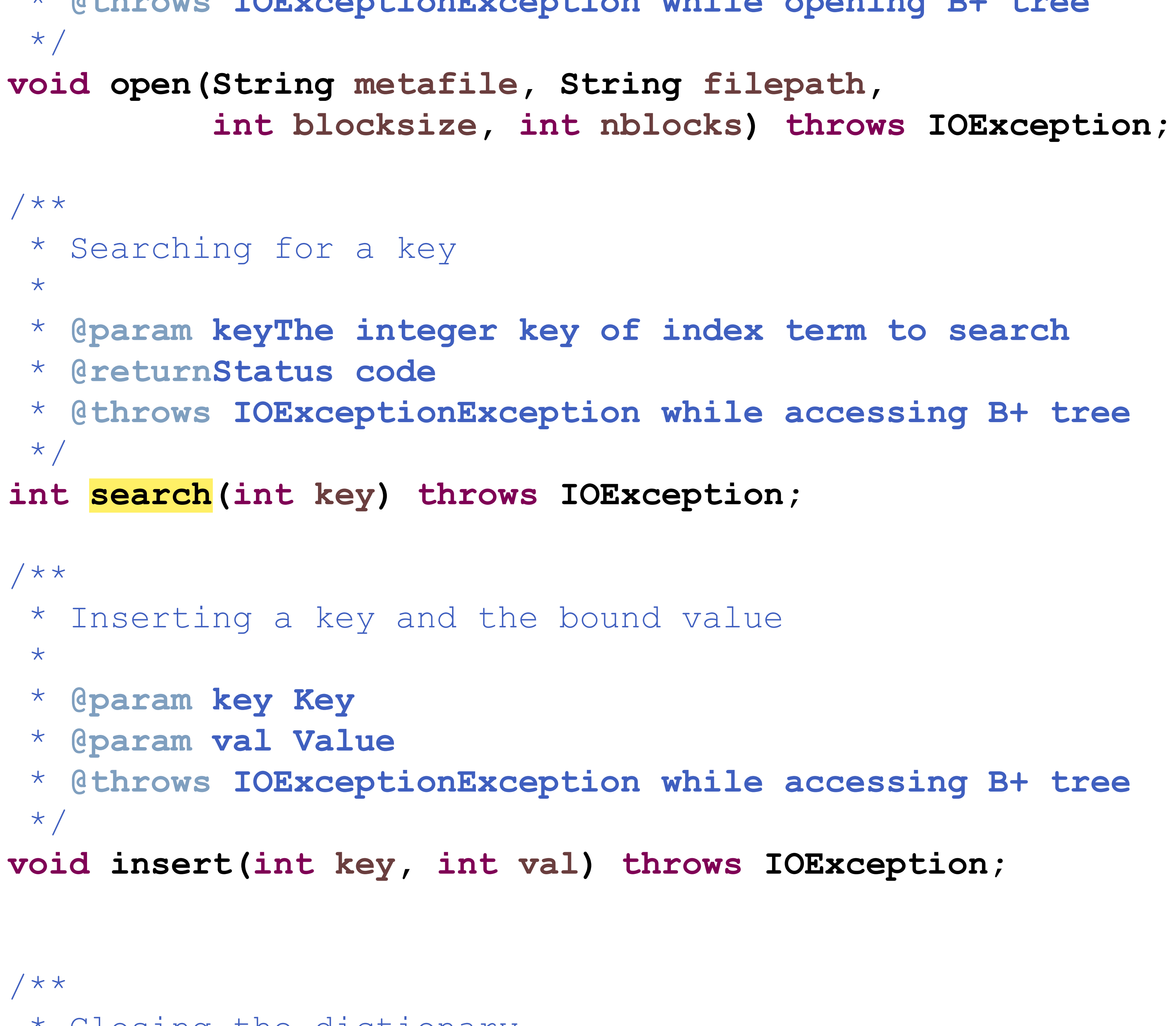
- Suppose that
  - Integer tuples (e.g., <5, 10000000>) are inserted into a B+ tree one by one
    - The first integer of each tuple is a key
    - And the second integer is a pointer (=value)
- Goal
  - Implement B^{+}-tree with interfaces
    - Search
      - Input: an integer search-key
      - Output: an integer value
    - Insert
      - Input: a pair of integer key and integer value
      - Output: none

Code Template

- We provide a package of
  - A maven project created in Eclipse
- It contains
  - Template codes (`edu.hanyang.submit.HanyangSEBPlusTree.java`)
  - Interface: BPlusTree

Interface

```
public interface BPlusTree {

    /**
     * Opening and initializing the directory
     *
     * @param metafile A meta-file with configurations for the di
     * @param filepath Directory or path for opening the dictiona
     * @param blocksize Available blocksize in the main memory of
B+ tree
     * @param nblocks Available block numbers in the main memory
B+ tree
     * @throws IOException Exception while opening B+ tree
     */
    void open(String metafile, String filepath,
              int blocksize, int nblocks) throws IOException;

    /**
     * Searching for a key
     *
```



```
 * @throws IOExceptionException while opening B+ tree
 */
void open(String metafile, String filepath,
          int blocksize, int nblocks) throws IOException;

/**
 * Searching for a key
 *
 * @param keyThe integer key of index term to search
 * @returnStatus code
 * @throws IOExceptionException while accessing B+ tree
 */
int search(int key) throws IOException;

/**
 * Inserting a key and the bound value
 *
 * @param key Key
 * @param val Value
 * @throws IOExceptionException while accessing B+ tree
 */
void insert(int key, int val) throws IOException;


/**
 * Closing the dictionary
```

```
    int search(int key) throws IOException;

    /**
     * Inserting a key and the bound value
     *
     * @param key Key
     * @param val Value
     * @throws IOExceptionException while accessing B+ tree
     */
    void insert(int key, int val) throws IOException;


    /**
     * Closing the dictionary
     *
     * @throws IOExceptionException while closing B+ tree
     */
    void close() throws IOException;
}
```

Example of A Metafile (Ascii file)

30209	Index of root node
284	Fan-out
4096	Blocksize

```
/**
     * Opening and initializing the directory
     *
     * @param metafile A meta-file with configurations for the dictionary
(e.g., pagesize)
     * @param filepath Directory or path for opening the dictionary
     * @param blocksize Available blocksize in the main memory of the current
system for B+ tree
     * @param nblocks Available block numbers in the main memory of the
current system for B+ tree
     * @throws IOException Exception while opening B+ tree
     */
    void open(String metafile, String filepath,
              int blocksize, int nblocks) throws IOException;
```

Sample: open

```
@Override public void open(String metapath, String filepath, int
blocksize, int nblocks) throws IOException {
    this.blocksize = blocksize;
    this.nblocks = nblocks;
    this.buf = new byte[blocksize];
    this.buffer = ByteBuffer.wrap(buf);
    this.maxKeys = (blocksize - 16) / 8;

    raf = new RandomAccessFile(filepath, "rw");
}
```

RandomAccessFile

- java.io.RandomAccessFile
  - The RandomAccessFile class treats the file as an array of Bytes
  - You can write your data in any position of the Array
  - It uses a pointer that holds the current position
- Example

```
public class RandomAccessFileEx {

  static final String FILEPATH = "C:/Users/nikos7/Desktop/input.txt";

  public static void main(String[] args) {
    try {
      System.out.println(new String(readFromFile(FILEPATH, 150, 23)));

      writeToFile(FILEPATH, "JavaCodeGeeks Rocks!", 22);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
```

RandomAccessFile

- Example

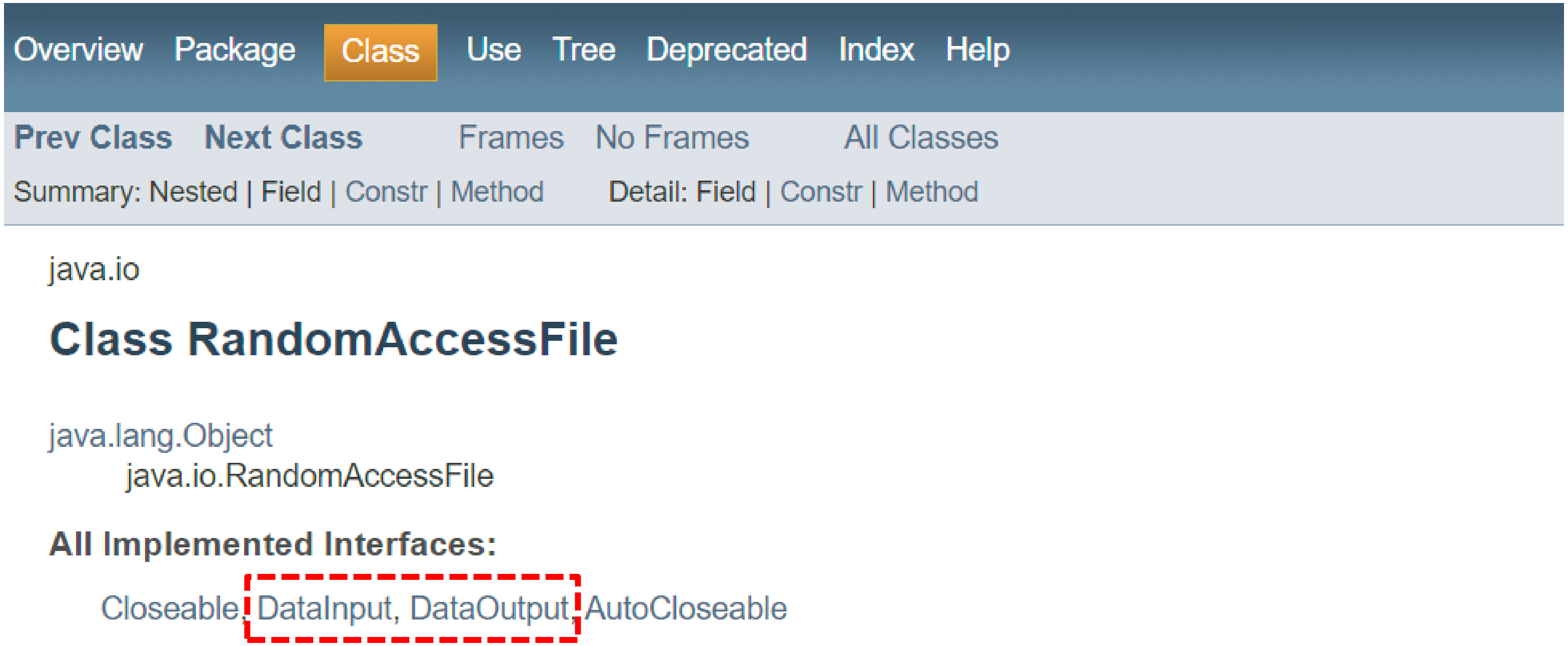
```
private static byte[] readFromFile(String filePath, int position, int size)
        throws IOException {
     RandomAccessFile file = new RandomAccessFile(filePath, "r");
    file.seek(position);
    byte[] bytes = new byte[size];
    file.read(bytes);
    file.close();
    return bytes;
 }

private static void writeToFile(String filePath, byte[] data, int position)
        throws IOException {
     RandomAccessFile file = new RandomAccessFile(filePath, "rw");
    file.seek(position);
    file.write(data);
    file.close();
}
}
```

https://examples.javacodegeeks.com/core-java/io/randomaccessfile/java-randomaccessfile-example/

RandomAccessFile: Methods

- getFilePointer()
  - to get the current position of the pointer
- seek(int)
  - to set the position of the pointer
- read(byte[] b)
  - to reads up to b.length bytes of data from the file into an array of bytes
- write(byte[] b)
  - to write b.length bytes from the specified byte array to the file, starting at the current file pointer

Class RandomAccessFile



java.io

Class RandomAccessFile

java.lang.Object
    java.io.RandomAccessFile

All Implemented Interfaces:

Closeable, DataInput, DataOutput, AutoCloseable

```
public final int readInt() throws IOException
```

SAMPLE SKELETONS

search

```
public int search(int key) throws IOException {
  Block rb = readBlock(rootindex);
  return _search(rb, key);
}

private int _search(Block b, int key) throws IOException {
  if (b.type == 1) { // non-leaf
    …
    if (block.keys[i] < key) {
      child = readBlock(b.vals[i]);
    }
    …
  } else { // leaf
   /* binary or linear search */
   // if exists,
   return val;
   // else
   return -1;
  }
}
```

insert

```
public void insert(int key, int val) throws IOException {
  Block block = searchNode(key);

  if (block.nkeys + 1 > maxKeys) {
      Block newnode = split(block, key, val);
      insertInternal(block.parent, newnode.my_pos);
  } else {
      …
  }
}
```

```
procedure insert(value $V$, pointer $P$)
    find the leaf node $L$ that should contain value $V$
    insert_entry($L, V, P$)
end procedure

procedure insert_entry(node $L$, value $V$, pointer $P$)
    if ($L$ has space for $(V, P)$)
        then insert $(V, P)$ in $L$
        else begin /* Split $L$ */
            Create node $L'$
            Let $V'$ be the value in $L.K_1, \ldots, L.K_{n-1}, V$ such that exactly
                $\lceil n/2 \rceil$ of the values $L.K_1, \ldots, L.K_{n-1}, V$ are less than $V'$
            Let $m$ be the lowest value such that $L.K_m \geq V'$
            /* Note: $V'$ must be either $L.K_m$ or $V$ */
            if ($L$ is a leaf) then begin
                move $L.P_m, L.K_m, \ldots, L.P_{n-1}, L.K_{n-1}$ to $L'$
                if ($V < V'$) then insert $(P, V)$ in $L$
                else insert $(P, V)$ in $L'$
            end
            else begin
                if ($V = V'$) /* $V$ is smallest value to go to $L'$ */
                    then add $P, L.K_m, \ldots, L.P_{n-1}, L.K_{n-1}, L.P_n$ to $L'$
                    else add $L.P_m, \ldots, L.P_{n-1}, L.K_{n-1}, L.P_n$ to $L'$
                delete $L.K_m, \ldots, L.P_{n-1}, L.K_{n-1}, L.P_n$ from $L$
                if ($V < V'$) then insert $(V, P)$ in $L$
                else if ($V > V'$) then insert $(V, P)$ in $L'$
                /* Case of $V = V'$ handled already */
            end
            if ($L$ is not the root of the tree)
                then insert_entry($parent(L), V', L'$);
                else begin
                    create a new node $R$ with child nodes $L$ and $L'$ and
                        the single value $V'$
                    make $R$ the root of the tree
                end
            if ($L$) is a leaf node then begin /* Fix next child pointers */
                set $L'.P_n = L.P_n$;
                set $L.P_n = L'$
            end
        end
end procedure
```

Test Setting

- Heapsize (for both inserting and searching)
  - 16Mb
- Datasize
  - About 100 Mb (will be available today) → about 15M keys
- Order to insert key and value
  - Random order
- Query keys: 1M keys
  - Not known
- Evaluation
  - Insertion time + Query time