

B+ tree implementation assignment

2013011800 구장희

Summary of Algorithm

(0) Base

- Node class 는 node들의 공통적인 속성을 정의한다. Leaf와 NonLeaf의 개별적인 속성은 Node class 를 상속받는 LeafNode , NonLeafNode class 안에 정의된다.
- 이는 level(각 노드의 레벨) , id(각 노드의 id) , int형 array p[] (b X 2 배열. leaf 노드일 경우 key-value 쌍이 저장되고 NonLeaf 노드일 경우 key-ptr 쌍이 저장된다.) , b(max # of child/value) , r(right most pointer)
- NonLeaf 노드는 p배열의 ptr로 자식의 id값을 저장한다. Leaf노드는 p배열의 value로 key값에 해당하는 value를 저장한다.
- leaf 노드를 level 0으로 해서, 트리의 depth가 늘어난다면 순차적으로 늘어난다.(1,2,3 ...)
- 각 노드는 level과 id값을 가지는데, id값은 레벨마다 첫번째 key로 노드를 정렬 한 뒤 왼쪽부터 0번을 받는다.
- level 마다 node를 담는 ArrayList가 사용된다.
- level을 key로, 해당 level의 ArrayList를 value로 갖는 HashMap을 사용한다.

(1) Insertion

[code]

```
public static void Insertion(int key, int value) {
    LeafNode newleafnode = null;
    if (root == null) {
        root = new LeafNode(0, 0, b, key, value, -1);
        addNodeToMap(0, root);
        maxlevel = 0;
        return;
    } else if (root.isLeafNode) {
        ((LeafNode) root).InsertAndSort(key, value);
        if (((LeafNode) root).isOverflowed()) {
            overflowInLeafNode((LeafNode) root);
        }
    } else if (!root.isLeafNode) {
        temp = whereToInsert(root, key);
        if (!temp.isLeafNode) {
            System.out.println("Fail!!![0]");
        }
        newleafnode = (LeafNode) temp;
        newleafnode.InsertAndSort(key, value);
        if (newleafnode.isOverflowed()) {
            overflowInLeafNode((LeafNode) newleafnode);
        }
    }
}
```

[description]

삽입함수가 호출되면 key와 value를 인자로 받는다.

##1. root가 null이면

root를 새로 만들고 , addNodeToMap 함수를 호출하여 그 root를 HashMap에 넣는다.

##2. root가 leaf 노드이면

key와 value를 root에 삽입하고 정렬(InsertAndSort 함수 호출)한다.

그 다음 오버플로가 발생하면 overflowInLeafNode를 호출한다.

##3. root가 non-leaf 노드이면

whereToInset 함수를 호출하여 key가 삽입 될 leaf 노드를 찾는다.

찾은 노드에 key를 삽입하고 정렬(InsertAndSort 함수 호출)한다.

그 다음 오버플로가 발생하면 overflowInLeafNode를 호출한다.

(2)Deletion

[code]

```
public static void Deletion(int key) {
    LeafNode temp;
    NonLeafNode parent;
    temp = (LeafNode) findLeafNodeByKey(key);
    parent = (NonLeafNode) findParentNode(temp);
    if (temp == null)
        return;
    temp.DeleteAndSort(key);
    if (parent.isContainKey(key) != -1) {
        int tmp = parent.isContainKey(key);
        parent.p[tmp][0] = temp.p[0][0];
    }
    if (temp.isUnderflowed()) {
        underflowInLeafNode(temp);
    }
}
```

[description]

삭제할 키를 가지고 있는 leaf 노드를 찾는다. 없다면 return.

찾았으면 DeleteAndSort함수를 호출해서 해당 key값을 삭제하고 배열을 정렬한다.

부모가 그 key를 가지고 있는 경우는, 부모 노드의 배열에서 그 key값의 위치에(isContainKey 함수를 호출해서 index를 찾고)

새로 정렬된 첫번째 key를 넣는다.

삭제 후 언더플로가 발생하면 underflowInLeafNode 함수를 호출한다.

(3)Search Key Search

[code]

```
public static void SearchSingleKey(Node entry, int key) {
    Node temp = null;
    int index = 0;
    if (entry.isLeafNode) {
        index = SearchKeyInNode(entry, key);
        if (index != -1) {
            System.out.println(entry.p[index][1]);
            return;
        } else {
            System.out.println("NOT FOUND");
            return;
        }
    } else {
        for (int i = 0; i < entry.m - 2; i++) {
            System.out.print(entry.p[i][0] + ", ");
        }
        System.out.print(entry.p[entry.m - 2][0]);
        System.out.println();

        if (key >= entry.p[entry.m - 2][0]) {
            temp = findNodeByLevAndId(entry.level - 1, entry.r);
        } else {
            for (int i = 0; i < entry.m - 1; i++) {
                if (key < entry.p[i][0]) {
                    temp = findNodeByLevAndId(entry.level - 1, entry.p[i][1]);
                    break;
                }
            }
        }
        SearchSingleKey(temp, key);
    }
}
```

[description]

인자로 entry 노드와 찾을 key값을 받는다.(recursive하게 key를 찾기 위함)

##1. entry가 leaf 노드일때

SearchKeyInNode 함수를 호출해서 key가 있다면, 배열의 index를 return 받고 해당 key의 value를 출력한다.

##2. entry가 non leaf 노드일때

entry의 모든 key값을 출력한다.

그후, entey의 key값을 배열의 key값과 비교해서 탐색할 자식 노드를 찾는다.

찾았으면, 인자를 그 자식노드로 하고 key값을 그대로 해서 recursive하게 호출.

(4)Ranged Search

[code]

```
public static void RangedSearch(int key1, int key2) {
    LeafNode temp = null;
    for (int i = 0; i < nmap.get(0).size(); i++) {
        temp = (LeafNode) nmap.get(0).get(i);
        for (int j = 0; j < temp.m; j++) {
            if ((key1 <= temp.p[j][0]) && (temp.p[j][0] <= key2)) {
                System.out.println(temp.p[j][0] + "," + temp.p[j][1]);
            }
        }
    }
}
```

[description]

HashMap에서 level 0의 노드들을 가지고있는 ArrayList를 꺼내고,
그 ArrayList의 모든 노드를 반복문으로 돌면서,
배열의 key값이 key1 과 key2 사이에 들어간다면 출력한다.

Detailed description

(1) Method in bptree.java

```
/**
 * Write meta-data into .dat file using PrintWriter
 * @param path      path of .dat file
 * @param filename   name of .dat file
 * @return void
 */
public static void writeMetaData(String path , String filename) throws FileNotFoundException

/**
 * When new node is created, add to HashMap.
 * @param level      level of new node
 * @param node       new node
 * @return void
 */
public static void addNodeToMap(int level, Node node)

/**
 * Find Node by its level and id
 * @param level      level of node
 * @param id         id of node
 * @return Node      that node
 */
public static Node findNodeByLevAndId(int level, int id)

/**
 * Find LeafNode by its key
 * @param key        key of a LeafNode
 * @return Node      that LeafNode
 */
public static Node findLeafNodeByKey(int key)

/**
 * Find appropriate node when insertion is invoked
 * @param entry      start point. It can be root or not
 * @param key        the key to be inserted
 * @return Node      appropriate LeafNode to be inserted
 */
public static Node whereToInsert(Node entry, int key)

/**
 * Print all node by level(ascending order)
 * @param void
 * @return void
 */
public static void printCurrentStatus()
```

```

/**
 * Find parent node
 * @param Node      child node
 * @return Node      parent of the child node
 */
public static Node findParentNode(Node node)

/**
 * Sort all node by its first key using HashMap,ArrayList
 * As a result, all own id and pointer to child node is re-distributed
 * @param void
 * @return void
 */
public static void sortAllNode()

/**
 * Transfer to left sibling(in LeafNode)
 * @param Leafnode      where underflow occurred
 * @return void
 */
public static void transferToLeftLeafSibling(LeafNode node)

/**
 * Transfer to left sibling(in NonLeafNode)
 * @param NonLeafnode      where underflow occurred
 * @return void
 */
public static void transferToLeftNonLeafSibling(NonLeafNode node)

/**
 * Merge to left sibling(in LeafNode)
 * @param Leafnode      where underflow occurred
 * @return void
 */
public static void mergeToLeftLeafSibling(LeafNode node)

/**
 * Merge to left sibling(in NonLeafNode)
 * @param NonLeafnode      where underflow occurred
 * @return void
 */
public static void mergeToLeftNonLeafSibling(NonLeafNode node)

/**
 * Transfer to right sibling(in LeafNode)
 * @param Leafnode      where underflow occurred
 * @return void
 */
public static void transferToRightLeafSibling(LeafNode node)

/**
 * Transfer to right sibling(in NonLeafNode)
 * @param NonLeafnode      where underflow occurred

```

```

        * @return void
*/
public static void transferToRightNonLeafSibling(NonLeafNode node)
/**
    * Merge to right sibling(in LeafNode)
    * @param Leafnode      where underflow occurred
    * @return void
*/
public static void mergeToRightLeafSibling(LeafNode node)

/**
    * Merge to right sibling(in NonLeafNode)
    * @param NonLeafnode    where underflow occurred
    * @return void
*/
public static void mergeToRightNonLeafSibling(NonLeafNode node)

/**
    * Handle overflow in LeafNode
    * @param Leafnode      where overflow occurred
    * @return void
*/
public static void overflowInLeafNode(LeafNode node)

/**
    * Handle overflow in NonLeafNode
    * @param NonLeafnode    where overflow occurred
    * @return void
*/
public static void overflowInNonLeafNode(NonLeafNode node)

/**
    * Handle underflow in LeafNode
    * @param Leafnode      where underflow occurred
    * @return void
*/
public static void underflowInLeafNode(LeafNode node)

/**
    * Handle underflow in NonLeafNode
    * @param NonLeafnode    where underflow occurred
    * @return void
*/
public static void underflowInNonLeafNode(NonLeafNode node)

/**
    * Insert key,value to B+ tree
    * @param key            key to be inserted
    * @param value          value to be inserted
    * @return void
*/
public static void Insertion(int key, int value)

/**

```



```

        * Delete key to B+ tree
        * @param key          key to be deleted
        * @return void
    */
    public static void Deletion(int key)

    /**
        * Find index(where the key is stored in the array of node)
        * @param node          node to be searched
        * @param key          key in that node
        * @return index        index of key
    */
    public static int SearchKeyInNode(Node node, int key)

    /**
        * Search single key
        * @param node          entry node as starting point of searching
        * @param key          key to be searched
        * @return void
    */
    public static void SearchSingleKey(Node entry, int key)

    /**
        * Search key in specific range(print all keys between key1 and key2)
        * @param key1
        * @param key2
        * @return void
    */
    public static void RangedSearch(int key1, int key2)

```

(2) Method in Node.java

```
/**
 * When the node has key, return index of array
 * @param key
 * @return index
 */
public int isContainKey(int key)

/**
 * Delete a key and sort own array
 * @param key
 * @return void
 */
public void DeleteAndSort(int key)

public void arraySwap(int[][] arr, int a, int b)

public void printNode()
```

(3) Method in LeafNode.java , NonLeafNode.java

```
/**
 * Insert a key and sort own array
 * @param key
 * @return void
 */
public void InsertAndSort(int key, int value)

/**
 * Delete half of own array especially when overflow occurred
 * @param void
 * @return void
 */
public void deleteHalf()

public boolean isOverflowed()
```

Instruction for compiling

*Settings

[1]

*.java file은
/Users/~~~~/bptree_1/src/bptree_1/
에 둔다.

/Users/~~~~/bptree_1/src/
에 input.csv , delete.csv를 둔다.



```
src — bash — 106x26
[K00ui-MacBook-Pro:bptree_1 KJH$ pwd
/Users/KJH/eclipse-workspace/bptree_1/src/bptree_1
[K00ui-MacBook-Pro:bptree_1 KJH$ ls
LeafNode.class      Node.java            bptree$1.class
LeafNode.java       NonLeafNode.class   bptree.class
Node.class           NonLeafNode.java     bptree.java
[K00ui-MacBook-Pro:bptree_1 KJH$ cd ..
[K00ui-MacBook-Pro:src KJH$ pwd
/Users/KJH/eclipse-workspace/bptree_1/src
[K00ui-MacBook-Pro:src KJH$ ls
bptree_1  delete.csv  index.dat  input.csv
[K00ui-MacBook-Pro:src KJH$
```

[2]

bptree.java 의 main method에서
String path = "/Users/~~~~/bptree_1/src/";
로 바꾼다.

```
712
713     public static void main(String[] args) throws IOException {
714         String path = "/Users/KJH/eclipse-workspace/bptree_1/src/";
715         String indexfile, datafile;
716         String line;
717         int startkey, endkey, searchkey;
718         PrintWriter pw;
719     }
```

*Compile

지정한 path(/Users/~/bptree_1/src/) 에서

```
$ pwd
/Users/KJH/eclipse-workspace/bptree_1/src
```

[컴파일]

```
$ javac bptree_1/bptree.java
```

[실행]

```
$ java bptree_1/bptree -c index.dat 4
$ java bptree_1/bptree -i index.dat input.csv
$ java bptree_1/bptree -d index.dat delete.csv
$ java bptree_1/bptree -s index.dat 15
$ java bptree_1/bptree -r index.dat 15 50
```

Composition of index.dat

[-c 옵션 실행하면]

c+ ” ”+b(max # of child/value) 쓴다.

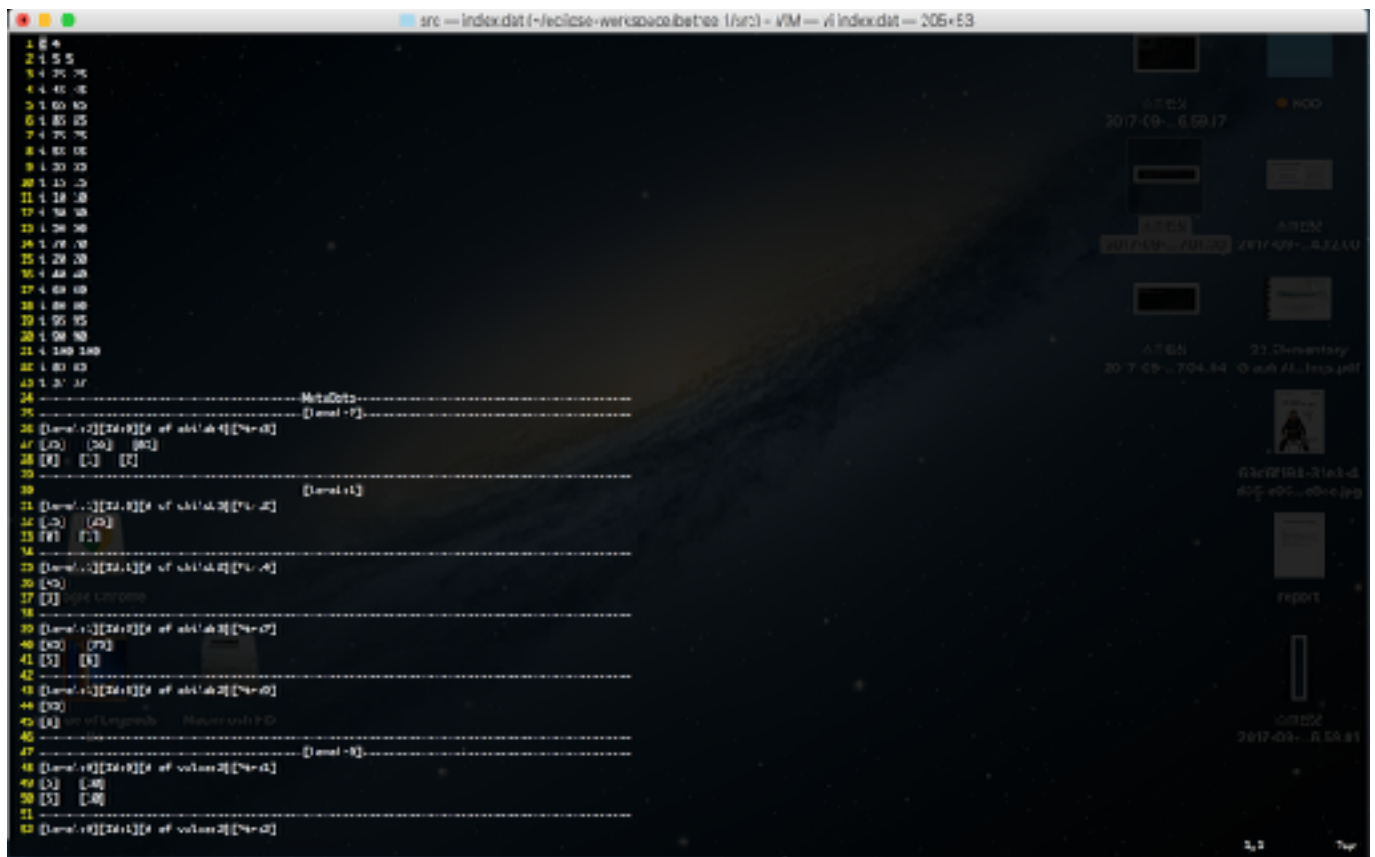
[-i 옵션 실행하면]

i+ ” ”+key+ ” ”+value 쓴다.

[mata-data]

-i 옵션 , -d 옵션 실행후, meta data 갱신한다.

node의 level , id , m , b , p[] , r 모두 출력



```
1 1 1 1 1 1
2 1 1 1 1 1
3 1 1 1 1 1
4 1 1 1 1 1
5 1 1 1 1 1
6 1 1 1 1 1
7 1 1 1 1 1
8 1 1 1 1 1
9 1 1 1 1 1
10 1 1 1 1 1
11 1 1 1 1 1
12 1 1 1 1 1
13 1 1 1 1 1
14 1 1 1 1 1
15 1 1 1 1 1
16 1 1 1 1 1
17 1 1 1 1 1
18 1 1 1 1 1
19 1 1 1 1 1
20 1 1 1 1 1
21 1 1 1 1 1
22 1 1 1 1 1
23 1 1 1 1 1

-----MetaData-----
24 [level:1][id:1][m:1][b:1][p:1][r:1]
25 [level:1][id:1][m:1][b:1][p:1][r:1]
26 [level:1][id:1][m:1][b:1][p:1][r:1]
27 [level:1][id:1][m:1][b:1][p:1][r:1]
28 [level:1][id:1][m:1][b:1][p:1][r:1]
29 [level:1][id:1][m:1][b:1][p:1][r:1]
30 [level:1][id:1][m:1][b:1][p:1][r:1]
31 [level:1][id:1][m:1][b:1][p:1][r:1]
32 [level:1][id:1][m:1][b:1][p:1][r:1]
33 [level:1][id:1][m:1][b:1][p:1][r:1]
34 [level:1][id:1][m:1][b:1][p:1][r:1]
35 [level:1][id:1][m:1][b:1][p:1][r:1]
36 [level:1][id:1][m:1][b:1][p:1][r:1]
37 [level:1][id:1][m:1][b:1][p:1][r:1]
38 [level:1][id:1][m:1][b:1][p:1][r:1]
39 [level:1][id:1][m:1][b:1][p:1][r:1]
40 [level:1][id:1][m:1][b:1][p:1][r:1]
41 [level:1][id:1][m:1][b:1][p:1][r:1]
42 [level:1][id:1][m:1][b:1][p:1][r:1]
43 [level:1][id:1][m:1][b:1][p:1][r:1]
44 [level:1][id:1][m:1][b:1][p:1][r:1]
45 [level:1][id:1][m:1][b:1][p:1][r:1]
46 [level:1][id:1][m:1][b:1][p:1][r:1]
47 [level:1][id:1][m:1][b:1][p:1][r:1]
48 [level:1][id:1][m:1][b:1][p:1][r:1]
49 [level:1][id:1][m:1][b:1][p:1][r:1]
50 [level:1][id:1][m:1][b:1][p:1][r:1]
51 [level:1][id:1][m:1][b:1][p:1][r:1]
52 [level:1][id:1][m:1][b:1][p:1][r:1]
```