

# 2020 B+ Tree Implementation Assignment

Database Systems (ITE2038)

컴퓨터소프트웨어학부

2019054957 이용우

## 1. Environment

OS: Windows

Language: Java (jdk 1.8.0\_241)

## 2. Configuration Files

BpTree.java

- Root 노드에 대하여 Insert, Delete, Search를 수행한다.
- Disk에 노드 정보를 저장하고, 이를 다시 메모리로 불러오는 함수가 구현되어 있다.

Node.java

- Abstract Class로 만들어 LeafNode와 NonLeafNode의 공통된 structure와 함수가 구현되어 있다.

LeafNode.java

- LeafNode를 구성하는 멤버 변수와 함수들이 구현되어 있다.

NonLeafNode.java

- NonLeafNode를 구성하는 멤버 변수와 함수들이 구현되어 있다.

Main.java

- Command Line에서의 명령을 처리한다.

### 3. Summary Algorithm && Description of Codes

Insertion

- LeafNode.java

```
public Node insert(Node root, int key, int value)
```

Key 가 들어갈 위치를 Binary Search 로 찾아 삽입하여 정렬된 상태를 유지한다.

Key 를 삽입하여 List 의 크기가 Degree - 1 을 초과하면 Split 한다.

새로 NonLeafNode 를 생성하여 Right Sibling 의 Leftmost Key 값을 넣어준다.

Split 이 일어나는 노드가 Root 노드라면 부모 노드를 반환하여 Root 노드를 갱신한다.

이미 존재하는 Key 를 삽입할 때에는 예외처리를 한다.

- NonLeafNode.java

```
public Node insert(Node root, int key, int value)
```

Child 의 위치를 Binary Search 로 찾아 Insert 를 재귀적으로 수행한다.

해당 Child 에 Overflow 가 발생하면 Split 을 수행한다.

Child 가 LeafNode 이면 Right Sibling 의 Leftmost Key 값을, NonLeafNode 이면 중간 지점의 Key 값을 새로운 NonLeafNode 에 넣어준다.

Split 이 일어나는 노드가 Root 노드라면 부모 노드를 반환하여 Root 노드를 갱신한다.

```
public Node split()
```

SplitNode 를 생성하여 Key 와 values 를 절반으로 나눈다.

LeafNode 에서는 Right Node 를 가리키는 포인터를 연결해준다.

- BpTree.java

```
public void insert(int key, int value) {  
    Node temp = root.insert(root, key, value);  
    if(temp != null) {  
        root = temp;  
    }  
}
```

Insert 수행 후 Split 에 의해 새로운 노드가 반환되면 Root 노드를 갱신한다.

## Deletion

### - LeafNode.java

```
public Node delete(Node root, int key)
```

해당 노드에 Key 와 일치하는 Key 값이 있는지 찾는다.

존재하면 Key 와 Value 를 지워주고 존재하지 않으면 예외처리를 한다.

### - NonLeafNode.java

```
public Node delete(Node root, int key)
```

Child 의 위치를 Binary Search 로 찾아 Delete 를 재귀적으로 수행한다.

LeafNode 의 Leftmost Key 가 지워지면 부모 노드의 Key 를 갱신한다.

Child 에 Underflow 가 일어나면 Sibling 을 찾아 Borrow 나 Merge 를 수행한다.

Root 노드에 Merge 가 일어날 경우 Root 노드를 자식 노드로 갱신한다.

### - Node.java

```
public boolean canBorrow() {  
    return getSize() - 1 >= (degree - 1) / 2;  
}
```

Borrow 가 가능한 지 확인한다.

```
void borrowFromLeft(Node parentNode, Node siblingNode, int loc)
```

Left Sibling 에 대해서 Borrow 를 수행한다.

LeafNode 에서는 Left Sibling 에게서 Key 를 Borrow 하고 부모 노드의 Key 를 갱신한다.

NonLeafNode 에서는 부모 노드의 Key 를 하나 가져오고, 그 위치에 Left Sibling 의 Key 값으로 채운다.

```
void borrowFromRight(Node parentNode, Node siblingNode, int loc)
```

Right Sibling 에 대해서 Borrow 를 수행한다.

```
public void merge(Node parentNode, Node siblingNode, int loc)
```

노드를 병합하는 과정이다.

LeafNode 에서는 병합 후에 부모 노드의 키를 지워준다.

NonLeafNode 에서는 부모 노드의 키를 가져온 후 Sibling 과 병합을 수행한다.

- BpTree.java

```
public void delete(int key) {  
    Node temp = root.delete(root, key);  
    if(temp != null) {  
        root = temp;  
    }  
}
```

Delete 수행 후 Merge 에 의해 자식 노드가 반환되면 Root 노드를 자식 노드로 갱신한다.

### Single Key Search

- LeafNode.java

```
public int getValue(int key)
```

Key 가 존재하면 Value 를 반환한다.

- NonLeafNode.java

```
public int getValue(int key)
```

Child 의 위치를 Binary Search 로 찾아 재귀적으로 Search 를 수행한다.

Search 경로 중 NonLeafNode 에 방문하면 그 노드의 모든 Key 값을 출력한다. (Stdout)

LeafNode 에 도착하면 Value 만 출력한다.

- BpTree.java

```
public int keySearch(int key) {  
    return root.getValue(key);  
}
```

Search 결과인 Value 를 그대로 반환한다.

- Main.java

```
BpTree bptree = BpTree.Load(input, order);  
int value = bptree.keySearch(key);  
if(value != -1) {  
    System.out.println(value);  
}  
else {  
    System.out.println("NOT FOUND");  
}
```

받은 Value 를 다음과 같이 핸들링한다.

## Ranged Search

- LeafNode.java

```
void rangeSearch(int startKey, int endKey)
```

Keys 를 루프를 돌면서 startKey 와 endKey 의 범위 안에 있으면 각각 출력한다. (Stdout)

Right Node 가 null 이 아니면 Right Node 에 대해 재귀적으로 Search 를 수행한다.

방문한 노드의 Keys 의 Leftmost Key 가 endKey 보다 크다면 Return 하여 재귀함수를 끝낸다.

- NonLeafNode.java

```
void rangeSearch(int startKey, int endKey)
```

Child 의 위치를 Binary Search 로 찾아 rangeSearch 를 재귀적으로 수행한다.

- BpTree.java

```
public void rangeSearch(int start, int end) {  
    root.rangeSearch(start, end);  
}
```

Root 노드에 대해서 rangeSearch 를 수행한다.

## Save

index.dat 에 Tree 정보를 저장한다.

- BpTree.java

```
public void save(PrintWriter output, Node node) {  
    if(node == null) {  
        return;  
    }  
    if(node instanceof NonLeafNode) {  
        NonLeafNode nonLeafNode = (NonLeafNode)node;  
        for (int i = 0; i < nonLeafNode.getChildNode().size(); i++) {  
            save(output, nonLeafNode.getChildNode().get(i));  
        }  
        output.print(nonLeafNode.toString() + "\n");  
    }  
    else if(node instanceof LeafNode) {  
        output.print(node.toString() + "\n");  
    }  
}
```

Post-Order Traversal 을 이용하여 LeafNode 들의 정보부터 파일에 기록한다.

형식) LeafNode: key1,key2,key3... value1,value2,value3

NonLeafNode: key1,key2,key3...

Load

- BpTree.java

```
public static BpTree load(Scanner input, int order)
```

Index.dat 에 기록된 데이터를 메모리로 불러온다.

```
while(input.hasNextLine()) {  
    String info = input.nextLine();  
    String[] nodeInfo = info.split(" ");
```

데이터를 줄 단위로 읽어와서 파싱한다.

```
for(int i = 0; i < count.size(); i++) {  
    if(keysInfo.length + 1 == count.get(i).size()) {  
        NonLeafNode nonLeafNode = new NonLeafNode(order, keys, count.get(i));  
        count.get(i).clear();  
        if(count.size() == i + 1) {  
            count.add(new ArrayList<>());  
        }  
        count.get(i + 1).add(nonLeafNode);  
        root = nonLeafNode;  
        break;  
    }  
}  
keys.clear();
```

트리의 Level 별로 담아 놓는 count 라는 2D List 를 만든다.

Child 노드들의 수는 Keys + 1 이라는 것을 이용하여, NonLeafNode 의 Child 들을 구분한다.

Child 가 전부 count[i]에 들어가면 NonLeafNode 를 새로 만들어 count[i]를 자식 노드로 설정한다.

새로 만들어진 NonLeafNode 를 Root 로 설정한다.

파일 끝까지 반복하면 트리를 메모리에 모두 불러올 수 있다.

Save 된 Index.dat 파일의 구성은 아래 테스트 부분에서 자세히 확인할 수 있다.

#### 4. How to Compile Source Codes

```
C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment>java -version
java version "1.8.0_241"
Java(TM) SE Runtime Environment (build 1.8.0_241-b07)
Java HotSpot(TM) 64-Bit Server VM (build 25.241-b07, mixed mode)

C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment>cd Source

C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment\Source>ls
Main.java  bpTree

C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment\Source>javac Main.java bpTree/*.java

C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment\Source>ls
Main.class Main.java  bpTree

C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment\Source>java Main -c index.dat 20

C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment\Source>ls
Main.class Main.java  bpTree  index.dat
```

- 1) Source 폴더로 이동한다. (Main.java 와 bpTree 패키지가 존재한다.)
  - 2) javac Main.java bpTree/\*.java 명령어를 실행시킨다.  
(Main.java 와 bpTree 폴더 내의 java 확장자의 모든 파일을 컴파일한다.)
  - 3) class 파일이 생성된 것을 확인한다.
- 컴파일에 문제가 생긴다면 jdk 1.8.0\_241 버전을 다운받아 환경변수로 설정한다.

##### 실행 방법

- 1) Main.class 를 이용할 경우 다음과 같은 형식의 명령어를 사용한다.  
Ex) java Main -c index.dat 20
- 2) 제공된 jar 파일을 이용할 경우 B-tree\_Assignment 폴더에 있는  
B-tree\_Assignment.jar 로 다음과 같은 형식의 명령어를 사용한다.  
Ex) java -jar B-tree\_Assignment.jar -c index.dat 20

## 5. Testing

과제 채점과 동일하게 백만개 정도의 데이터를 가지고 테스트를 진행했다.

```
dataCreator.py > ...
1  import random
2
3  select = input("Insert(i) or Delete(d) : ")
4
5  keys = [i for i in range(1, 1000001)]
6
7  random.shuffle(keys)
8
9  if select == "i":
10     with open("insertTest.csv", "w") as f:
11         for key in keys:
12             f.write(str(key) + "," + str(key) + "\n")
13
14     elif select == "d":
15         with open("deleteTest.csv", "w") as f:
16             for i in range(1, 300001):
17                 f.write(str(keys[i]) + "\n")
18
19     else:
20         print("Input case is only (i) or (d).")
```

다음과 같은 코드를 작성하여 테스트할 csv 파일을 만들었다.

Insert 파일은 1 에서 1000000 까지의 숫자를 무작위로 넣어주었다.

Delete 파일은 1 에서 1000000 까지의 숫자 중 300000 개를 무작위로 넣어주었다.

Insert 시 Key 와 Value 는 동일하게 넣어주었다.

```
C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment>java -jar B-tree_Assignment.jar -c index.dat 20
C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment>java -jar B-tree_Assignment.jar -i index.dat insertTest.csv
C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment>java -jar B-tree_Assignment.jar -d index.dat deleteTest.csv
C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment>
```

다음과 같이 명령어들을 입력하였다. (Create -> Insert -> Delete)

Degree 는 20 으로 설정했다.



```
index.dat - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

Order: 20
LeafNode: 1,2,3,4,5,6,7,8,9,10,11,12 1,2,3,4,5,6,7,8,9,10,11,12
LeafNode: 13,14,15,16,17,18,19,20,21,22,23,24,25,26 13,14,15,16,17,18,19,20,21,22,23,24,25,26
LeafNode: 27,28,29,30,31,32,33,34,35,36,37 27,28,29,30,31,32,33,34,35,36,37
LeafNode: 38,39,40,41,42,43,44,45,46,47 38,39,40,41,42,43,44,45,46,47
LeafNode: 48,49,50,51,52,53,54,55,56,57,58 48,49,50,51,52,53,54,55,56,57,58
LeafNode: 59,60,61,62,63,64,65,66,67,68,69,70 59,60,61,62,63,64,65,66,67,68,69,70
LeafNode: 71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89 71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89
LeafNode: 90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105 90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105
LeafNode: 106,107,108,109,110,111,112,113,114,115,116,117,118 106,107,108,109,110,111,112,113,114,115,116,117,118
LeafNode: 119,120,121,122,123,124,125,126,127,128,129,130 119,120,121,122,123,124,125,126,127,128,129,130
LeafNode: 131,132,133,134,135,136,137,138,139,140,141,142,143 131,132,133,134,135,136,137,138,139,140,141,142,143
LeafNode: 144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160 144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160
NonLeafNode: 13,27,38,48,59,71,90,106,119,131,144
LeafNode: 161,162,163,164,165,166,167,168,169,170,171,172 161,162,163,164,165,166,167,168,169,170,171,172
LeafNode: 173,174,175,176,177,178,179,180,181,182,183 173,174,175,176,177,178,179,180,181,182,183
LeafNode: 184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200 184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200

Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

Insert 수행 시 index.dat

백만개의 데이터에 대하여 삽입을 수행했을 때 저사양 노트북 기준으로 5 초 정도가 소요되었다.

```
index.dat - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

Order: 20
LeafNode: 1,3,4,5,7,8,9,10,11,12 1,3,4,5,7,8,9,10,11,12
LeafNode: 13,14,15,18,19,20,21,22,25,28,31,32,34,35 13,14,15,18,19,20,21,22,25,28,31,32,34,35
LeafNode: 36,37,38,39,41,44,46,47,48,49,51,54,55,56,57 36,37,38,39,41,44,46,47,48,49,51,54,55,56,57
LeafNode: 61,63,64,65,66,67,68,69,70 61,63,64,65,66,67,68,69,70
LeafNode: 71,75,77,78,80,83,84,85,86,87,88,89 71,75,77,78,80,83,84,85,86,87,88,89
LeafNode: 93,94,95,98,99,101,104,105,107,109,111,113,114,115,117 93,94,95,98,99,101,104,105,107,109,111,113,114,115,117
LeafNode: 119,121,122,123,124,125,127,128,129,132,133,137,139,141,143 119,121,122,123,124,125,127,128,129,132,133,137,139,141,143
LeafNode: 144,145,146,147,148,149,150,151,152,155,156,157,158,159 144,145,146,147,148,149,150,151,152,155,156,157,158,159
LeafNode: 161,163,164,165,166,168,170,171,174,175,176,177,180,181,183 161,163,164,165,166,168,170,171,174,175,176,177,180,181,183
LeafNode: 186,187,189,192,193,196,197,198,199,200 186,187,189,192,193,196,197,198,199,200
LeafNode: 201,202,203,204,205,207,208,209,212,213,214,215,218 201,202,203,204,205,207,208,209,212,213,214,215,218
LeafNode: 219,220,221,222,223,224,225,227,229,231,232,233,234,236,237 219,220,221,222,223,224,225,227,229,231,232,233,234,236,237
LeafNode: 239,240,241,242,243,244,245,246,247,248 239,240,241,242,243,244,245,246,247,248
LeafNode: 249,250,251,253,254,255,256,260,261,263,264,270,271 249,250,251,253,254,255,256,260,261,263,264,270,271
LeafNode: 273,274,275,276,278,279,280,281,282 273,274,275,276,278,279,280,281,282
LeafNode: 284,285,286,287,288,289,290,291,292,293,295,296 284,285,286,287,288,289,290,291,292,293,295,296

Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

Delete 수행 시 index.dat

30 만개의 데이터를 삭제하여 몇몇 Key 들이 사라진 것을 확인할 수 있다.

```
명령 프롬프트
C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment>java -jar B-tree_Assignment.jar -s index.dat 642324
507428
541718, 576904, 611324, 639829, 676669, 713251, 747841, 777689, 809415, 837356, 888575, 914974, 947171, 972032
642080, 644047, 647603, 651661, 655202, 659269, 662641, 665488, 667931, 670510, 672976
642272, 642474, 642734, 642956, 643129, 643321, 643540, 643736, 643905
642286, 642303, 642317, 642340, 642358, 642380, 642398, 642420, 642441, 642456
642324

C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment>java -jar B-tree_Assignment.jar -r index.dat 875320 875350
875320, 875320
875321, 875321
875322, 875322
875323, 875323
875324, 875324
875325, 875325
875327, 875327
875329, 875329
875330, 875330
875331, 875331
875332, 875332
875333, 875333
875335, 875335
875336, 875336
875337, 875337
875338, 875338
875340, 875340
875341, 875341
875344, 875344
875345, 875345
875346, 875346
875347, 875347
875348, 875348
875349, 875349
875350, 875350

C:\Users\dyddn\2020_ite2038_2019054957\B-tree_Assignment>
```

임의의 숫자를 입력하여 Single Key Search 와 Ranged Search 를 수행하였다.

백만개라는 많은 데이터를 다룬 것이 처음인데, 빠른 시간내에 Value 들을 찾을 수 있었다.

Search 가 굉장히 빠르다는 점이 B+ Tree 의 강점임을 다시 한번 깨닫게 되었다.