

Database System Work #1

B+ tree implementation

2015004857 이영수

1) Summary of algorithm

전체적 구조는 일반적인 B+ 트리를 따른다. 노드에서 key 와 자식 포인터의 기준은 key 보다 작은 값은(미만) 원쪽, key 보다 같거나 큰(이상) 값은 오른쪽에 위치하도록 정의하였다.

Leaf node에서 key 와 value 는 pair로 함께 저장되며, Non-leaf node에서는 key 와 child pointer 가 한 쌍의 pair로 저장된다. 이 때 자식 수는 키보다 한 개 더 많을 수 있는데, 이를 위해 "r"이라는 특수한 포인터가 사용된다. "r"은 rightmost child 를 의미하는데, 오로지 트리의 가장 오른쪽 노드만 "r"을 가지며, 이외의 노드들은 "r"값을 항상 null로 고정시킨다.

Insertion 이후 노드의 자식 수가 b 이상이 될 경우 노드를 쪼개 주어야 하는데, 이 때 새로운 노드를 기존 노드의 원쪽에 위치시키는 방법으로 포인터의 꼬임을 방지한다. 이 방법을 통해 새로운 노드는 항상 최 우측 노드가 아니게 되며, 고로 "r"에 대한 수정을 할 필요가 없어진다.

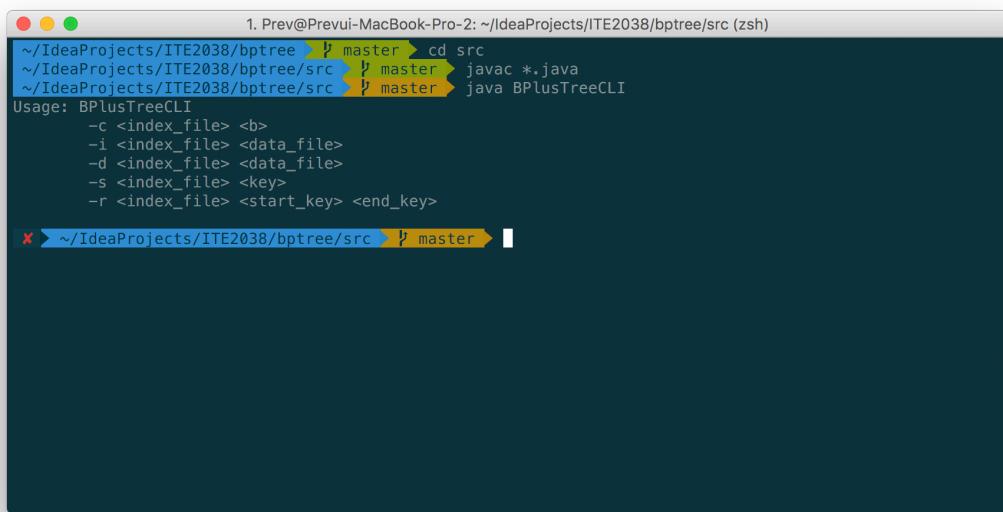
Deletion에서 병합이 발생할 때도 비슷한 원칙을 사용하는데, 원쪽 노드를 오른쪽 노드에 합치는 방식으로 작업을 진행한다. 최 우측 노드는 삭제되지 않으므로 마찬가지로 "r"에 대한 수정 작업은 이루어지지 않는다.

"r"에 새로운 값이 생성되는 경우는 오직 한 경우 밖에 없다. Insertion 이후 전파를 통해 루트 노드가 쪼개질 때, 기존 노드는 새 루트 노드의 가장 원쪽 pair에 저장되며, 새로 만들어진 오른쪽 노드는 루트 노드의 "r"로 연결되게 된다.

Insertion에 의한 split이나, deletion에 의한 redistribute 이후에, 바뀐 노드에 대응되는 키 값은 키 바로 오른쪽 노드의 left-most-key로 갱신하며 key에 대한 갱신을 진행한다.

2) Instructions for compiling source codes

```
cd src  
javac *.java  
java BPlusTreeCLI
```



The terminal window shows the following sequence of commands:

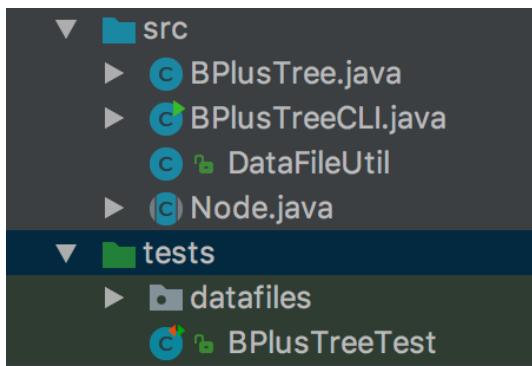
1. Prev@Prevui-MacBook-Pro-2: ~/IdeaProjects/ITE2038/bptree/src (zsh)
- ~/IdeaProjects/ITE2038/bptree > **p** master > cd src
- ~/IdeaProjects/ITE2038/bptree/src > **p** master > javac *.java
- ~/IdeaProjects/ITE2038/bptree/src > **p** master > java BPlusTreeCLI

Usage: BPlusTreeCLI

- c <index_file>
- i <index_file> <data_file>
- d <index_file> <data_file>
- s <index_file> <key>
- r <index_file> <start_key> <end_key>

x ~ /IdeaProjects/ITE2038/bptree/src > **p** master > █

3) Detailed description of codes



프로젝트 코드는 크게 3 가지로 나눌 수 있다. B+ tree 내부가 구현된 구현 부 ("BPlusTree.java" 및 "Node.java"), 구현된 트리를 바탕으로 커맨드라인을 통해 명령 해석 부 ("BPlusTreeCLI.java"), 그리고 위 코드들을 테스트하는 테스트 부 ("BPlusTreeTest.java") 가 있다.

3-1) 구현 부

BPlusTree.java

- BPlusTree Class
- SearchResult Class

Node.java

- LeafNode Class
- Node Abstract Class
- LeafNode Class
- NonLeafNode Class

Node.java

Node Abstract Class

Non-leaf 노드와 Leaf 노드의 공통적인 특징을 합친 abstract class 이다. 부모 노드 및 이웃, 형제 노드를 관리, 반환하는 함수를 가지고 있으며, key 와 keyLength 를 반환하는 abstract function 을 포함하고 있다.

```
12  o\ abstract class Node implements Serializable {
13      static final long serialVersionUID = 1L;
14
15      private NonLeafNode parent;
16
17      /**
18      * Constructor
19      * @param parent: parent node (if root, use null)
20      */
21      Node(NonLeafNode parent) { this.parent = parent; }
22
23      /**
24      * Get parent node
25      * @return
26      */
27      public NonLeafNode getParent() { return this.parent; }
28
29      /**
30      * Set parent node
31      * @param node
32      */
33      public void setParent(NonLeafNode node) { this.parent = node; }
34
35      /**
36      * Get siblings of current node (include itself)
37      * @return ArrayList<Node>
38      */
39      public ArrayList<Node> getsiblings() {
40          if (this.parent == null)
41              return new ArrayList<>();
42          else
43              return this.parent.getChildren();
44
45      /**
46      * Get neighbors
47      * @return Pair<Node, Node>
48      *         pair.left is left-neighbor
49      *         pair.right is right-neighbor
50      */
51      public Pair<Node, Node> getNeighbors() {
52          Pair<Node, Node> ret = new Pair<>( left: null, right: null );
53          ArrayList<Node> siblings = this.getsiblings();
54
55          int index = siblings.indexOf(this);
56
57          if (index > 0)
58              ret.left = siblings.get(index - 1);
59          if (index < siblings.size() - 1)
60              ret.right = siblings.get(index + 1);
61
62          return ret;
63      }
64
65      /**
66      * Get keys of node
67      * @return
68      */
69      abstract public int[] getKeys();
70
71      /**
72      * Get length of keys
73      * @return
74      */
75      abstract public int getKeyCounts();
76
77
78
79
80
81
82  o\}
83 }
```

Non-leaf node class

Leaf 노드가 아닌 나머지 노드가 이에 해당된다. 속성 p 는 (Key, 자식노드) 쌍의 배열을 갖고 있으며, 속성 r 은 rightmost child 를 가르킨다.

Insert 와 getChildren 함수는 p 에 대한 삽입 및 조회를 도와주는 wrapper 함수이다.

```
86     class NonLeafNode extends Node {
87         ArrayList<Pair<Integer, Node>> p;
88         Node r;
89
90         NonLeafNode(NonLeafNode parent) {
91             super(parent);
92             this.r = null;
93             this.p = new ArrayList<>();
94         }
95
96         @Override
97         public int[] getKeys() {
98             int[] ret = new int[this.p.size()];
99             for (int i = 0; i < ret.length; i++) {
100                 Pair<Integer, Node> pair = this.p.get(i);
101                 ret[i] = pair.left;
102             }
103             return ret;
104         }
105
106         @Override
107         public int getKeyCounts() { return p.size(); }
108
109         public ArrayList<Node> getChildren() {
110             ArrayList<Node> ret = new ArrayList<>();
111
112             for (Pair<Integer, Node> pair: this.p)
113                 ret.add(pair.right);
114
115             if (this.r != null)
116                 ret.add(this.r);
117
118             return ret;
119         }
120
121         public int getChildrenCounts() {
122             int ret = this.getKeyCounts();
123             if (this.r != null)
124                 ++ret;
125             return ret;
126         }
127
128         public void insert(Pair<Integer, Node> pair) {
129             this.p.add(pair);
130             this.p.sort(Comparator.comparingInt(o -> o.left));
131         }
132
133         public void insert(int key, Node node) { this.insert(new Pair<>(key, node)); }
134     }
```

Leaf node class

Non-Leaf 노드가 아닌 나머지 노드가 이에 해당된다. 속성 p 는 (Key, Value) 쌍의 배열을 갖고 있으며, 속성 r 은 바로 오른쪽의 Leaf node 를, 속성 l 은 바로 왼쪽의 Leaf node 를 가르킨다.

```
141  class LeafNode extends Node {
142      ArrayList<Pair<Integer, Integer>> p;
143
144      LeafNode l;
145      LeafNode r;
146
147      LeafNode(NonLeafNode parent) {
148          super(parent);
149          this.r = null;
150          this.l = null;
151          this.p = new ArrayList<>();
152      }
153
154      @Override
155      public int[] getKeys() {
156          int[] ret = new int[this.p.size()];
157          for (int i = 0; i < ret.length; i++) {
158              Pair<Integer, Integer> pair = this.p.get(i);
159              ret[i] = pair.left;
160          }
161          return ret;
162      }
163
164      @Override
165      public int getKeyCounts() { return p.size(); }
166
167      public void insert(Pair<Integer, Integer> pair) {
168          this.p.add(pair);
169          this.p.sort(Comparator.comparingInt(o -> o.left));
170      }
171
172      public void insert(int key, int value) { this.insert(new Pair<>(key, value)); }
173
174  }
```

Pair class

일반적인 Pair ADT 를 별도로 구현하였다.

```
180  class Pair<L,R> implements Serializable {
181      private static final long serialVersionUID = 1L;
182
183      public L left;
184      public R right;
185
186      public Pair(L left, R right) {
187          this.left = left;
188          this.right = right;
189      }
190
191      @Override
192      public int hashCode() { return left.hashCode() ^ right.hashCode(); }
193
194      @Override
195      public boolean equals(Object o) {
196          if (!(o instanceof Pair)) return false;
197          Pair pairo = (Pair) o;
198          return this.left.equals(pairo.left) &&
199                  this.right.equals(pairo.right);
200      }
201
202  }
```

BPlusTree.java

최상위 노드인 rootNode 와 "b"를 의미하는 maxChildNodes 속성이 존재한다.
maxChildNodes는 getter로만 사용할 수 있다.

```
1 import java.io.Serializable;
2 import java.util.*;
3
4 /**
5  * B+ tree of integer keys and integer values
6  *
7  * @author Prev (0soo.2@prev.kr)
8 */
9
10 public class BPlusTree implements Serializable {
11     private static final long serialVersionUID = 1L;
12
13     public Node rootNode;
14     private int maxChildNodes;
15
16     BPlusTree(int maxChildNodes) {
17         this.maxChildNodes = maxChildNodes;
18         this.rootNode = new LeafNode( parent: null );
19     }
20
21     /**
22      * "b" of B+ Tree
23      * @return int
24      */
25     public int getMaxChildNodes() { return maxChildNodes; }
26
27 }
```

삽입 부분

새로운 값 삽입은 insert 함수로 실행하며 내부적으로 _insertValue, _splitNode, _insertNode 등의 함수가 추가로 사용된다.

```
30 /**
31  * Insert (key, value) pair to tree
32  * @param key
33  * @param value
34  * @return true if insertion is success
35  *         false if ignored because key is duplicated
36 */
37 public Boolean insert(int key, int value) {
38     LeafNode node = this.search(key).leafNode;
39
40     if (Arrays.binarySearch(node.getKeys(), key) >= 0)
41         // Ignore if key exists
42         return false;
43
44     _insertValue(key, value, node);
45     return true;
46 }
47
48 private void _insertValue(int key, int value, LeafNode node) {
49     node.insert(key, value);
50
51     if (node.getKeyCounts() >= this.maxChildNodes)
52         _splitNode(node);
53 }
```

자식 수가 maxChildNodes 이상이면 노드를 쪼개야한다. 노드는 leaf node 일 때와 Non-leaf node 일 때를 구분하여 진행되는데, 공통적으로 왼쪽에 새로운 노드를 삽입한다는 원칙을 사용한다. Leaf node 는 l 과 r 을 수정해주어 iteration 에 문제가 없도록 해주며, Non-leaf node 일 때는 자식의 parent 를 갱신해준다. 작업이 끝난 후 부모 노드의 자식 수가 maxChildNodes 이상이면 부모노드도 분할을 진행해준다.

```

55     private void _splitNode(Node node) {
56         Node newNode = null;
57
58         if (node == this.rootNode) {
59             this.rootNode = new NonLeafNode( parent: null );
60             node.setParent((NonLeafNode) this.rootNode);
61
62             ((NonLeafNode) this.rootNode).r = node;
63         }
64
65         if (node instanceof LeafNode) {
66             newNode = new LeafNode(node.getParent());
67             ((LeafNode) newNode).p = new ArrayList<>(
68                 ((LeafNode) node).p.subList(0, (node.getKeyCounts() + 1) / 2)
69             );
70             ((LeafNode) node).p = new ArrayList<>(
71                 ((LeafNode) node).p.subList((node.getKeyCounts() + 1) / 2, node.getKeyCounts())
72             );
73
74             // Change directions
75             if (((LeafNode) node).l != null) ((LeafNode) node).l.r = ((LeafNode) newNode);
76             ((LeafNode) newNode).l = ((LeafNode) node).l;
77             ((LeafNode) newNode).r = ((LeafNode) node);
78             ((LeafNode) node).l = ((LeafNode) newNode);
79
80
81         }else if (node instanceof NonLeafNode) {
82             newNode = new NonLeafNode(node.getParent());
83             ((NonLeafNode) newNode).p = new ArrayList<>(
84                 ((NonLeafNode) node).p.subList(0, (node.getKeyCounts() + 1) / 2)
85             );
86             ((NonLeafNode) node).p = new ArrayList<>(
87                 ((NonLeafNode) node).p.subList((node.getKeyCounts() + 1) / 2, node.getKeyCounts())
88             );
89
90             for (Node child: ((NonLeafNode) newNode).getChildren())
91                 child.setParent((NonLeafNode) newNode);
92         }
93
94         _insertNode(newNode, node, node.getParent());
95
96         if (node.getParent().getKeyCounts() >= this.maxChildNodes) {
97             _splitNode(node.getParent());
98         }
99     }
100 
```

노드를 새로 추가할 때에는 새로운 키 값으로 “오른쪽 노드의 가장 왼쪽 값”을 사용한다.

```

102     private void _insertNode(Node leftNode, Node rightNode, NonLeafNode parentNode) {
103         // Get left-most Node from root
104         Node leftMostOfRightNode = rightNode;
105         while (leftMostOfRightNode instanceof NonLeafNode)
106             leftMostOfRightNode = ((NonLeafNode) leftMostOfRightNode).p.get(0).right;
107
108         int key = leftMostOfRightNode.getKeys()[0];
109         parentNode.insert(key, leftNode);
110     }

```

제거 부분

값 제거는 remove 함수로 실행하며 내부적으로 _removeValue, _removeValue, _balancingNode, _mergeNode 등의 함수가 추가로 사용된다.

```
113  /**
114  * Remove data by key
115  * @param key
116  * @return true if deletion is succeed
117  *         false if key not found
118 */
119 public Boolean remove(int key) {
120     LeafNode node = this.search(key).leafNode;
121     int index = Arrays.binarySearch(node.getKeys(), key);
122
123     if (index < 0)
124         // Ignore if key do not exists
125         return false;
126
127     _removeValue(key, node);
128
129     return true;
130 }
```

먼저 search 를 통해 leaf 노드를 찾고, leaf 에서 value 를 지운다. 이 때 해당 leaf node 가 B+ tree 의 제약조건을 어기게 되면(키가 b/2 보다 작아지면) 노드 밸런싱을 해주어야 한다.

```
132 /**
133  * private void _removeValue(int key, LeafNode node) {
134  *     int index = Arrays.binarySearch(node.getKeys(), key);
135  *     node.p.remove(index);
136
137     if (node.getKeyCounts() < (this.maxChildNodes-1) / 2 && node != this.rootNode)
138         _balancingNode(node);
139 }
```

노드 밸런싱은 크게 2 가지 방법으로 진행된다. 해당 노드와 옆 노드의 자식 수가 b 미만이면 둘을 합친다(Merge). 그렇지 않으면 재분배(Redistribute) 한다. 삭제 작업도 삽입과 비슷하게 오른쪽 노드를 우선하며, 왼쪽 노드를 지우는 방향으로 진행된다.

```
139 /**
140  * private void _balancingNode(Node node) {
141  *     Pair<Node, Node> neighbors = node.getNeighbors();
142
143     if (neighbors.right != null && neighbors.right.getKeyCounts() + node.getKeyCounts() < this.maxChildNodes )
144         // case 1-1: merge (with right)
145         _mergeNode(node, neighbors.right, node.getParent());
146
147     else if (neighbors.left != null && neighbors.left.getKeyCounts() + node.getKeyCounts() < this.maxChildNodes )
148         // case 1-2: merge (with left)
149         _mergeNode(neighbors.left, node, node.getParent());
150
151     else {
152         // case 2-1: Redistribute
153         if (neighbors.right != null) {
154             if (node instanceof LeafNode)
155                 ((LeafNode) node).insert(
156                     ((LeafNode) neighbors.right).p.remove( index: 0 )
157                 );
158             else
159                 ((NonLeafNode) node).insert(
160                     ((NonLeafNode) neighbors.right).p.remove( index: 0 )
161                 );
162
163             _updateKey(node.getParent(), node, neighbors.right);
164
165         }else if (neighbors.left != null) {
166             if (node instanceof LeafNode)
167                 ((LeafNode) node).insert(
168                     ((LeafNode) neighbors.left).p.remove( index: neighbors.left.getKeyCounts()-1 )
169                 );
170             else
171                 ((NonLeafNode) node).insert(
172                     ((NonLeafNode) neighbors.left).p.remove( index: neighbors.left.getKeyCounts()-1 )
173                 );
174
175         }
176     }
177 }
```

병합 작업은 split 작업과 비슷하게 진행되는데, split에서 새로운 노드를 왼쪽에 삽입했던 것과 반대로 삭제 시엔 왼쪽 노드부터 지우게 된다. 삭제된 노드가 만약 leaf node라면 l과 r 설정을 통해 iteration을 보장하게 한다. 병합 이후 부모 노드가 또 제약조건에 위배된다면 재귀 호출을 통해 부모에서도 re-balancing을 진행한다.

```
179     private void _mergeNode(Node leftNode, Node rightNode, NonLeafNode parentNode) {
180         if (leftNode instanceof LeafNode)
181             for (Pair<Integer, Integer> pair: ((LeafNode) leftNode).p)
182                 ((LeafNode) rightNode).insert(pair);
183
184         else if (leftNode instanceof NonLeafNode)
185             for (Pair<Integer, Node> pair: ((NonLeafNode) leftNode).p)
186                 ((NonLeafNode) rightNode).insert(pair);
187
188         for (int i = 0; i < parentNode.p.size(); i++) {
189             if (parentNode.p.get(i).right == leftNode) {
190                 parentNode.p.remove(i);
191
192                 if (leftNode instanceof LeafNode) {
193                     LeafNode ll = ((LeafNode) leftNode).l;
194                     if (ll != null)
195                         ll.r = (LeafNode) rightNode;
196                     ((LeafNode) rightNode).l = ll;
197                 }
198                 break;
199             }
200         }
201
202         if (parentNode == this.rootNode && parentNode.getChildrenCounts() == 1) {
203             rightNode.setParent(null);
204             this.rootNode = rightNode;
205             return;
206         }
207
208         if (parentNode.getKeyCounts() < this.maxChildNodes / 2)
209             _balancingNode(parentNode);
210     }
211
212     private void _updateKey(NonLeafNode parentNode, Node leftNode, Node rightNode) {
213         // Get left-most Node from root
214         Node leftMostOfRightNode = rightNode;
215         while (leftMostOfRightNode instanceof NonLeafNode)
216             leftMostOfRightNode = ((NonLeafNode) leftMostOfRightNode).p.get(0).right;
217
218         int key = leftMostOfRightNode.getKeys()[0];
219
220         for (Pair<Integer, Node> pair: parentNode.p)
221             if (pair.right == leftNode) {
222                 pair.left = key;
223                 break;
224             }
225
226     }
227 }
```

검색 부분

검색 결과는 value 이외에 다양한 데이터가 더 필요함으로 별도의 SearchResult 클래스를 사용한다. 검색은 노드의 key 들을 iteration 하며 검색을 진행하고 재귀 호출을 통해 leaf node 를 발견할 때까지 반복된다. Leaf node 를 찾으면 해당 노드의 pair 배열을 순회하여 최종 value 를 찾게 된다.

```
255     /**
256      * Search for value by key
257      * @param key
258      * @return SearchResult Object{
259      *     Boolean hit
260      *     int value (result)
261      *     LeafNode leafNode;
262      *     ArrayList<Node> history;
263      * }
264      */
265     public SearchResult search(int key) {
266         ArrayList<Node> history = new ArrayList<>();
267         LeafNode leafNode = _searchProc(key, this.rootNode, history);
268
269         if (leafNode == null)
270             return SearchResult.miss(leafNode, history);
271
272         for (Pair<Integer, Integer> pair: leafNode.p) {
273             if (pair.left == key)
274                 return SearchResult.hit(pair.right, leafNode, history);
275         }
276
277         return SearchResult.miss(leafNode, history);
278     }
279
280     @
281     private LeafNode _searchProc(int key, Node node, ArrayList<Node> history) {
282         if (history != null)
283             history.add(node);
284
285         if (node == null)
286             return null;
287
288         if (node instanceof LeafNode)
289             return (LeafNode) node;
290
291         else {
292             for (Pair<Integer, Node> p: ((NonLeafNode) node).p) {
293                 if (key < p.left)
294                     return _searchProc(key, p.right, history);
295             }
296         }
297     }
298 }
```

범위 검색 부분

범위 검색은 검색과 비슷하게 진행되는데, startKey 와 endKey 의 범위 내에 있는 모든 자식 노드를 재귀 탐색하며 검색 결과를 찾게 된다. 검색 결과는 여러 개가 될 수 있기에 ArrayList 를 반환한다.

```
300      /**
301       * Search from B+Tree by range
302       * @param startKey
303       * @param endKey
304       * @return ArrayList<Integer>
305      */
306     public ArrayList<Pair<Integer, Integer>> rangedSearch(int startKey, int endKey) {
307         ArrayList<Pair<Integer, Integer>> ret = new ArrayList<>();
308
309         for (LeafNode node: _rangedSearchProc(startKey, endKey, this.rootNode)) {
310             for (Pair<Integer, Integer> pair: node.p) {
311                 if (pair.left >= startKey && pair.left <= endKey)
312                     ret.add(pair);
313             }
314         }
315
316         return ret;
317     }
318
319     private ArrayList<LeafNode> _rangedSearchProc(int startKey, int endKey, Node node) {
320         ArrayList<LeafNode> ret = new ArrayList<>();
321
322         if (node == null)
323             return ret;
324
325         if (node instanceof LeafNode)
326             ret.add((LeafNode) node);
327
328         else {
329             for (Pair<Integer, Node> p: ((NonLeafNode) node).p)
330                 if (startKey < p.left)
331                     ret.addAll(_rangedSearchProc(startKey, endKey, p.right));
332
333                 if (node.getKeys()[node.getKeyCounts()-1] <= endKey)
334                     ret.addAll(_rangedSearchProc(startKey, endKey, ((NonLeafNode) node).r));
335             }
336
337         }
338
339     }
```

검색 결과 SearchResult 클래스

검색 결과는 다양한 용도로 쓰이므로 관리가 용이하도록 별도의 클래스를 제작하였다. 성공 여부인 hit, 성공 시의 값인 value, leaf node 및 검색 history 가 포함되어 있다.

```
350  class SearchResult {
351      Boolean hit;
352      int value;
353      LeafNode leafNode;
354      ArrayList<Node> history;
355
356      /**
357      * SearchResult Constructor
358      * @param hit: True if value is found
359      *           False if not found
360      * @param value: Value of search result
361      *           If not found, -1
362      * @param leafNode: LeafNode including value
363      * @param history: Hierarchical history of searching
364      */
365      SearchResult(Boolean hit, int value, LeafNode leafNode, ArrayList<Node> history) {
366          this.hit = hit;
367          this.value = value;
368          this.leafNode = leafNode;
369          this.history = history;
370      }
371
372      @
373      static SearchResult hit(int value, LeafNode leafNode, ArrayList<Node> history) {
374          return new SearchResult(hit: true, value, leafNode, history);
375      }
376      @
377      static SearchResult miss(LeafNode leafNode, ArrayList<Node> history) {
378          return new SearchResult(hit: false, value: -1, leafNode, history);
379      }
}
```

3-2) 명령 해석 부

Program arguments를 해석해서 옵션별로 다른 동작을 하도록 명령을 해석하고 인터페이스를 제공하는 클래스가 별도로 존재한다.

```
1  /***
2   * B+ tree CLI
3   *
4   * Usage: BPlusTreeCLI
5   *   -c <index_file> <b>
6   *   -i <index_file> <data_file>
7   *   -d <index_file> <data_file>
8   *   -s <index_file> <key>
9   *   -r <index_file> <start_key> <end_key>
10  *
11  * @author Prev (0soo.2@prev.kr)
12  */
13
14  public class BPlusTreeCLI {
15
16      static final String PROGRAM_NAME = "BPlusTreeCLI";
17
18  public static void main(String[] args) {
19      if (args.length < 1) {
20          System.out.println(CLUtil.getHelpMessage());
21          System.exit( status: -1);
22      }
23      switch (args[0]) {
24          case "-c" :
25              CLIUtil.guaranteeArgs(args, number: 3);
26              create(
27                  args[1],
28                  Integer.parseInt(args[2])
29              );
30              break;
31
32          case "-i" :
33              CLIUtil.guaranteeArgs(args, number: 3);
34              insert(args[1], args[2]);
35              break;
36
37          case "-d" :
38              CLIUtil.guaranteeArgs(args, number: 3);
39              delete(args[1], args[2]);
40              break;
41
42          case "-s" :
43              CLIUtil.guaranteeArgs(args, number: 3);
44              search(
45                  args[1],
46                  Integer.parseInt(args[2])
47              );
48              break;
49
50          case "-r" :
51              CLIUtil.guaranteeArgs(args, number: 4);
52              rangedSearch(
53                  args[1],
54                  Integer.parseInt(args[2]),
55                  Integer.parseInt(args[3])
56              );
57              break;
58
59          default:
60              System.out.println(CLUtil.getHelpMessage());
61              System.exit( status: -1);
62      }
63  }
```

create 및 insert 함수

```
66  /**
67  * Create B+ tree
68  * @param indexFileName
69  * @param b
70  */
71  static void create(String indexFileName, int b) {
72      BPlusTree tree = new BPlusTree(b);
73      DataFileUtil.saveTree(indexFileName, tree);
74  }
75
76
77  /**
78  * Insert Dataset to B+ tree
79  * @param indexFileName
80  * @param dataFileName
81  */
82  static void insert(String indexFileName, String dataFileName) {
83      BPlusTree tree = DataFileUtil.loadTree(indexFileName);
84      int[][] data = DataFileUtil.loadIntCSV(dataFileName);
85      int insertedRows = 0;
86
87      for(int[] row: data) {
88          if (tree.insert(row[0], row[1]))
89              insertedRows++;
90      }
91
92      System.out.printf("%d lines are inserted\n", insertedRows);
93      DataFileUtil.saveTree(indexFileName, tree);
94  }
```

delete 함수

```
96  /**
97  * Delete Dataset from B+ tree
98  * @param indexFileName
99  * @param dataFileName
100 */
101 static void delete(String indexFileName, String dataFileName) {
102     BPlusTree tree = DataFileUtil.loadTree(indexFileName);
103     int[][] data = DataFileUtil.loadIntCSV(dataFileName);
104
105     int deletedRows = 0;
106
107     for(int[] row: data) {
108         if (tree.remove(row[0]))
109             deletedRows++;
110     }
111
112     System.out.printf("%d keys are deleted\n", deletedRows);
113     DataFileUtil.saveTree(indexFileName, tree);
114 }
```

search 함수

```
116 /**
117 * Search from B+ tree
118 * @param indexFileName
119 * @param key
120 */
121 static void search(String indexFileName, int key) {
122     BPlusTree tree = DataFileUtil.loadTree(indexFileName);
123
124     for (Node node: tree.search(key).history) {
125         if (node == null) {
126             System.out.println("NOT FOUND");
127             return;
128         }
129
130         else if (node instanceof LeafNode) {
131             for (Pair<Integer, Integer> p: ((LeafNode) node).p) {
132                 if (p.left == key) {
133                     System.out.println(p.right);
134                     return;
135                 }
136             }
137             System.out.println("NOT FOUND");
138
139         } else {
140             StringBuilder msg = new StringBuilder();
141             int[] keys = node.getKeys();
142             for (int i = 0; i < keys.length; i++) {
143                 msg.append(keys[i]);
144                 if (i != keys.length - 1)
145                     msg.append(',');
146             }
147             System.out.println(msg.toString());
148         }
149     }
150 }
```

ranged-search 함수

```
153     /**
154      * Search by range in B+ tree
155      * @param IndexFileName
156      * @param startKey
157      * @param endKey
158      */
159     static void rangedSearch(String indexFileName, int startKey, int endKey) {
160         BPlusTree tree = DataFileUtil.loadTree(indexFileName);
161
162         for (Pair<Integer, Integer> pair: tree.rangedSearch(startKey, endKey)) {
163             System.out.printf("%d,%d\n", pair.left, pair.right);
164         }
165     }
166
167 }
```

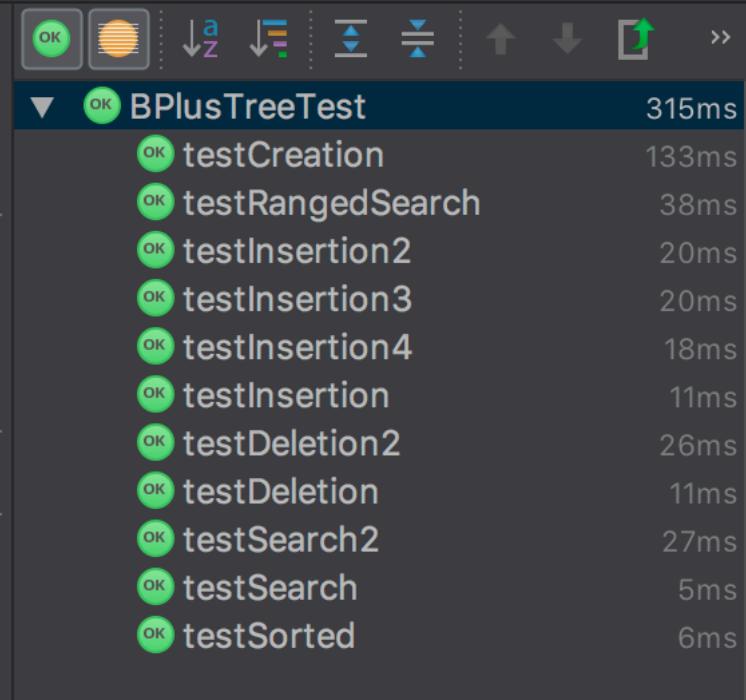
CLI를 위한 유ти리티 클래스 CLIUtil도 이 파일에 있으며, help message를 출력하거나, 인자의 개수를 보장하는 메소드가 있다.

```
169 class CLIUtil {
170
171     @
172     static String getHelpMessage() {
173         return "Usage: " + BPlusTreeCLI.PROGRAM_NAME + "\n\t" +
174             "-c <index_file> <b>\n\t" +
175             "-i <index_file> <data_file>\n\t" +
176             "-d <index_file> <data_file>\n\t" +
177             "-s <index_file> <key>\n\t" +
178             "-r <index_file> <start_key> <end_key>\n";
179
180     static void guaranteeArgs(String[] args, int number) {
181         if (args.length != number) {
182             System.out.println(getHelpMessage());
183             System.exit(status: -1);
184         }
185     }
186 }
```

3-3) 테스트 부

tests/BPlusTreeTest.java

JUnit을 사용해서 각 기능 별로 테스트 코드를 작성하였다. Input 셋과 output에 대한 expected 값을 지정하고 실제로 옳은지 체크하는 정도의 함수들로 구성되어 있다.



		315ms
▼	OK BPlusTreeTest	
	OK testCreation	133ms
	OK testRangedSearch	38ms
	OK testInsertion2	20ms
	OK testInsertion3	20ms
	OK testInsertion4	18ms
	OK testInsertion	11ms
	OK testDeletion2	26ms
	OK testDeletion	11ms
	OK testSearch2	27ms
	OK testSearch	5ms
	OK testSorted	6ms

```
9  /**
10   * B+ tree Tests
11   *
12   * @author Prev (0soo.2@prev.kr)
13  */
14
15  public class BPlusTreeTest {
16
17      static final String TEST_INDEX_FILE = "tests/datafiles/_index.dat";
18      static final String TEST_INSERT_FILE = "tests/datafiles/insert.csv";
19      static final String TEST_INSERT_FILE2 = "tests/datafiles/insert2.csv";
20      static final String TEST_INSERT_FILE0 = "tests/datafiles/insert0.csv";
21      static final String TEST_DELETE_FILE = "tests/datafiles/delete.csv";
22      static final String TEST_DELETE_FILE0 = "tests/datafiles/delete0.csv";
23
24
```

```

25      /**
26       * Check that tree is satisfying the properties of B+ Tree
27       * @param tree
28       */
29     private void guaranteeBPlusTree(BPlusTree tree) {
30         Queue<Node> que = new LinkedList<>();
31         que.offer(tree.rootNode);
32
33         int b = tree.getMaxChildNodes();
34
35         while (!que.isEmpty()) {
36             Node node = que.poll();
37
38             if (node == tree.rootNode) {
39                 if (node instanceof NonLeafNode)
40                     assertTrue( condition: ((NonLeafNode) node).getChildrenCounts() >= 2);
41
42                 else if (node instanceof LeafNode)
43                     assertTrue( condition: node.getKeyCounts() <= b-1);
44
45             } else if (node instanceof NonLeafNode) {
46                 int childrenCounts = ((NonLeafNode) node).getChildrenCounts();
47
48                 assertTrue( condition: childrenCounts >= b / 2);
49                 assertTrue( condition: childrenCounts <= b);
50
51             } else if (node instanceof LeafNode) {
52                 assertTrue( condition: node.getKeyCounts() >= (b-1) / 2);
53                 assertTrue( condition: node.getKeyCounts() <= b-1);
54             }
55
56
57             if (node instanceof NonLeafNode) {
58                 for (Pair<Integer, Node> p : ((NonLeafNode) node).p)
59                     que.offer(p.right);
60             }
61         }
62     }
63
64
65     @Test
66     public void testCreation() {
67         int maxChildNodes = 10;
68
69         BPlusTreeCLI.create(TEST_INDEX_FILE, maxChildNodes);
70         BPlusTree tree = DataFileUtil.loadTree(TEST_INDEX_FILE);
71
72         assertEquals(tree.getMaxChildNodes(), maxChildNodes);
73     }
74
75
76
77     @Test
78     public void testInsertion() {
79         BPlusTreeCLI.create(TEST_INDEX_FILE, b: 4);
80         BPlusTreeCLI.insert(TEST_INDEX_FILE, TEST_INSERT_FILE);
81
82         BPlusTree tree = DataFileUtil.loadTree(TEST_INDEX_FILE);
83
84         tree.traversal();
85         ArrayList<Pair<Integer, Integer>> actual = tree.getList();
86
87         assertEquals( expected: 1, tree.rootNode.getKeyCounts());
88
89         for (int i = 0; i < 5; i++) {
90             assertEquals( expected: i+1, (long) actual.get(i).left);
91             assertEquals( expected: i+1, (long) actual.get(i).right);
92         }
93
94         guaranteeBPlusTree(tree);
95     }
96

```

```

97     @Test
98     public void testInsertion2() {
99         BPlusTreeCLI.create(TEST_INDEX_FILE, b: 3);
100        BPlusTreeCLI.insert(TEST_INDEX_FILE, TEST_INSERT_FILE);
101
102        BPlusTree tree = DataFileUtil.loadTree(TEST_INDEX_FILE);
103
104        tree.traversal();
105        ArrayList<Pair<Integer, Integer>> actual = tree.getList();
106
107        assertEquals( expected: 2, tree.rootNode.getKeyCounts());
108
109        for (int i = 0; i < 5; i++) {
110            assertEquals( expected: i+1, (long) actual.get(i).left);
111            assertEquals( expected: i+1, (long) actual.get(i).right);
112        }
113
114        guaranteeBPlusTree(tree);
115    }
116
117    @Test
118    public void testInsertion3() {
119        BPlusTreeCLI.create(TEST_INDEX_FILE, b: 3);
120        BPlusTreeCLI.insert(TEST_INDEX_FILE, TEST_INSERT_FILE2);
121
122        BPlusTree tree = DataFileUtil.loadTree(TEST_INDEX_FILE);
123
124        tree.traversal();
125        ArrayList<Pair<Integer, Integer>> actual = tree.getList();
126
127        for (int i = 1; i < actual.size(); i++) {
128            assertTrue( condition: actual.get(i-1).left < actual.get(i).left);
129            assertTrue( condition: actual.get(i).left * 10 == actual.get(i).right);
130        }
131
132        guaranteeBPlusTree(tree);
133    }
134
135    @Test
136    public void testInsertion4() {
137        BPlusTreeCLI.create(TEST_INDEX_FILE, b: 3);
138        BPlusTreeCLI.insert(TEST_INDEX_FILE, TEST_INSERT_FILE0);
139
140        BPlusTree tree = DataFileUtil.loadTree(TEST_INDEX_FILE);
141
142        tree.traversal();
143
144        assertTrue(tree.search( key: 10).hit);
145        assertTrue(tree.search( key: 86).hit);
146        assertTrue(tree.search( key: 20).hit);
147        assertTrue(tree.search( key: 37).hit);
148        assertTrue(tree.search( key: 87).hit);
149
150        guaranteeBPlusTree(tree);
151    }
152
153
154
155    @Test
156    public void testDeletion() {
157        BPlusTreeCLI.create(TEST_INDEX_FILE, b: 3);
158        BPlusTreeCLI.insert(TEST_INDEX_FILE, TEST_INSERT_FILE);
159        BPlusTreeCLI.delete(TEST_INDEX_FILE, TEST_DELETE_FILE);
160
161        BPlusTree tree = DataFileUtil.loadTree(TEST_INDEX_FILE);
162
163        tree.traversal();
164
165        assertFalse(tree.search( key: 1).hit);
166        assertFalse(tree.search( key: 2).hit);
167        assertTrue(tree.search( key: 3).hit);
168        assertTrue(tree.search( key: 4).hit);
169        assertFalse(tree.search( key: 5).hit);
170
171        guaranteeBPlusTree(tree);
172    }

```

```
174 @Test
175 public void testDeletion2() {
176     BPlusTreeCLI.create(TEST_INDEX_FILE, b: 3);
177     BPlusTreeCLI.insert(TEST_INDEX_FILE, TEST_INSERT_FILE0);
178     BPlusTreeCLI.delete(TEST_INDEX_FILE, TEST_DELETE_FILE0);
179
180     BPlusTree tree = DataFileUtil.loadTree(TEST_INDEX_FILE);
181
182     tree.traversal();
183
184     assertTrue(!tree.search(key: 26).hit);
185     assertTrue(tree.search(key: 10).hit);
186     assertTrue(!tree.search(key: 87).hit);
187     assertTrue(tree.search(key: 86).hit);
188     assertTrue(tree.search(key: 20).hit);
189     assertTrue(!tree.search(key: 84).hit);
190     assertTrue(!tree.search(key: 37).hit);
191
192     guaranteeBPlusTree(tree);
193 }
194
195
196 @Test
197 public void testSearch() {
198     BPlusTreeCLI.create(TEST_INDEX_FILE, b: 8);
199     BPlusTreeCLI.insert(TEST_INDEX_FILE, TEST_INSERT_FILE);
200
201     BPlusTree tree = DataFileUtil.loadTree(TEST_INDEX_FILE);
202
203     for (int i = 1; i <= 5; i++)
204         assertEquals(i, tree.search(i).value);
205 }
206
207 @Test
208 public void testSearch2() {
209     BPlusTreeCLI.create(TEST_INDEX_FILE, b: 8);
210     BPlusTreeCLI.insert(TEST_INDEX_FILE, TEST_INSERT_FILE0);
211
212
213     BPlusTreeCLI.search(TEST_INDEX_FILE, key: 68);
214
215     BPlusTree tree = DataFileUtil.loadTree(TEST_INDEX_FILE);
216     assertEquals(expected: 97321, tree.search(key: 68).value);
217 }
218
219
220 @Test
221 public void testRangedSearch() {
222     BPlusTreeCLI.create(TEST_INDEX_FILE, b: 4);
223     BPlusTreeCLI.insert(TEST_INDEX_FILE, TEST_INSERT_FILE0);
224
225     BPlusTreeCLI.rangedSearch(TEST_INDEX_FILE, startKey: 26, endKey: 68);
226
227     BPlusTree tree = DataFileUtil.loadTree(TEST_INDEX_FILE);
228
229     ArrayList<Pair<Integer, Integer>> ret = tree.rangedSearch(26, 68);
230     assertEquals(expected: 26, (long) ret.get(0).left);
231     assertEquals(expected: 37, (long) ret.get(1).left);
232     assertEquals(expected: 68, (long) ret.get(2).left);
233 }
234
235 @Test
236 public void testSorted() {
237     BPlusTreeCLI.create(TEST_INDEX_FILE, b: 8);
238     BPlusTreeCLI.insert(TEST_INDEX_FILE, TEST_INSERT_FILE);
239
240     BPlusTree tree = DataFileUtil.loadTree(TEST_INDEX_FILE);
241
242     ArrayList<Pair<Integer, Integer>> actual = tree.getList();
243
244     for (int i = 0; i < 5; i++) {
245         assertEquals(expected: i+1, (long) actual.get(i).left);
246         assertEquals(expected: i+1, (long) actual.get(i).right);
247     }
248 }
249 }
```