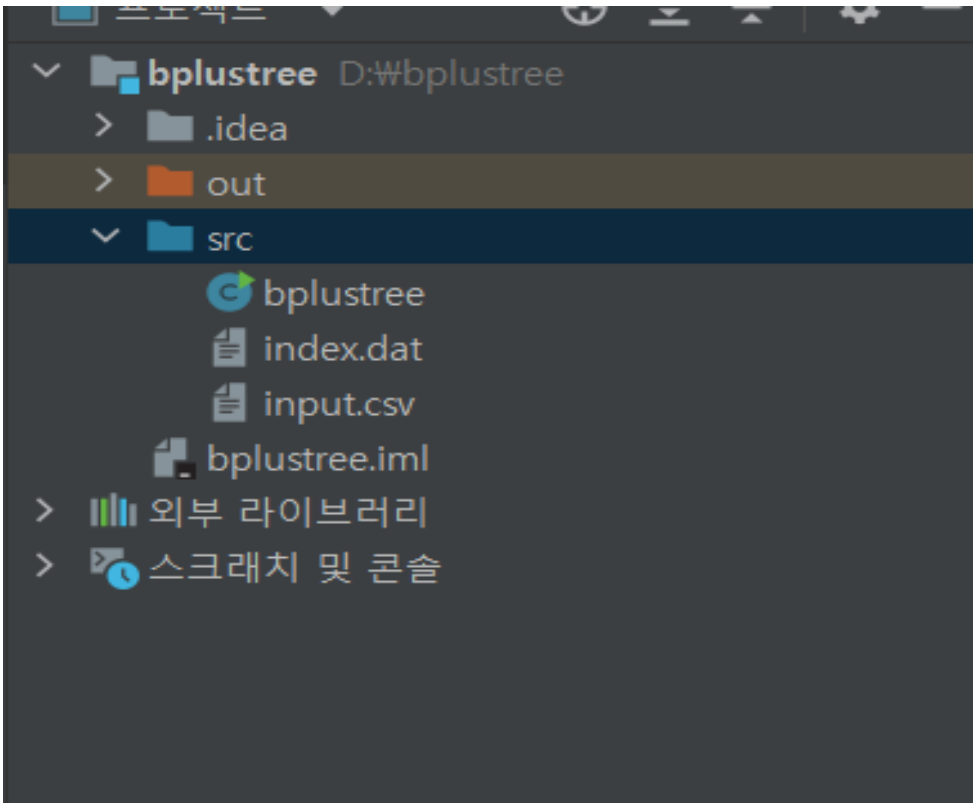
DatabaseSystem Assignment #1

이름 : 탁민주

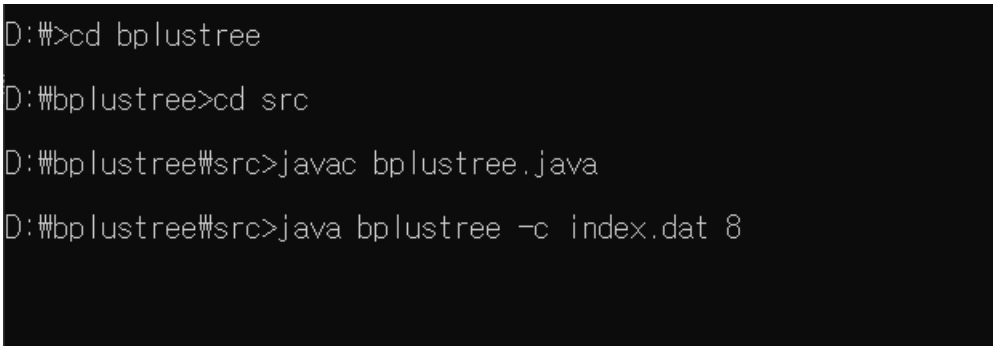
학번 : 2021037156

1. 실행 환경

```
C:₩Users₩tmj55>java -version
java version "18.0.2" 2022-07-19
Java(TM) SE Runtime Environment (build 18.0.2+9-61)
Java HotSpot(TM) 64-Bit Server VM (build 18.0.2+9-61, mixed mode, sharing)
```



IDE는 IntelliJ 를 사용하였고 java version은 18.0.2 에서 실행하였습니다.
bplustree 프로젝트 -> src 폴더 내에 bplustree.java 와 input.csv 를 넣고 실행하였습니다. cmd창에서 명령어를 통해 (cd src) bplustree.java와 input.csv가 함께 있는 폴더까지 가서 컴파일하였습니다.



-cmd 창에서의 실행 화면

- compile

  javac bplustree.java

- creation

  java bplustree -c index.dat <degree>

- insertion

  java bplustree -i index.dat input.csv

- single key search

  java bplustree -s index.dat <key to find>

- ranged key search

  java bplustree -r index.dat <range_start> <range_finish>

2. 알고리즘 요약

✓ Class

- Node

  index node와 leaf node가 공통적으로 사용하는 parent 를 가진다.

  해당 node가 leaf node인지 판단하는 함수를 가진다.

- nLNode

  leaf가 아닌 index node에 해당하는 클래스로, Node 를 상속한다.

  pair형 List, rightmost child node를 가진다.

  index node에서 element를 삽입하는 함수를 가진다.

- Lnode

  leaf node에 해당하는 클래스로, Node를 상속한다.

  pair형 List, rightmost child node를 가진다.

  leaf node에서 element를 삽입하는 함수를 가진다.

- pair

  cloneable 하며 key, value, left child 를 가진다.

- Tree

  root를 통해 전체 node를 관리하는 static 클래스

- degree, leave, index_num 등의 static 변수

✓ creation

- 명령어 java bplustree -c index.dat <degree> 를 실행창에 친다. 파일명에 해당하는 파일을 만들고 degree는 static 변수에 저장한다. 첫줄에 degree 값을 작성한다.

✓ insertion

- 명령어 java bplustree -i index.dat input.csv 를 실행창에 친다. 기존 tree가 있을 경우 그 tree를 읽어온다. 각 줄에서 key 와 value를 읽으며 insert 해준다.
- root가 null인 경우, root가 leaf일 경우는 Lnode의 insert 함수로 실행된다. root가 null 이 아닌 경우에는 nLNode의 insert 함수를 사용해서 범위에 맞는 leaf node 까지 내려간다.
- 요소를 크기에 맞게 저장 후 해당 노드가 overflow가 된 경우에는 차수에 맞 인덱스 (= ceil((degree-1)/2)-1) 값을 기준으로 left node, right node 를 나눈다. 해당 노드가 root가 아닌 경우에는 parent에 인덱스 요소를 삽입한다. 만약 parent에 넣었을 때 overflow가 일어나게 된다면 nLNode의 insert_ 함수를 사용한다. 해당 노드의 사이즈가 degree-1 보다 클 동안 left, right node를 구분한다. 만약 해당 노드가 root 라면 더 이상 신경 쓸 node가 없으므로 parent와 child 관계를 정리한다. 만약 이 overflow된 parent도 parent가 있는 상태라면 다시 parent에 요소를 저장하고 이 함수를 반복하며 overflow가 생기지 않고 root까지 도달하도록 한다.

✓ single key search

- 명령어 java bplustree -s index.dat <key to find> 를 실행창에 친다. 먼저 index.dat에 있는 tree를 불러온다. key를 찾기 위해 insert에서 leaf노드까지 가는 알고리즘을 쓴다. 단, leaf 노드에 도달하기 전까지 만나는 nonleaf node 에서의 요소들을 다 출력한다. 만약 해당하는 leaf node까지 도달했다면 그 leaf node 내에서 key 값을 찾은 후 value 값을 출력한다.

✓ ranged key search

- 명령어 java bplustree -s index.dat <start> <finish> 를 실행창에 친다. 먼저 index.dat에 있는 tree를 불러온다. 먼저 start에 해당하는 leaf node로 내려간다. 이 leaf node에서 start 값보다 작다면 다음 요소로 넘어가는 동작을 반복한다. 만약 start와 finish 내의 요소들을 만났다면 요소들을 출력한다. 만약 그 node의 right sibiling이 있다면 넘어가 위 동작을 반복한다.

3. 코드 상세

✓ static value

```
public static int D; //degree
public static String leave = "LeafNode: ";

public static int index_num = 0;
```

D는 degree를 저장한다. leave는 tree를 write할 때 leaf node의 인덱스를 저장하기 위해 사용된다. index_num 은 index를 따로 저장하는 변수이다.

✓ Class

```
public static class Node {
    int m = 0;
    nLNode parent;


    public Node() {
        this.m = 0;
        this.parent = null;
    }
```

```
    public boolean notLeaf(Node node) {
        if (node instanceof nLNode) {
            return true; //nonleaf
        }
        return false; //leaf
    }

    public void insert_key(int key, int value) throws
CloneNotSupportedException {}

}
```

Node 클래스는 nLNode와 Lnode의 공통된 부분을 담고 있다. degree는 차수를, nLNode parent 가 변수로 들어있다. notLeaf는 해당 노드가 nLNode의 객체이면 true를, LNode의 객체이면 false를 리턴한다.

```
public static class nLNode extends Node { //nonleaf node
    List<pair> p;


    Node rm_childNode;

    public nLNode(int key, int value) {
        m++;
        p = new ArrayList<>();
        p.add(new pair(key, value));

    }

    public nLNode() {
        p = new ArrayList<>();
    }

    public void insert_key(int key, int value) throws
CloneNotSupportedException {
…
public void insert_(nLNode node) throws CloneNotSupportedException {
```

nLNode는 Node를 상속하며 pair List의 p와 오른쪽 child를 담을 rm_childNode를 변수로 가진다. 두가지 insert 함수로 이루어져있다. insert_key는 leaf노드로 내려가는 함수이고 insert_는 nLNode 객체의 overflow가 생겼을 때 해결하는 함수이다.

```
CloneNotSupportedException {
…
```

Lnode는 Node를 상속하며 pair list의 p_, 오른쪽 형제 노드를 담을 r_siblingNode를 가진다. insert_key는 leaf node에서의 삽입을 다루는 함수이다.

root 를 가르키는 Tree 클래스이다.

✓ creation

-c 를 입력받았다면 createFile 함수로 넘어간다.

"./"+ 설정할 파일 이름으로 파일 객체를 만든 후 BufferedWriter 로 해당 파일을 작성한다. D를 전달받은 degree로 설정하고 파일에 degree에 대한 정보를 작성해준다.

✓ insertion

인수로 '-i'를 전달받았다면 작성할 파일과 읽을 파일의 이름을 insertion 함수에 넘겨준다.

기존의 트리가 있을 경우를 대비하여 *readFile을 통해 기존의 트리를 다시 저장한다. 불러온 이후 한줄 한줄 읽어들이며 ","를 기준으로 key와 value를 구분한다. insert 함수에 key와 value를 넘겨준다.

1) LNode 의 insert_key

Leaf 노드에 삽입하는 것이므로 해당 노드의 p에서 key의 위치를 찾아서 요소를 리스트에 저장해준다. (만약 key가 중복이라면 돌아간다.) 만약 차수가 D-1 보다 큰 경우, 즉 overflow 된 경우에는 find index () 함수를 사용하여 split의 기준이 될 index num 을 얻어온다. overflow 되었으므로 left, right 노드로 split 한다. left 에는 0~i-1번째, right에는 i~p.size()-1번째 요소만큼 저장하고 올려질 노드에는 index 번째 요소를 저장한다. 만약 해당 노드가 root인 상태라면 nLNode형 parent 에 index번째 요소를 저장한다. parent의 첫번째 요소의 left child는 left, parent의 right child는 right이다. left, right는 leaf node 인 상태이므로 rm_childNode를 통해 연결해준다. left, right의 parent는 parent 이며 root가 parent를 가르키도록한다. 만약 해당 노드가 root 가 아니라면 해당 노드의 parent 노드를 가르키는 임시 parent를 선언한다. 위의 방법과 똑같이 key의 크기를 비교하며 parent의 리스트에 넣는다. 만약 부모노드에 넣

넣었을 때 overflow가 일어난다면 nLNode의 insert_에 부모노드를 전달한다.

2) nLNode 의 insert_key

nLNode에 insert 된 경우에는 넣을 key와 nLNode의 key를 비교하며 작은 경우는 left child, 큰 경우에는 right child 로 내려간다. 만약 leafNode를 만났을 경우에는 위에 있는 LNode 의 insert_key를 실행시킨다.

3)nLNode 의 insert_

이 함수는 nLNode에서 overflow가 걸렸을 때 들어오게 되는 함수이다. 전달받은 (overflow 걸린) node의 p.size()가 D-1 보다 클 동안 싸이클을 돌게 된다. left와 right 노드를 split 해주는데 이때 index를 기준으로 left는 왼쪽 요소들, right는 오른쪽 요소들만 저장하고 index에 해당하는 요소는 새로운 노드에 저장해준다. 부모 형제 자식 관계를 정리한다. node 가 Tree.root 라면 index를 저장한 노드와 left와 right 를 연결해주고 root에 index를 저장한 노드를 대입한다. 만약 node가 root가 아니라면 그 node의 parent에서 index에 해당하는 요소를 삽입해주고 부모 자식 관계를 정리해준 뒤 node가 그 부모를 가르키도록 할당해준다. 만약 node가 또 overflow 된다면 while 문을 빠져나가지 못하고 재실행된다.

- ✓ single key search

  명령어로 -s 를 입력받으면 singleKey_Search 함수로 넘어간다. readFile을 사용하여 기존의 트리를 불러온 후 root에서 찾으려고 하는 key가 있는 범위의 leaf 노드까지 내려간다. 이때 만나는 nonleaf 노드의 요소를 출력해준다. 만약 leaf 노드를 만났다면 해당 노드의 리스트를 돌면서 key와 일치하는 값을 찾고 해당하는 value 값을 출력한다.

- ✓ range key search

명령어 -r을 입력 받으면 rangedKey_Search 함수로 넘어간다. readFile을 사용항 기존의 트리를 불러온다. 범위의 시작 key가 node의 요소보다 작으면 left child, 크거나 같으면 right child로 내려가며 leaf node로 내려간다. 이때 노드의 요소를 돌면서 시작 범위보다 작은 key들은 무시, 크거나 같은 key 부터 시작하여 right sibling node로 넘어가며 끝나는 범위 내의 요소들을 출력한다.