과제명 : <B+ tree implementation assignment>

1. 클래스 및 변수 설명

- **BpTree : main함수가 들어있는 핵심 클래스. command-line interface와 관련된 명령문을 처리하는 클래스이다.**

Node bptree : B+ tree의 root node
String command : command-line에서 -c, -i등의 명령이 들어가는 변수
String fileName : B+ tree의 정보가 저장되는 파일의 이름 (ex. index.dat)
String info1 : 각 동작에 따라 tree의 degree나 파일의 이름 등으로 사용되는 변수

- **Node : B+ tree의 node에 해당하는 클래스이다.**

static int degree : B+ tree의 degree
int num : 현재 node에 담겨있는 key의 개수
LinkedList<Pair> pair : <key, value>혹은 <key, pointer>의 pair가 저장되는 LinkedList
Node ptr : 가장 오른쪽 child를 가리키는 포인터
boolean leaf : 현재 노드가 leaf인지 알려주는 변수. leaf일 때 true이고 leaf가 아닐 때 false이다.
boolean root : 현재 노드가 root인지 알려주는 변수. root일 때 true이고 root가 아닐 때 false이다.
int height : 현재 node의 높이. leaf일 때 0이고 위로 올라갈수록 1씩 증가한다.

- **Pair : B+ tree의 node내부에 있는 <key, left_child_node>pair나 <key, value> pair를 구현하기 위한 클래스이다.**

int key : key가 저장되는 변수
int value : <key, value> pair의 value가 저장되는 변수
Node ptr : <key, pointer>의 pointer가 저장되는 변수

2. 알고리즘 요약

- **Data File Creation**

command-line으로부터 파일 이름과 B+ tree의 degree를 전달받아서 PrinterWriter 클래스와 FileOutputStream 클래스로 B+ tree의 정보를 기록할 파일을 새롭게 생성한다.

- **Insertion**

B+ tree에서 새로운 key가 들어갈 적절한 leaf node를 찾아서 key를 삽입한 다음 해당 node의 key값의 개수가 degree에 도달했다면 node를 split하여 새롭게 생긴 subtree의 root node를 한 단계 위로 올려준다. 만약 split해서 생긴 root node를 삽입한 node의 key 개수가 degree에 도달하면 또 다시 split을 하고, 이런 식으로 더 이상 split하지 않아도 될 때 까지 split과 arrange를 반복하여 B+ tree의 균형을 맞춰준다.

- **Deletion**

삭제해야 할 key가 들어있는 leaf node와 그 node의 parent node를 찾고 삭제될 key가 height가 1인 node의 가장 왼쪽에 있는 leaf node의 first key는 상위 root에 key로 존재하기 때문에 이를 찾아서 같은 leaf node의 두 번째 key값으로 바꿔주거나(두 번째 값이 존재하는 경우), 부모가 같은 바로 옆 sibling의 첫 번째 key의 값으로 바꿔준다. 그리고 leaf node에서 key를 삭제한다. key를 삭제하고 나서 leaf node에서 underflow가 발생했다면 sibling node로부터 key를 빌리는 것을 시도하고, key를 빌려올 수 없다면 MergeInLeafNode()함수를 호출하여 sibling node와 merge한다. 새로 merge를 통해 생겨난 node의 parent에서 underflow가 발생했다면 먼저 parent로부터 key를 빌릴 수 있는지 확인하고, key를 빌릴 수 없다면 sibling node와 merge한다. 이 과정을 끝내도 underflow가 발생한다면 같은 과정을 여러번 반복하여 underflow를 해소한다.

- **Single Key Search**

각 node의 값을 담을 Node 변수(temp)를 하나 새롭게 생성하여 temp에 있는 key 값들을 출력하고 temp가 leaf node가 아닐 때 temp 내부의 key 값들과 찾고자 하는 key값의 대소를 비교하여 temp 값을 업데이트한다. SearchKey()함수를 재귀적으로 호출하여 temp가 leaf가 되었을 때, 찾고자 하는 key가 해당 node 안에 존재하면 그 key의 value를 출력하고 존재하지 않으면 'NOT FOUND'를 출력한다.

- **Ranged Search**

빈 Node변수 하나를 생성하여(temp) B+ tree의 root를 담고 key1값과 temp의 key값들의 대소비교를 통해 temp가 leaf가 아닐 때 temp의 값을 업데이트한다. leaf node인 temp를 찾으면 temp내에서 key1~key2 범위의 <key,value>를 출력하고 다음 node도 key1~key2 범위의 key값을 가지고 있다면 temp.ptr로 temp를 업데이트 하여 출력을 반복한다.

3. 함수 설명

- **public void CreateBptree(String file, int num)** : Data File Creation을 할 때 사용되는 함수이다. PrinterWriter클래스와 FileOutputStream클래스를 사용하여 전달받은 file이름의 파일을 새롭게 생성하며 파일에 degree의 정보를 함께 저장한다.
- **public void InsertData(String file, String data, int degree)** : Insertion을 할 때 사용되는 함수이다. Scanner클래스와 FileinputStream클래스를 이용해 파일로부터 degree값을 읽어온다. 새롭게 Node를 생성하고, while문을 이용하여 data file에 남은 내용이 없을 때까지 FileInputStream으로 data file를 한줄씩 읽고 split함수를 통해 ',' 를 기준으로 key와 value를 구분한다. 그리고 InsertKey()함수를 호출해서 insert를 진행한다. 모든 과정이 끝나면 정렬된 input값들의 key와 value를 전달받은 파일에 저장한다.
- **public void DeleteData(String file, String data, int degree)** : Deletion을 할 때 사용되는 함수이며 Scanner클래스와 FileinputStream클래스를 이용해 파일로부터 degree값과 key, value 정보를 읽어와 B+ tree를 만들고 FileInputStream클래스와 Scanner클래스를 사용해 Deletekey() 함수를 호출하여 file안에 들어있는 B+ tree에서 data안에 있는 key값들을 삭제한다. 삭제가 완료되면 삭제가 완료된 상태로 정렬된 key와 value를 전달받은 파일에 저장한다.
- **public void SingleSearch(String file, int num)** : Single Search를 할 때 사용되는 함수로, Scanner클래스와 FileInputStream클래스를 통해 B+ tree를 다시 구축한 뒤 SearchKey()함수를 호출하여 single search를 진행한다.
- **public void RangeSearch(String file, int num1, int num2)** : Range Search를 할 때 사용되는 함수로, Scanner클래스와 FileInputStream클래스를 통해 B+ tree를 다시 구축한 뒤

RSearchKey()함수를 호출하여 range search를 진행한다.

- **public void InsertKey(int degree, String file, int key, int value)** : Insertion을 할 때 사용되는 함수이다. FindLeafNode()함수를 호출하여 key가 들어갈 leaf node를 찾아 key를 삽입하고 만약 그 node의 key값의 개수(num)가 degree에 도달했을 경우 SplitNode()함수를 호출하여 split을 진행한다.
- **public Node SplitNode(Node rootnode, int degree)** : overflow된 node가 호출하여 split을 진행하는 함수이다. (degree/2)번째 index인 key가 parent(split key)가 되게끔 split한다. 새로운 node를 3개 생성하여 각각을 parent, leftchild, rightchild로 사용한다. split이 끝나면 새로 만들어진 subtree의 root값을 갖게 되며, 그 root가 전체 B+ tree의 root가 아닐 경우 ArrangeNode()함수를 호출하여 B+ tree의 알맞은 위치에 subtree를 삽입한다. arrange까지 끝낸 후에 생긴 최종 root를 return한다.
- **public Node ArrangeNode(Node root, Node origin, int degree)** : split을 통해 만들어진 subtree의 root를 원래 B+ tree의 적절한 위치에 삽입할 때 사용하는 함수이다. 이 함수의 실행이 완료되기 전까지는 전체 B+ tree가 split이 완료된 형태로 업데이트되기 전이므로, split을 하기 전 상태의 child node를 origin으로 받아와서 FindParentNode()함수를 이용해 split된 child의 parent node를 삽입해야 할 위치인 target을 구하고 target에 split을 통해 생긴 parent node를 삽입한다.
- **public Node FindLeafNode(int key)** : 인자로 받은 key를 삽입하기에 적합한 leaf node를 찾는 함수이다. 빈 Node인 temp를 선언하고 FindLeafNode()함수를 호출한 node의 key들과 가진 key의 값을 비교하여 temp의 값을 업데이트하며 FindLeafNode() 함수를 호출한 node가 leaf node가 아닐 때 FindLeafNode()함수를 재귀적으로 호출한다. 최종적으로 얻은 Node를 return해서 leaf node를 반환한다.
- **public Node FindNonLeafNode(Node node, int key)** : B+ tree의 root node를 전달받아서 인자로 받은 key값이 들어있는 non-leaf node를 찾아 반환하는 함수이다. 반환되는 node가 null일때는 해당 key값이 들어있는 non-leaf node가 없음을 의미한다.
- **public void SearchKey(int key)** : 빈 노드 temp를 this로 설정하고, 먼저 temp에 있는 key값들을 출력한다. 그리고 temp의 각 key 값들과 찾고 싶은 key값의 대소를 비교하여 temp의 값을 업데이트 하여 SearchKey()함수를 재귀적으로 호출한다. this가 leaf가 아니게 되었을 때 loop를 벗어나며 그 때의 this가 해당 key가 있는(혹은 B+ tree 내에 해당 key가 존재했다면 key가 있었을)node이다. for문으로 key를 찾아 key가 존재하면 value를 출력하고 key가 존재하지 않으면 'NOT FOUND'를 출력한다.
- **public void RSearchKey(int key1, int key2)** : 빈 노드 temp를 this로 설정하고, temp가 leaf가 아닐 때 key1과 temp의 key값들의 대소비교를 통해 temp 값을 업데이트 하며 RSearchkey()함수를 재귀적으로 호출하고 temp가 leaf가 되면 loop를 빠져나오고 key1이 속해있는(혹은 속해있어야 하는) leaf node를 찾게 된다. temp의 pair의 첫 번째 key가 key2의 값보다 작거나 같을 때만 while문을 실행시켜 key1과 key2 범위 내에 있는 <key,value>를 출력하며 한 노드의 출력이 끝나면 temp.ptr로 temp의 값을 업데이트한다.
- **public void DeleteKey(int degree, String file, int key)** : B+ tree에서 key값을 찾아 삭제하고 key값을 삭제한 leaf node(target node)에서 underflow가 발생했는지 확인한다. key값을 삭제하기 전에 상위 node에 삭제할 key가 있는지 확인하고 있다면 값을 바꿔준다. leaf node의 first key가 index를 나타내기 위한 key로 사용되는데, 그 중에서도 높이가 1인 node의 가장 왼쪽에 있는 child의 first key는 높이가 2이상인 노드들의 key값으로 사용된다. 이 값들은 삭제하기 전에 target의 두 번째 key값이나 target.ptr에 있는 첫 번째 key값으로 바꾸어준다. key를 삭제하고 underflow가 발생했다면, 오른쪽에 있는 sibling node로부터 key를 빌려올 수

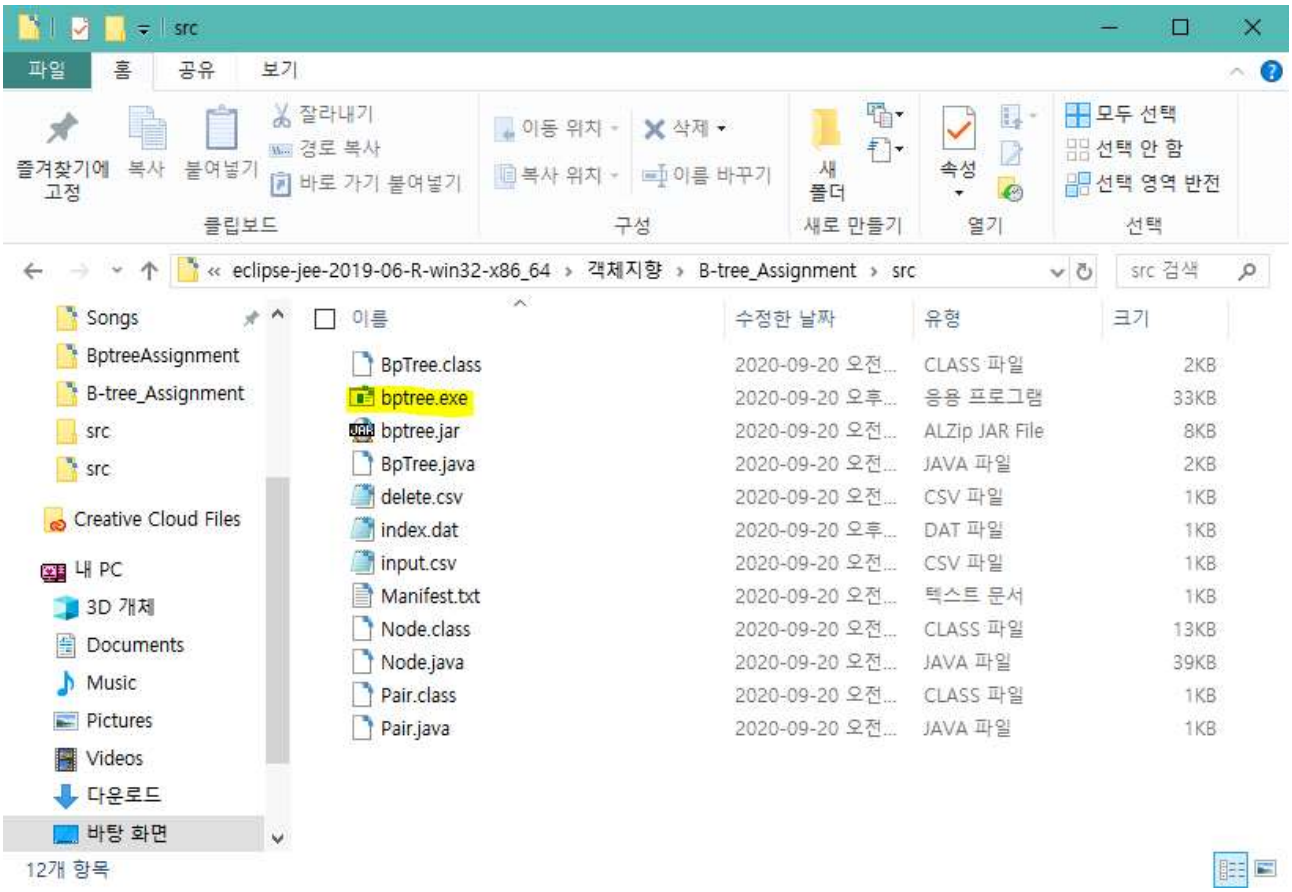
있는지 확인하고, key를 빌려올 수 있으면 key를 빌려오면서 underflow를 해결한다. key를 빌려올 수 없다면 MergeInLeafNode()함수를 호출하여 merge를 수행한다. 이 과정을 수행하고 나서 target node의 parent node에서 underflow가 발생했는지 확인하고 underflow가 발생했다면 UnderflowInNonLeafNode()함수를 호출하여 underflow를 해결한다.

- **public Node MergeInLeafNode(Node root, Node parent, int key, int degree)** : leaf node에서 merge를 수행하고 난 뒤 leaf node의 parent node를 반환하는 함수이다. key값에 따라 바로 옆에 있는 node를 sibling으로 지정해준다. underflow가 일어난 node와 sibling의 위치에 상관없이 무조건 오른쪽에 있는 노드의 <key,value>를 왼쪽에 있는 node에 삽입한 다음 왼쪽에 있는 node의 포인터를 수정해준다. 두 node의 merge를 수행하고 나면 두 node의 split key를 parent node에서 찾아 제거해준다. merge를 끝낸 후 overflow가 발생한다면 SplitNode()함수를 호출하여 split한다.
- **public Node MergeInNonLeafNode(Node root, Node parent, int key, int degree)** : non-leaf node를 merge할 때는 왼쪽으로 merge한다. merge를 해야 하는 node를 target, target과 함께 merge될 node를 sibling으로 정해 key의 위치에 따라 sibling node를 지정해준다. node를 왼쪽으로 merge하게 되면 원래는 sibling이 target의 왼쪽에 위치하게 되지만, target이 parent의 첫 번째 child일 경우 sibling이 target의 오른쪽에 위치하게 되므로 경우를 나누어주었다. target이 parent의 첫 번째 child인 경우 sibling에 먼저 parent의 첫 번째 key와 target의 ptr을 Pair로 만들어 저장하고, target에 있는 <key,ptr>을 sibling으로 옮겨준다. 그리고 parent에서 첫 번째 key를 지운다. 만약 parent가 단일 root였다면 merge로 생겨난 node가 새로운 root가 되므로 root의 값을 true로 바꿔준다. 그 이외에 sibling이 target의 왼쪽에 위치하는 경우에는 sibling에 parent에 있는 splitkey과 sibling의 ptr을 Pair로 만들어 저장해주고, target에 있는 <key, ptr>을 sibling에 삽입해준다. 그 후 sibling의 ptr을 taget의 ptr로 바꿔주고 parent에 있는 key 개수와 parent에서 siling이 몇 번째 index에 위치해있는가에 따라 parent가 sibling을 가리키도록 지정해준다. 그리고 parent에서 key값을 삭제해주고 이 과정을 끝낸 뒤에 새롭게 생긴 merge node(sibling)의 key 개수가 degree를 넘는다면 split을 진행한다.
- **public Node FindParentNode(Node root)** : 특정 node의 parent node를 찾아 return해주는 함수이다. 특정 node를 나타낼 수 있는 key값을 지정하여 key값의 대소비교를 통해 특정 node가 어느 위치에 있을지 탐색한다. temp에 root node를 넣어 key에 값에 따라 temp를 변화시키며 만약 어느 시점에 key의 값에 따른 temp의 ptr값이 특정 node와 같아진다면 그 temp를 반환하게 된다.
- **public Node UnderflowInNonLeafNode(Node parent, Node target, int minKeyNum)** : non-leaf node에서 underflow가 발생했을 때 호출하는 함수이다. parent로부터 key를 빌리는 것을 시도하는 것과 sibling node와 merge하는 과정이 결합된 함수이다. underflow가 해소될 때까지 반복적으로 호출된다. underflow가 일어난 node를 target node로 설정하고, target node의 parent node를 찾아서 target node의 위치에 따라 적절한 sibling node를 찾는다. parent에 있는 key값의 개수가 1개 이상이고, sibling이 최소 key개수보다 많은 key를 가지고 있다면 parent에서 key를 빌려온다.그것이 안된다면 MergeInNonLeafNode()함수를 호출하여 sibling node와 merge한다. 위의 과정이 끝난 뒤에 parent node에 underflow가 발생하지 않았는지 확인하고 발생했다면 underflow가 발생하지 않을 때까지 UnderflowInNonLeafNode()함수를 반복적으로 호출한다.

4. 컴파일 방법(1) - exe파일을 사용

조교님께서는 exe파일을 사용해서 실행해주셨으면 좋겠습니다! 그런데 컴파일 방법이라고 하셔서 .jar 파일을 .exe파일로 바꾸는 과정을 생략한 것이 조금 걱정이 돼서 하단에 .jar 파일을 사용하는 방법을 함께 첨부해두었습니다

1. cmd에서 bptree.exe 파일이 있는 위치까지 이동한다. 이 위치에는 input.csv 파일과 delete.csv 파일이 존재해야 한다. 이 exe파일은 아래에서 만든 bptree.jar을 Launch4j 프로그램을 사용해서 exe파일로 만든 것이며 이 과정은 생략한다. (이 과정을 생략하는 것이 혹시 문제가 된다면 (2)번 방법을 참고해주세요)



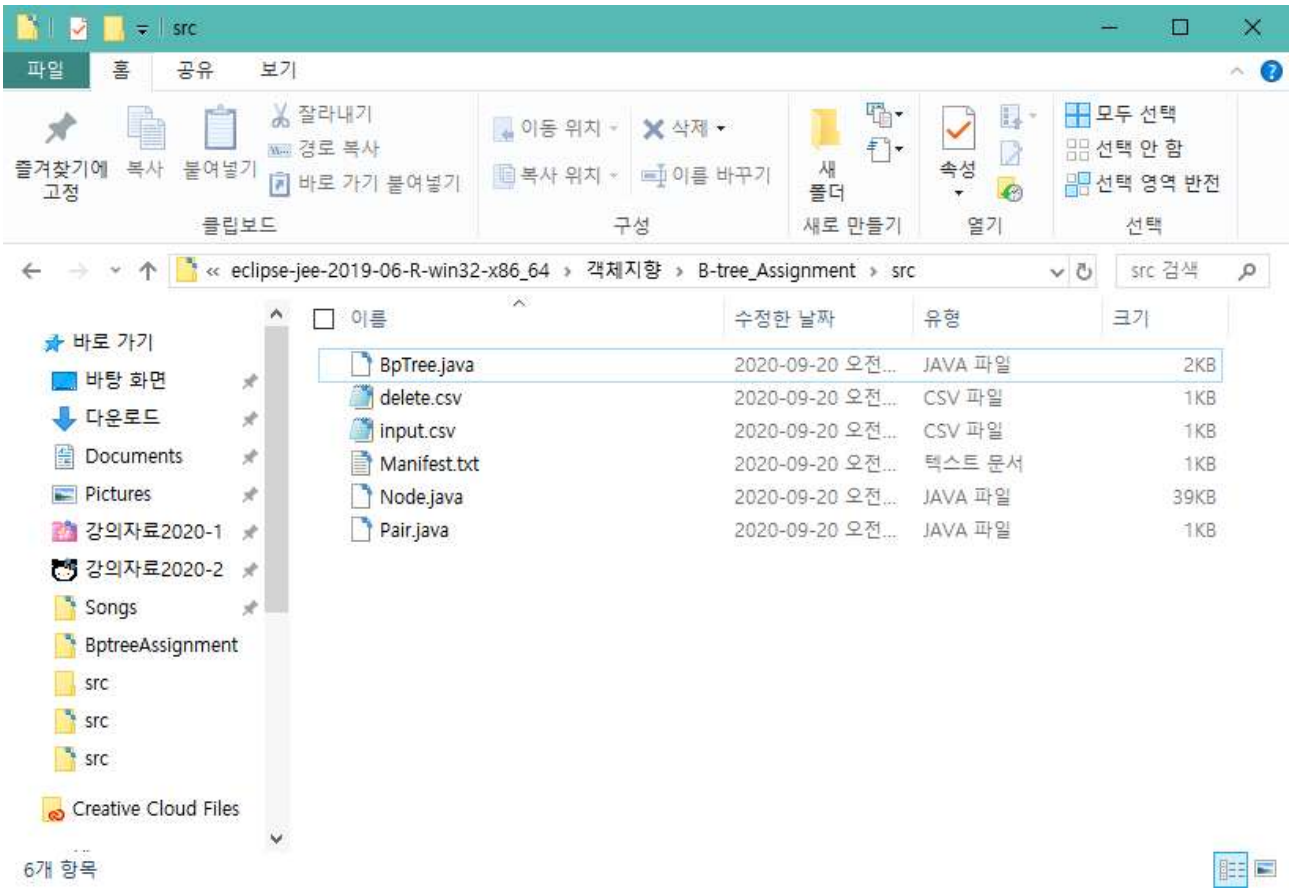
2. cmd에서 명령을 입력하여 파일을 실행시킨다.

```
ex) bptree -c index.dat 4
    bptree -i index.dat input.csv
    bptree -d index.dat delete.csv
    bptree -r index.dat 1 100
    bptree -s index.dat 17
```

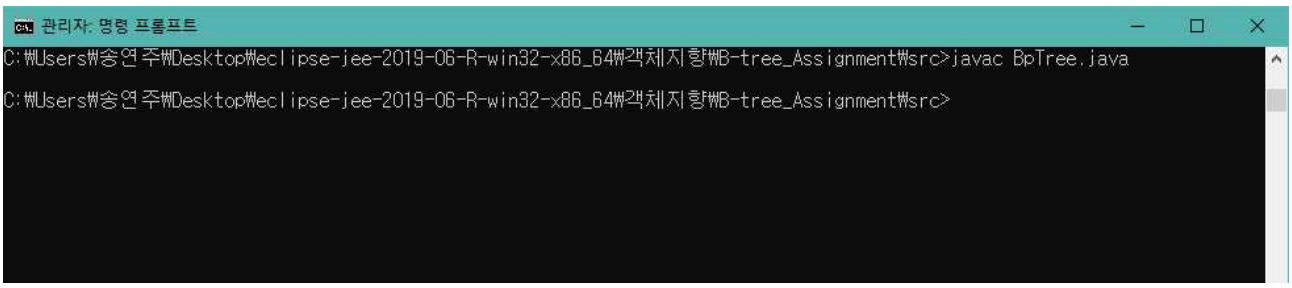
```
C:₩Users₩송연주₩Desktop₩eclipse-jee-2019-06-R-win32-x86_64₩객체지향₩B-tree_Assignment₩src>bptree -c index.dat 4

C:₩Users₩송연주₩Desktop₩eclipse-jee-2019-06-R-win32-x86_64₩객체지향₩B-tree_Assignment₩src>bptree -i index.dat input.csv

C:₩Users₩송연주₩Desktop₩eclipse-jee-2019-06-R-win32-x86_64₩객체지향₩B-tree_Assignment₩src>bptree -d index.dat delete.csv

C:₩Users₩송연주₩Desktop₩eclipse-jee-2019-06-R-win32-x86_64₩객체지향₩B-tree_Assignment₩src>bptree -r index.dat 1 100
6,60
7,70
8,80
17,170
19,250
21,210
24,244
28,200
30,300
32,322
42,420
43,433
44,444
55,555
76,760
77,777
82,822

C:₩Users₩송연주₩Desktop₩eclipse-jee-2019-06-R-win32-x86_64₩객체지향₩B-tree_Assignment₩src>bptree -s index.dat 17
24,44
8,19
170

C:₩Users₩송연주₩Desktop₩eclipse-jee-2019-06-R-win32-x86_64₩객체지향₩B-tree_Assignment₩src>
```

4. 컴파일 방법(2) - jar파일을 사용

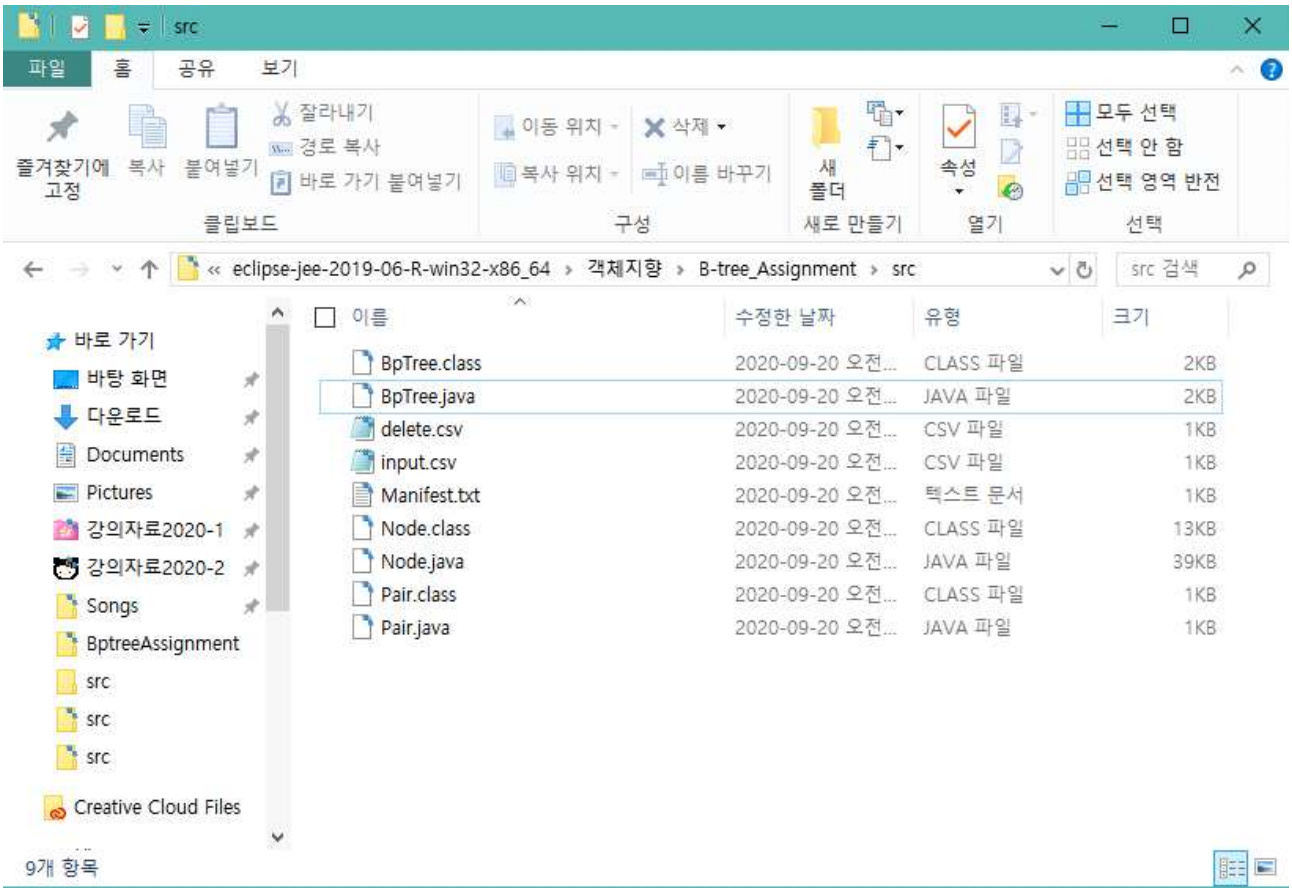
1. .java 파일들과 Manifest.txt 파일이 있는 위치로 이동한다. 이 위치에 input.csv 파일과 delete.csv 파일도 넣어두어야 한다.



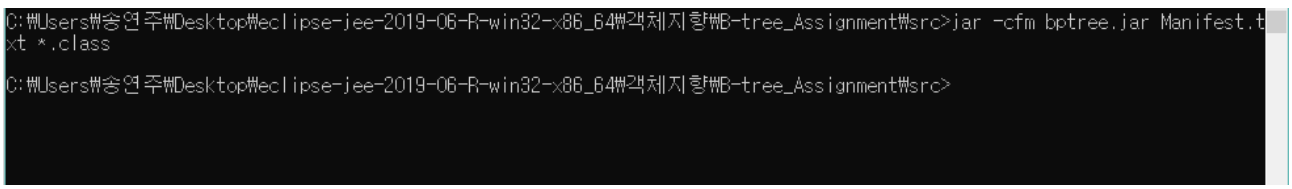
2. cmd에서 **javac BpTree.java**를 입력한다.



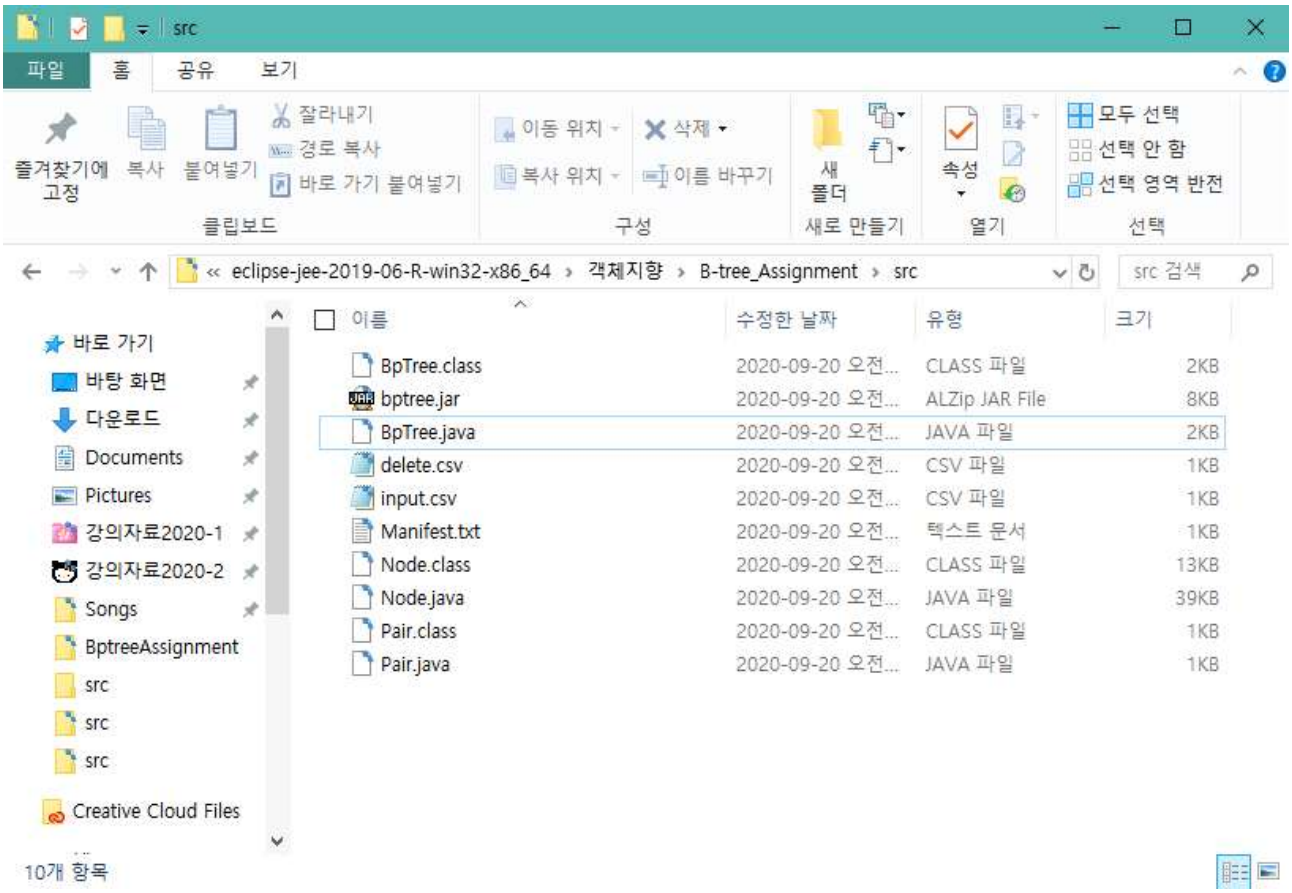
이 과정이 정상적으로 완료되면 .class파일이 생긴다.



3. cmd에서 **jar -cfm bptree.jar Manifest.txt *.class** 를 입력한다



4. 이 과정을 마치면 .jar 파일이 생성된 것을 확인할 수 있다.



5. cmd에서 jar 파일은 **java -jar bptree.jar** 을 이용해 실행시킬 수 있다.

ex) java –jar bptree.jar –c index.dat 3
java –jar bptree.jar –i index.dat input.csv
java –jar bptree.jar –r index.dat 1 100
java –jar bptree.jar –s index.dat 30