2017 B+ tree implementation assignment

Course name: Data Science

Professor: Sang-Wook Kim

TAs: Jangwan Koo , Tae-Ri Kim, Yeon-Chang Lee

2016025987 조민지

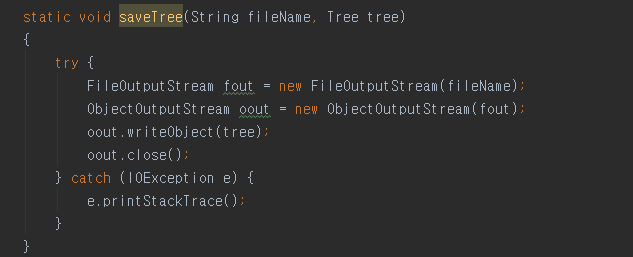
* Data File Creation

예시 ) java bptree.BPlusTree -c index3.dat 4

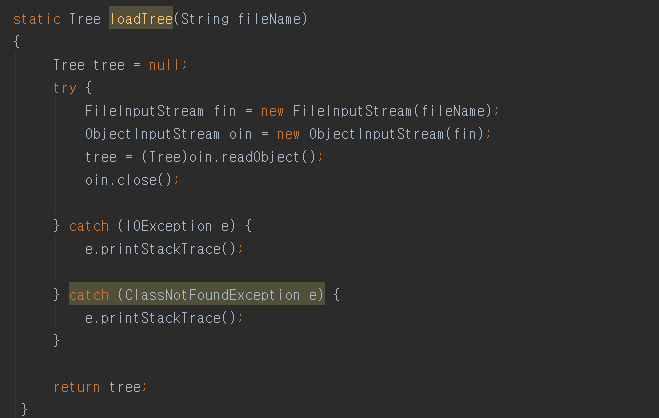
* Summary of algorithm

1. 알맞은 order 의 크기를 입력 받아 그 크기에 맞는 새로운 노드를 만든다.
2. 성공하면 "create new file success" 를 출력한다.
3. 새롭게 만든 노드를 saveTree() 함수를 이용해 저장한다.

* Detailed description



saveTree() 함수 : 노드의 루트를 기억해준다.



loadTree() 함수 : 노드의 루트를 반환해준다.

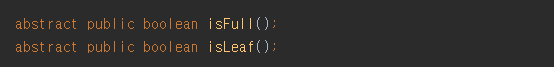
자바에는 포인터가 없어서 **Pair.java** 를 만들어줬습니다.

LeafNode 일 때는 키값과 밸류값을 저장할 수 있고 NonLeafNode 일 때는 키값과 자기의 자식을 저장할 수 있습니다.

**Node.java** 에는 right, parent, left, rs, maxsize 변수가 있다.

Maxsize 변수는 args[2] 에 저장된 값이다.

마찬가지로 자바는 포인터가 없기 때문에 새로운 노드를 하나 만들 때 마다 Node.java 클래스가 실행돼서 각각의 변수에 알맞은 값을 넣어 만든다.



Node 의 정보를 나타내는 isFull() , isLeaf() 함수를 간단하게 구현했습니다.



Node의 부모정보를 입력하고 반환 받기 위해 setParent() , getParent() 함수를 만들었다. LeafNode 일 때와 NonLeafNode 일 때 Pair.java 에 받는 값이 다르기 때문에 abstract class 로 선언하고 각각 클래스를 선언해주었다.

* CMD (스크린샷)



Create new file success 가 출력되었습니다.

* Insertion

예시 ) java bptree.BPlusTree -i index3.dat in50.csv

* Summary of algorithm

1. 이미 키가 있는지 없는지 체크해줍니다.

2. 키가 들어갈만한 적절한 LeafNode 를 찾아 넣어줍니다.

3. 만약 maxsize 에 위반하지 않았다면 -> 괜찮습니다.

4. 위반했다면 노드를 나눠줘야 합니다.

만약 leafNode이고 root 노드이면 중간을 잘라 중간값을 새로 만든 부모에 넣어주고 부모 노드의 양 옆에 자른 2노드를 붙여줍니다.

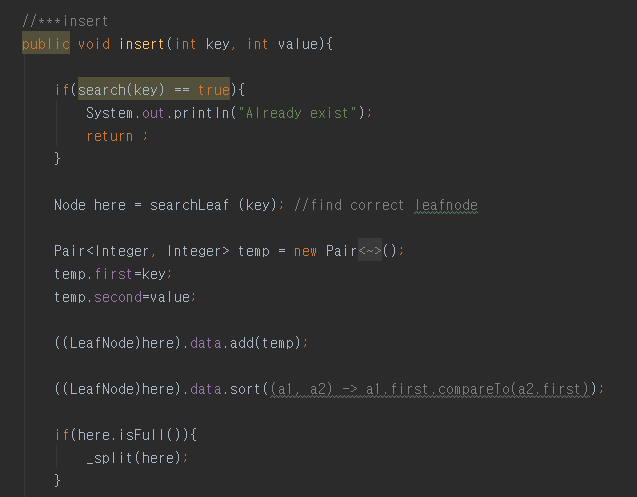
만약 leafNode이고 root 노드가 아니라면 중간을 잘라 중간 값을 이미 존재하는 부모 에 더해주고 새로 추가한 부모노드의 양 옆에 자른 2노드를 붙여줍니다.

만약 NonleafNode 이고 root 노드이면 중간을 잘라 중간 값을 새로 만든 부모에 넣어주고 그 노드의 자식들도 각각 처리해주고 부모 노드의 양 옆에 자른 2노드를 붙여줍니다.

만약 NonleafNode 이고 root 노드도 아니라면 중간을 잘라 중간 값을 이미 존재하는 부모에 더해주고 그 노드의 자식들도 각각 처리해주고 부모 노드의 양 옆에 자른 2노드를 붙여줍니다.

\*그 후 노드의 부모를 조사해서 isFull() 이 참이 경우 4번으로 돌아갑니다.

* Detailed description

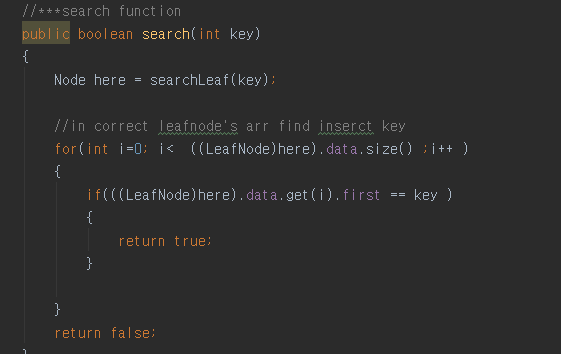


Insert 함수입니다.

Pair.java 를 불러와 입력 받은 key 와 value 값을 저장합니다.

적절한 노드에 추가한 뒤 정렬합니다.

isFull() 함수를 사용해서 넘쳤는지 안 넘쳤는지 조사합니다.

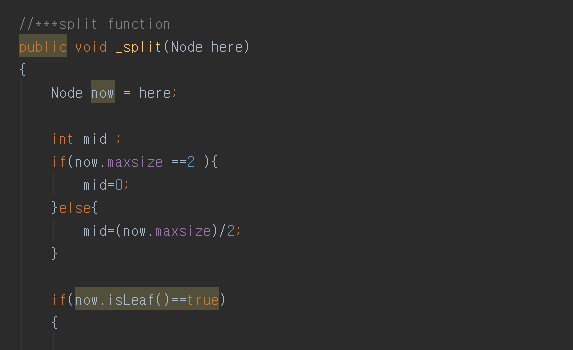


키와 밸류를 받아 search() 함수를 이용해서 이미 존재하는지 안 하는지 체크합니다.



입력 받은 key 값으로 적절한 위치에 있는 LeafNode 를 찾아서 반환해줍니다.

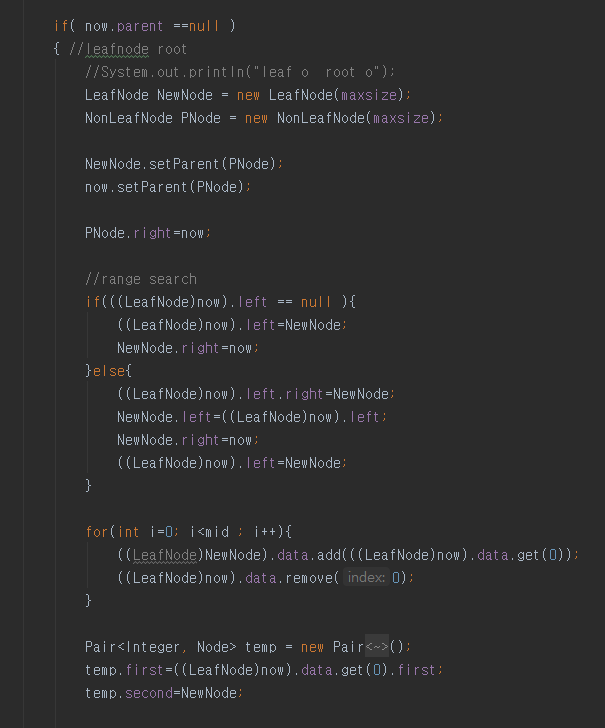
isLeaf() 함수를 이용해서 제일 밑에 도달할 때까지 적절한 연산을 통해 계속 내려갑니다.



isFull() 함수를 이용해서 true 값을 받으면 \_split() 함수로 넘어갑니다.

이 함수는 Node 를 파라 미터로 받습니다.

Case 1 :LeafNode 이고 root 일 때



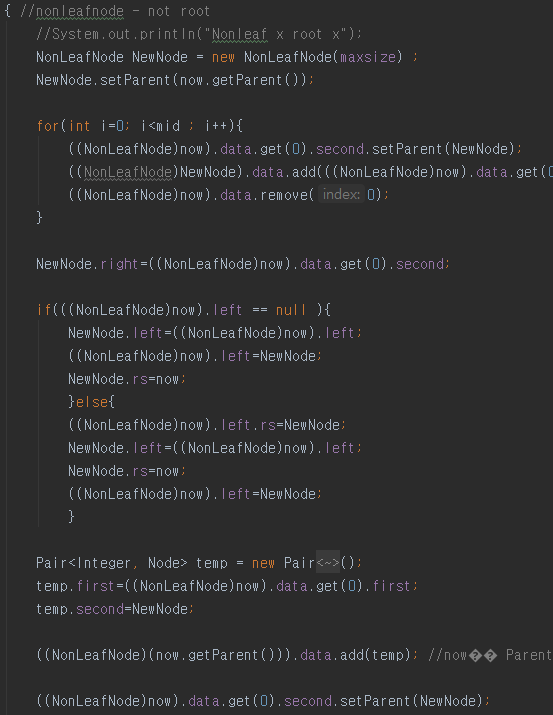
Case 2:leafNode 이고 root 일 때

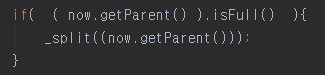


Case 3 : Nonleaf 이고 root 일 때



Case 4 : Nonleaf 이고 root 도 아닐 때





Insert 는 총 4가지의 case로 나뉩니다. 각각 Case 의 마지막에서 그 노드의 부모의 key 개수는 체크하고 재귀합니다.

* CMD (스크린샷)

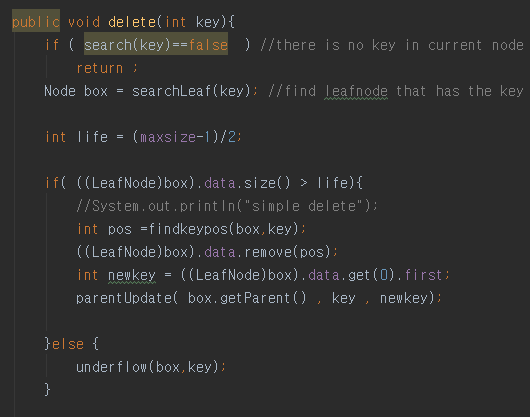


50개의 키 값 중 중복된 값이 있어서 Already exist 가 출력됐습니다.

* Deletion
  + Summary of algorithm
    1. 제거할 키값을 받아서 그 키가 존재하는지 안 하는지 체크합니다.
    2. 키가 있으면 그 키를 갖고 있는 LeafNode 를 찾습니다.
    3. Life 는 maxsize 에 따라 달라집니다. 그 노드가 최소한 몇 개의 키를 가지고 있어야 하는지는 알려줍니다.
    4. 키를 제거하고도 충분하다면 간단한 제거를 합니다. Simple delete -> 키의 위치만 변경해줍니다.
    5. 만약 키를 제거한 뒤 life 의 사이즈에 위반했을 경우 underflow 가 발생해서 underflow() 함수로 갑니다.
    6. 그 노드가 LeafNode 이고 left 에서 borrow 를 해야 한다.
    7. 그 노드가 LeafNode 이고 right 에서 borrow 를 해야 한다.
    8. 그 누드가 LeafNode 이고 left 에서 Merge 를 해야 한다.
    9. 그 노드가 LeafNode 이고 right 에서 Merge 를 해야 한다.
    10. 그 노드가 NonLeafNode 이고 left 에서 borrow 를 해야 한다.
    11. 그 노드가 NonLeafNode 이고 right 에서 borrow 를 해야 한다.
    12. 그 노드가 NonLeafNode 이고 left 에서 Merge 를 해야 한다.
    13. 그 누드가 NonLeafNode 이고 right 에서 Merge 를 해야 한다

각각의 노드의 상황에 맞게 8개의 case의 함수를 재귀적으로 실행한다.

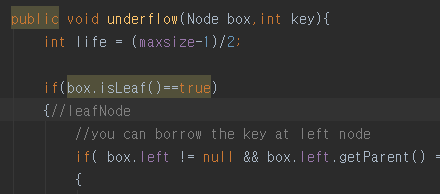
* + Detailed description



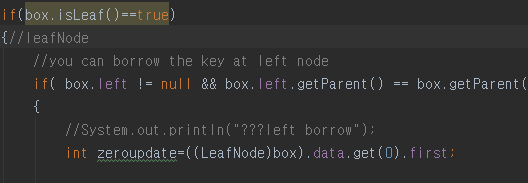
Delete() 함수입니다. Serarch() 함수와 searchLeaf() 함수를 사용합니다.

Life 의 값을 설정해줍니다.

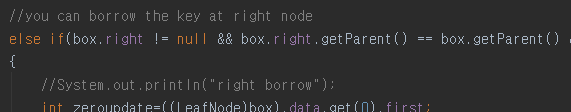
Node 의 키 값이 제거 후에서 life 에 위반되지 않으면 바로 delete 를 할 수 있습니다. 만약 제거 후에 life 에 위반했으면 underflow() 함수를 실행합니다.



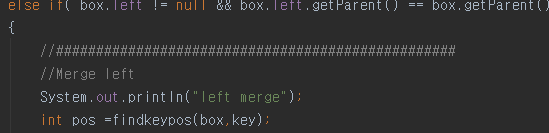
Underflow () 함수입니다. 이 함수는 적절한 노드와 delete 할 키값을 파라 미터로 갖습니다. 이중 if 문으로 case 를 8개로 나눴습니다.



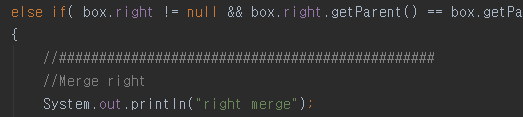
Case 1 : leafNode 이고 left 에서 빌려올 때



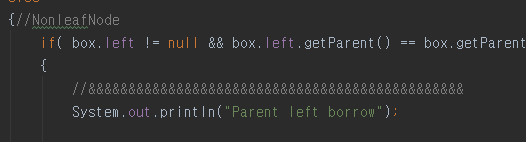
Case 2 : leafNode 이고 right 에서 빌려올 때



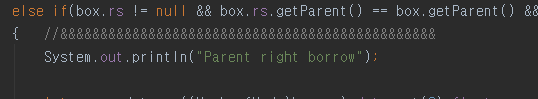
Case 3 : leafNode 이고 left 에서 merge 할 때



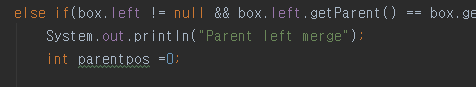
Case 4 : leafNode 이고 right 에서 merge 할때



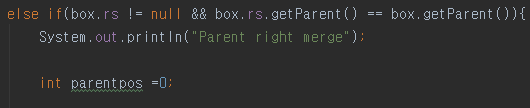
Case 5: NonleafNode 이고 left 에서 빌려올 때



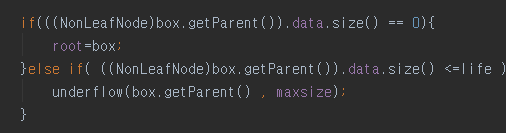
Case 6: NonleafNode 이고 right 에서 빌려올 때



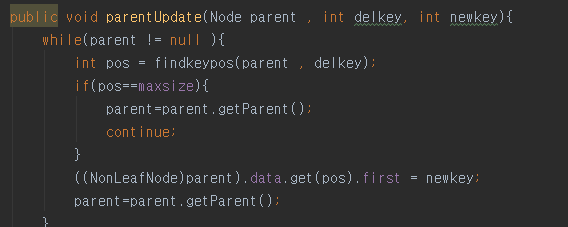
Case 7: NonleafNode 이고 left 에서 merge 할 때



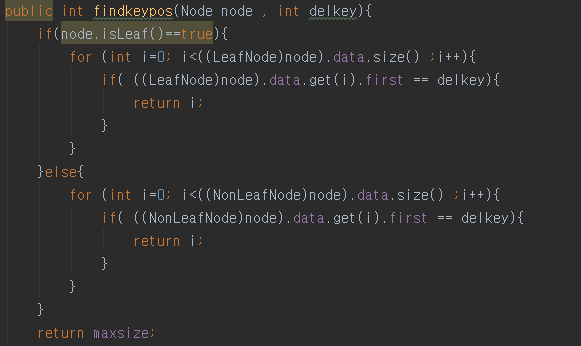
Case 8: NonleafNode 이고 right 에서 merge 할 때



각각의 Merge Case 의 마지막에는 노드의 부모의 키 개수를 체크해서 underflow() 가 일어날 경우 와 root 갱신을 해줘 야할 경우로 나눠서 insert(0 와 비슷한 구조로 재귀함수를 호출합니다.



parentUpdate 함수는 borrow case 의 경우 노드의 개수에 추가 삭제 없이 키값이 변경되는 경우가 많아서 이 함수를 만들었습니다. Merge 할 때도 부모의 키값을 찾을 때 사용합니다.

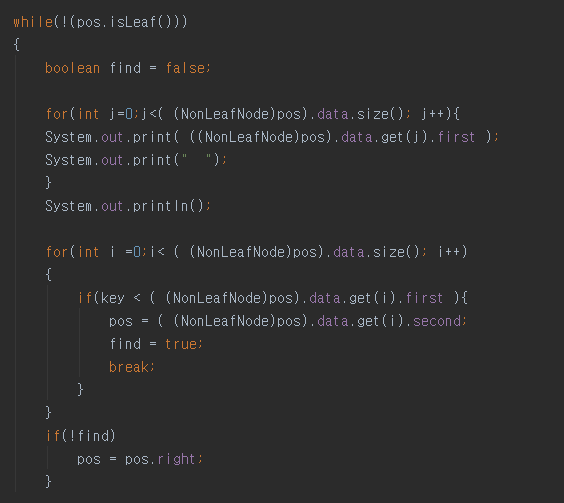


Findkepos () 함수는 노드와 key 값을 파라 미터로 받아 그 노드의 사이즈만큼 for 문들 돌려 key 값이 있는 노드의 위치를 반환해주는 함수입니다. 만약 find를 실패하면 maxsize 값을 반환해줍니다. 이것은 parentUpdate 에서 예외 처리해줬습니다.

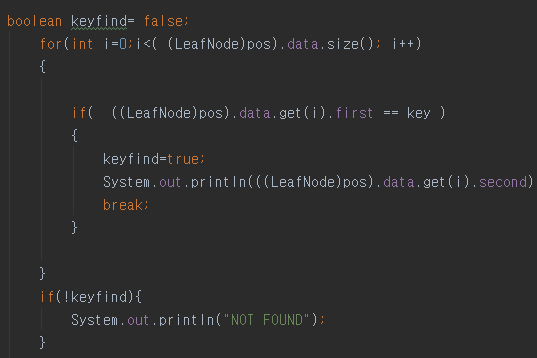
* + CMD (스크린샷)



* Single Key Search
  + Summary of algorithm
    1. 찾으려는 키 값을 입력 받아서 isLeaf() 함수를 이용해서 그 키가 있을만한 마지막 노들을 찾습니다.
    2. Leaf Node 에 도달하면 while 문을 멈추고 그 노드의 키값이 있는지 없는지 노드의 사이즈만큼 for 문을 돕니다.
    3. 만약 키값이 있으면 value 값을 리턴해주고
    4. 만약 키값이 없으면 NOT FOUND 를 출력합니다.
  + Detailed description



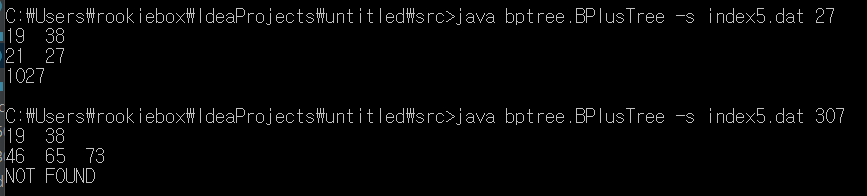
그 키가 있을만한 LeafNode 에 다가갑니다. 가는 동안 만난 노드의 모든 키값을 출력해줍니다.



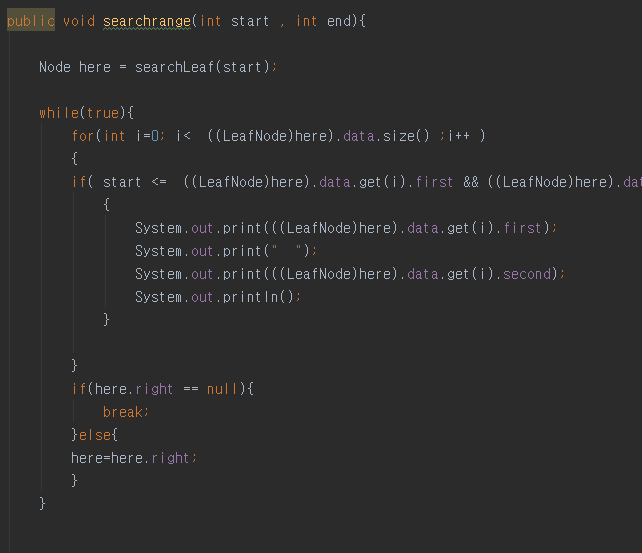
LeafNode 의 사이즈만큼 for 문을 돌아 찾는 키와 노드안에 키가 일치하는지 if 문을 통해 검사합니다. 맞으면 value 값을 리턴 합니다.

* + CMD (스크린샷)

Delte 를 하기전 search



* Ranged Search
  + Summary of algorithm
    1. 첫 번째로 입력 받은 key 값으로 그 키가 있을만한 노드를 탐색합니다.
    2. 찾은 LeafNode 의 사이즈만큼 for 문을 돌려 start – end 사이에 있는지 없는지 탐색합니다.
    3. 한 노드의 탐색이 끝나면 다음 노드로 넘어갑니다.
  + Detailed description



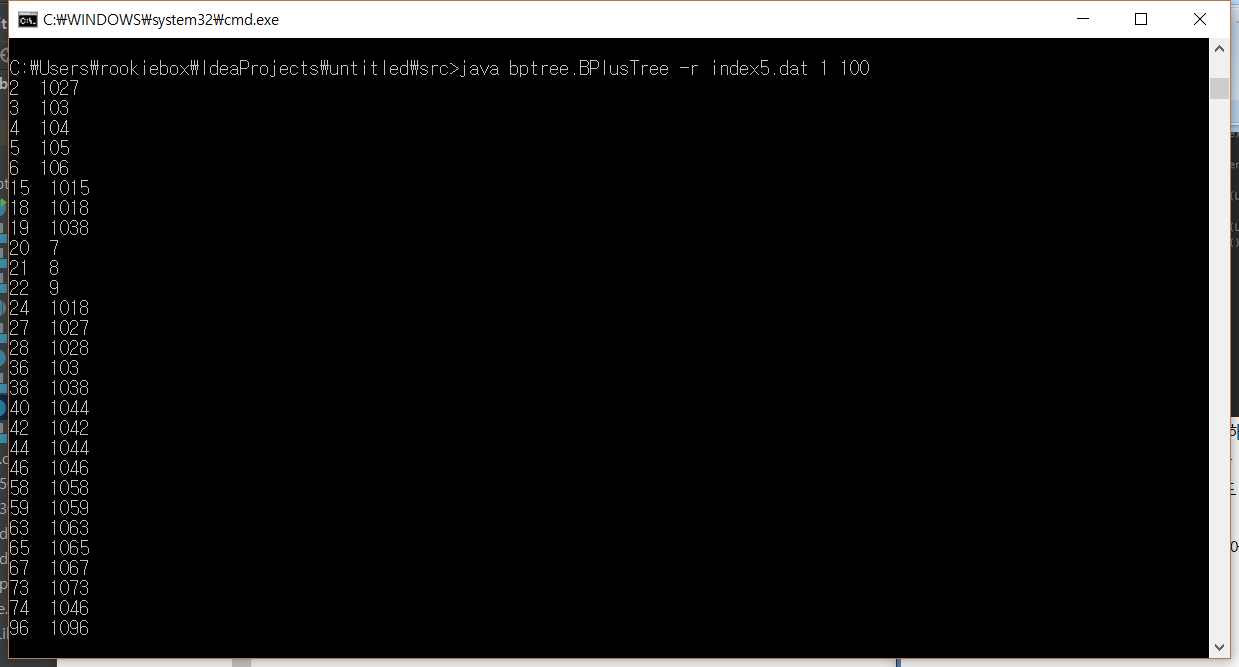
SearchLeaf() 함수를 이용해서 알맞은 LeafNode 를 찾습니다.

그 노드의 사이즈만큼 for 문을 돌려 if 조건에 맞는 키들을 출력합니다.

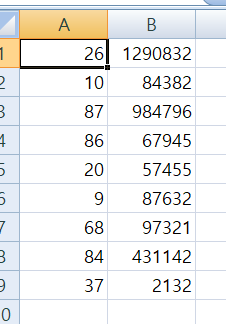
만약 노드 옆에 다른 노드가 있으면 그 노드로 가서 while 문을 계속 돌립니다.

* + CMD (스크린샷)

Delete 함수를 하기전 search 에는 20 이란 값이 있습니다.



실제 input.csv ---- delete 할때 옵션으로 넣었습니다.



Delete 실행후 range search 20 이라는 값이 없어졌습니다.

