데이터베이스시스템과제

B+Tree

Implementation

제출일 2020.09.20
담당교수 김상욱 교수님
전공 컴퓨터공학과
학번 2015005169
이름 최윤석

1. Class introduction

1.1 Key

이 클래스는 (int)Key값과 (int)Value값을 클래스 변수로 가지고 있다.
실제 value값에는 보통 Key값과 연동되어 있는 데이터의 주소 값을 가리키지만 이번 과제에서는 Value값을 Integer형식으로 제한하였다.

변수 명	자료형	설명
key	int	Key of the B plus Tree
value	int	The value associated with Key

1.2 Node

이 클래스는 Key의 값을 저장하는 배열, 자식 노드들을 저장하는 배열, 노드가 리프 노드일 경우 형제 노드를 가리키기 위한 변수, 노드에 포함된 키의 개수를 저장하는 변수를 가지고있다.
BplusTree를 구성 할 노드를 나타내는 클래스이다.

변수 명	자료형	설명
keys	List<Key>	ArrayList for Key objects
childNodes	List<Node>	ArrayList for Node objects
r	Node	To store sibling node for leafNode
n	int	The number of keys

1.3 BplusTree

이 클래스는 BplusTree의 차수를 저장하는 변수 degree와 제일 상위 노드를 저장하는 root라는 변수를 가지고있다. 값의 삽입, 삭제, 탐색 등의 역할도 수행한다.

변수 명	자료형	설명
degree	int	The degree of BplusTree
root	Node	The variable to store the top Node

1.4 Main

따로 특정 변수를 가지고 있진 않다. 이 클래스는 main method를 포함하고 있으며 argument 값으로 명령어와 파일을 입력 받아 그에 해당하는 기능을 수행한다. 변수 대신 명령어에 대한 간단한 사용법만 알아보자.

명령어	기능	형식
-c	Create file	java Main -c [filename1] [degree] Filename1: file to store BplusTree information (String) Degree: The degree of BplusTree (Integer type)
-i	Insert key to B+tree	java Main -i [filename1] [filename2] Filename1 : file to store BplusTree information (String) Filename2: file have (key,value)pair for BplusTree
-d	Delete key from B+tree	java Main -d [filename1] [filename2] Filename1 : file to store BplusTree information (String) Filename2: file have key for delete from BplusTree
-s	Search key from B+tree	java Main -s [filename] [Key] Filename : file to store BplusTree information (String) Key: the integer for searching from BplusTree
-r	Range search from B+tree	java Main -r [filename] [startKey] [endKey] startKey: the integer for searching from BpluTree endKey: the integer for searching from BpluTree

2. Function introduction with algorithm

2.1 BplusTree

2.1.1 public void insert (int key, int value)

이 메소드는 새로운 (key , value)를 입력 받아 BplusTree에 삽입하는 역할을 하는 메소드이다. 세가지 경우의 수가 존재하는데 첫 번째, BplusTree에 아무런 노드가 포함되어 있지 않을 경우이다. 이 경우에 key와 value로 새로운 노드를 생성하여 이 노드를 BplusTree의 루트로 지정한다. 두 번째, 만약 노드가 루트 노드 한 개만 존재하고 이 노드가 키를 추가했을 때 overflow(노드의 키 개수와 bplusTree의 차수가 같거나 커질 경우)가 일어나지 않는다면 루트 노드에 key와 value를 (앞으로는

키를 추가한다고 쓰겠다.) 추가한다. 이 때 다른 메소드 insertInLeafNode를 이용한다. 마지막으로 그 이외의 경우들, 루트 노드에 값을 추가하여서 overflow가 일어나는 상태이거나 루트 노드가 자식 노드를 가지고 있을 경우 임시 노드 (여기서는 sNode가 그 역할을 한다)를 생성하여 sNode.getChildNodes().isEmpty() 가 아닐 경우, 즉 sNode가 nonLeafNode인 경우에 한하여 자식 노드에 접근한다. 이 때 선택할 자식 노드는 binSearch 메소드를 이용하여 고른 다. sNode가 leafNode라면 해당 노드에 키를 추가한 후 만약 leafNode가 overflow가 일어났다면 splitLeafNode 메소드를 이용하여 해당 노드를 분할해 준다.

변수 명	자료형	설명
key	int	the key to be inserted
value	int	the value to be inserted

2.1.2 private void insertInLeafNode (Node node, int key, int value)

이 메소드는 매개변수로 받은 node에 key와 value를 알맞은 위치에 삽입한다. 알맞은 위치는 binSearch 메소드를 이용해 얻어낸다. 만약 key가 이미 해당 노드에 존재한다면 이번 프로젝트에서는 중복된 키가 들어오지 않는다는 가정하에 진행되므로 중복된 키라고 알려준 후 메소드를 중단한다. 만약 Node에 해당하는 키가 없다면 key와 value값을 가진 새로운 Key object를 생성하고 node의 key 배열에 삽입한다. 후에 해당 노드의 n값(키의 개수)를 한 개 늘려준다.

변수 명	자료형	설명
node	Node	Key object is inserted into this node
key	int	the key to be inserted
value	int	the value to be inserted

2.1.3 public int binSearch (int key, List<Key> keys)

이 메소드는 매개변수로 받은 key값이 들어갈 위치를 keys 배열에서 찾아 int값으로 반환한다. keys 노드가 가지고 있는 key값 보다 크거나 같다면 해당 index+1 값을, 작다면 index 값을 반환한다. 첫 시작 값은 0 마지막 값은 keys리스트의 사이즈 -1 (index가 0에서 시작하니 -1을 해준다) 로 정한 후 만약 key가 keys배열의 첫 번째 값(index == 0)보다 작다면 0을 반환한다. 만약 key값이 리스트의 마지막 키 값보다 크거나 같다면 end+1 (마지막 index보다 커야 하기 때문)을 반환한다. 만약 중간에 위치한 값이라면 mid를 (start+end)/2로 설정하고 key가 리스트의 mid 위치의 값보다 작고

mid-1의 값보다 크거나 같을 경우에 해당 값을 반환한다. 아닐 경우 start와 end를 적절히 조절해 가며 알맞은 mid값을 찾아낸다. 다만 나의 경우 start가 0이고 end가 1이면 첫 번째 조건 확인 시 에러가 발생하여 mid가 0이 나오는 경우는 따로 분리하여 처리해주었다.

변수 명	자료형	설명
key	int	the key to be inserted
keys	List<Key>	Array to search for where key will be placed

2.1.4 private void splitLeafNode (Node mNode)

매개 변수 mNode는 overflow상태여서 분할해야 하는 노드 객체이다. 중간 key값을 기준으로 nonLeafNode 1개 좌측 LeafNode 1개 우측 LeafNode 1개로 분할을 하는데 이 때 분할의 기준이 되는 값 minN은 정가운데 값이거나(차수가 홀수일 경우) 가운데보다 오른쪽이다.(차수가 짝수일 경우) 예를 들어 차수가 6인 b+tree는 가운데 노드가 0,1,2,3,4,5 중 2와 3 둘 다 가능한데 내가 만든 B+tree에서는 오른쪽 값을 기준으로 분할을 진행한다. nonleafNode가 될 upNode 와 우측 leafNode가 될 rightNode를 생성한다. upNode에는 mNode의 midN에 위치한 키값을 삽입하고 childNode배열에 mNode와 rightNode를 추가한다. B+tree는 bTree와 다르게 nonLeafNode의 키값이 leafNode에도 존재함으로 rightNode의 key배열을 세팅할 때 midN위치의 키값을 포함하여 설정한다. upNode와 rightNode의 생성을 도와준 mNode의 Key배열중에 minN을 포함한 우측배열 값들을 subList()를 이용해 정리하고 mNode의 n값을 재설정해준다. 만약 mNode가 우측 siblingNode를 가지고 있었다면 해당 노드를 rightNode의 r값에 저장해준다.

변수 명	자료형	설명
mNode	Node	leafNode where the splition will occur

2.1.5 private Node findParent (Node mNode)

매개 변수 mNode의 상위 노드, 부모 노드를 찾는 것이 목적인 메소드이다. 만약 mNode가 B+tree의 root 값이라면 부모 노드는 존재할 수 없으므로 null값을 반환하고 하위 노드가 존재한다면 root를 tmpNode에 저장하고 하위 노드로 향하면서 탐색을 진행하다가 만약 tmpNode와 mNode가 같다면 parentNode를 반환한다. 탐색 메소드 binSearch에 들어갈 key값은 mNode의 키 값들 중 아무거나 이용하면 된다. 이 코드에서는 첫 번째 키 값을 사용하여 탐색을 진행한다.

변수 명	자료형	설명

mNode	Node	method return the parent node of this node

2.1.6 private void mergeNonLeafNode

(Node insertedNode, Node existedNode)

existedNode는 삽입이 진행될 노드, insertedNode는 삽입될 노드 insertedRNode는 insertedNode의 자식노드 중 왼쪽 노드가 가지고 있던 노드 변수 r에 대한 값이다. 만약 existedNode가 null이라면 상위 노드가 존재하지 않다는 것이기 때문에 B+tree의 root를 insertedNode로 설정한 후 메소드를 종료한다. 만약 null이 아니라면 insertedNode의 키 값(이 메소드의 매개변수로 넘어온 insertedNode는 모두 overflow로 인하여 넘어온 변수이므로 키 값이 전부 1개이다)을 이용하여 existedNode에 들어갈 위치를 찾아 삽입을 한다. 키 값이 들어갔다면 해당 index의 1증가된 childNode배열의 위치에 insertedNode의 오른쪽 childNode를 추가해준다. 후에 existedNode의 키의 개수가 degree와 일치한다면 overflow이므로 splitNonLeafNode 메소드를 이용해 분할을 진행한다.

변수 명	자료형	설명
insertedNode	Node	Node to be inserted
existedNode	Node	the parent node of insertedNode

2.1.7 private void splitNonLeafNode (Node currNode, Node parentNode)

currNode가 분할이 진행될 노드. parentNode는 currNode의 부모노드. 인덱스 mid의 값은 degree의 절반으로 한다. Bplustree의 nonLeafNode분할은 Btree와 동일하므로 mid index의 키 값을 가질 노드 midNode와 우측 키 값들과 자식 노드들을 가져갈 rightNode를 생성한다. rightNode에 currNode가 가지고 있던 mid+1 ~key size 만큼의 키와 mid+1 ~ childNodes size 만큼의 자식 노드들을 subList()를 이용해 넘겨준다. midNode에 mid index위치의 키 값을 넣어준 후 currNode와 rightNode를 차례대로 midNode의 자식 노드에 집어넣는다. 전부 끝났다면 currNode에서 minNode와 rightNode에 넘겨주었던 값들을 subList().clear()를 이용해 지워준다. 만약 parentNode가 null이라면 상위 노드가 없다는 소리이므로 root를 midNode로 설정한 후 메소드를 종료하고 null이 아니라면 mergeNonLeafNode 메소드를 이용해 상위 노드와 합친다.

변수 명	자료형	설명
currNode	Node	nonLeafNode to be splited
parentNode	Node	the parent node of currNode

2.1.8 public List<String> search (int key)

Main 메소드에서 search시에 사용할 메소드이다. key값을 입력 받아 상위 노드에서부터 하위 노드를 향해 내려가며 탐색을 진행한다. List<String> lastPrint 리스트에 탐색을 지나면서 거쳐갔던 키 값들을 저장한다. 여기에 index가 0일 경우 왼쪽으로 내려가므로 index가 0일때는 키 값의 왼쪽 노드로 이외에는 키 값의 오른쪽 노드로 향했다는 메시지를 출력하여 조금 더 쉽게 알아볼 수 있도록 만들었다. 만약 리프 노드까지 탐색 후 해당 키 값이 없다면 NOT FOUND를 리스트에 추가하고 모든 탐색이 종료되면 lastPrint 리스트를 반환하며 함수를 종료한다.

변수 명	자료형	설명
key	int	key to be searched

2.1.9 public List<Key> search (int startKey, int endKey)

Main 메소드에서 range search시에 사용할 메소드이다. 2.1.8과 동일하게 키 값을 입력 받지만 중간에 거쳐가는 경로는 따로 저장하지 않는다. 하위 노드를 향해 탐색을 진행할 때 기준으로 삼는 키는 startKey이며 leafNode에 도착하였다면 해당 노드를 시작으로 startKey보다 크거나 같고 endKey보다 작거나 같은 key값들을 순차적으로 탐색해가며 해당하는 값들을 List<Key> findKeys 리스트에 저장한다. 노드가 끝나면 해당 노드에 연결된 sibling node로 r을 이용해 이동하여 탐색을 지속하다가 key값이 endKey보다 커지면 탐색을 종료하고 findKeys 리스트를 반환한다.

변수 명	자료형	설명
startKey	int	startkey to be searched
endKey	int	endkey to be searched

2.1.9 public void delete (int deletedKey)

이 메소드는 지역 변수가 조금 많다. 현재 노드를 표시할 **currNode**, nonLeafNode에서 deletedKey와 일치하는 값이 있을 경우 후에 교체해주기 위하여 nonLeafNode를 저장해 놓을 **parentNode**, parentNode에 deletedKey가 위치한 index를 저장할 **pfindKeyIndex**, 탐색을 진행할 **index**, leafNode에서 값이 나왔다면 해당 위치를 저장할 **leafIndex**, 발견 여부를 저장할 boolean타입의 **nonLeafFind**와 **leafFind**가 있다. 변수를 설명하면서 메소드의 실행과정을 간단하게 설명하였는데 root부터 시작하여 leafNode를 향해 내려가면서 deletedKey와 일치하는 key값이 있는지 확인한다 만약 nonLeafNode에 해당 값이 있다면 nonLeafFind, parentNode, pfindKeyIndex 값을 설정해준다. 마지막 leafNode에서 deletedKey가 없다면 트리에 해당하는 키가 없다고 출력을 해준 후 메소드를 종료한다. 만약 값이 존재한다면 deleteFromLeafNode를 호출한다.

변수 명	자료형	설명
deletedKey	int	key to be deleted

2.1.10 private void deleteFromLeafNode (boolean nonLeafFind, boolean leafFind, int parentIndex, int leafIndex, Node pFindNode, int pfindKeyIndex, Node leafNode)

이 메소드는 리프 노드에서 key값을 삭제하는 과정을 진행한다. 좌,우 노드가 값을 빌려줄 수 있는지 여부를 확인해야 하므로 minOfKey에 Math.ceil((degree-1)/2.0) 값을 저장한다. 위 값은 insert를 진행하면서 자연스럽게 나온 특성이라 삭제 후에도 같은 특징을 유지하기 위함이다. 만약 leafNode가 root라면 다른 값들을 확인할 필요가 없으므로 leafNode에서 값을 제거한 후 메소드를 종료한다. 다만 n값 조정 후 leafNode.getN()의 값이 0이라면 노드에 아무런 값이 없는 것이기 때문에 root를 null로 만들고 종료한다. 그 이외에 경우 leafNode의 상위 노드를 찾고 nonLeafFind가 true일 경우 키값을 교체할 때 사용할 변수 tmpKey를 만든다. 만약 leafNode.getN()>miinOfKey 라면(앞으로는 Landable하다면이라고 표현하겠다) 즉 leafNode가 landable하다면 leafNode에서 값을 삭제한다. 만약 nonLeafFind가 true라면 leafNode에 남아있는 최솟값과 해당 키 값을 교체해준다. leafNode가 landable하지 않다면 좌,우 노드들의 landable여부를 확인한다. 만약 왼쪽 노드가 landable하다면 왼쪽노드의 가장 오른쪽 키 값을 해당 부모 노드에서 왼쪽 노드와 오른쪽 노드 사이에 위치한 key값과 교체해준 후 부모 노드에 있었던 key값을 우측 노드의 가장 첫번째 key값으로 설정한다. 만약 nonLeafFind가 true라면 leafNode의 가장 첫 번째 값을 해당 노드 pfindKeyIndex의 위치에 기존키를 삭제하고 삽입한다. 우측 노드가 landable하다면 바로 위의 경우와 반대로 행해주면 된다 우측 노드의 제일 처음 값, 부모 노드 키값 좌측 노드의 제일 마지막 값을 조정해주고

nonLeafFind의 경우도 동일하게 해주면 된다. 마지막으로 양쪽에서 모두 빌려올 수 없을 경우 우선 leafNode에서 값을 삭제하고 nonLeafFind가 true일 때 leafNode의 n값이 0일 경우 tmpKey에 leafNode의 우측 노드에서 첫 번째 값을 저장한다. 단 제일 우측 노드일 경우 빌려올 노드가 없으므로 tmpKey에 null을 저장한다. 만약 leafNode에 값이 1개라도 있다면 leafNode의 첫번째 값을 tmpKey에 저장한다. 여기서 balanceLeafNode에 값을 넘겨주어야 하는데 leafNode가 부모노드의 첫 번 노드일 경우를 제외하고 leafNode와 우측 ciblingNode와 부모노드와 좌측 노드가 부모노드의 몇 번째 childNode인지 나타내는 값을 넘겨주면 된다.

변수 명	자료형	설명
nonLeafFind	boolean	whether the deletedKey exists in the nonLeafNode
leafFind	boolean	whether the deletedKey exists in the LeafNode
parentIndex	int	index of parnetNode where the leafNode is
leafIndex	int	deletedKey index of leafNode
pFindNode	Node	the Node which has deletedKey
pfindKeyIndex	int	deletedKey index of pFindNode
leafNode	Node	leafNode 😊 where deleted Key exist?

2.1.11 private void balanceLeafNode (Node node1, Node node2 Node parentNode, int leftChildIndex)

이 메소드는 node1 node2 parentNode의 balance를 맞추는 메소드이다. Node1에 node2의 모든 키 값을 추가하고 node1의 n을 변경한다. 부모 노드의 childNodes의 leftChildIndex+1 위치의 노드를 삭제한다. 부모 노드 key에서 leftChildindex 위치 키 삭제. node1의 r 재설정한다. 만약 밸런싱 후 parentNode가 또 밸런스가 깨진다면 상위 노드에서 밸런스를 맞춰야 하니 해당 여부를 파악하기 위하여 upper 변수를 만든다. 만약 부모 노드가 landable하지 않고 부모 노드가 root가 아니라면 upper는 true로 만든다. 만약 부모 노드가 n이 0이고 부모 노드의 부모 노드가 null이라면 node1을 root로 만들고 메소드를 종료한다. 만약 upper가 true라면 여러 조건(부모 노드의 n값이 0이어서 비교할 값이 없는지, parentNode가 그 상위 노드에 어디에 위치해 있는지 등)을 확인한 후에 알맞은 값들을 찾아 balanceNonLeafNode에 값을 넘겨준다.

변수 명	자료형	설명
node1	Node	left Node for balancing
node2	Node	right Node for balancing
parentNode	Node	parent Node for balancing
leftChildIndex	int	index of parentNode.childNodes where node1 is

2.1.12 private void balanceNonLeafNode (Node nodeL, Node parentNode, Node nodeR, int leftChildIndex)

이 메소드는 nodeL nodeR parentNode의 balance를 맞추는 메소드이다. 똑같이 밸런싱 여부를 위하여 upper변수를 선언한다. 후에 nodeL이나 nodeR중에 Landable한 노드가 있다면 balanceLeafNode와 동일한 과정을 통하여 값을 옮겨준다. 둘 다 Landable하지 않다면 nodeL에 parentNode의 키 1개와 nodeR의 키 값을 전부 nodeL에 옮겨주고 nodeR의 모든 자식 노드를 전부 nodeL에 옮겨준다. 만약 이 때 parentNode의 모든 키를 가져다 썼다면 nodeL을 root로 설정하고 메소드를 종료한다. parentNode의 키가 남아있고 upper가 true라면 balanceLeafNode의 과정과 동일한 과정을 거쳐서 balanceNonLeafNode를 재호출한다.

변수 명	자료형	설명
nodeL	Node	left Node for balancing
nodeR	Node	right Node for balancing
parentNode	Node	parent Node for balancing
leftChildIndex	int	index of parentNode.childNodes where node1 is

2.2 Main

2.2.1 public static void main (String[] args)

bplusTree의 실행을 도와줄 main method. Argument를 입력 받아 해당하는 명령어에 알맞은 기능을 수행한다. 명령어의 설명은 1.4 Main에 위치하여 있으니 읽어 보길 바란다. 각각의 기능에 따라서 파일(index.dat)에서 BplusTree 정보를 가져와 빠르게 재구성을 하여 BplusTree 클래스의 기능을 이용해 명령을 마치거나 트리를 수정하고 다시 메소드를 통하여 BplusTree정보를 간단히 변환하여 파일에 덮어쓰기 해준다. 관련된 기능은 아래에서 살펴보자.

2.2.2 private static List<String[]> readFromCsvFile (String fileName)

Main 클래스에서 argument로 -i 혹은 -d를 입력 받았을 때 작동하는 메소드이다. CsvFile로부터 데이터를 읽어와 List<String[]>list 배열에 저장하고 반환한다.

변수 명	자료형	설명
fileName	String	The file which has input data or delete data

2.2.3 private static BplusTree makeBplusFromFile (String fileName)

이번 과제 B+tree implementation에서 상당히 중요한 역할을 하는 메소드이다. 프로그램 내에서 BplusTree정보를 유지하고 있는 것이 아닌 외부 파일에 B+tree 구조에 대한 정보를 가지고 있다가 필요할 때 마다 불러오고 새로운 데이터 추가나 삭제가 일어날 때마다 트리 수정 후 파일을 갱신해주는 것이 목적인 메소드이다. 처음엔 Serialize를 이용하여 BplusTree object 전체를 index.dat 파일에 저장하여 읽고 쓰기를 사용하였는데 입력키 값이 100만개를 넘어서자 B+tree의 차수를 크게 증가시키지 않는 한 계속하여 에러가 (StackOverFlow) 발생하였다. 문제를 해결하기 위해 고민을 한 결과 List<Node>를 이용하여 파일을 저장하기로 하였다. 우선 List<Node> mainNodeList와 List<Integer> numChilds를 만들어주었다. mainNodeList는 index.dat에서 정보를 읽어와 Key와 Value값이 들어가 있는 새로운 노드를 만들어 저장하기 위한 리스트이며 numChilds는 mainNodeList와 같은 index에 저장된 노드의 childNode의 개수가 저장되어 있는 리스트이다. 파일 속 데이터는 ‘,’를 이용하여 구분하며 자세한 Index.dat 파일 내부 형식은 [2.2.4 writeBplusToFile]에서 설명하도록 하겠다. 파일에서 읽어온 String[] 정보를 저장하기 위한 list 리스트를 생성하여 index.dat의 내용을 읽어낸다. 만약 list.size()가 0이라면 파일을 잘못 읽어온 것이므로 index.dat파일을 먼저 만들어달라고 출력하고 프로그램을 종료한다. list.size()가 1이라면 해당 파일에 아무런 노드도 저장되어 있지 않은 상태, root가 null인 상태를 의미하므로 트리의 degree만 설정하여 비어있는 상태의 BplusTree bt를 만들고 이를 반환한다. 그 외의 경우 Bplustree의 degree를 설정하고 list의 1번 index부터 마지막 index까지 진행을 하며 새로운 노드를 만들고 n값을 설정해준다. 입력받은 n값에 따라서 새로운 Key, Value 값들을 노드의 Keys 리스트에 추가해준다. 키 입력이 끝난다면 해당 노드를 mainNodeList의 제일 뒤에 저장해 준다. 그리고 list의 마지막 열에 저장된 값은 노드의 chlidNode의 개수를 의미하므로 노드가mainNodeList에 저장된 index와 동일한 곳에 노드의 개수를 numChilds리스트에 추가한다. 그 다음 넘겨받은 정보 (degree, mainNodeList, numChilds)들을 이용하여 새로운 BplusTree를 빠르게 구성해준다. mainNodeList에 저장된 순서는 루트부터 순서대로 하위 노드로, 같은 depth에 있는 노드들이라면 왼쪽에서 오른쪽을 향해서 저장이 되어 있으므로 손쉽게 childNode와 r을 설정할 수 있다. 예를 들어 내가 mainNodeList에 0번을 루트에 저장한 후 childNode를 설정하고 싶다면 numChilds.get(0)에서 하위 노드의 개수를 알아낸 후 반복문을 통해 mainNodeLlist.get(0)의 child노드에 순서대로 추가한다. 이때 mainNodeList의 노드들 중에 ChildNode 설정에 사용된 노드를 구분하기 위해 useIndex라는 변수를 활용한다. useIndex의 값을 해당 노드의 자식 노드 수만큼 증가시켜 사용하는 것이다. 다음 mainNodeList의 1번 노드의 childNode를 추가하고 싶다면 useIndex부터 자식 노드 개수만큼의 노드를 추가하는 방식이다. 만약 자식 노드의 개수가 0이라면 childNode가 아닌 r을 설정하여 leafNode를 연결해준다. 모든 과정이 종료되면 bplusTree를 반환하며 메소드를 종료한다.

변수 명	자료형	설명
fileName	String	The file you want to read for make BpluTree

2.2.4 private static void writeBplusToFile (String filename, BplusTree b)

2.2.4 메소드 역시 중요한 역할을 하는 메소드이다. makeBplusFromFile 메소드가 파일에서 정보를 읽어와 BplusTree를 구성하여 프로그램에 넘겨주었다면 이번 메소드는 프로그램으로부터 BplusTree b를 넘겨받아 파일에 나중에 읽어올 수 있도록 저장하는 메소드이다. 파일에 저장하는 방식은 다음과 같다. 우선 BplusTree의 차수를 읽어와 파일의 제일 첫 줄에 입력을 한다. 만약 빈 트리라면 차수만 넘겨주고 메소드를 종료한다. 추가적인 구성요소가 있다면 root를 먼저 List<Node> pushNodeList에 입력한 후 currNode 객체를 이용하여 root를 참조한다. currNode의 자식 노드 개수가 0일때까지, 즉 nonLeafNode일 동안 탐색을 계속하여 진행하며 위에서 아래로, 왼쪽에서 오른쪽으로 순차적으로 BplusTree의 모든 노드를 pushNodeList에 저장한다. 저장이 끝났다면 파일에 정보를 입력하기 시작하는데 정보는 다음과 같이 구성되어 있다.

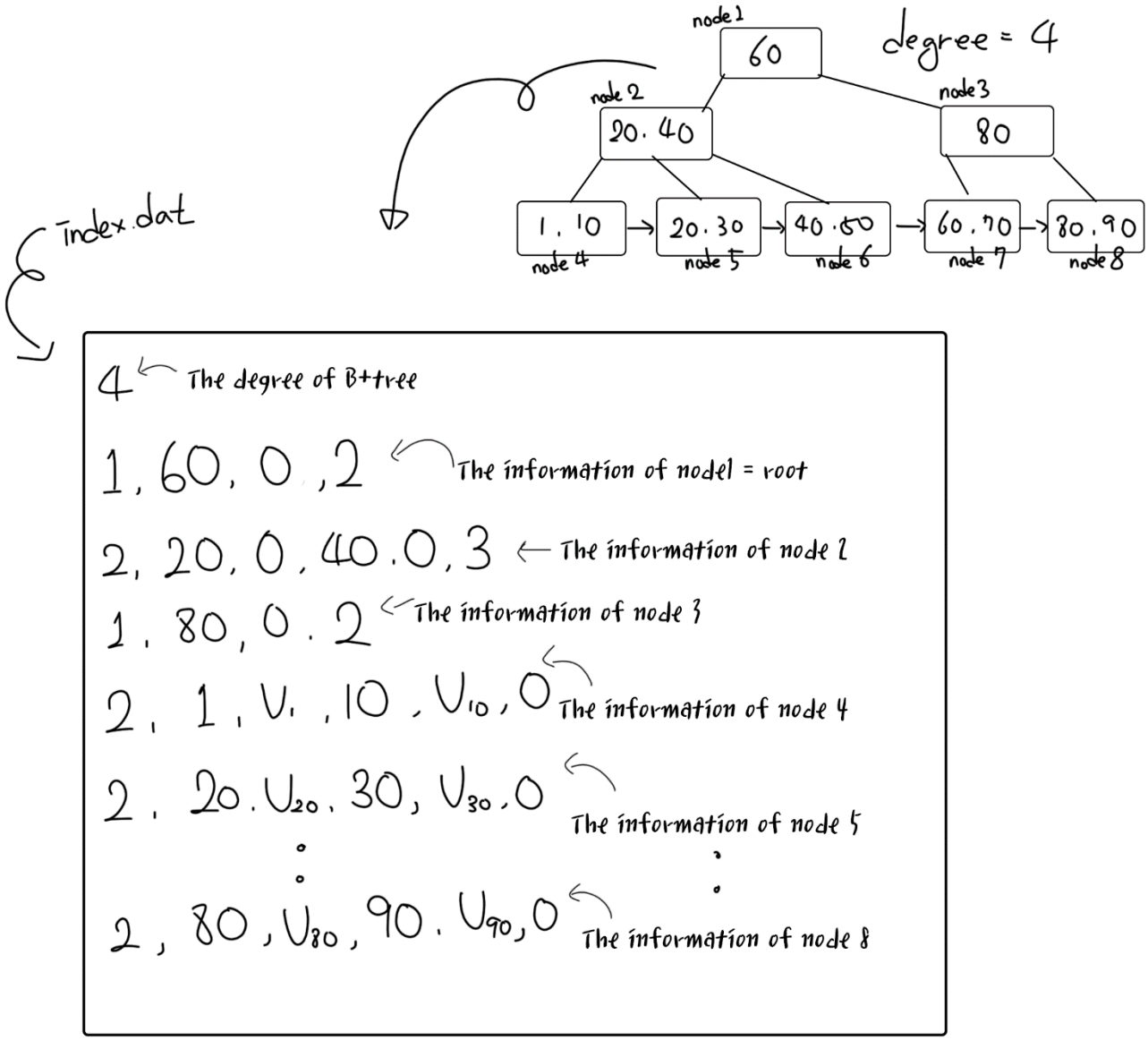
key의 개수(n)	(key 1, value 1), (key 2, value 2), ··· ,(key n, value n)	childNode의 개수

마지막 노드까지 전부 파일에 입력한 다음 메소드를 종료한다.

변수 명	자료형	설명
fileName	String	The file you want to write about BpluTree
b	BplusTree	BplusTree object you want write to file

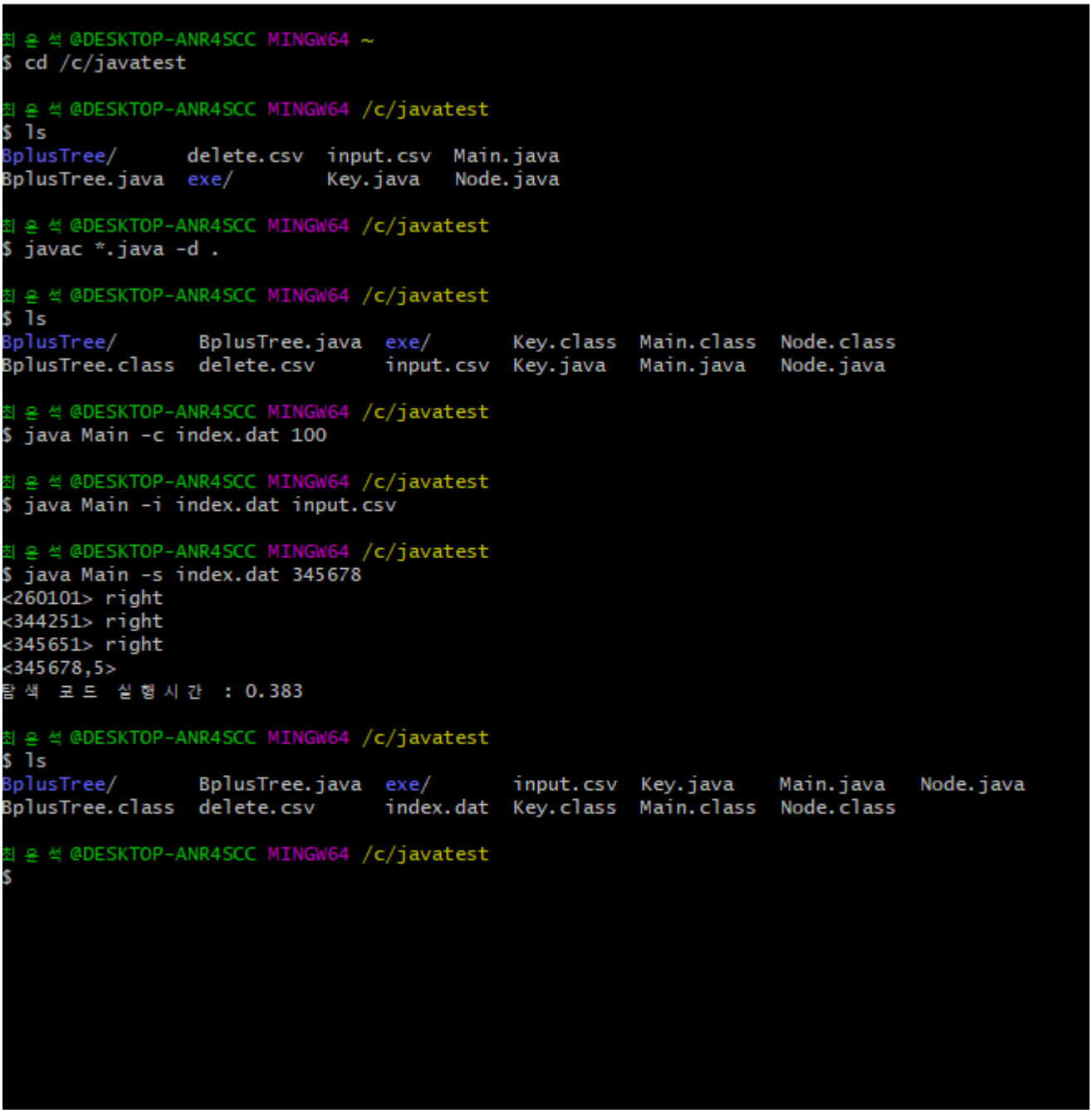
3.Index.dat file explanation wit example

간단한 예시를 통하여 Index.dat 의 구성에 대해 알아보겠다.



4. Instructions for compiling and result from my computer

만약 처음부터 컴파일을 할 예정이라면 모든 java파일을 한 군데 넣은 후 java 명령어를 이용하여 컴파일 하면 된다. 실행 결과는 다음과 같다.



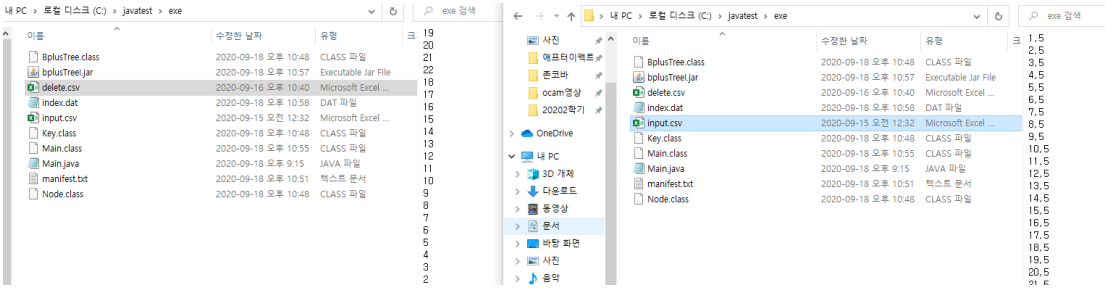
Jar 파일을 만드는 과정은 다음과 같다 밑의 두 png파일은 각각 jar파일 생성과 jar파일을 이용한 실행 테스트 결과이다.

MINGW64:/c/javatest/exe

```
최 윤 석 @DESKTOP-ANR4SCC MINGW64 ~
$ cd /c/javatest/exe

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ ls
BplusTree.class  Key.class  Main.class  Main.java  manifest.txt  Node.class

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ cat manifest.txt
Main-Class: Main

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ jar -cvmf manifest.txt bplusTreeI.jar Main.class BplusTree.class Node.class Key.class
added manifest
adding: Main.class(in = 7180) (out= 3881)(deflated 45%)
adding: BplusTree.class(in = 8506) (out= 4328)(deflated 49%)
adding: Node.class(in = 1481) (out= 729)(deflated 50%)
adding: Key.class(in = 1078) (out= 555)(deflated 48%)

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ ls
BplusTree.class  bplusTreeI.jar  Key.class  Main.class  Main.java  manifest.txt  Node.class

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ java -jar bplusTreeI.jar -c index.dat 100

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ ls
BplusTree.class  bplusTreeI.jar  index.dat  Key.class  Main.class  Main.java  manifest.txt  Node.class

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$
```

```
최 윤 석 @DESKTOP-ANR4SCC MINGW64 ~
$ cd /c/javatest/exe

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ ls
BplusTree.class  bplusTreeI.jar  delete.csv  index.dat  input.csv  Key.class  Main.class  Main.java  manifest.txt  Node.class

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ java -jar bplusTreeI.jar -c index.dat 100

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ java -jar bplusTreeI.jar -i index.dat input.csv

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ java -jar bplusTreeI.jar -s 21
Wrong use with -s parameter
-s [The file, contain BplusTreeData] [the key you want find (int type)]

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ java -jar bplusTreeI.jar -s index.dat 21
<130051> left
<2551> left
<51> left
<21,5>
탐색 코드 실행시간 : 0.356

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ java -jar bplusTreeI.jar -d index.dat delete.csv

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ java -jar bplusTreeI.jar -r index.dat 21
Wrong use with -r parameter
-r [The file, contain BplusTreeData] [the key you want find start (int type)] [the key you want find end (int type)]

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$ java -jar bplusTreeI.jar -s index.dat 21
<260101> left
<5101> left
<101> left
Not Found
탐색 코드 실행시간 : 0.377

최 윤 석 @DESKTOP-ANR4SCC MINGW64 /c/javatest/exe
$
```

밑 사진은 테스트한 input.csv와 delete.csv 파일 내용이다. Input.csv는 1~1000000까지의 key값을 저장해두었고 delete.csv파일은 1~35까지의 key값을 가지고 있다.



마지막으로 exe파일은 jsmooth를 이용하여 만들었으며 해당 사진은 exe파일을 가지고 실행하는 모습이다.