1. 알고리즘 요약

b+tree는 키에 의해서 각각 식별되는 레코드의 효율적인 삽입, 검색과 삭제를 통해 정렬된 데이터를 표현하기 위한 [트리자료구조](https://ko.wikipedia.org/w/index.php?title=%ED%8A%B8%EB%A6%AC%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0&action=edit&redlink=1)의 일종이다.

알고리즘은 크게 삽입, 삭제로 나뉘며 삽입에서는 split 작업이, 삭제에서는 merge와 redistribution(재분배) 작업이 필요하다.

먼저 Node에 관한 생성자를 두 개 만들었다. 하나는 non-leaf의 경우를 받고, 다른 하나는 leaf의 경우를 받는다. 논리프의 경우에는 키 값만 필요하지만 리프 노드의 경우 키 값과 포인터가 모두 필요하기 때문에 경우를 나누었다.

Insert 함수는 재귀적으로 짰으며, 리프의 오버플로우가 발생하면 두 개의 노드로 분할하고

키 값들을 절반씩 분배해서 저장했으며, 분할된 왼쪽노드에서 제일 큰 키 값을 부모 인덱스 노드로 저장했다.

Delete함수도 재귀를 이용했다. underflow발생시 redistribution과 merge 두 경우가 있다. redistribution은 인덱스 키 값을 조정하는 작업을 거쳤고, merge시는 인덱스 값을 삭제했다.

또한 루트일때와 루트가 아닐 때 경우를 나누었다. 따라서 경우의 수가 많이 발생했고, 2.번에 나올 함수들을 통해 경우의 수를 처리하였다.

1. 소스코드 설명
2. bptree.java

Bptree : 노드 사이즈(최대 자식 수)를 받는다.

insertList : 인풋 사이즈만큼 입력을 받은 후, insert함수로 넘긴다

Insert : 본격적인 insert 작업을 수행하는 함수이다. Leaf node일때와 non-leaf node일때 두 가지 경우로 나누었다. 리프 노드일때는 키 값과 포인터 값이 모두 필요하기 때문에 그 둘에 관한 배열을 만들었다. Full일 경우에는 splitNode라는 함수로 넘겨주었다. 논리프 노드일 경우에는 또 두가지 경우가 있는데, 일단 full에서 루트일 경우 InsertAndsplitRoot함수를, 그렇지 않은 경우 InsertToNode함수로 넘겨 주었다. 이 함수는 주로 다른 함수에 값을 전달해 주는 방식으로 구현했다.

deleteList : 입력을 받고 deleteInRoot함수로 넘겼다.

deleteInRoot : 리프 노드인 경우, deleteInLeaf함수로 넘겼다. 리프 노드가 아니라면 먼저 다음 노드를 찾고, 해당하는 노드와 키값을 delete함수에 전달했다. 노드가 비어있으면 deleteNode를, 그리고 비어있는 노드가 루트라면 루트를 다음 노드로 설정했다. (지워지기 때문에)그리고 반복문을 다 돈 뒤에는 남은 키들을 확인했다.

Delete : insert처럼 본격적인 delete작업을 수행하는 함수이다. 먼저 리프 노드일 경우에는 크게 오른쪽 sibling (sibling[1]) 을 보는 경우와 왼쪽 sibling(sibling 0)을 보는 경우가 있다. 노드가 부족하고 (사이즈보다 크고), 오른쪽 sibling 노드가 꽉차거나 비었을 경우 moveLeftOneLeaf라는 함수를 통해 한 칸 왼쪽으로 옮겨 준다. 오른쪽으로 노드를 옮길 수 없기 때문에 왼쪽으로 가는 것이다. 반대로 왼쪽 sibling 노드가 꽉차거나 비었을 경우에는 moveRightOneLeaf함수를 사용한다.

또 다른 경우는 sibling이 없는 경우인데, 이때는 merge 작업이 필요하기 때문에 mergeTwoLeafNode로 이동한다(왼쪽 오른쪽에 따라 1,2로 나뉨)

리프 노드가 아닐 경우에도 같은 방식으로 수행한다.

SearchSingle, searchRange : single search와 range search 수행

readCSV : csv파일을 읽는 함수로 ArrayList를 이용했으며 file I/O 작업을 수행한다

irregularFormat : args, 즉 커맨드 명령어의 길이가 맞지 않을 경우 에러 메시지를 출력한다

Show : index.dat에 결과값 출력

Main: args[0]에 따른 작업 수행(create, insert, delete, search)

1. Node.java

Node : 두 노드 생성자는 리프와 논리프로 나누어진다. Int n만 받는 생성자는 리프노드의 초기 세팅을 해준다. int n, int [] inputKeys, int [] inputValues를 받는 생성자는 키와 value를 가지고 리프노드를 설정해 준다. int n, int [] inputKeys, Node [] nodes를 받는 생성자는 논리프노드이다. 논리프노드는 value를 가지고 있지 않다.

InsertToNode : 마찬가지로 리프, 논리프 각각 다른 생성자를 가진다. for문을 돌면서 키값을 하나씩 밀어 준다. 논리프의 경우 deepleftMostKey 함수로 이동한다.

insertAndSplit : insert 후 split작업을 수행하는 함수이다. 계산을 통해 해당하는 위치의 노드의 키와 포인터 값을 변경시켜 주고, 새 노드를 만들며 sibling값도 재설정해 준다.

insertAndSplitRoot : split작업을 수행하는데 이 경우는 root일때를 나누어 주었다. 루트를 split하는 경우는 다른 노드와 다르게 트리의 높이가 변화할 수 있기 때문이다.

findPath, findPathInt : 경로를 찾으면서 포인터, 키 값을 반환한다.

isthereKeyInNode : 노드 안에 키가 있는지 확인

leftMostKey : 가장 왼쪽 키 리턴

deleteInLeaf : 키를 찾으면 key와 value를 하나씩 왼쪽으로 옮긴다. 값을 지우기 때문에 배열에 배치된 순서가 바뀌기 때문이다.

IsFull, IsLack, isEnough : 노드가 꽉찼을때, 비었을때, n/2를 채웠을때 ( b+tree의 조건)를 확인해주는 함수

moveLeftOneLeaf, moveRightOneLeaf : 리프일 경우 왼쪽, 혹은 오른쪽으로 하나씩 옮겨준다(key, value를) children에도 변화가 일어난다.

mergeTwoLeafNode, mergeTwoLeafNode2 : merge를 하는 함수로 children의 수에 변화가 일어난다.

IsEmpty : 노드가 비었는지 확인

checkAllKey, checkKey, deepleftMostKey : 재귀함수이며 leftmost key확인

deleteNode : 노드를 지움. 배열에서 index를 감소시켜준다.

moveRightOneNonLeaf, moveLeftOneNonLeaf : 오른쪽, 왼쪽으로 한칸씩 이동

mergeTwoNonLeafNode, mergeTwoNonLeafNode2 : 논리프의 merge로 위에서 sibling[0], [1] 즉 왼쪽과 오른쪽에 따라 구분했다. 합쳐지기 때문에 자식 수가 증가하고 포인터도 다시 설정해 주어야 한다.

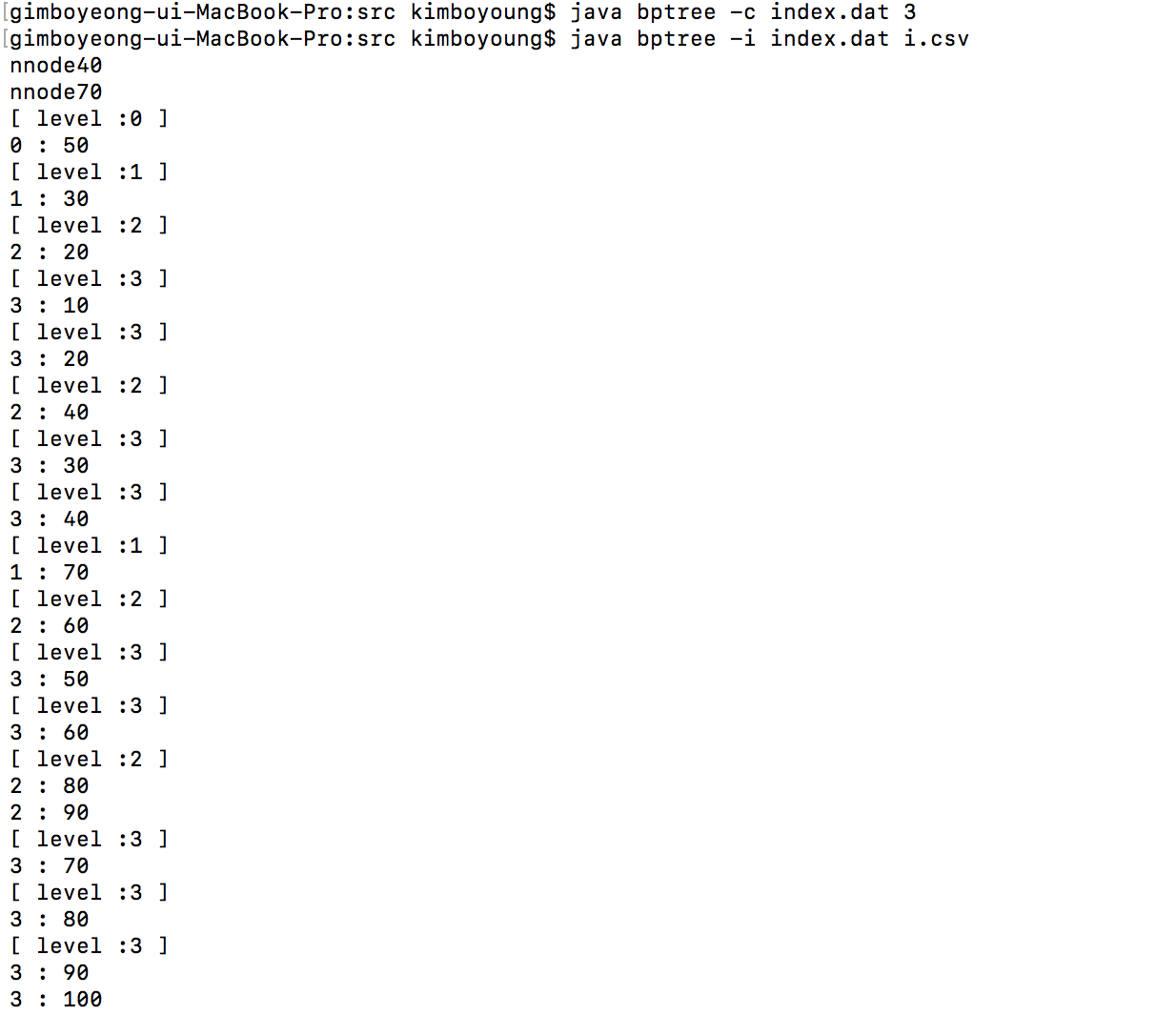
searchPath : 키가 없으면 not found를 출력하고, 있는 경우 findPath를 통해 해당 키를 출력한다.

Range search : 범위 안에서 키 값을 출력한다.

1. 컴파일 방법

1) javac으로 실행전 세팅

2) create

3) insert : level별로 출력. root는 0레벨

4) delete

5) Single search

6) Range search

