<B+ tree 보고서>

한양대학교 컴퓨터소프트웨어학부
2019061721 최성훈

1. 알고리즘 정리

B+ tree 알고리즘을 구현했다. Main 클래스에 있는 main문을 보면 흐름을 이해할 수 있을 것이다. 구현 조건 중 하나인 java bpTree "-c, -i, -d, -s, -r" 5개 중 1개를 입력 받아서 args[0]에, index.dat라는 파일명을 args[1]에, 그리고 각 함수별로 필요한 추가적인 내용 (-c 일 경우, degree의 크기, -i일 경우, input.csv라는 파일명 등)을 args[2]에 저장하였다.

1) java bpTree -c ...
이 경우 index.dat 파일을 새롭게 생성하는 단계이다. args[1]의 내용일 index.dat 파일을 새로 만들거나 현존되어 있으면 덮어 쓰우기를 한다. index.dat 파일 내에는 args[2]에 저장되어 있는 degree를 저장한다.

2) java bpTree -i ...
이 경우 input.csv 파일에 있는 key와 value들을 B+ tree에 삽입을 하게 된다. 따라서 input.csv 파일의 내용들을 입력받아 string 내의 함수인 split을 통해서 ,을 기준으로 key와 value를 구별하고 read를 통해 다음 줄이 없을 때 까지 반복해서 나오는 key와 value를 B+ tree에 삽입하게 된다. 이후 index.dat 파일에 B+ tree에 관련된 내용을 저장한다.

3) java bpTree -d ...
이 경우 delete.csv 파일에 있는 key를 받아와 index.dat에 구현되어 있는 B+ tree를 보고 B+ tree내에 지우고자 하는 key가 있으면 key들을 삭제하고, 다시 index.dat 파일에 B+ tree에 관련된 내용을 저장한다.

4) java bpTree -s ...
이 경우 index.dat라는 파일을 통해 저장되어 있는 B+ tree 내용을 불러온 후, args[2]에 들어온 key 값을 single search 하게 된다. 이때 그 key 값이 B+ tree에 존재하지 않는다면, "NOT FOUND"를 출력하고, key 값이 B+ tree에 현존한다면 어떤 parents key를 통해서 찾게 되었는지 출력하고 그 key 값의 value를 출력한다.

5) java bpTree -r ...
이 경우 index.dat라는 파일을 통해 저장되어 있는 B+ tree 내용을 불러온 후, args[2]와 args[3]에 들어온 key 값을 range search 하게 된다. index.dat 파일 내에 있는 B+ tree를 참고에 key 값이 args[2] 이상이고 args[3] 이하인 것을 찾아, <key, value> 형식으로 출력한다.

6) File Save & Load
정말 여러 가지 방법을 시도해보았고, 어떻게하면 최대한 빠른 시간내에 처리할 수 있을지에 대해서 생각해보고 구현을 해봤으나 실패하여 결국에는 leafnode에 있는 모든 key와 value

값들만 저장을 하고 그 저장된 값을 다시 불러서 insert를 한 후 이후 단계 (Search나 delete)를 실행하도록 했다.

2. 함수 및 클래스 설명

* 함수 내에 나오는 V나 K는 value와 key의 type으로 이번 과제 내에서는 실수인 int라고 가정한다. (이는 main에서 생성할 때 정해준다.)

1) bptree

(1) 클래스 설명

B+ Tree을 구현하는데 있어 가장 큰 클래스로 bptree의 node_size를 저장하고 있고, root Node를 저장하고 있다.

(2) 함수 설명

public bptree (node_size) : bptree 생성자로 node_size를 저장하고, B+ tree의 Leaf_Node 타입인 root도 함께 생성하는 bptree를 생성하는 생성자

public V serach (K key) : root에서 SingleSearch를 할 수 있는 함수

public HashMap<K,V> searchRange(K key1, K key2) : root에서 RangeSearch를 할 수 있는 함수

public void insert(K key, V value) : root에서 Key 값과 valus를 인자로 받아 B+ tree에 삽입하는 함수

public void delete(K key) : root에서 Key 값을 입력받아 그 key와 같은 값이 B+ tree안에 있다면, key 값과 value값을 삭제한다.

public void FileSave(String filename, bptree BPlusTree) : 입력받은 파일명에 bptree와 관련된 내용을 넣는다. 첫 번째 줄에는 degree의 값이 들어가고 이후에는 leaf node에 있는 key와 value의 값이 차례대로 들어간다.

public void FileLoad(String filename, bptree BPlusTree) : 입력받은 파일명의 파일을 통해 bptree와 관련된 내용을 입력받는다. 첫 번째 줄에 있는 degree의 값을 입력받아 bptree(degree)를 통해 bptree를 생성하고, 이후 key와 value 값을 차례로 받아 삽입해준다.

2) Node 내 함수

* Node 클래스가 abstract 클래스이고, 모든 함수가 abstract이기 때문에 자세히 설명하지는 않겠다.

(1) 클래스 설명

Node (Leaf_Node와 Inter_Node를) 구현하기 위한 abstract 클래스로 여러 개의 abstract

함수를 보유하고 있다. 또한 K (이번 과제에서는 int) 타입의 keys라는 배열을 가지고 있다. 이는 각 node의 키 값들을 저장하는 배열이 될 것이다.

2) 함수 설명
int _max_key() : Keys 리스트에 몇 개의 key가 들어있는지 반환하는 함수이다.

abstract V find_V(K key) : SingleSearch를 실행하는 함수
abstract void delete_v(K key) : 삭제를 실행하는 함수
abstract void insert_v(K key, V value) : key 값과 value 값을 삽입하는 함수
abstract void K first_leaf_key() : leaf node에서 첫 번째 children 값을 구하는 함수
abstract HashMap<K,V> get_Range(K key1, K key2) : 두 개의 key 값들을 입력받아 RangeSearch를 실행하는 함수
abstract void merge (Node s) : B+ tree를 구현하고 삭제할 때 2개의 노드가 하나로 합쳐져야할 때 사용하는 함수
abstract Node split() : B+ tree를 구현하고 삽입을 할 때 1개의 노드가 두 개로 나눠져야할 때 사용하는 함수
abstract boolean flag_over() : 노드가 overflow인지 확인하는데 사용하는 함수
abstract boolean flag_under() : 노드가 underflow인지 확인하는데 사용하는 함수

3) Inter_Node 내 함수
(1) 클래스 설명
Inter_Node 즉, Non_Leaf_Node를 구현하기 위한 클래스이다. 이 클래스에는 children을 node 타입의 배열로 가지고 있는데, 이는 Inter_Node의 자식들을 배열로 저장하기 위함이다.

(2) 함수 설명
Inter_Node() : Inter_Node의 생성자, keys와 children를 arraylist로 가지고 있다.

Node getChild(K key) : Collections에 내장되어 있는 binarySearch를 통해서 loc (위치)를 찾고, loc이 0 이상이면 loc+1을, 0 미만이면 -loc-1의 위치의 children의 값을 반환한다. 즉, key를 통해 위치를 찾고 그 위치를 통해 그 key에 알맞은 child를 반환한다.

void delete_c(K key) : Collections에 내장되어 있는 binarySearch를 통해서 loc을 찾고, 그 loc 위치에 있는 key 값과 child값을 각각 keys와 children 배열에서 삭제한다.

void insert_c(K key, Node child) : Collections에 내장되어 있는 binarySearch를 통해서 loc을 찾고, loc이 0 이상이면 loc+1을, 0 미만이면 -loc -1 값 저장하고, 그 값이 음수이면 저장한 위치에 key와 저장한 위치 + 1에 child를 각각 keys와 children 배열에 삽입한다. 또한 저장한 값이 양수이면, Key 값이 있다는 이야기로 받아드리고 value 값만 변경한다.

Node left_s(K key) : key 값을 기준으로 왼쪽 children 노드를 반환한다.

Node right_s(K key) : key 값을 기준으로 오른쪽 children 노드를 반환한다.

Override 함수들

V find_V(K key) : 그 노드에서 keys 배열에 있는 모든 key를 출력하고, 찾고자 하는 key에 알맞은 자식노드를 찾아서 find_V(key)를 통해 원하는 키의 value 값을 찾는다.

void delete_v(K key) : Child 노드 내에서 key 값을 찾아 삭제한다. 다만 그 노드의 children이 underflow (즉, flag_under가 true)일 때에는 왼쪽 자식노드와 오른쪽 자식노드를 합친다. 또한 본인 노드에서 지우는 키가 포함되어 있다면 그 키와 value도 지운다.

insert_v(K key, V value) : Child 노드 내에서 key 값이 들어갈 공간을 찾아, 입력 받은 key 값과 value 값을 삽입한다. 이 때, child 노드가 overflow (즉, flag_over가 true)일 때에는 child노드를 spilt 함수를 이용하여 나눠주고, parents 노드에 왼쪽 child 노드의 맨 오른쪽 key의 값을 삽입해 준다. 이 후 root가 undeflow 일 경우에는 새로운 노드를 만들어 그 안에 두 노드 안에 있는 모든 child값을 children 배열에 넣고, root를 새로운 노드로 바꾸어 준다.

K first_leaf_key : children 배열에서의 가장 왼쪽 즉, 첫 번째 child 노드에서 first_leaf_key를 실행해 나오는 key 값을 구한다.

HashMap<K,V> get_Range(K key1, K key2) : key1 값이 있는 곳의 children 배열을 찾아 거기서부터 key2까지 get_Range(key1, key2)를 실행한다.

void merge(Node s) : 입력 받은 Node s에 있는 모든 키 값과, children 값을 this 즉 현재 노드에 더 넣어서 합쳐 준다.

Node split() : 새로운 노드에 들어갈 key의 index는 원래 있던 key 배열의 _max_key() / 2 + 1 (start) 이고 마지막 index는 _max_key (end) 일 것이다. 이때 새로운 노드를 만들어 start부터 end의 값을 새로운 노드의 keys 배열에 넣고, start부터 end + 1의 child를 새로운 노드의 children 배열에 넣는다. 그리고 원래 노드에 있던 keys와 children들을 삭제해준다.

bolean flag_over() : children 배열의 크기보다 node_size가 작으면 참을 반환하다,

boolean flag_under() : children 배열의 크기보다 (node_size + 1) / 2가 크면 참을 반환한다.

4) Leaf_Node 내 함수

(1) 클래스 설명

Leaf Node를 구현하기 위해 필요한 함수들과 V (이번 과제에서는 int) 타입의 values 배열과 Leaf_Node 타입의 next를 가지고 있다. values 배열은 value 값들을 저장하는 배열이 될 것

이며, next는 오른쪽 형제 노드를 가르키는 LeafNode를 가지고 있을 것이다.

(2) 함수 설명
Leaf_Node() : Leaf_Node를 생성하는 생성자로 keys는 K 타입의 arraylist를 values는 V 타임의 arraylist를 만든다.

Override 함수들
V find_V(K key) : Collections에 있는 binarySearch를 이용하여 위치를 찾고, 그 위치의 값을 loc라는 변수에 저장한다. 만약 위치의 값이 음수라면 찾을 수 있는 값이 없는 것임으로 null을 반환한다.

void delete_v(K key) : Collections에 있는 binarySearch를 이용하여 위치를 찾고, 그 위치의 값을 loc라는 변수에 저장한다. 만약 위치의 값이 양수라면, keys 배열과 values 배열에서 index (loc) 위치의 내용을 삭제한다.

insert_v(K key, V value) : Collections에 있는 binarySearch를 이용하여 위치를 찾고, 그 위치에 key와 value값을 삽입한다. 그때, root의 노드가 overflow일 경우에 새로운 노드를 만들어 그 노드에 원래 노드의 오른쪽 반의 키와 value 값을 저장한다. 그리고 그 오른쪽 노드의 첫 번째 값을 위에 부모 노드에 삽입하여 준다.

K first_leaf_key : keys 배열에서 첫 번째 (index 0)의 키를 반환한다.

HashMap<K,V> get_Range(K key1, K key2) : Hashmap을 이용하여 rangesearch를 구현한다. 원래는 hashmap에 다 모든 값을 넣고 출력하려고 했으나, 그 경우 순서대로 나오지 않아 바로 출력하는 형태를 취하였다. iterator를 사용하여 key 값을 비교하고 key값이 key1과 key2 사이에 있다면 iterator의 위치를 바꿔가면서 key와 value의 값을 출력한다.

void merge(Node s) : 노드가 underflow일 때를 대비해, merge 함수를 구현한다. 입력받은 node에 있는 모든 key들과 value의 값들은 현재 노드에 다 넣어주고, 현재 노드의 next를 입력받은 node의 next로 넘겨준다.

Node split() : 노드가 overflow일 때를 대비해, split 함수를 구현한다. 새로운 Leaf_Node 타입의 변수를 만들어 원래 노드의 (_max_key() + 1) / 2번째부터 _max_key번째까지의 데이터 (key와 value 값)들을 새로운 노드의 첫 번째부터 넣는다. 그리고 원래 노드의 (_max_key() + 1) / 2번째부터 _max_key번째까지의 데이터를 삭제한다.

bolean flag_over() : values 배열의 크기보다 node_size -1 이 작으면 참을 반환한다.

boolean flag_under() : values 배열의 크기보다 node_sizze /2가 크면 참을 반환한다.

5) Main 내 함수

새로 정의한 함수는 없고, main내에서는 이미 설명한 함수들을 이용해 실행한다.

3. 실행 방식

먼저 실행 파일이 있는 폴더 전 (즉, src)의 폴더에 input.csv, delete.csv, input.dat 파일이 필요하다면 생성하고 그 파일에 내용을 넣어 저장한다.

이후 그 폴더 내에서 javac ./bpTree/*.java를 통해 컴파일 해준다.
main 문이 main.java 안에 있기 때문에, 실행시 java main ("-i", "-c", "-s", "-d", "-r")와 그 구문에 알맞은 값들 (index.dat, degree의 크기, input.csv, delete.csv, 찾고자 하는 키의 값, rangeSearch시 찾고자 하는 키의 첫 번째와 마지막 값)을 넣어준다. 이 때 가장 중요한 것은 그 어떠한 실행보다 java main -c index.dat (degree의 크기)를 제일 먼저 실행해줘야한다.

```
C:₩Users₩Sunghun₩Desktop₩Sunghun₩Hanyang₩Grade 2₩2 semester₩데이터베이스시스템및응용₩2020_ite2038_2019061721₩B-tree_Assignment₩Source>javac ./bptree/*.java
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:₩Users₩Sunghun₩Desktop₩Sunghun₩Hanyang₩Grade 2₩2 semester₩데이터베이스시스템및응용₩2020_ite2038_2019061721₩B-tree_Assignment₩Source>java bptree.main -c index.dat 3

C:₩Users₩Sunghun₩Desktop₩Sunghun₩Hanyang₩Grade 2₩2 semester₩데이터베이스시스템및응용₩2020_ite2038_2019061721₩B-tree_Assignment₩Source>java bptree.main -i index.dat input.csv

C:₩Users₩Sunghun₩Desktop₩Sunghun₩Hanyang₩Grade 2₩2 semester₩데이터베이스시스템및응용₩2020_ite2038_2019061721₩B-tree_Assignment₩Source>java bptree.main -s index.dat 9
37
20
10

C:₩Users₩Sunghun₩Desktop₩Sunghun₩Hanyang₩Grade 2₩2 semester₩데이터베이스시스템및응용₩2020_ite2038_2019061721₩B-tree_Assignment₩Source>java bptree.main -r index.dat 10 40
10,6
20,9
26,5
37,13

C:₩Users₩Sunghun₩Desktop₩Sunghun₩Hanyang₩Grade 2₩2 semester₩데이터베이스시스템및응용₩2020_ite2038_2019061721₩B-tree_Assignment₩Source>java bptree.main -d index.dat delete.csv
```

<실행 예시>