김상욱 교수님

한양대학교 컴퓨터소프트웨어학부

이영석, 2016025732

2019.10.02

Database Systems

B Plus Tree

본 코드는 hash나 treemap을 사용하지 않고 일일이 구현한 코드입니다.

1. Overview

```
Non-leaf node
      m: # of keys
       p: an array of b <key, left_child_node> pairs
       r: a pointer to the rightmost child node
Leaf node
       m: # of keys
       p: an array of b <key, value(or pointer to the value)> pairs
       r: a pointer to the right sibling node
```

Node의 변수로 m(int), p(list), r(int) 을 설정하였습니다.

p라는 list안에 리프노드인 경우에는 [key, value], 논리프노드인 경우 [key, pointer] 를 페어 형태로 집어넣어 구현하였습니다.

(참고) `# key : p[n][0] , value || pointer : p[n][1]`

2. 코드 구성

<Class1. Node>

is_full : p의 길이가 m과 같을 때, overflow를 감지하는 함수

add(key, value) : p의 적당한 자리에 [key, value], 혹은 [key, pointer]를 넣는 함수

split : overflow가 일어났을 때 right노드를 만들고, leaf와 nonleaf, parent의 유무를 구분하여 절반값을 right노드로 이동시켜주는 함수

show : nonleaf일때는 p에 있는 key값만, leaf일때는 key와 value값을 print해주는 함수

is_underflow : 현재 노드가 underflow인지를 감지해주는 함수

sibling_underflow : 형제노드에게서 빌렸을 때 형제노드가 언더플로가 나는지를 감지해주는 함수. 또한 현재 노드에서 하나를 빼도 되는지 여부를 확인하는 데에도 쓰임.

index_of_the_node : 현재 노드가 parent의 p에서 어떤 index를 차지하고 있는지를 return해주는 함수

which_sibling : delete구현 시에, 왼쪽과 오른쪽 노드중 어떤 노드에게서 빌릴지를 return값을 통해 확인 가능하고, borrow를 할지 merge를 할 지에 대해서도 tuple형태로 알려주는 함수

borrow(tuple) : which_sibling에서 받은 tuple에서 왼쪽에서 borrow를 할 지 오른쪽에서 borrow를 할지를 구분하고, leaf노드인지 아닌지를 확인하여 각각에 맞는 빌리는 과정을 구현한 함수

merge(tuple) : borrow와 마찬가지로 왼쪽과 merge할 지, 오른쪽과 merge할 지 구분하고, leaf인지 아닌지를 판단하여 각각에 맞는 병합하는 과정을 구현한 함수.

<Class2. BplusTree>

insert(key,some) : 아래의 find_node 함수를 통해 찾은 node에 add해주는 함수 여기서 some은 value나 pointer를 의미합니다

delete(key) : find_node 함수를 통해 찾은 적당한 위치의 노드에 key값을 indexing해주고, 언더플로가 안일어날경우 와 일어날 경우로 나누어 각각의 deletion구현하는 함수.

만약 언더플로가 일어난다면, while문을 통해 underflow가 안일어나는 부분까지 borrow나 merge과정을 수행합니다.

find : 본인이 찾고싶은 key를 포함 '할것 같은' 노드를 리턴해주는 함수. 다만 중간에 show를 포함하여 path를 나타냅니다.

find_node : find함수와 같지만 show를 사용하지 않아 리턴되는 노드값만 사용하게 만든 함수

search : find함수를 통해 path를 출력하고, 그 노드에서 키가 있는지를 확인해주는 함수. 만약 없다면 'NOT FOUND'를 출력합니다.

range_search(key1, key2) : key1과 key2가 포함된 노드를 find_node함수를 통해 정한후,

leaf의 r과 show함수를 사용하여 계속해서 그 노드의 key와 value를 출력하다가, key2가 출력되면 끝나는 함수

in_this_tree(key) : 트리안에 key가 있는지를 확인해주는 함수. 실질적으로 사용x

show_all : 루트부터 leaf까지를 DFS와 비슷하게 순회하며 트리의 모든 구조를 볼 수 있게 출력하는 함수

3. input, delete.csv 생성

	A	B
1	108	1019
2	76	402
3	56	2996
4	13	1102
5	10	504
6	18	2599
7	57	1642
8	52	2143
9	61	2950
10	111	2824
11	15	1641
12	116	1491
13	27	89
14	30	639
15	54	1418
16	128	2097
17	106	2215
18	84	1469
19	121	780
20	126	2594
21	114	943
22	36	2839
23	67	1161
24	44	1619
25	2	49
26	73	95

<input.csv파일 일부>

input.csv : 1부터 129까지의 난수를 생성하여 대입하였습니다.

delet.csv : 1부터 129의 숫자중 임의의 숫자들을 대입하였습니다.

4. 컴파일 실행 화면

```
C:₩Users₩O|영석₩Desktop₩ProjectSeries₩BplusTree>py BplusTree.py -c index.dat 4

C:₩Users₩O|영석₩Desktop₩ProjectSeries₩BplusTree>py BplusTree.py -i index.dat input.csv

C:₩Users₩O|영석₩Desktop₩ProjectSeries₩BplusTree>py BplusTree.py -s index.dat 8
76
30 52
10 18
3 5 8
8 ,  2351

C:₩Users₩O|영석₩Desktop₩ProjectSeries₩BplusTree>py BplusTree.py -r index.dat 0 13
1 ,  2720
2 ,  49
3 2091
4 2069
5 320
6 1458
7 2675
8 2351
9 673
10 504
11 1961
12 1102
13 ,  1102

C:₩Users₩O|영석₩Desktop₩ProjectSeries₩BplusTree>py BplusTree.py -d index.dat delete.csv

C:₩Users₩O|영석₩Desktop₩ProjectSeries₩BplusTree>py BplusTree.py -r index.dat 0 13
9 ,  673
11 1961
12 1102
13 ,  1102
```

cmd파일에서 제대로 작동하는 것을 확인하였습니다.

(참고1 : m = 3 경우에는 merge와 borrow 구현 중 구분 조건을 몇가지 놓쳐 에러가 나는 것을 확인하였습니다. 따라서 insert경우에는 문제없이 실행이 되나, delete경우에는 m=3인 경우를 제외할 것을 알려드립니다.)