1. Basic Concept To B+ tree

1) 각 key, value들은 leaf node에 있다. B-tree는 leaf node가 아닌 각자의 key마다 value를 가지지만, B+tree는 leaf node에 모든 value값을 가진다.
2) 각 leaf node들은 linked list의 형태를 띈다
3) leaf node의 parent key는 leaf node의 첫 번째 key값보다 작거나 같다.
4) Node는 최대 order개 부터 order/2개 만큼의 자식을 가질 수 있다.
5) Node에는 최대 order-1개 부터 [order/2]-1 개의 키를 가질 수 있다

2. How to process the each command

2-1) INIT

이미 구현되어있지만, 살펴보면 INIT 커맨드 뒤에오는 숫자를 int형으로 변환시켜 그 숫자의 order만큼 B_PLUS_TREE를 생성하는 커맨드이다.

2-2) INSERT

INSERT 커맨드의 경우에는 아래의 세 함수를 사용해 구현하였다

```
def insert_key (self ,key ,node):
```

이 함수는 B_PLUS_TREE 클래스에 있는 함수이며, 새로 들어가야할 key와 node를 반환하는 함수이다.
새로 들어가야할 key의 위치를 찾아 넣고, 이 과정에서 leaf node가 order를 넘었다면 split 해주어야한다.
새로 들어가야할 key의 위치는 biscet.bisect() 함수를 사용해 들어갈 노드에 위치할 인덱스를 반환해주었고, 그 인덱스 사이에[idx:idx]에 key값을 넣고, leaf node에는 자기 자신이 포함되어야 하므로 [idx+1:idx+1]위치에 넣었다.
이후 order이 넘으면 Split()함수를 실행시킨다.
node가 leaf node이면 위의 과정을 바로 실행해도 되지만, 그렇지 않을 경우, 순회를 통해 child node에서 들어갈 위치를 찾아 위의 과정을 실행시킨다.

```
def Split (self):
```

이 함수는 insert된 key의 leaf node가 order이 넘었으면 split 해주어야 한다는 것이다. 함수를 적절히 자르고 중간에 있는 key와 새로운 node를 반환해준다.
이 함수는 Node에 관련되어있으므로 Node 클래스에 작성하였다.
Split함수의 컨셉은 간단하다. keys의 절반을 중심으로 왼쪽은 기존의 keys에, 오른쪽은 자기 자신을 포함하여 새로운 child node로 만들어 넣는다.

```
def insert (self ,value):
```

위의 함수들은 key값과 새로생긴 node를 반환한다고 하였다.
따라서 이 함수는 새로운 함수를 만들고, child에 node를 넣음과 함께, 새로운 key까지 넣어주는 과정을 포함한다.

2-3) DELETE

우선 삭제의 경우에는 능력이 안되어 구현하지 못했다.
모든 Command를 구현하는데에 약 6시간정도 소요되었다면, DELETE 하나에만 3일의 시간을 사용했는데 결국 어떻게 해도 잘 안되어 포기하였다...(test_bp.txt 파일의 경우에는 성공할 수 있도록, 즉 부모노드에 하나의 key만 있을 때만 고려했다)

아래는 DELETE 알고리즘의 개요이다.

(구글링으로 찾았다. 이걸 알아도 못하는 내가 참 한심하다)

1) Start at the root and go up to leaf node containing the key K
2) Find the node n on the path from the root to the leaf node containing K
    A. If n is root, remove K
        a. if root has more than one key, done
        b. if root has only K
            i) if any of its child nodes can lend a node
                Borrow key from the child and adjust child links
            ii) Otherwise merge the children nodes. It will be a new root
        c. If n is an internal node, remove K
            i) If n has at least ceil(m/2) keys, done!
            ii) If n has less than ceil(m/2) keys,
                If a sibling can lend a key,
                Borrow key from the sibling and adjust keys in n and the parent node
                    Adjust child links
                Else
                    Merge n with its sibling
                    Adjust child links
        d. If n is a leaf node, remove K
            i) If n has at least ceil(M/2) elements, done!
                In case the smallest key is deleted, push up the next key
            ii) If n has less than ceil(m/2) elements
            If the sibling can lend a key
                Borrow key from a sibling and adjust keys in n and its parent node
            Else
                Merge n and its sibling
                Adjust keys in the parent node

```
def isUnderflow (self )->bool :
def isRoot (self )->bool :
def _find (self ,key ):
def delete (self ,key ):
def search (self ,key ):
```

이 5개의 함수는 결국 사용되지 못했다..

(사실 추가로 4개의 함수를 더 만들었다가 말도안되는 것 같아서 삭제했다..)

2-4) ROOT & PRINT

ROOT 커맨드의 경우에는 그저 self.root.keys 의 key들을 전부 출력해준다.

PRINT 커맨드가 의외로 애먹었는데, self.root의 list들을 level 1 (맨 윗줄의 트리)로 잡고, 시작한다.

그 이후 level 1 tree를 순회하는데, 이 과정에서 child node가 있으면(+Leaf node 이라면) 그 child 노드또한 순회하며 key값들을 찾도록 했다.(여기서 그냥하면 안되기에, iterable 하게하는 extend() 함수가 있다는 것을 알았다.)

그냥 출력하는것이면 간단했는데, 교수님이 주신 Format에 맞추어 출력을 하려다보니 조금 힘들었다.

2-5) FIND

FIND 커맨드는 아래 두 개의 함수가 필요했다.

```
def find_node (self ,key ,node ):
def find (self ,key ):
```

위의 함수는 tree 안에 key가 있으면, 다시 그 key값을 반환해준다

사실 위의 함수는 아래와 같이 구현하려고 하였다.

```
    node =self.root
```

```
    while not isinstance (node , Node):
      node, index =self._find (node , key)
    for i, item in enumerate (node .keys):
      if key == item:
        return node.values [i]
    return None
```
하지만 왠지 잘 안되어 새로운 방식으로, child node를 전부 뒤져서(재귀함수를 사용) key값을 찾도록 하였다.

이후 FIND는 그 반환된 key를 사용해 직접 PRINT해주는 함수인데 2-4의 print_tree() 함수에서

```
if k in node.keys:
```
이 조건 하나만 추가해 그 key값이 들어있는 노드만 출력해주도록 하였다.

2-6) RANGE

이것 또한 print_tree() 함수에서

```
if int (node .keys [i])>= k_from and int (node .keys [i])<= k_to :
```
위의 조건만 추가해주었으며,(크거나 같을 때 && 작거나 같을 때)

Format을 맞추기 위해 조금만 다르게 수정하였다.

3. lessons I learned (과제하며 배운 것)

사실 소프트웨어학과의 모든 과제는 (객프제외) 모두 C언어로 사용하기에, Python은 정말정말 사용하는 방법을 몰랐다. (1~10 이라고 하면 2정도) 기본 문법도 잘 모르는 상태로 파이썬으로 작성하려니 굉장히 곤욕이었고, 그것이 DELETE 커맨드를 구현할 때 정말 여실히 들어난 것 같다.

~~(C언어였으면 포인터로 왔다갔다 거리면서 잘 만들었을 것 같다...)~~

그렇기에 Python에 대한 지식이 굉장히 많이 늘어난 것 같고, B+tree에 대한 컨셉을 다시금 정말 잘 알게될 수 있는 기회가 된 것같다.

(나 빼고 다들 잘 한것같은데 정말 속상하고 속상하다.. 끝까지 잘 하고싶었는데 3일밤, 거의 4일동안 이 과제에 매달리다보니 다른 과제들이 밀려 결국 손놓을 수밖에 없는 내가 너무 답답했다.. 화도 많이나고 그랬지만 그래도 정말 재밌었었던 시간이었다.)

Thank you

report_by 김영훈