스켈레톤 코드 변경 사항

* B\_PLUS\_TREE.init(self, order) 에서 root를 생성할 때 root.isLeaf 의 값을 True로 설정해줌
* Node.location(self, node, key) : node에서 key가 index될 위치를 return
* Node.indexing(self, root) : leaf node에서 delete 발생 시 indexing을 다시 해 줌
* Node.indexMerge(self, order, root) : leaf node를 제외한 index node에서 left borrow, right borrow, merge가 필요한 경우 호출해서 재귀적으로 merge 수행
* Node.delete(self, k, order, root) : node에서 k값 삭제 후 left borrow, right borrow, merge 수행
* Node.insert(self, loc, k) : node의 keys[loc]에 k값 삽입
* Node.split(self, order, root) : oder-1 보다 key length가 길면 재귀적으로 split 수행

1. Key 삽입
2. Root에서 부터 leaf가 나올 때 까지 key가 들어갈 node를 찾아간다.
3. Leaf node에서 크키 순으로 알맞은 곳에 key 삽입
4. Leaf node 의 key length가 order 이상일 경우 split함수 호출
5. 다른 leaf node 생성하고 order/2 초과 번째의 key를 떼어줌. Leaf node 사이에 link도 수행
6. Leaf node 의 order/2 번째의 key를 부모 노드에 추가
7. 추가된 부모 노드도 order 이상일 경우 재귀적으로 위의 과정을 수행
8. Key 삭제
   1. Root에서 부터 leaf가 나올 때 까지 key가 들어간 node를 찾아간다.
   2. Leaf node에서 key 삭제
   3. Node의 length가 order/2 -1 미만일 경우 borrow or merge 수행
      1. prev node 와 부모가 서로 같고 prev node length – 1 이 order/2-1 이상일 경우 borrow left
      2. elif next node와 부모가 서로 같고 next node length – 1 이 order/2-1 이상일 경우 borrow right
      3. elif prev node와 부모가 서로 같을 경우 merge left
      4. elif next node와 부모가 서로 같을 경우 merge right
   4. leaf node 단에서 delete와 merge가 끝났으면 indexing을 해준다.
   5. Leaf node 에서의 merge로 인해 parent key length가 order/2 – 1 미만 일 경우 indexMerge()호출
      1. prev node와 부모가 서로 같을 경우 merge left
      2. elif next node와 부모가 서로 같을 경우 merge right
      3. 위 과정을 재귀적으로 수행