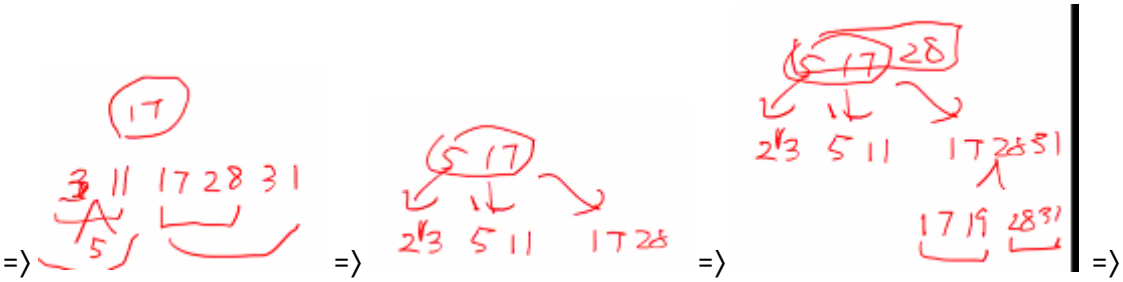
Discussion 05/23

Discussion 11-7

Construct a B+-tree for the following set of key values in order of insertion. Assume that the tree is initially empty and 4 pointers (and 3 key values) fit in one node.

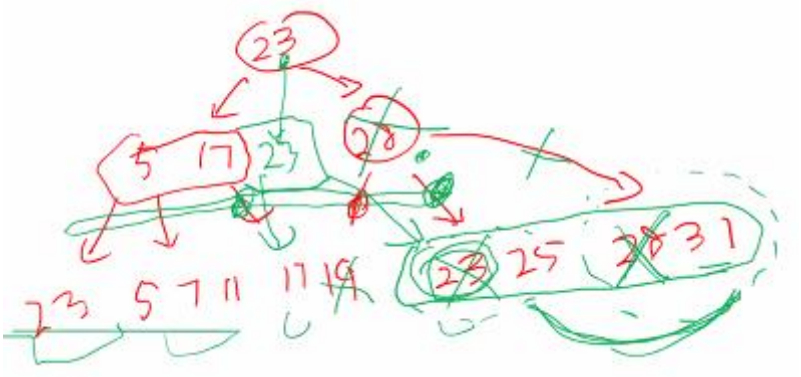
28, 11, 17, 3, 5, 31, 2, 19, 23, 7, 25



Discussion 11-8

28, 31, 19, 23 을 삭제한 뒤의 트리 - 28 을 삭제하면 23 25 31 이 합쳐지고 23 이 위로 올라감. / 31 을 삭제하면 그냥 삭제 / 19 를 삭제하면 5 7 과 11 17 이 분배되고, 상위 노드가 5 11 이 됨. / 23 을 삭제하면 해당 노드가 25 31 이 되고 상위 노드가 25 가 됨.

=>



Discussion 11-9

What is the *complexity of a B+-tree update*? (Insertion and deletion)

Leaf node 까지 찾아가는 데 O(logn)이고 스플릿을 할 경우 두 노드의 value 의 개수가 k 개라 하면 O(k). 루트까지 전파한다 해도 이 과정을 logn 번 반복하니까 O(log_(k/2)n) + O(klog_(k/2)n) 그래서 best 는 O(log_(k/2)n), worst 는 O(klog_(k/2)n).

Delete 도 마찬가지일 듯?

Discussion 11-10

What would the occupancy of each leaf node of a B+-tree be, if index entries are inserted in sorted order? For example, 1, 2, 3, …, 100, 101, …

- Explain why.

맨 오른쪽 노드를 제외하고 모두 반씩 차있을 것. 왜냐면 정렬된 순서대로 들어오면 채워지다가 다 차면 반으로 split 되고가 반복인데 split 된 노드에 value 가 들어올 일이 없음.

Discussion 11-11

Suggest an efficient way of building a B+-tree index for an existing table? (eg., a new index on *phone_number* attribute for the *customer* table that already has 1M records)

phone_number 를 기준으로 정렬한 다음에..?

=〉 정렬한 다음에 꽉꽉 채워서 leaf node 를 만들고 그 leaf node 들로 또 상위 노드를 만들고 이런 식으로 하면 된다고 한다.