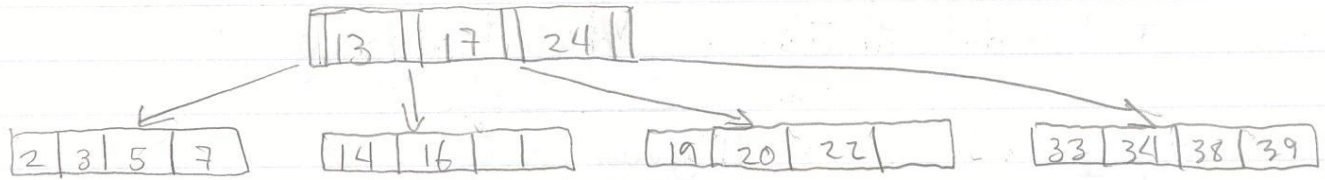
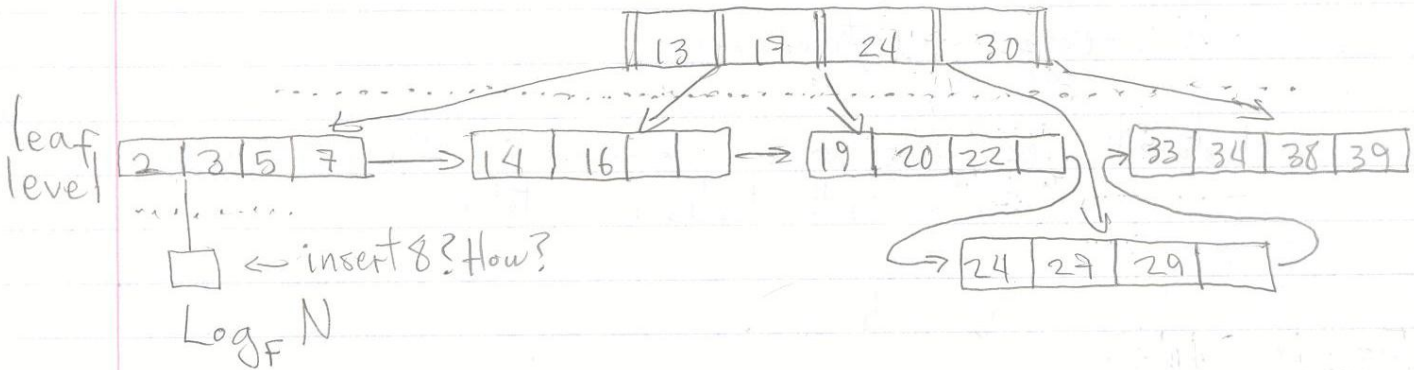


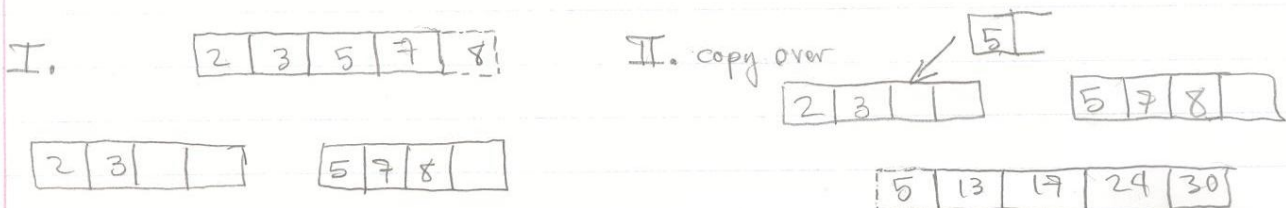
November 17, 2009



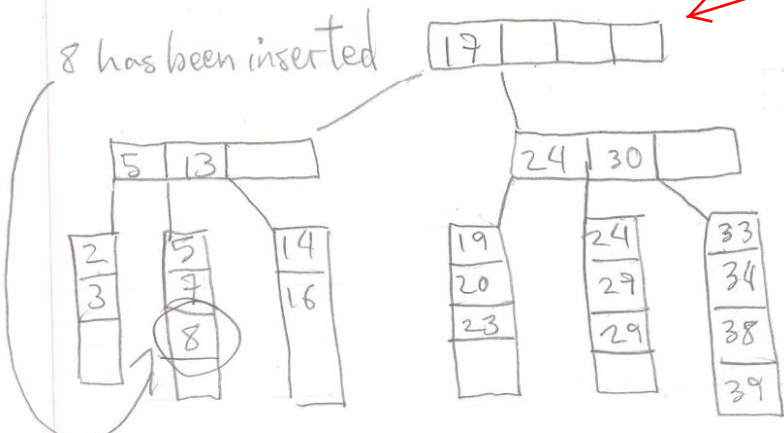
- (1) d-order
each node should have $[d, 2d]$ data entries, the root is an exception
- (2) F-fanout
leaf level: $F \in [d, 2d]$ key values and F pointers to data
non-leaf level: F key values and $F+1$ pointers to $F+1$ child nodes
- (3) leaf nodes are chained by the order of the search key



I. Insertion of B^+ -tree algorithm Insert (K_i)
Find the leaf node where K_i should stay
IF N_i is not full THEN add K_i to N_i
ELSE SPLIT (N_i, K_i)



SPLIT can be propagated higher
 - Intermediate node SPLIT:



instead of copying the middle value 17 to the higher level (as we did in splitting the leaf nodes), we push it up.

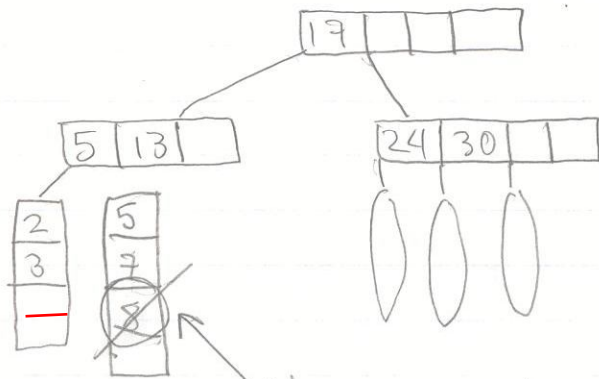
2. Deletion from B⁺-tree

Algorithm Delete (K)

- Find the leaf node N_i where K_i is
- delete K_i from N_i
- IF N_i is at least half full
 THEN EXIT
- ELSE REDISTRIBUTE (N_i)
- IF redistribute was not successful
 THEN MERGE

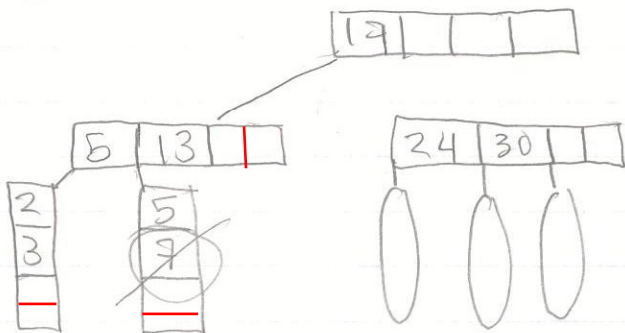
Redistribution: borrowing values from a sibling node. We are missing the example for a redistribution here. See other notes for such details.

Example: delete 8

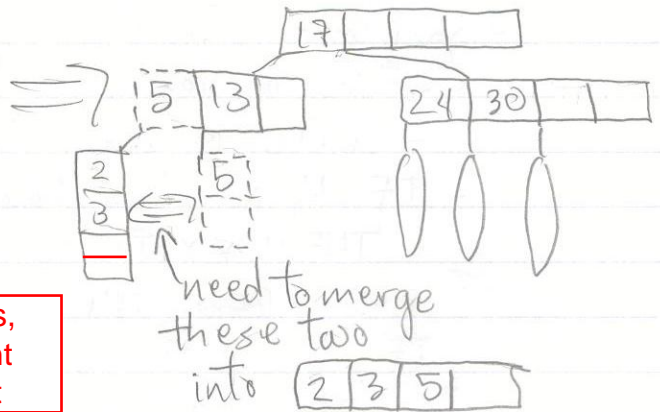


No issues, just erase 8

delete 7

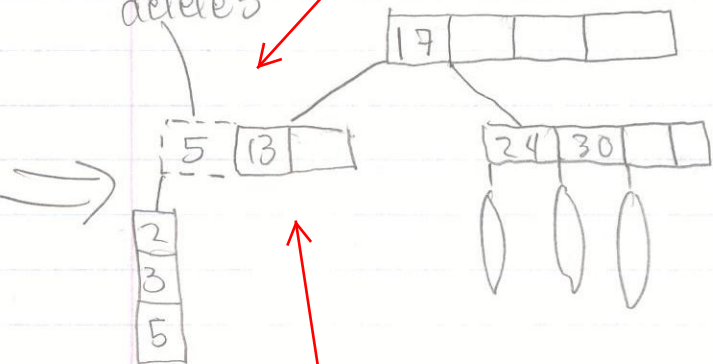


Problem!



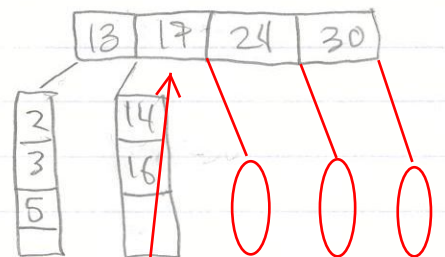
propagate higher deletes

In merging two leaf nodes, we just delete the relevant key value from the parent



After deleting 5, we face a underflow situation in this intermediate node, Redistribution or Merge will have to performed on this level.

Final Result



opposite to the Split operation, when merging two intermediate nodes, we move the value 17 down from the parent, instead of just deleting it from the parent node.