§ Chapter 13.0 2-4 Trees

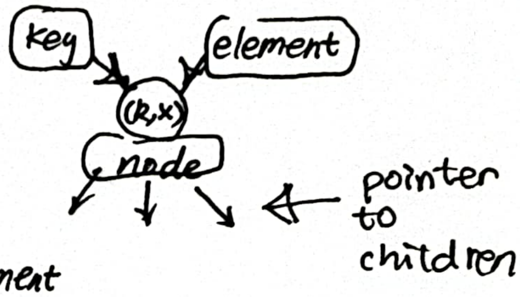
2-4 Trees

1. Multi-searching trees
   - They are search trees, but not necessarily binary.
   - They can have [2, 4] children, ie.

▷ each node

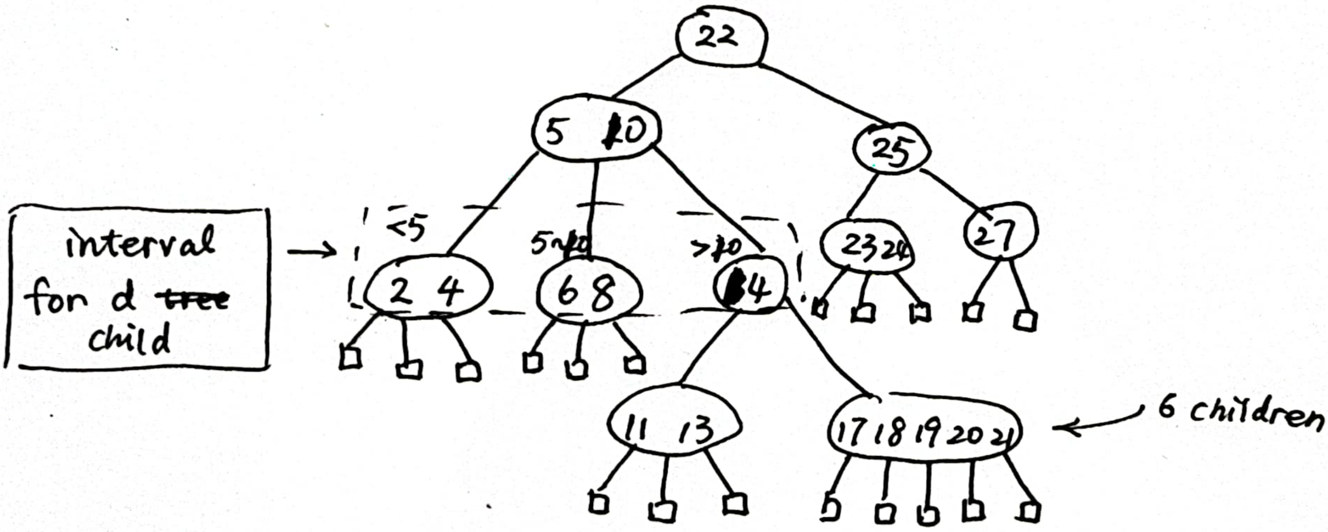
- has at least 2 children
- store a collection of item (k, x).

  key element



- contains $d-1$ items,

  number of children
- has pointer to children



▷ children of each internal node are between items

▷ all keys ~~between~~ in the subtree rooted at the child fall between keys of those items.

The searching procedure

If search key $s < k_1$, search leftmost child

⋮

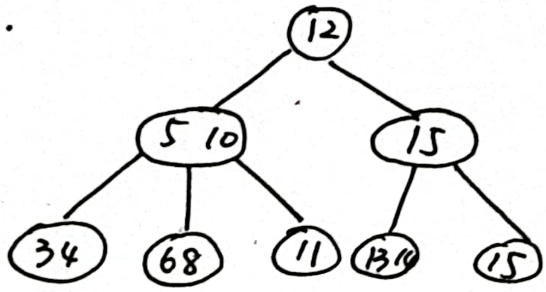
$s > k_{d-1}$, rightmost.

2、(2,4) trees

- Properties: have at most 4 children

  all leaf nodes at the same level

  ⇒ a node has up to 3 keys.

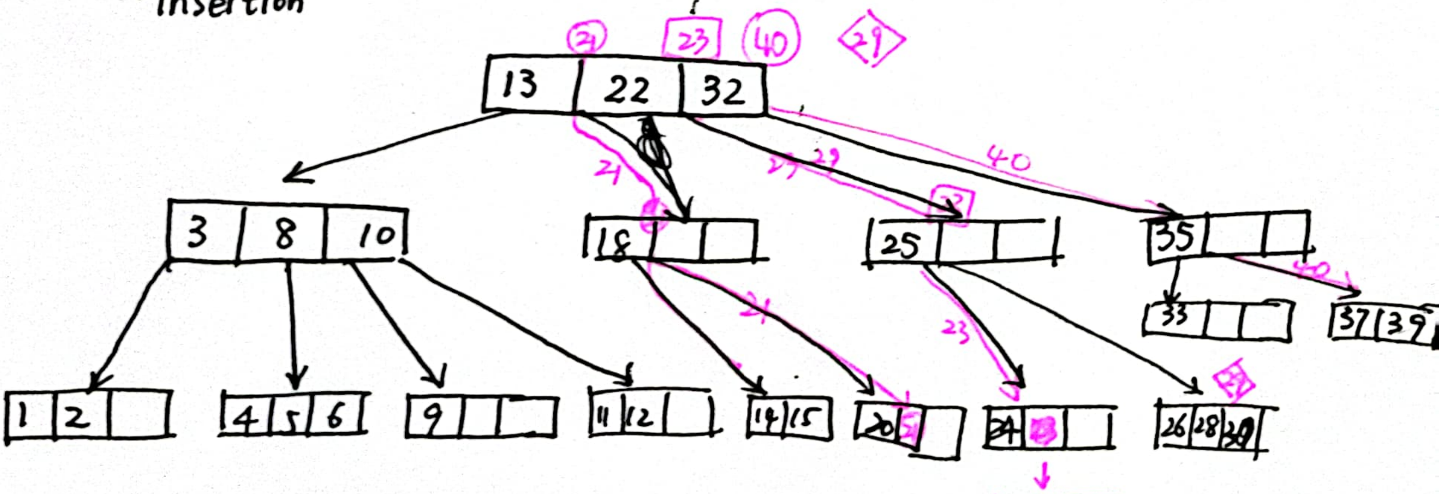


- Height $\log_4 n \le h \le \log_2 n$

- Search process : $3\log n$

  ↑ multiple nodes and multiple comparison

- Insertion

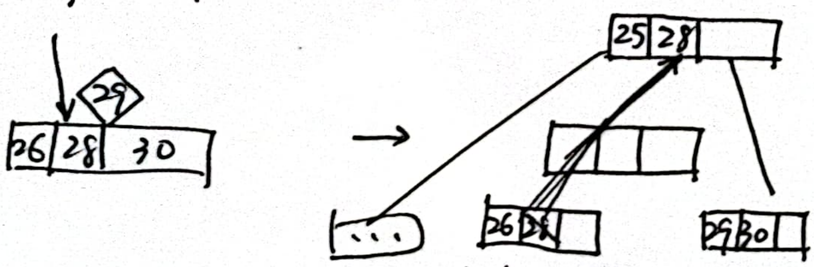


Insert 21 . Insert $\boxed{23}$. ~~Insert 40~~ Insert 40

▫ No problem if there are empty slots

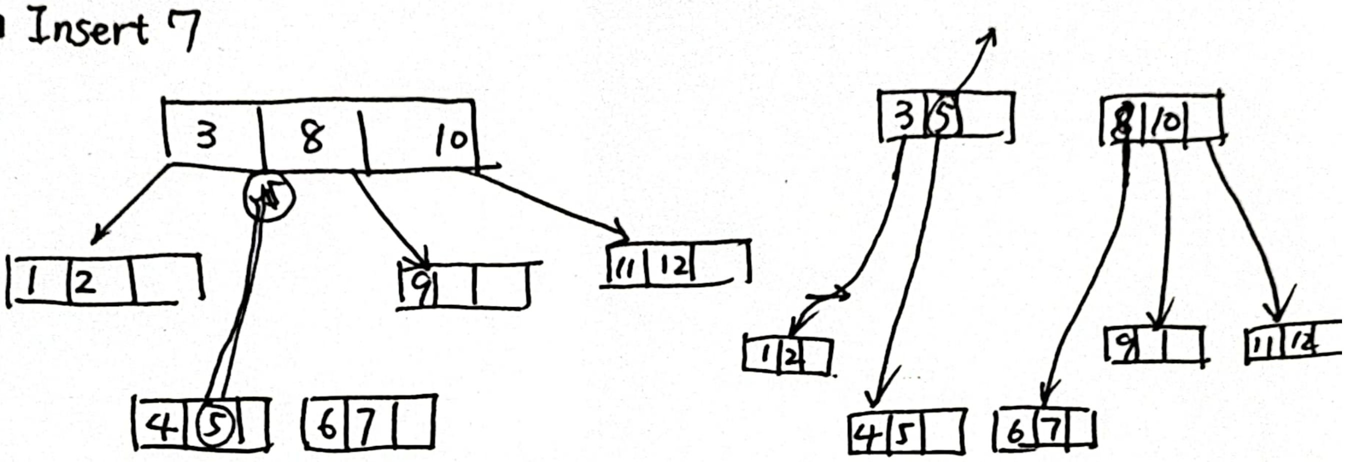
Insert 29

▫ Nodes get split if there's insufficient space



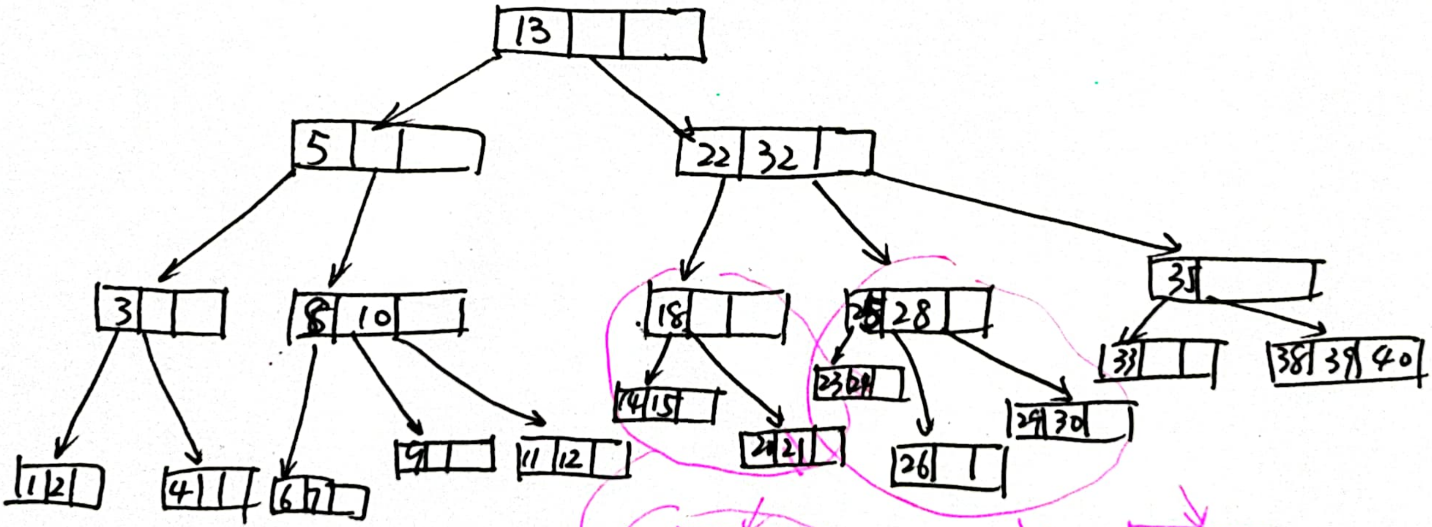
promote largest key　　smallest key

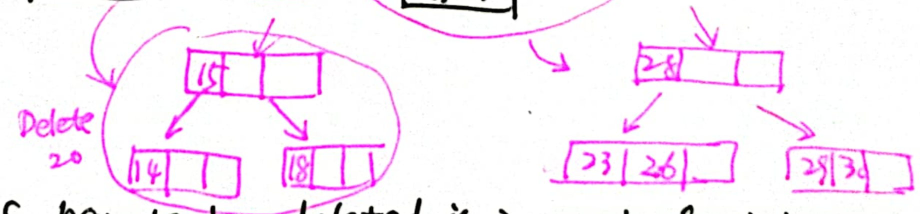
- Insert 7



- In this manner it can split cascade.
- create a new node if the cascade is NIL.

Will end up in





- Deletion
  - No problem if key to be deleted is in a leaf with at least 2 keys. (Delete 21, 24)
  - If the key to be deled is an internal node.
    - swap it with predecessor (leaf), go to case 1.
    - delete that leaf (~~Delete~~
  - If after deleting a key a node becomes empty then we borrow a key from its sibling (like rotation) Example. Delete 20.

- If sibling has only 1 key then we merge with it.

  (Delete 23)

- Moving a key down from parents corrsponds to deletion in the parent node
  - Same for the leaf node
  - Can lead to cascade (Reduced by 1).