

B-Trees

Balanced Binary Search Tree

- $H = O(\log(N))$. Insert, remove, and search all have complexity of $O(\log(n)) = O(\log_2(N))$
- Each Node has a maximum of 2 children

C-ary Tree in Tree

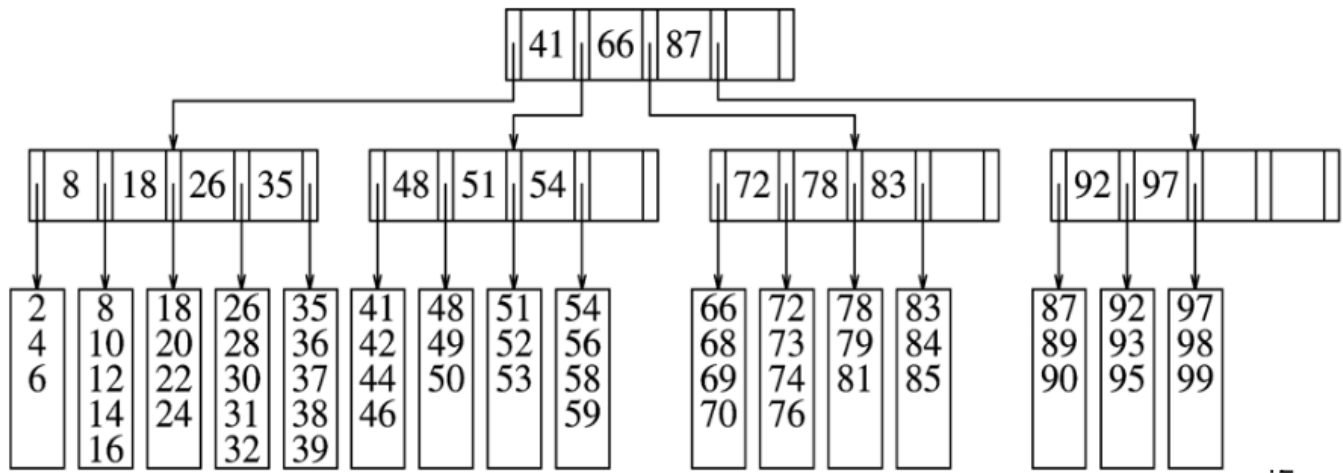
- $N = C^0 + C^1 + \dots + C^H$
- $N = \frac{C^{H+1}-1}{C-1} \approx C^H$
- So $H \approx \log_C(N) = \frac{\log_2(N)}{\log_2(C)}$
- If we increase the max number of children from 2 to C and maintain a balanced tree, the height is reduced by $(\log_2(C))$

M-ary Trees

- Allow up to M children for each node
 - Instead of 2 max for binary trees
- A complete M-ary tree of N nodes has a depth of $\log_M N$
- Each node has (M-1) keys to decide which branch follows
- Larger M, smaller tree depth
- **Balancing M-ary**
 1. Restrict the tree shape like in AVL
 2. Restrict the number of children each node can have
- B-Tree takes the second approach, easier to implement

Balanced Trees (B-Tree)

- B-Tree is an M-ary search tree with restrictions
 1. Data items stored at the leafs
 2. Non-leaf nodes store up to M-1 Keys to guide search
 3. The root can be a leaf or have between 2 to M children
 4. Non leaf Nodes except the root have between $\lceil M/2 \rceil$ and M children
 5. All leaves are at the same depth, have between $\lceil L/2 \rceil$ and L data items
- Keys in each node are sorted. The i'th key in a node is the smallest data in the i+1 subtree



Deletion

- Do a search to find the leaf node to delete
- If the leaf still has at least $L/2$ data entries, done
- Else, merge the data with neighboring leaf to ensure $L/2$ data entries

Deletion of 99

Causes combination of two leaves into one.
Can recursively combine non-leaves

