CSC3310 Algorithms
B-Trees

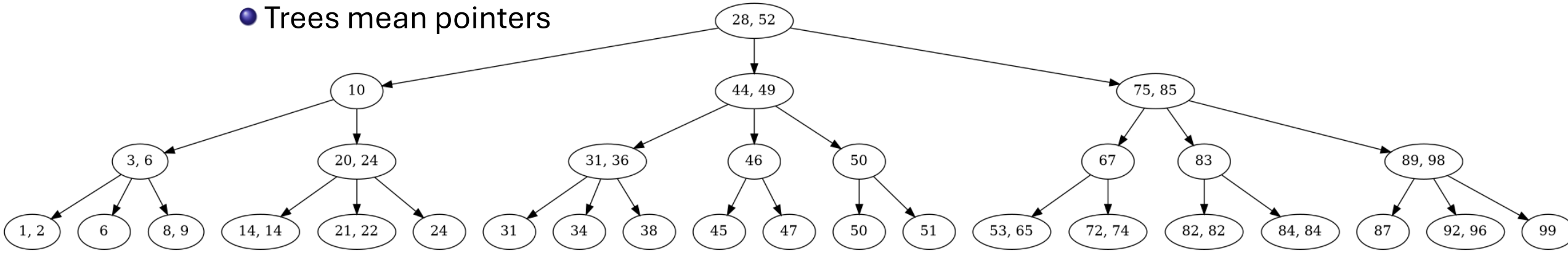
Matthew, Liam, Helina

Milwaukee School of Engineering

B-Trees

B-Tree

- Long-term (disk, ssd) storage is stored in blocks of data (non-contiguous)
- Most data structures are optimized/designed for random-access high-speed ram
- How do we store huge amounts of data efficiently?
- B-trees!
  - Auto-balancing
  - Widened levels&nodes
  - Trees mean pointers

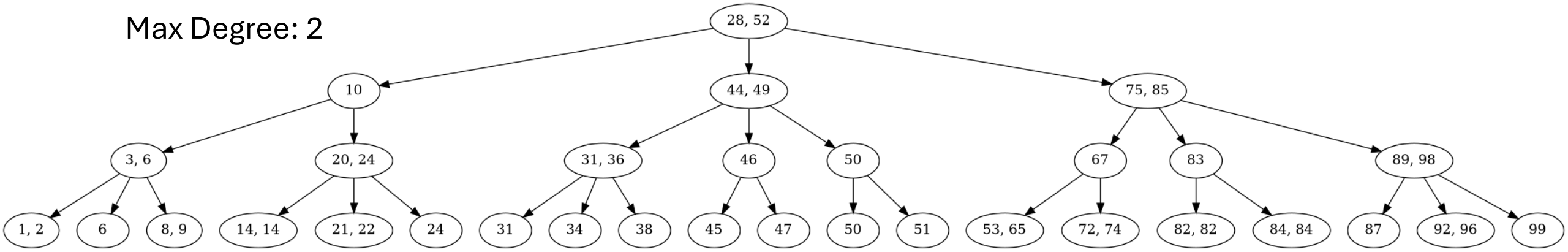


B-Trees

B-Tree Rules

- Nodes cannot have more than a set Max Degree number of Keys
- Nodes cannot have fewer than (Max Degree // 2) Keys
- Child Nodes contain keys between the respective keys of the parent Node

Max Degree: 2



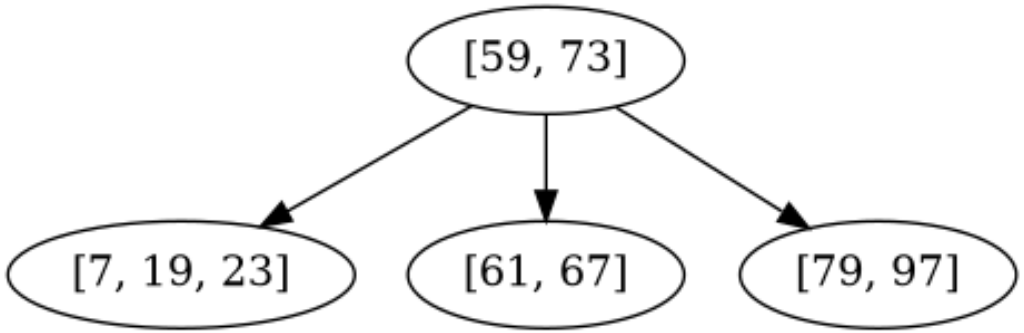
B-Tree Operations

`Insert`	Insert a new element
`Delete`	Remove an element
`Search`	Find an element in the B-Tree

Search

- Find an element in the B-Tree
  - Scanning each key
  - Recursing through children till key is found.
- Algorithm
  - Iterate through subtrees
    - If a key is found, return True
    - If a key is not found, and there is enough children, we recurse through the children.

Max Degree = 4

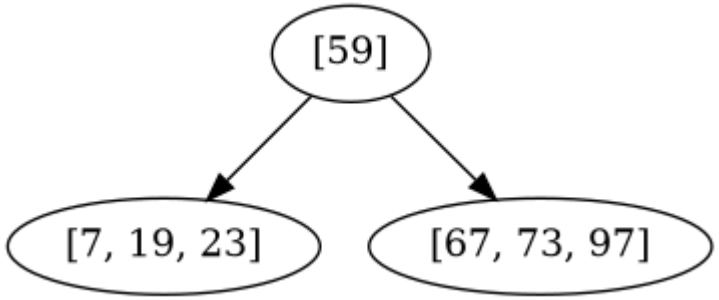


Insert

- Insert a new element
  - The data is also the separators (keys)
  - Simply: add to correct spot, split if we violate max degree
- Algorithm
  - Step into correct leaf node based off key separators
  - Add data to leaf
  - If we violate max degree
    - Split the values, add the middle value to the parent node
    - Do this step recursively and reassign root if necessary

Max Degree = 4

Insert: 79

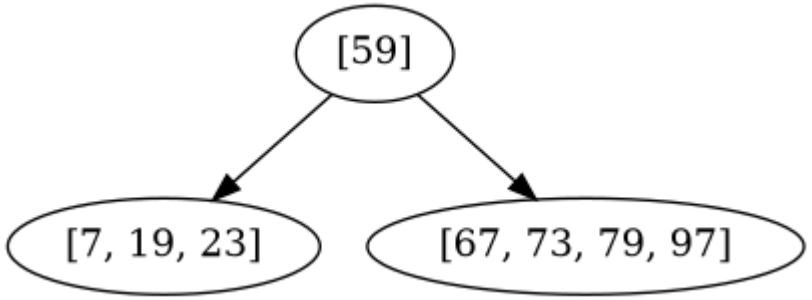


Insert

- Insert a new element
  - The data is also the separators (keys)
  - Simply: add to correct spot, split if we violate max degree
- Algorithm
  - Step into correct leaf node based off key separators
  - Add data to leaf
  - If we violate max degree
    - Split the values, add the middle value to the parent node
    - Do this step recursively and reassign root if necessary

Max Degree = 4

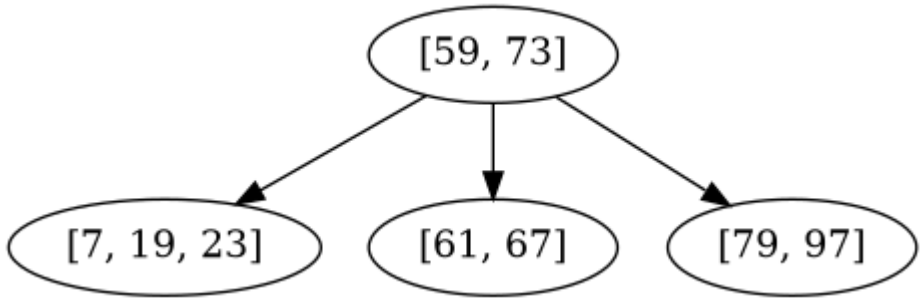
Insert: 61



Insert

- Insert a new element
  - The data is also the separators (keys)
  - Simply: add to correct spot, split if we violate max degree
- Algorithm
  - Step into correct leaf node based off key separators
  - Add data to leaf
  - If we violate max degree
    - Split the values, add the middle value to the parent node
    - Do this step recursively and reassign root if necessary

Max Degree = 4



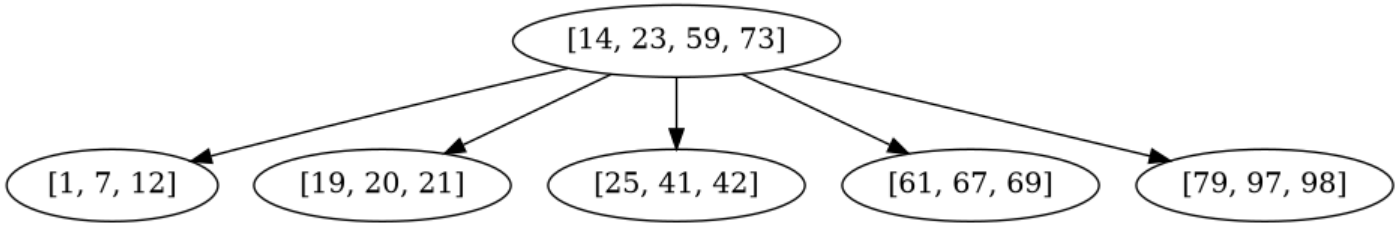
Let's look at another example

Insert

- Insert a new element
  - The data is also the separators (keys)
  - Simply: add to correct spot, split if we violate max degree
- Algorithm
  - Step into correct leaf node based off key separators
  - Add data to leaf
  - If we violate max degree
    - Split the values, add the middle value to the parent node
    - Do this step recursively and reassign root if necessary

Max Degree = 4

Insert: 2

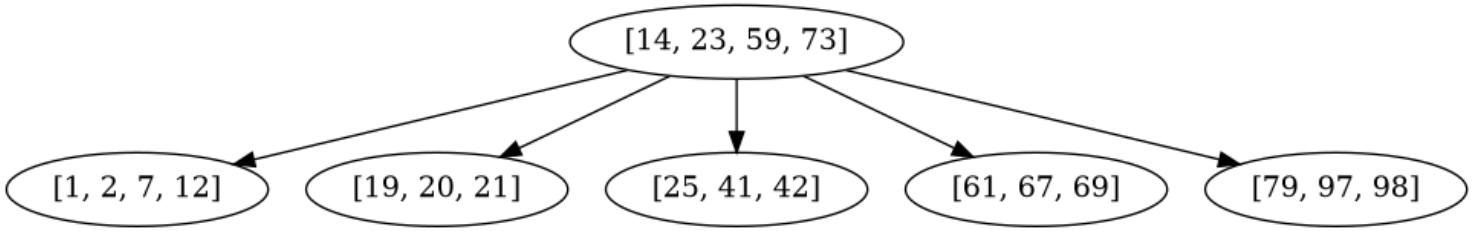


Insert

- Insert a new element
  - The data is also the separators (keys)
  - Simply: add to correct spot, split if we violate max degree
- Algorithm
  - Step into correct leaf node based off key separators
  - Add data to leaf
  - If we violate max degree
    - Split the values, add the middle value to the parent node
    - Do this step recursively and reassign root if necessary

Max Degree = 4

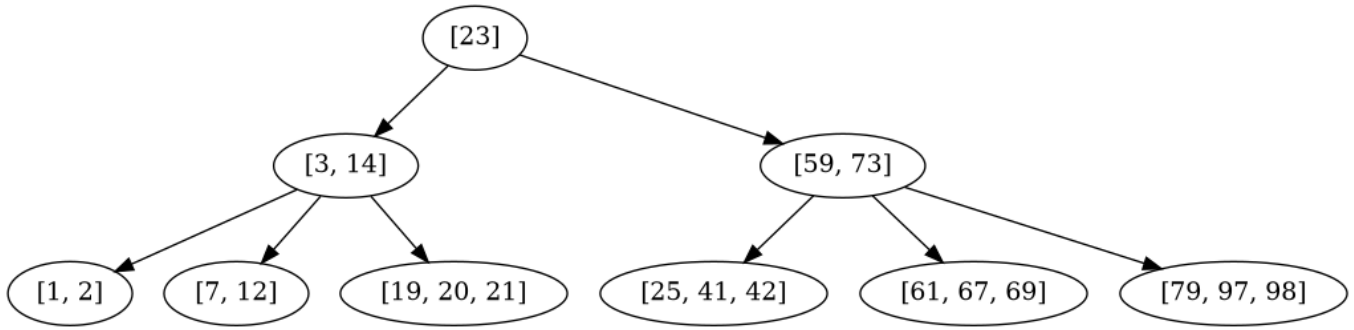
Insert: 3



Insert

- Insert a new element
  - The data is also the separators (keys)
  - Simply: add to correct spot, split if we violate max degree
- Algorithm
  - Step into correct leaf node based off key separators
  - Add data to leaf
  - If we violate max degree
    - Split the values, add the middle value to the parent node
    - Do this step recursively and reassign root if necessary

Max Degree = 4



Delete

Remove an element from the tree.

- Need to ensure that the rules are maintained throughout the tree.

Multiple cases

- Case I – Element to be deleted is in a leaf node
- Case II – Element to be deleted is in an internal node
- Case III – The height of the tree shrinks

Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion doesn’t violate any properties.
- Deletion results in a violation.

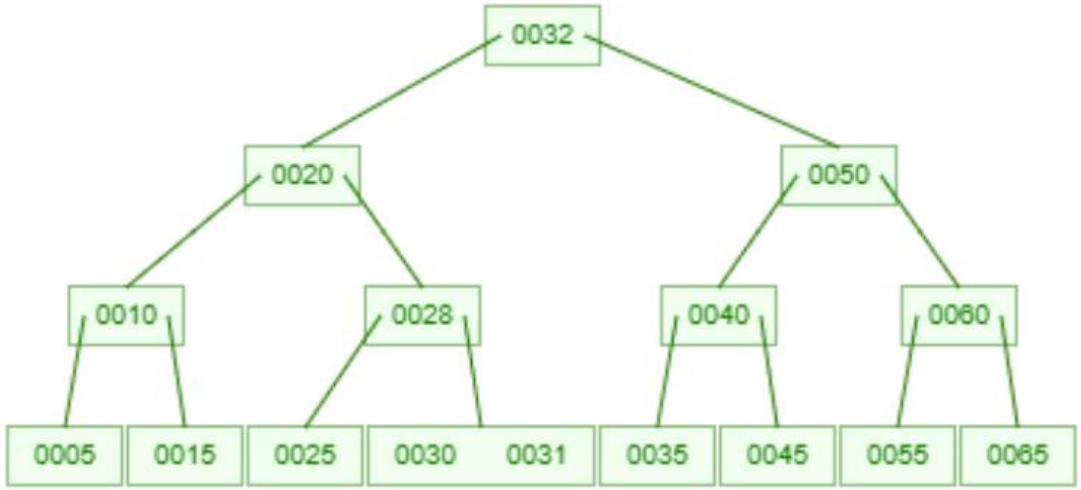
Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion doesn't violate any properties.
  - Traverse down the tree to the node
  - Delete the key

Max Degree = 2



Delete 31

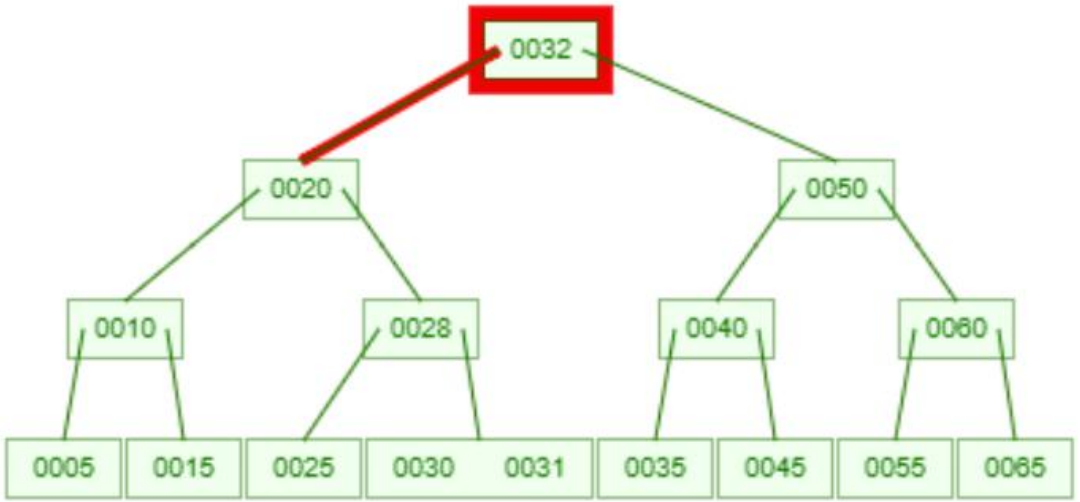
Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion doesn't violate any properties.
  - Traverse down the tree to the node
  - Delete the key

Max Degree = 2



Delete 31

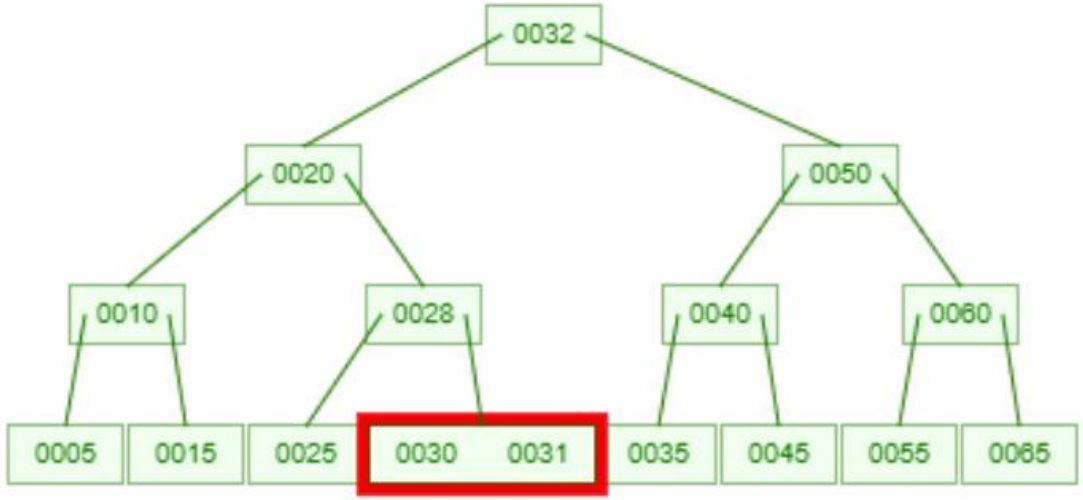
Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion doesn't violate any properties.
  - Traverse down the tree to the node
  - Delete the key

Max Degree = 2



Delete 31

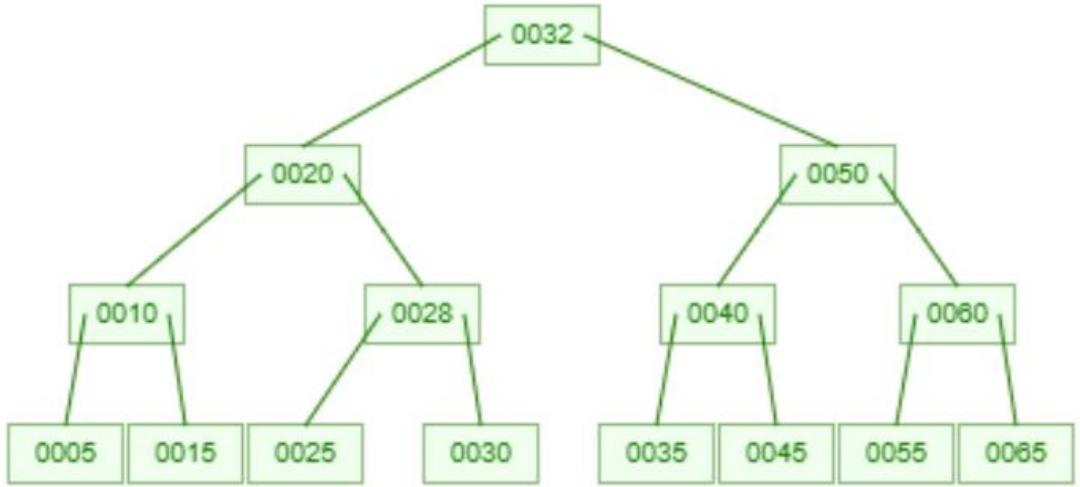
Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion doesn't violate any properties.
  - Traverse down the tree to the node
  - Delete the key

Max Degree = 2



Delete 31

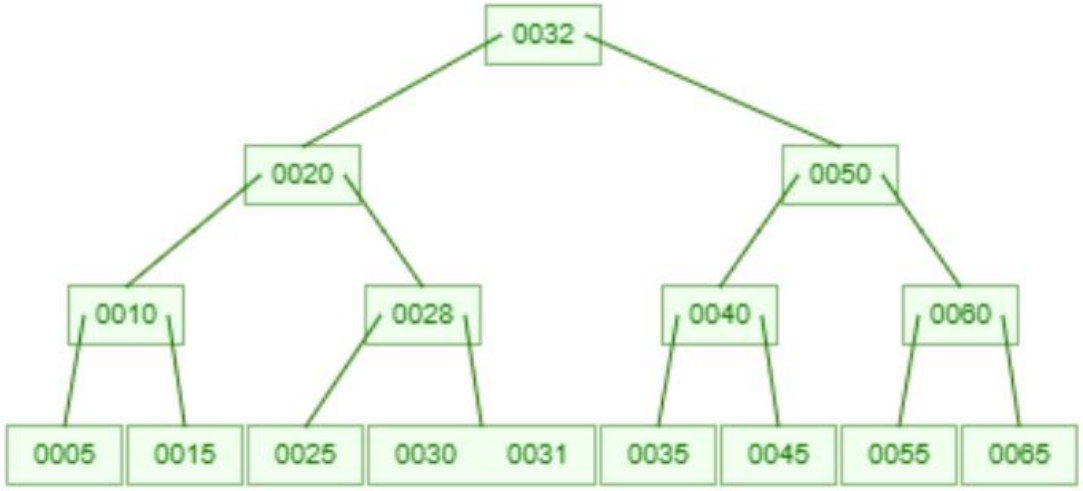
Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion results in a violation.
  - If either sibling has enough keys
    - Borrow from sibling
  - Else
    - Parent node will handle merging

Max Degree = 2



Delete 25

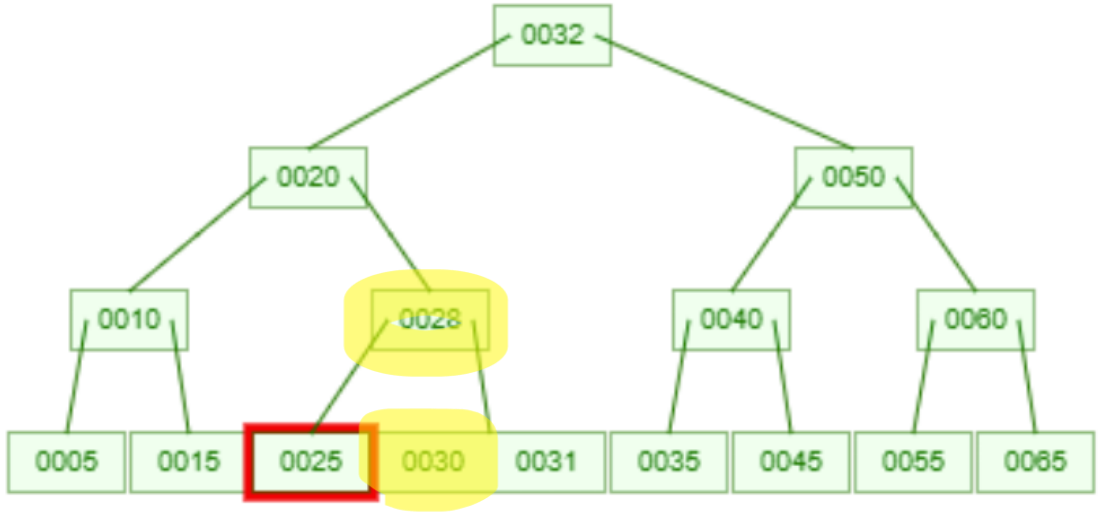
Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion results in a violation.
  - If either sibling has enough keys
    - Borrow from sibling
  - Else
    - Parent node will handle merging

Max Degree = 2



Delete 25

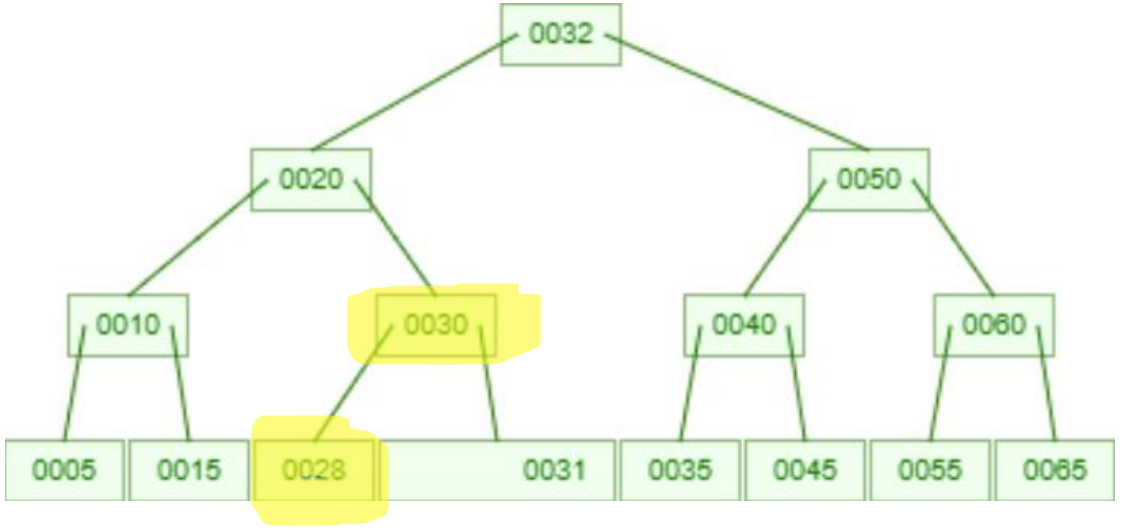
Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion results in a violation.
  - If either sibling has enough keys
    - Borrow from sibling
  - Else
    - Parent node will handle merging

Max Degree = 2



Delete 25

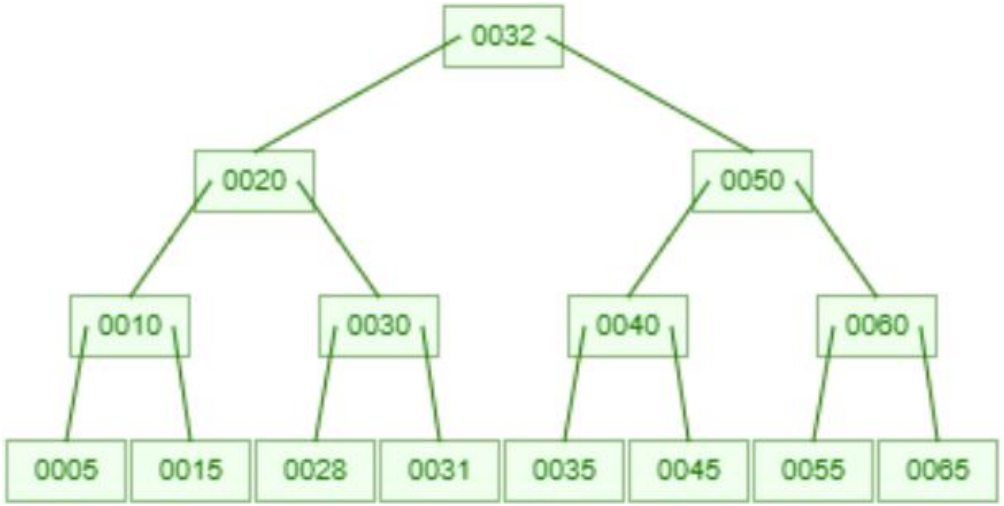
Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion results in a violation.
  - If either sibling has enough keys
    - Borrow from sibling
  - Else
    - Parent node will handle merging

Max Degree = 2



Delete 25

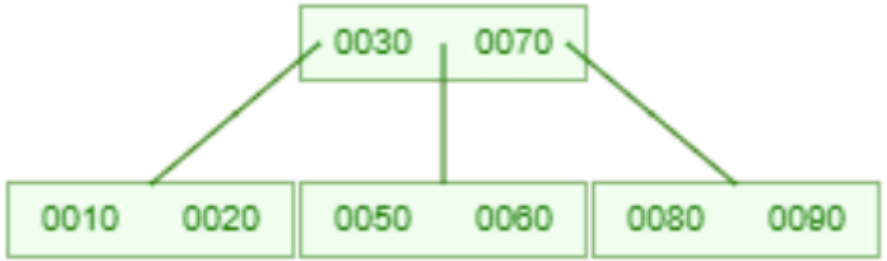
Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion results in a violation.
  - If either sibling has enough keys
    - Borrow from sibling
  - Else
    - Parent node will handle merging

Max Degree = 4



Delete 60

Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion results in a violation.
  - If either sibling has enough keys
    - Borrow from sibling
  - Else
    - Parent node will handle merging

Max Degree = 4

Delete 60

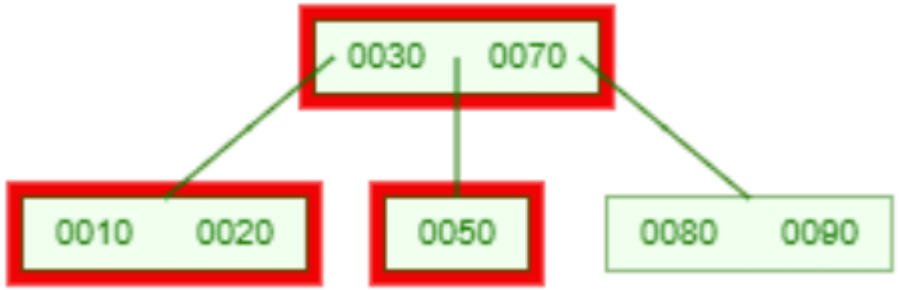
Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion results in a violation.
  - If either sibling has enough keys
    - Borrow from sibling
  - Else
    - Parent node will handle merging

Max Degree = 4



Delete 60

Case I

The element to be deleted is in a leaf node.

Two sub cases:

- Deletion results in a violation.
  - If either sibling has enough keys
    - Borrow from sibling
  - Else
    - Parent node will handle merging

Max Degree = 4

Delete 60

Case II

Element to be deleted is in an internal node

Two sub cases:

- Either child has more than the minimum number of keys
  - Take largest (left) or smallest (right) element from child
- Neither child has more than the minimum number of keys
  - Merge left and right child

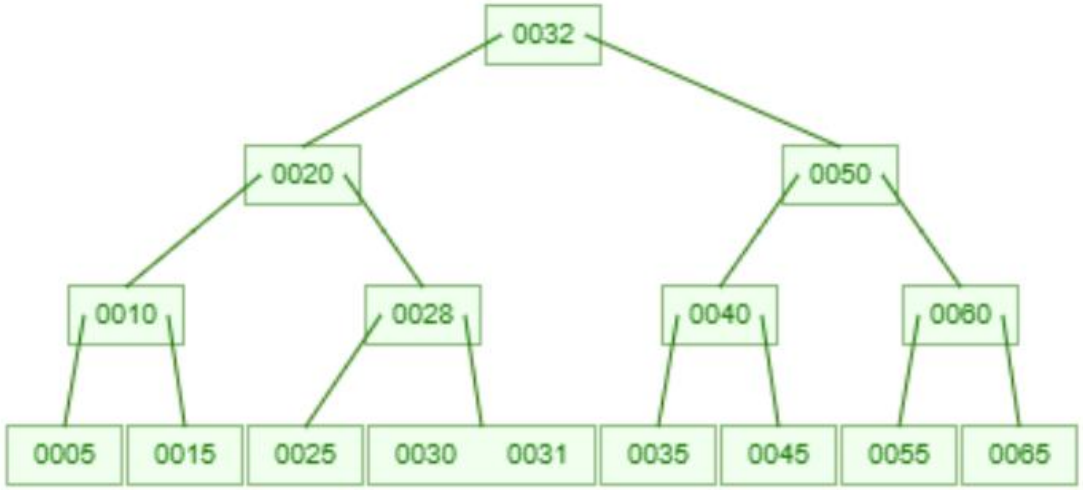
Case II

Element to be deleted is in an internal node

Two sub cases:

- Either child has more than the minimum number of keys
  - Take largest (left) or smallest (right) element from child
- Neither child has more than the minimum number of keys
  - Merge left and right child

Max Degree = 2



Delete 28

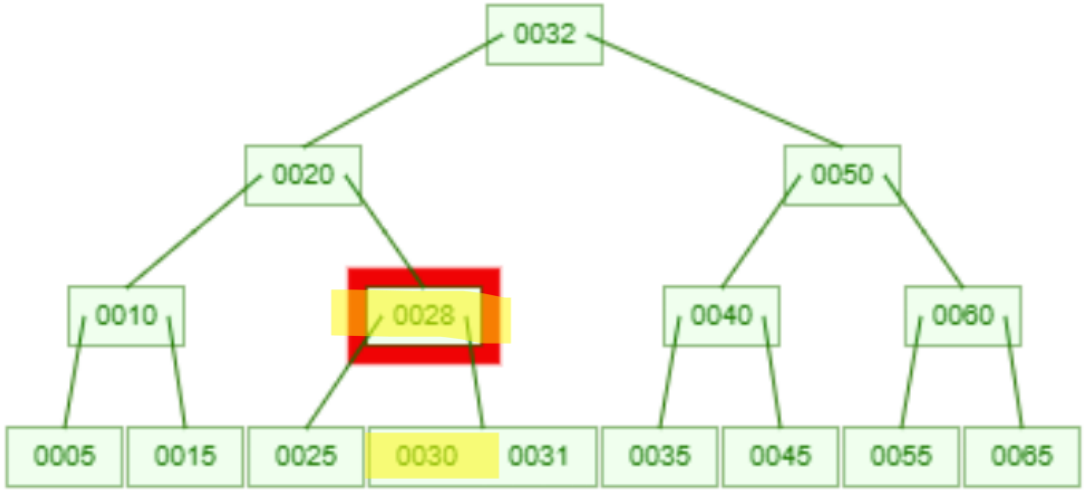
Case II

Element to be deleted is in an internal node

Two sub cases:

- Either child has more than the minimum number of keys
  - Take largest (left) or smallest (right) element from child
- Neither child has more than the minimum number of keys
  - Merge left and right child

Max Degree = 2



Delete 28

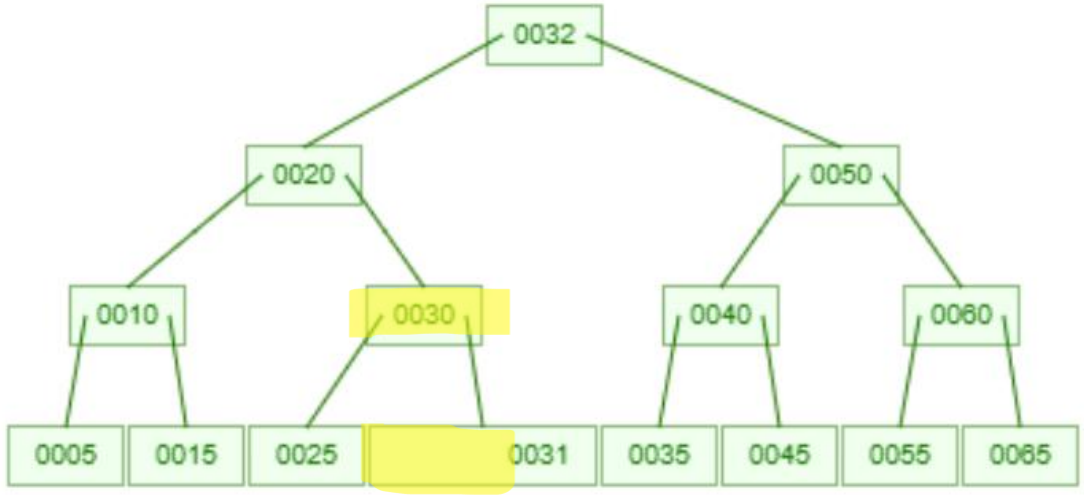
Case II

Element to be deleted is in an internal node

Two sub cases:

- Either child has more than the minimum number of keys
  - Take largest (left) or smallest (right) element from child
- Neither child has more than the minimum number of keys
  - Merge left and right child

Max Degree = 2



Delete 28

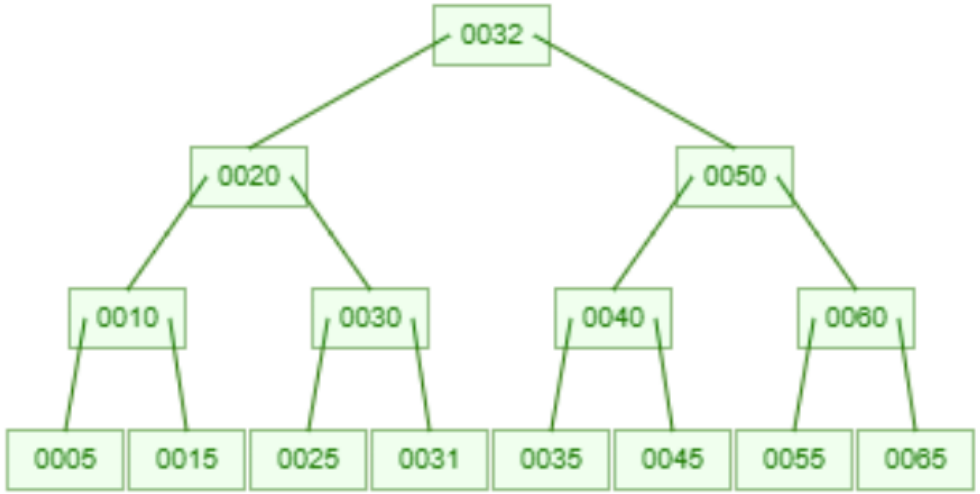
Case II

Element to be deleted is in an internal node

Two sub cases:

- Either child has more than the minimum number of keys
  - Take largest (left) or smallest (right) element from child
- Neither child has more than the minimum number of keys
  - Merge left and right child

Max Degree = 2



Delete 28

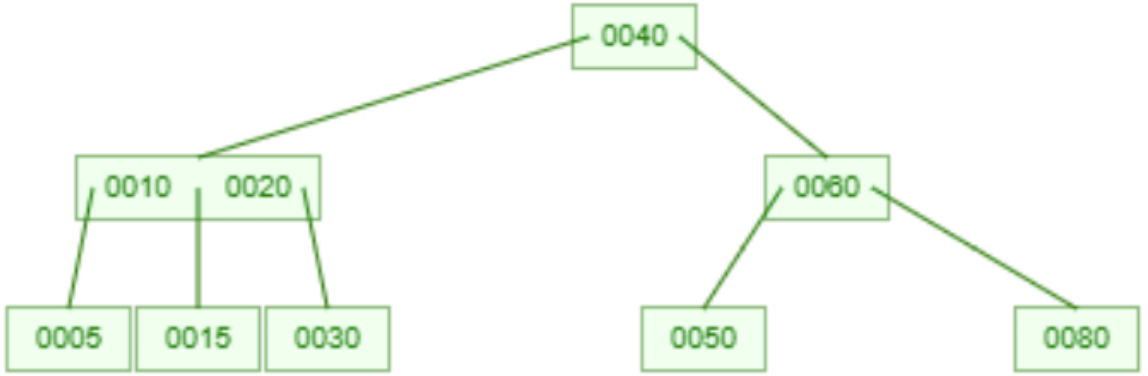
Case II

Element to be deleted is in an internal node

Two sub cases:

- Either child has more than the minimum number of keys
  - Take largest (left) or smallest (right) element from child
- Neither child has more than the minimum number of keys
  - Merge left and right child

Max Degree = 3



Delete 10

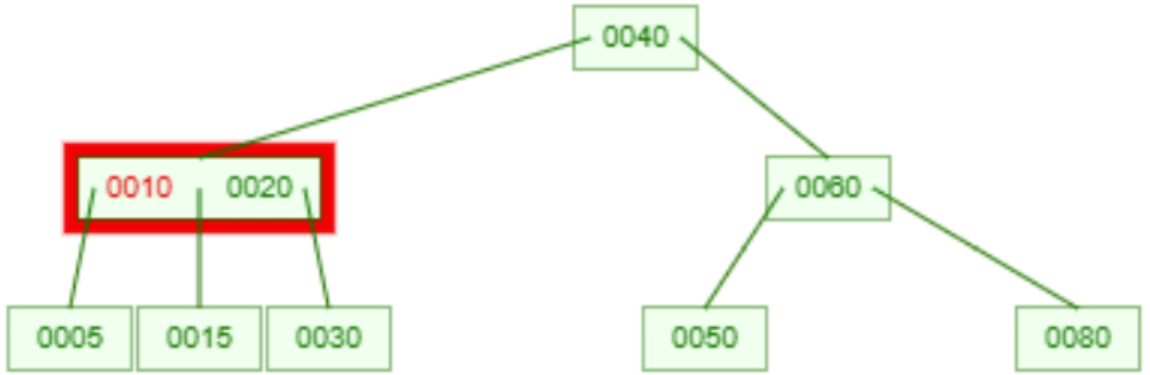
Case II

Element to be deleted is in an internal node

Two sub cases:

- Either child has more than the minimum number of keys
  - Take largest (left) or smallest (right) element from child
- Neither child has more than the minimum number of keys
  - Merge left and right child

Max Degree = 3



Delete 10

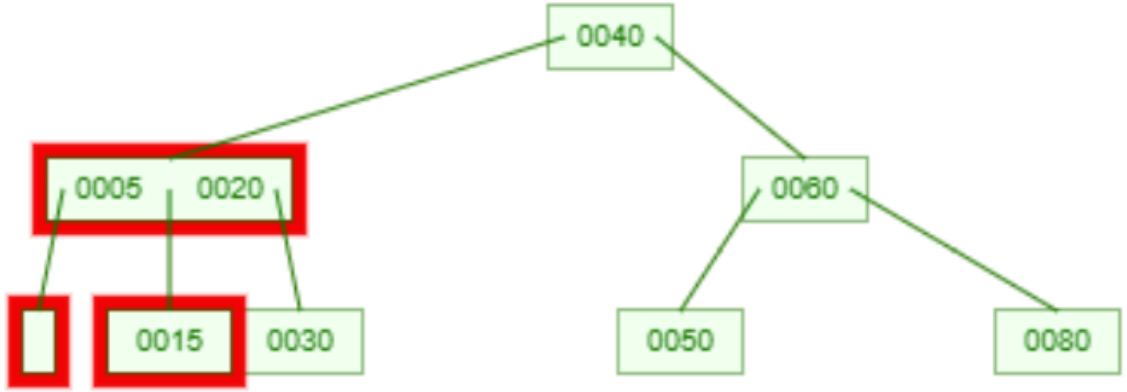
Case II

Element to be deleted is in an internal node

Two sub cases:

- Either child has more than the minimum number of keys
  - Take largest (left) or smallest (right) element from child
- Neither child has more than the minimum number of keys
  - Merge left and right child

Max Degree = 3



Delete 10

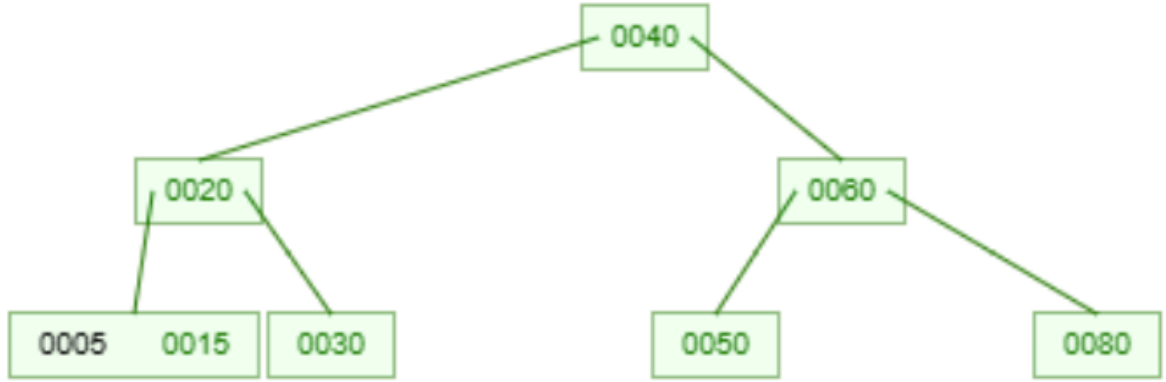
Case II

Element to be deleted is in an internal node

Two sub cases:

- Either child has more than the minimum number of keys
  - Take largest (left) or smallest (right) element from child
- Neither child has more than the minimum number of keys
  - Merge left and right child

Max Degree = 3



Delete 10

Case III

The height of the tree shrinks

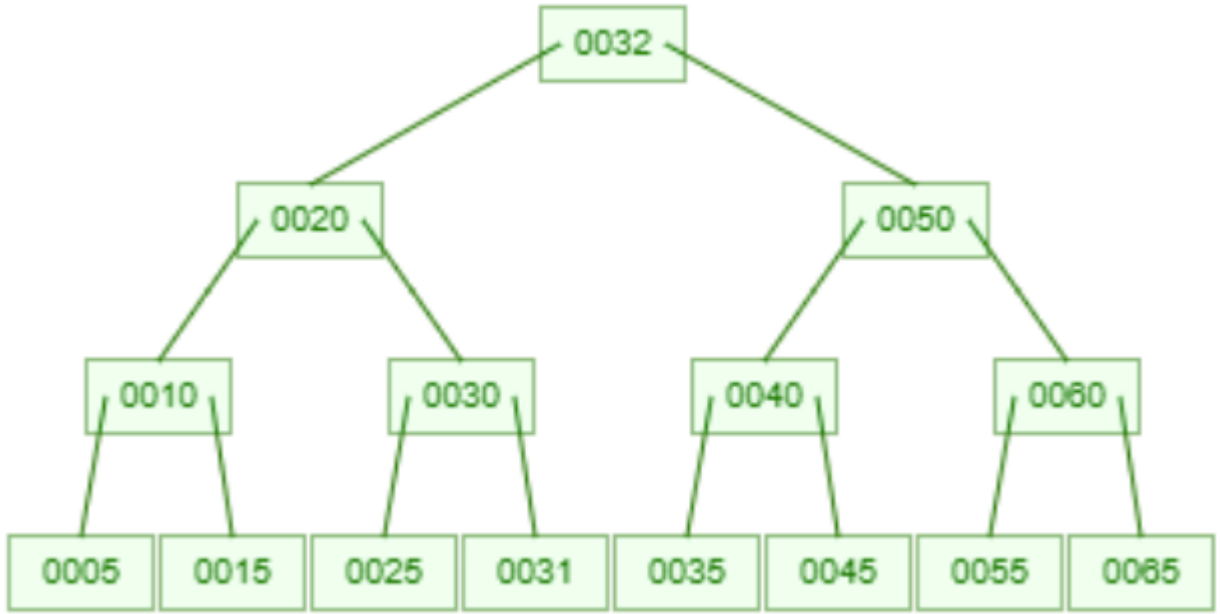
- Deletion of the key leads to fewer number of keys in the node
- If siblings don’t have enough keys, we need to merge recursively

Case III

The height of the tree shrinks

- Deletion of the key leads to fewer number of keys in the node
- If siblings don’t have enough keys, we need to merge recursively

Max Degree = 2



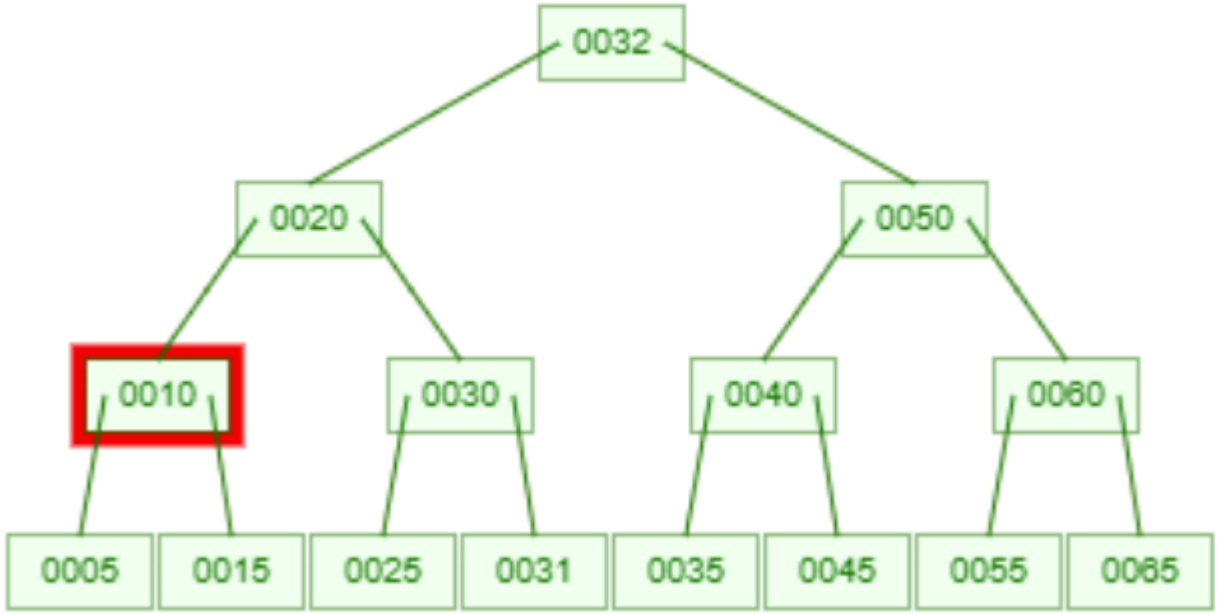
Delete 10

Case III

The height of the tree shrinks

- Deletion of the key leads to fewer number of keys in the node
- If siblings don't have enough keys, we need to merge recursively

Max Degree = 2



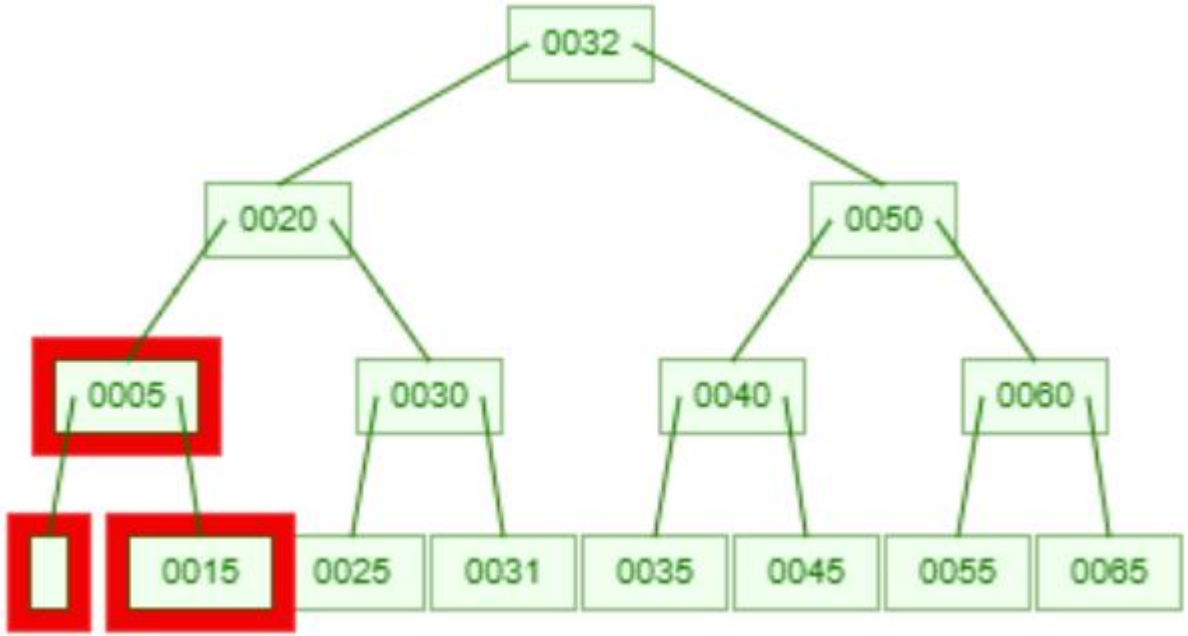
Delete 10

Case III

The height of the tree shrinks

- Deletion of the key leads to fewer number of keys in the node
- If siblings don’t have enough keys, we need to merge recursively

Max Degree = 2



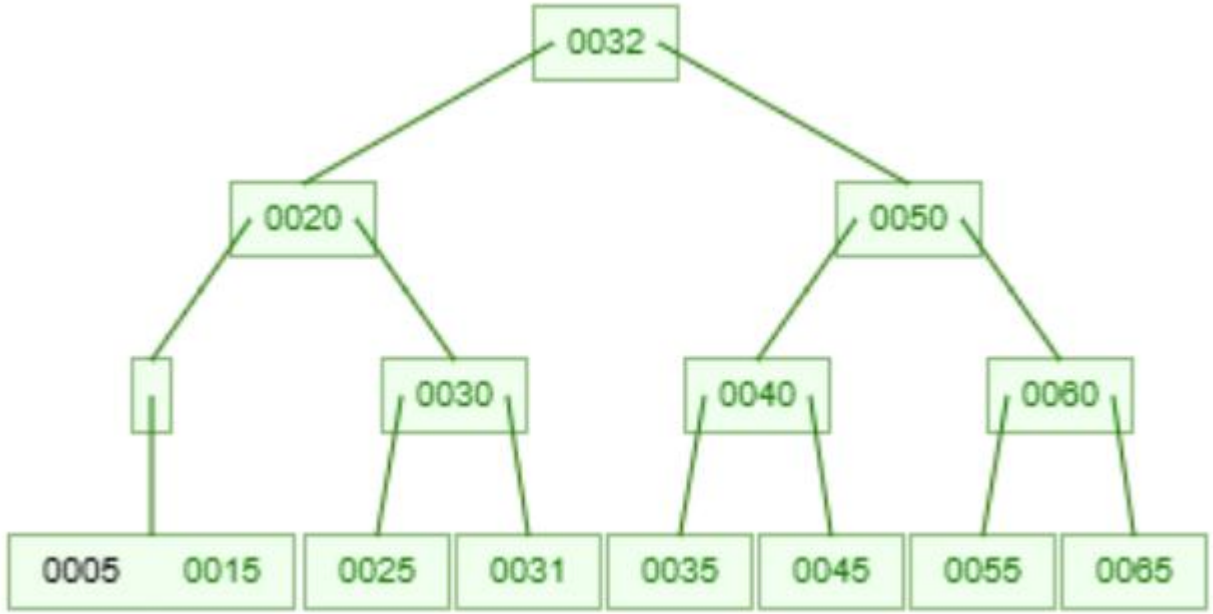
Delete 10

Case III

The height of the tree shrinks

- Deletion of the key leads to fewer number of keys in the node
- If siblings don’t have enough keys, we need to merge recursively

Max Degree = 2



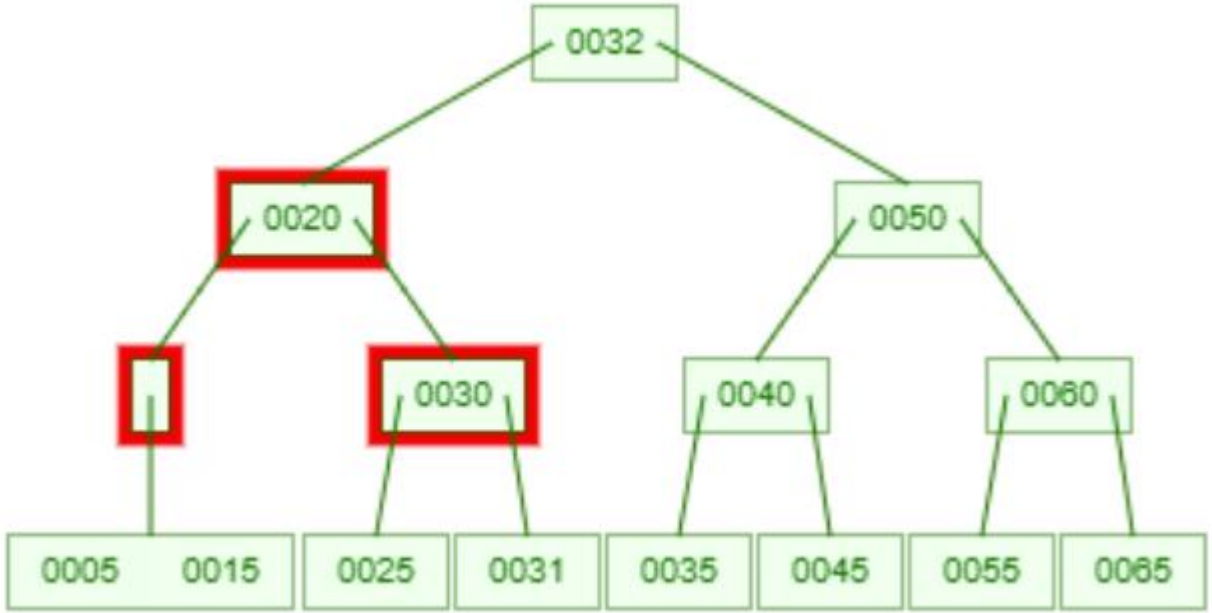
Delete 10

Case III

The height of the tree shrinks

- Deletion of the key leads to fewer number of keys in the node
- If siblings don’t have enough keys, we need to merge recursively

Max Degree = 2



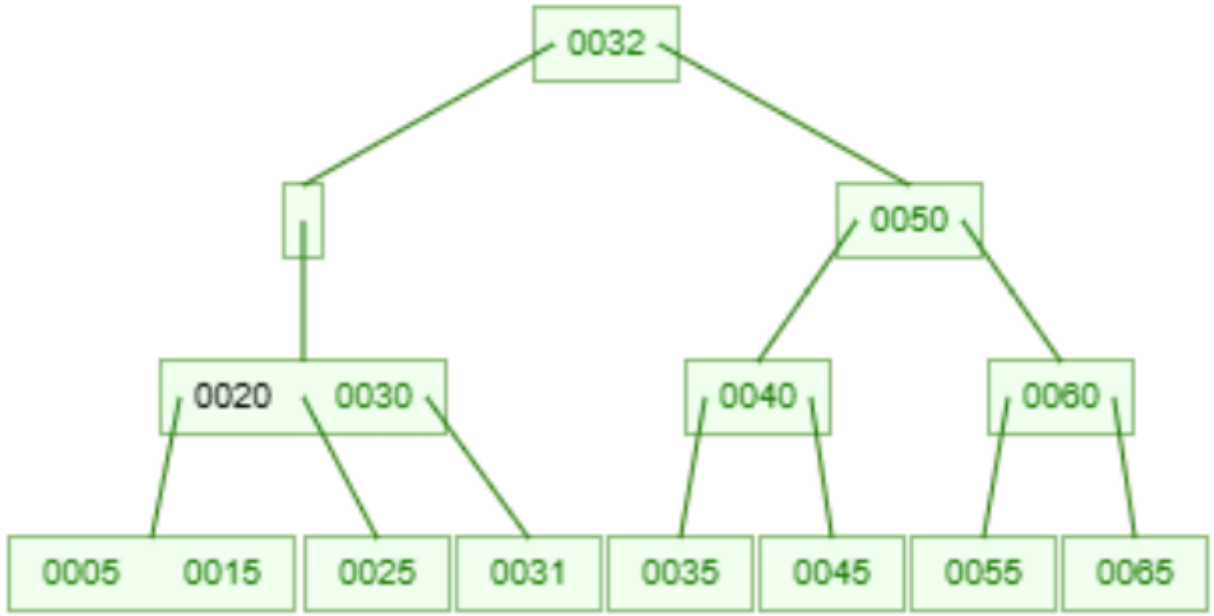
Delete 10

Case III

The height of the tree shrinks

- Deletion of the key leads to fewer number of keys in the node
- If siblings don’t have enough keys, we need to merge recursively

Max Degree = 2



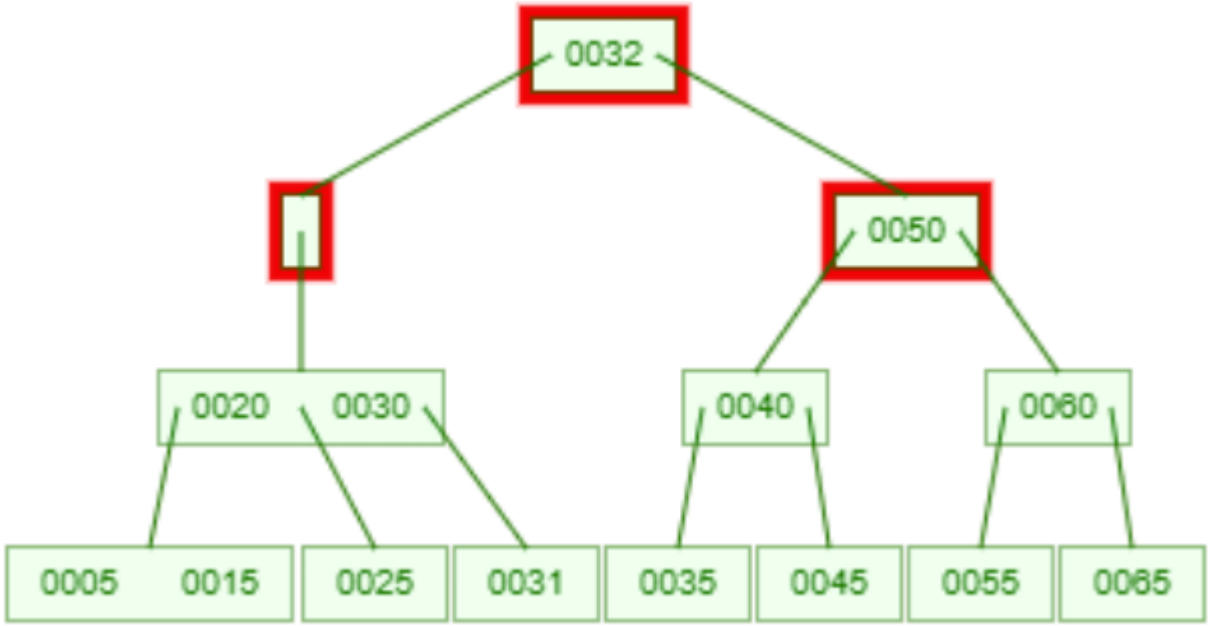
Delete 10

Case III

The height of the tree shrinks

- Deletion of the key leads to fewer number of keys in the node
- If siblings don’t have enough keys, we need to merge recursively

Max Degree = 2



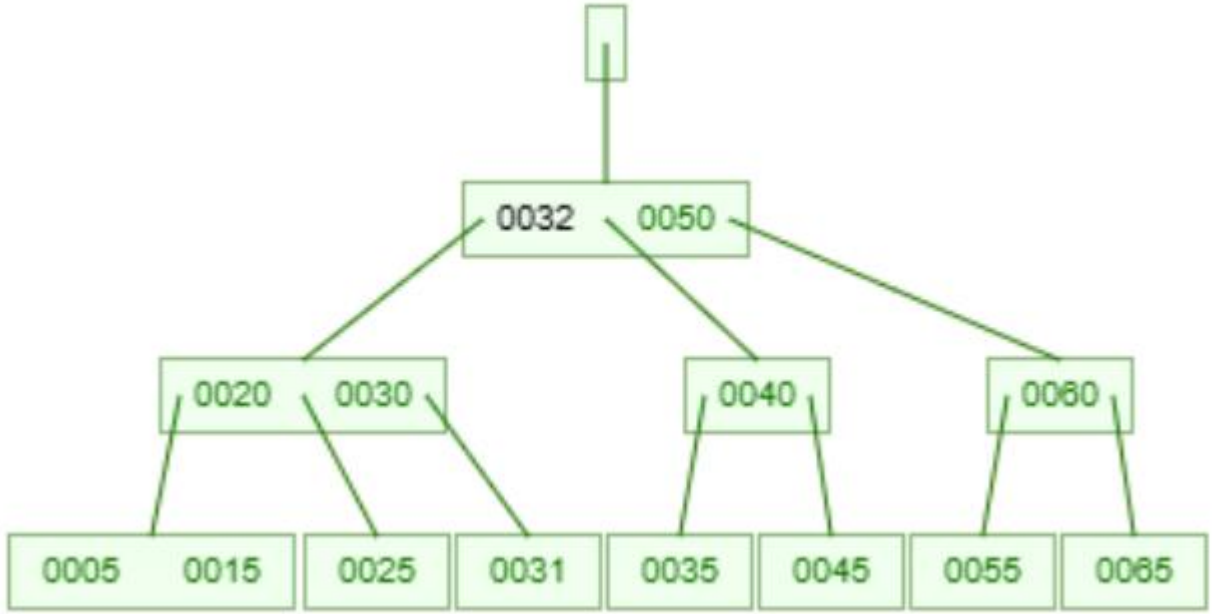
Delete 10

Case III

The height of the tree shrinks

- Deletion of the key leads to fewer number of keys in the node
- If siblings don’t have enough keys, we need to merge recursively

Max Degree = 2



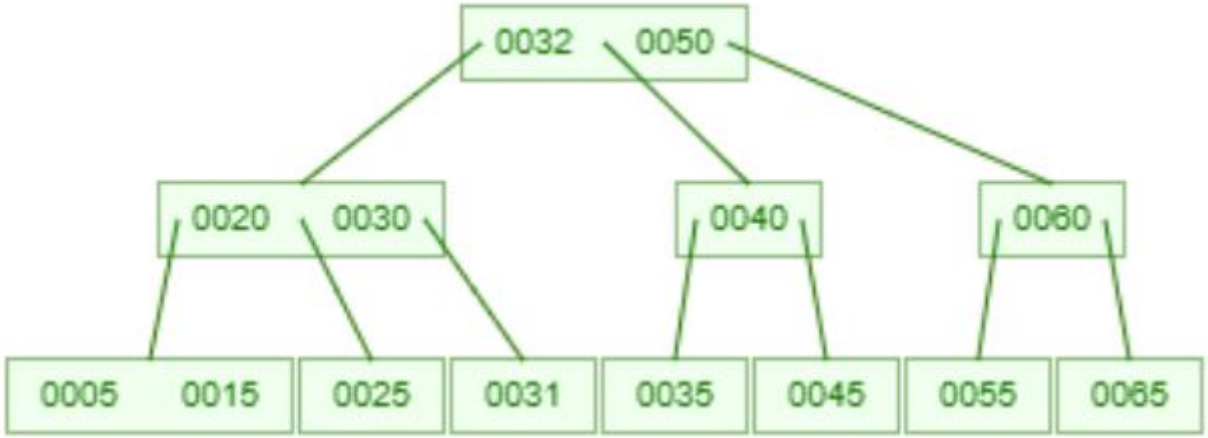
Delete 10

Case III

The height of the tree shrinks

- Deletion of the key leads to fewer number of keys in the node
- If siblings don’t have enough keys, we need to merge recursively

Max Degree = 2



Delete 10

Time Complexity

Operation	Time Complexity n = # of nodes in tree k = max degree
Search	$O(\log_k(n)\log(k))$
Insert	$O(\log_k(n)\log(k))$
Delete	$O(\log_k(n)\log(k))$

These can be simplified to O(log(n)) in all cases

$$\log_k(n) = \frac{\log_a(x)}{\log_b(k)}; \log_k(n) * \log(k) = log(n)$$

Advantages and Disadvantages

Advantages

- Very efficient for huge data storage
  - We set the max degree based on disk block sizes decreasing disk I/O – optimized for disk storage
  - If we are storing huge amounts of data in memory, the larger contiguous nodes result in fewer cache misses
- Auto-Balancing without sacrificing runtime
  - log(n) for all operations

Disadvantages

- Complex to implement and maintain
  - Memory/Disk Fragmentation
  - Non-numeric data
- A decent amount of overhead no matter the size of the tree.
- Data in range queries can be scattered across nodes
  - B+ Trees are often used which makes this more efficient

Questions?

“Bee Tree”