Multiway Tree

Data Structures and Algorithms

Tran Ngoc Bao Duy
Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM

Overview

Motivation

- Assume that we use an AVL tree to store about 20 million records.
- We end up with a very deep binary tree, that $\log_2(2 \times 10^7)$ is about 24.

Motivation

- Assume that we use an AVL tree to store about 20 million records.
- We end up with a very deep binary tree, that $\log_2(2 \times 10^7)$ is about 24.
- The solution is to use more branches and thus reduce the height of the tree!

Multiway Trees

Multiway Trees

Compared to binary tree

Whereas each node in a binary tree has only **one entry**, multiway trees have **multiple entries** in each node and thus may have **multiple subtrees**.

M-Way Search Trees

An **m-way tree** is a search tree in which each node can have from `0` to `m` subtrees, where m is defined as the B-tree order, with the following properties:

1. Each node has 0 to m subtrees.

M-Way Search Trees

An **m-way tree** is a search tree in which each node can have from `0` to `m` subtrees, where m is defined as the B-tree order, with the following properties:

1. Each node has 0 to m subtrees.
2. A node with $k < m$ subtrees contains k subtrees and $k - 1$ data entries.

M-Way Search Trees

An **m-way tree** is a search tree in which each node can have from `0` to `m` subtrees, where m is defined as the B-tree order, with the following properties:

1. Each node has 0 to m subtrees.
2. A node with $k < m$ subtrees contains k subtrees and $k - 1$ data entries.
3. The key values in subtrees are all greater than or equal to the key of the left data entry and less than the key of the right data entry.

M-Way Search Trees

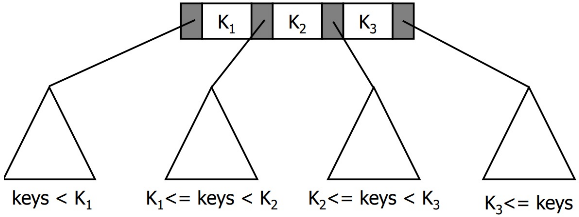
An **m-way tree** is a search tree in which each node can have from `0` to `m` subtrees, where m is defined as the B-tree order, with the following properties:

1. Each node has 0 to m subtrees.
2. A node with $k < m$ subtrees contains k subtrees and $k - 1$ data entries.
3. The key values in subtrees are all greater than or equal to the key of the left data entry and less than the key of the right data entry.
4. The keys of the data entries are ordered.

M-Way Search Trees

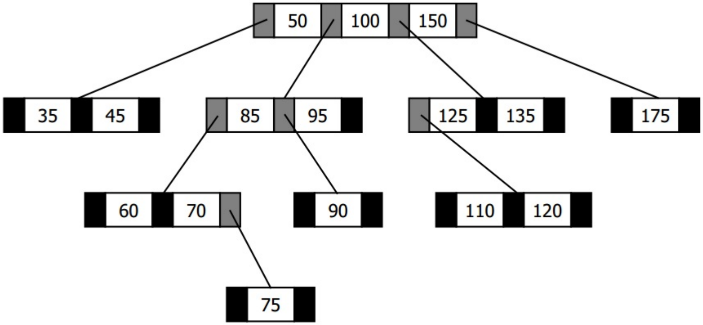
An **m-way tree** is a search tree in which each node can have from `0` to `m` subtrees, where m is defined as the B-tree order, with the following properties:

1. Each node has 0 to m subtrees.
2. A node with $k < m$ subtrees contains k subtrees and $k - 1$ data entries.
3. The key values in subtrees are all greater than or equal to the key of the left data entry and less than the key of the right data entry.
4. The keys of the data entries are ordered.
5. All subtrees are themselves multiway trees.



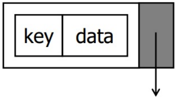
(Source: Data Structures - A Pseudocode Approach with C++)

M-Way Search Trees

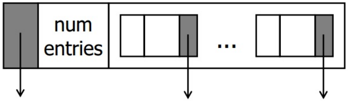


(Source: Data Structures - A Pseudocode Approach with C++)

M-Way Node Structure



```
entry
    key <key type>
    data <data type>
    rightPtr <pointer>
end entry
```



```
node
    firstPtr <pointer>
    numEntries <integer>
    entries <array[1 .. m-1] of entry>
end node
```

(Source: Data Structures - A Pseudocode Approach with C++)

B-Trees

B-Trees

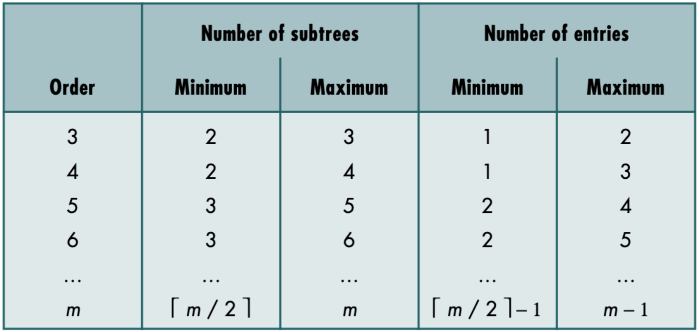
- M-way trees are unbalanced.
- Bayer, R. & McCreight, E. (1970) created B-Trees.

B-Trees

A B-tree is an m-way tree with the following additional properties ($m \geq 3$):

- The root is either a leaf or it has $[2, m]$ subtrees.
- All internal nodes have at least $\lceil m/2 \rceil$ nonnull subtrees and at most m nonnull subtrees.
- All other nodes have at least $\lceil m/2 \rceil - 1$ entries and at most $m - 1$ entries.
- All leaf nodes are at the same level.

Entries in B-Trees of Various Orders



	Number of subtrees		Number of entries	
Order	Minimum	Maximum	Minimum	Maximum
3	2	3	1	2
4	2	4	1	3
5	3	5	2	4
6	3	6	2	5
...	...	...	...	...
m	$\lceil m/2 \rceil$	m	$\lceil m/2 \rceil - 1$	$m - 1$

(Source: Data Structures - A Pseudocode Approach with C++)

B-Trees

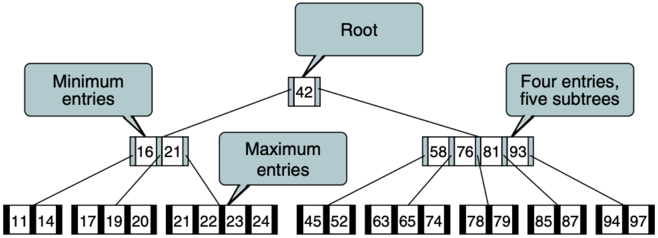


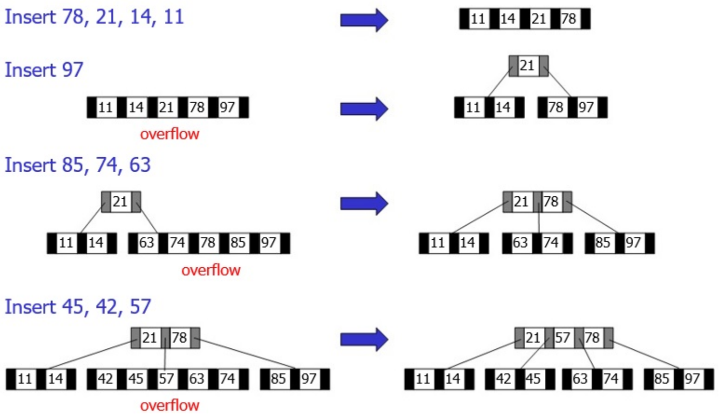
Figure: B-Tree with $m = 5$

(Source: Data Structures - A Pseudocode Approach with C++)

B-Tree Insertion

- Attempt to insert the new key into a leaf.
- If this would result in that leaf becoming too large (more than $m - 1$ entries, overflow), split the leaf into two, promoting the middle key to the leaf's parent.
- If this would result in the parent becoming overflow, split the parent into two, promoting the middle key.
- This strategy might have to be repeated all the way to the root.
- If necessary, the root is split in two and the middle key is promoted to a new root, making the tree one level higher.

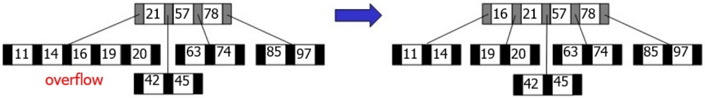
B-Tree Insertion



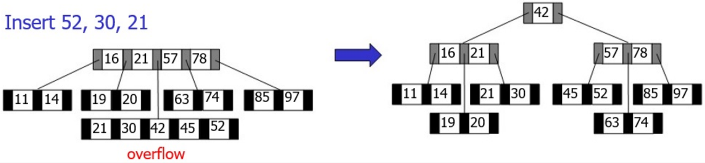
(Source: Data Structures - A Pseudocode Approach with C++)

B-Tree Insertion

Insert 20, 16, 19



Insert 52, 30, 21



(Source: Data Structures - A Pseudocode Approach with C++)

B-Tree Deletion

1. If the key is already in a leaf node, and removing it doesn't cause that leaf node to have too few keys, then simply remove the key to be deleted.

B-Tree Deletion

1. If the key is already in a leaf node, and removing it doesn't cause that leaf node to have too few keys, then simply remove the key to be deleted.

2. If the key is not in a leaf, just delete the key and promote the predecessor (largest on the left) or successor (smallest on the left) key to the non-leaf deleted key's position.

B-Tree Deletion

3. If (1) or (2) lead to a leaf node containing less than the minimum number of keys then look at the siblings immediately adjacent to the leaf in question:
    1. **Borrow**: If one of them has more than the min. number of keys, promote one of its keys to the parent and take the parent key into our lacking leaf.
    2. **Combine**: If neither of them has more than the min. number of keys then the lacking leaf and one of its neighbours can be combined with their shared parent entry and the new leaf will have the correct number of keys.

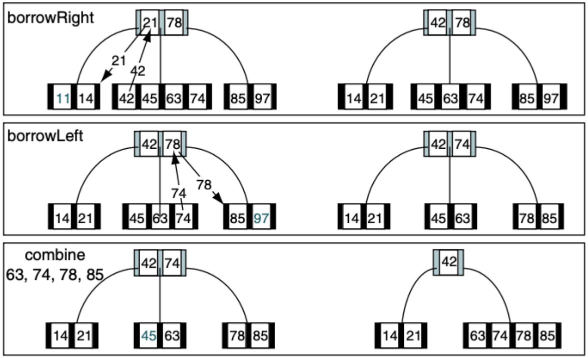
       Note: If this step leave the parent with too few keys, repeat the process up to the root itself, if required.

B-Tree Deletion

(a) Delete 11

(b) Delete 97

(c) Delete 45

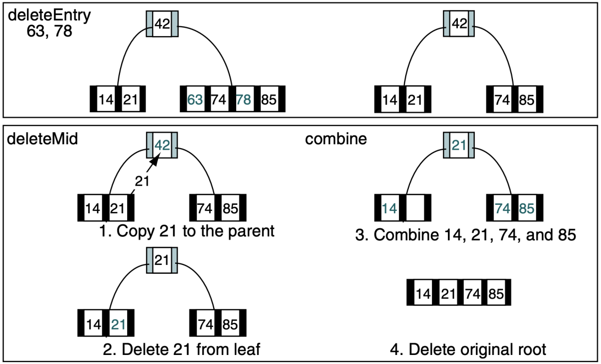


(Source: Data Structures - A Pseudocode Approach with C++)

B-Tree Deletion

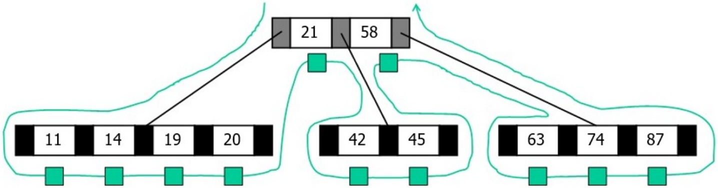
(d) Delete 63 and 78

(e) Delete 42



(Source: Data Structures - A Pseudocode Approach with C++)

B-Tree Traversal



B-Tree Variations

- B*Tree: the minimum number of (used) entries is two thirds.
- B+Tree:
  - Each data entry must be represented at the leaf level.
  - Each leaf node has one additional pointer to move to the next leaf node.

THANK YOU.