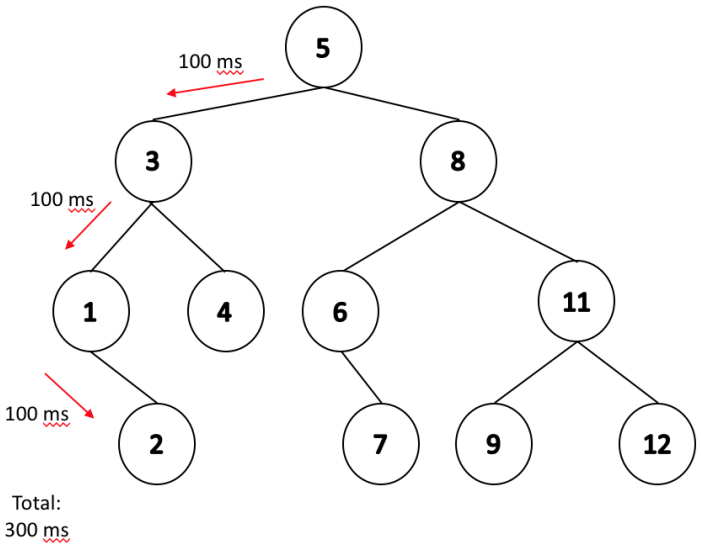
TA Lecture Notes: Lecture #23 - BTrees

March 12, 2018 • by *Milica Hadzi-Tanovic*

** BTree is **NOT** a binary tree **

Motivation:

- The definition of big-O assumes that every operation takes O(1) but that there is a non-constant n number of these O(1) operations.
- What if our data is not in the main memory, but on the cloud?
    - The operation to lookup/get the data from the cloud is not O(1).



- We need a data structure where data may not reside in main memory.
- Example:
    - Assume there was 500 million record ≅ 2.5 PB of data
    - In this case, AVL tree will have 30 levels → this means that the lookup time would be 3 seconds.
    - Our lookups on Facebook take less than 1 second and Facebook has much more than 2.5PB of data.
- We want to:
    - Keep the tree short.
    - Keep the data relevant.

BTrees (of order m)

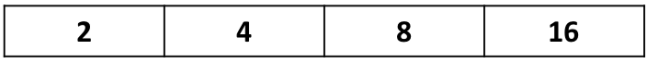
- Goal: minimize the number of reads.
- We starts with a sorted array (keys are ordered):
    - If BTree is of order m, the sorted array of size m - 1 is our tree.

-3	8	23	25	31	42	43	55

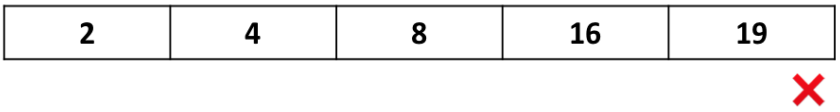
m = 9

- Building the tree:
    - Order tells us how many keys can be in a node.
        - # of keys = m - 1

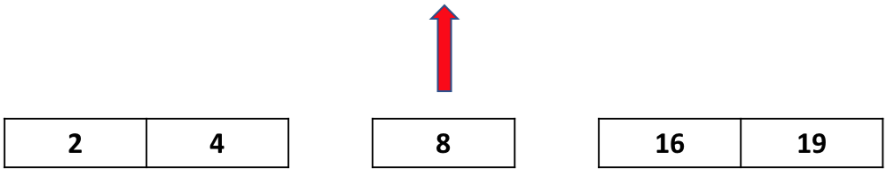
- We add to the tree until we reach the maximum number of keys in a node.
  - Consider a BTree of order 5.



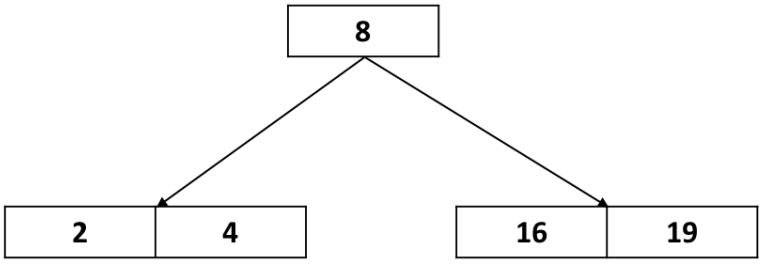
- If we insert for example 19, we will exceed the allowed number of keys in this node.



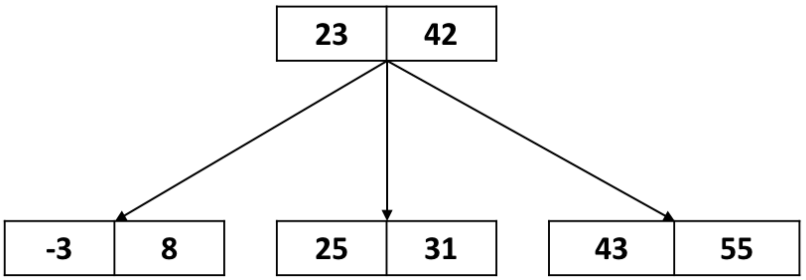
- In order to fix the node, we need to split the data and throw up the middle element.



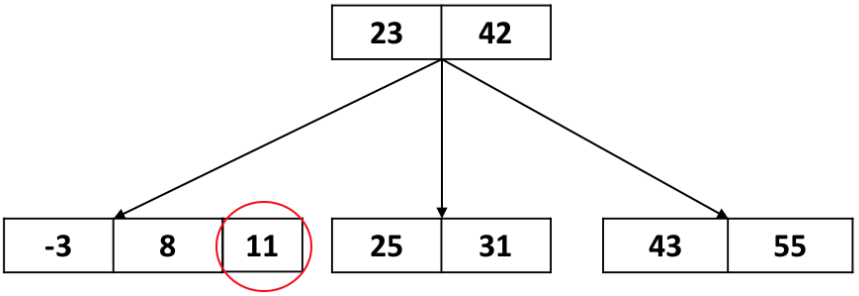
- Finally, we connect them and obtain a tree.



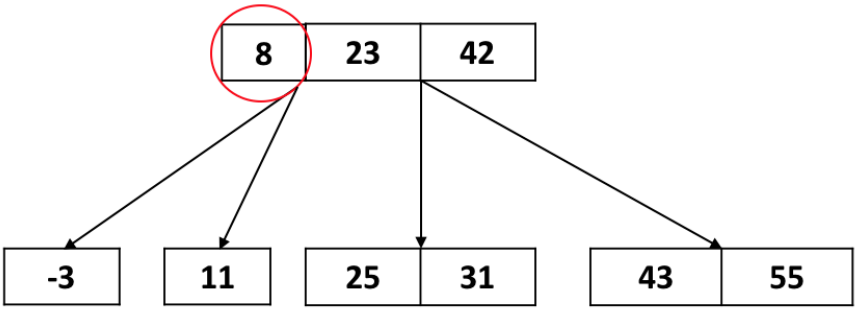
- Let's see what happens in the following case:
  - m = 3



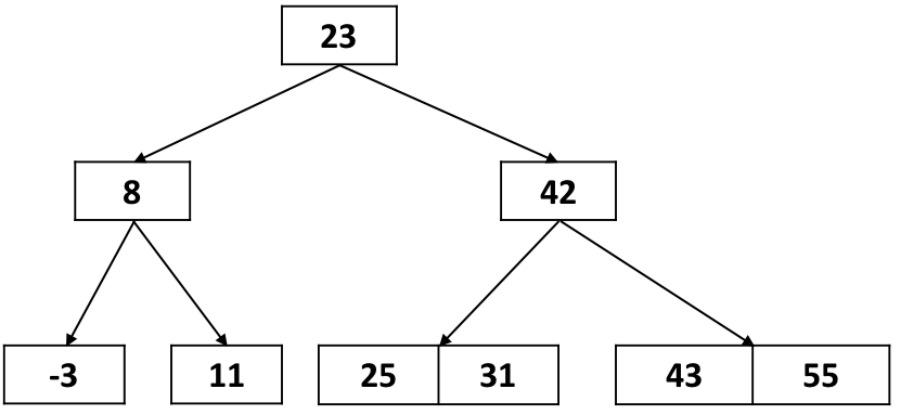
- Insert 11



- At the leftmost child node, we have too many keys. Therefore, we need to split data and throw up key number 8.



- We fixed the child node, but now the parent has 3 key which is illegal. Thereby, we split and throw up again.



✅

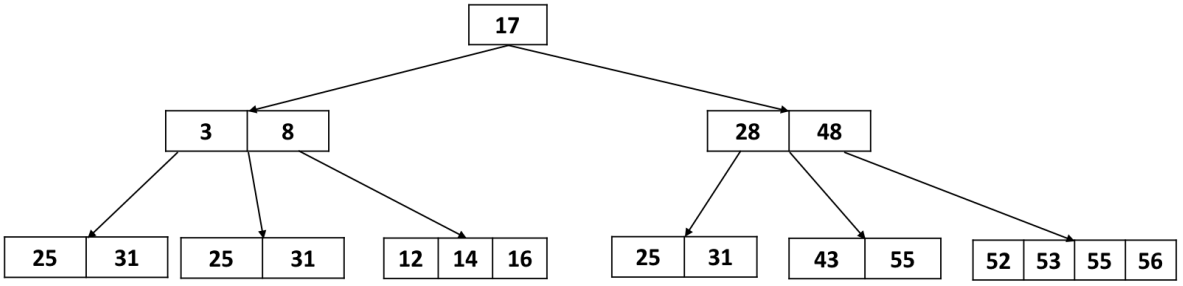
- Finally, we have a valid BTree of order 3.
- Note:
  - Everything on the left subtree is less than the left key.
  - Everything on the right subtree is greater than the right key.
  - Middle tree is the data in between left and right key.

- **BTree properties:**
  - A BTree of order m is an m-way tree.

- All keys within a node are ordered.
- All leaves contain no more than m - 1 keys.
- All internal nodes have exactly one more child than keys.
- Root node can be a leaf or have [2, m] children (because the only way to add a node is to grow).
- All non-root nodes have [ceiling(m/2), m] children.
- All leaves are on the same level.

What is the order of the following tree?

- Lower bound: order is at least 5, m ≥ 5 (because one of the nodes has 4 keys).
- Upper bound: m ≤ 7 (because of ceiling definition on children).
- Therefore m = {5 or 6}



- How big is a BTree going to get?
  - BTreeis shorter than AVL:
    - AVL has 2 children → h = $\log_2(n)$.
    - BTree has m children → h = $\log_m(n)$.
- Find in BTree works the same way as find in AVL.
  - Lines 8 & 9: Loop over keys → linear search.
    - If the data is not in memory linear search is better because we can search while reading-in the data.
    - If the data is in memory, we should do binary search.
    - However, this running time is not physical and in the function we are bounded by the physical time (this means that the running time at this point in code is not the most important).
  - Lines 11 & 12: Found the data and return.
  - Lines 15 to 20: Lookup data → this is the slowest part of the code. Lookups are slow. They take a lot of physical time.

BTree.cpp	

6
7
8
9
	bool Btree::_exists(BTreeNode & node, const K & key) {
	
	  unsigned i;
	  for (i = 0; i < node.keys_ct_ && key < node.keys_[i]; i++) { }

```
10
11      if ( i < node.keys_ct_ && key == node.keys_[i] ) {
12          return true;
13      }
14
15      if ( node.isLeaf() ) {
16          return false;
17      } else {
18          BTreeNode nextChild = node._fetchChild(i);
19          return _exists(nextChild, key);
20      }
21  }
```

- In order to delete from BTree, we need to “steal” keys, but we will not work on it.