**B+ Trees**

- Similar to B trees, with a few slight differences
- All data is stored at the leaf nodes (leaf pages); all other nodes (index pages) only store keys
- Leaf pages are linked to each other
- Keys may be duplicated; every key to the right of a particular key is $\geq$ to that key

**B+ Tree Insertion**

- Insert at bottom level
- If leaf page overflows, split page and copy middle element to next index page
- If index page overflows, split page and move middle element to next index page

**B+ Tree Insertion Example**

Insert 5

```
  9, 16
    12
      1
      7
      3, 4, 6
    18
      9
      2, 7
```

Split page, copy 5

```
  9, 16
    12
      1
      7
      3, 4, 5, 6
    18
      9
      2, 5, 7
```

In this example, the B+ tree is inserted with the value 5.
**B+ Tree Insertion Example 2**

**Insert 17**

9, 13, 16

3, 4, 6 9 14 16, 18, 20

---

**B+ Tree Insertion Example 2**

**Insert 17**

9, 13, 16

3, 4, 6 9 14 16, 17, 18, 20

---

**B+ Tree Insertion Example 2**

**Split leaf page, copy 18**

9, 13, 16, 18

3, 4, 6 9 14 16, 17 18, 20

---

**B+ Tree Insertion Example 2**

**Split index page, move 13**

13

9 16, 18

3, 4, 6 9 14 16, 17 18, 20

---

**B+ Tree Deletion**

- Delete key and data from leaf page
- If leaf page underflows, merge with sibling and delete key in between them
- If index page underflows, merge with sibling and move down key in between them

---

**B+ Tree Deletion Example**

**Remove 9**

13

9 16, 18

3, 4, 6 9 14 16, 17 18, 20
B+ Tree Deletion Example

Remove 9

Leaf page underflow, so merge with sibling and remove 9

Index page underflow, so merge with sibling and demote 13

B+ Tree Deletion Example

Threaded Trees

- Binary trees have a lot of wasted space: the leaf nodes each have 2 null pointers
- We can use these pointers to help us in inorder traversals
- We have the pointers reference the next node in an inorder traversal; called threads
- We need to know if a pointer is an actual link or a thread, so we keep a boolean for each pointer

Example code:

```java
class Node {
    Node left, right;
    boolean leftThread, rightThread;
}
```

Threaded Tree Example
Threaded Tree Traversal

- We start at the leftmost node in the tree, print it, and follow its right thread
- If we follow a thread to the right, we output the node and continue to its right
- If we follow a link to the right, we go to the leftmost node, print it, and continue

Follow thread to right, print node

Follow link to right, go to leftmost node and print

Follow thread to right, print node

Follow link to right, go to leftmost node and print

Follow thread to right, print node

Follow link to right, go to leftmost node and print
Threaded Tree Traversal

Follow thread to right, print node

Node leftMost(Node n) {
    Node ans = n;
    if (ans == null) {
        return null;
    }  
    while (ans.left != null) {
        ans = ans.left;
    }  
    return ans;
}

void inOrder(Node n) {
    Node cur = leftmost(n);
    while (cur != null) {
        if (cur.rightThread) {
            cur = cur.right;
        }  
        else if (cur == leftmost(cur.right)) {
            cur = null;
        }  
    }
}

Threaded Tree Traversal Code

Follow link to right, go to leftmost node and print

Threaded Tree Modification

- We’re still wasting pointers, since half of our leaves’ pointers are still null
- We can add threads to the previous node in an inorder traversal as well, which we can use to traverse the tree backwards or even to do postorder traversals
Threaded Tree Modification