1 B+ Trees

- Same properties (order, children, balance, ...) as B-Trees with a few differences
- Items (data) with keys are placed only in leaf nodes
- Index values (keys) are placed by themselves in internal nodes
- All leaf nodes are linked (all nodes on the bottom level are in a list)
- B+ Trees are efficient for storage on disk – used in database systems

2 B+ Tree differences (from B-Trees)

- Since we have keys in internal nodes separate from leaf nodes, we know that there could be keys in internal nodes that could lead us to the actual data item with that key – we store items with key \( \leq \) to a key in its right child
- Not all keys in internal nodes actually correspond to data items still in the B+ Tree (although they probably once did)
- When inserting into a B+ tree, we insert into the bottom as we did for B-Trees, except when we split, we copy the key (but not the data) up from the leaf node to an internal node; if this causes another split, we push the key up from the internal node to the internal node above it
- When removing, we remove from the bottom (since data only resides at the bottom), and merge if needed (if the node becomes too small), throwing away the key that was in between the merged nodes; note that we don’t need to throw away keys from internal nodes even if we remove their corresponding data items
- Sometimes on deletion, B+ Trees do “redistribution” of items, but we don’t need to worry about that...

3 Threaded Trees

- All binary search tree traversals we’ve covered rely upon recursion or stacks
- BSTs waste a lot of space (many nodes, such as leaf nodes, waste two pointers that point to null)
- Why not use the extra null links to help with an inorder traversal, obviating recursion?
- At each node with a null right link (in a BST), we place a “thread” (this has nothing to do with concurrency) that links the item to its inorder successor
- We must indicate somehow that threads are different from regular pointers – use two booleans, one for the right, one for the left, to indicate whether the links are threads or not
- When traversing, we don’t need recursion – at the start, we follow all the left links to the bottom left, output the node, then follow its right link/thread
• If we follow a thread to the right, we output the node
• If we follow a link to the right, we go to its leftmost and continue the process
• No recursion or stack is required for this traversal (since the threads take us up the tree) and little extra space is used (only two bits per node extra)