COMP20003
Algorithms and Data Structures
Balanced Trees

Kris Ehinger
Department of Computing and Information Systems
University of Melbourne
Semester 2

AVL Trees

- Good features:
  - Tree is always reasonably balanced
  - Actually, height ≤ 1.44 $\log_2 n$
  - Therefore complexity for any search is O(?)
- Less ideal features:
  - Very fiddly to code, must keep track of
    - insertion path and
    - size of all subtrees
  - Balancing adds time (but constant time)


```
node* insert ( node* tree, node* new_node )
{
    if ( tree == NULL )
        tree = new_node;
    else if ( new_node->key < tree->key ) {
        tree->left = insert ( tree->left, new_node );
        /* Fifty lines of left balancing code */
    }
    else {
        tree->right = insert ( tree->right, new_node );
        /* Fifty lines of right balancing code */
    }
    return tree;
}
```


Other resources for AVL trees

Tutorial on AVL trees by A. Gunawardena, Carnegie Mellon Institute
http://www.youtube.com/watch?v=EsgAUiXbOBo (25 minutes)

- Interactive Demo!
  https://www.cs.usfca.edu/~galles/visualization/AVLtree.html
- Alternative:
  - https://people.ok.ubc.ca/ylucet/DS/Algorithms.html

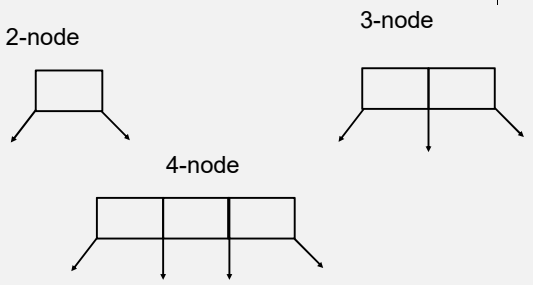
Balanced Trees (so far)

- AVL trees use rotation to keep the tree balanced
- Rotations are a general operation, used in other situations, not just AVL trees.

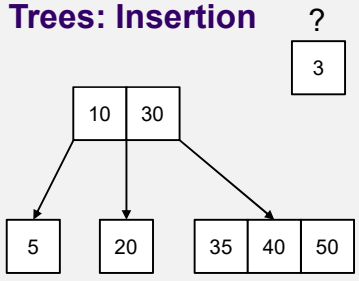
2,3,4-Trees: Overview

- Trees do not have to be binary!
- Nodes in 2,3,4-Trees have:
  - 1, 2, or 3 keys
  - 2, 3, or 4 pointers, correspondingly.
- Items are inserted *only into leaf nodes*
- When 4-nodes are full – split to accommodate new items.
- Tree height grow slowly

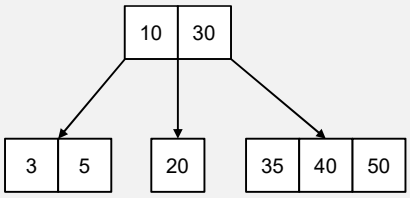
2-3-4 Tree nodes



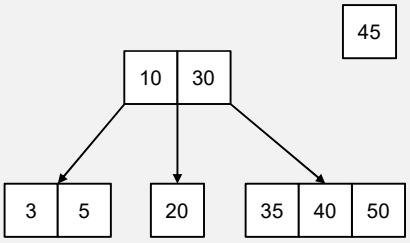
2,3,4 Trees: Insertion



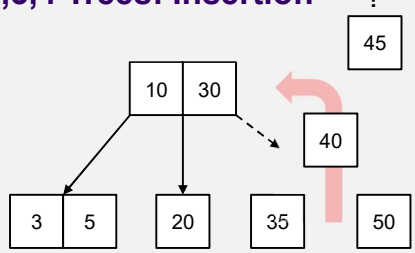
2,3,4 Trees: Insertion



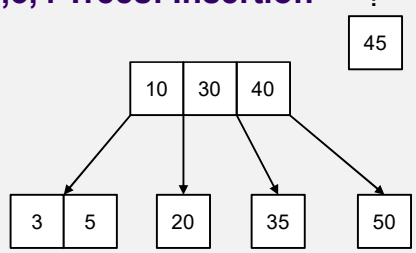
2,3,4 Trees: Insertion



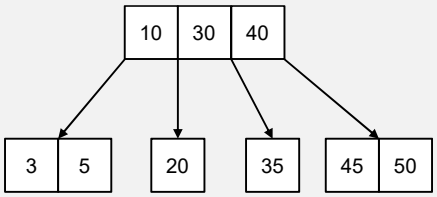
2,3,4 Trees: Insertion



2,3,4 Trees: Insertion



2,3,4 Trees: Insertion

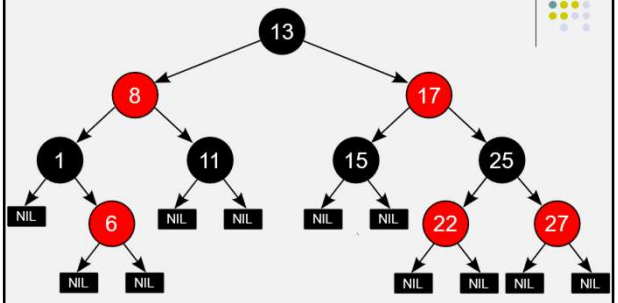


2,3,4-Trees

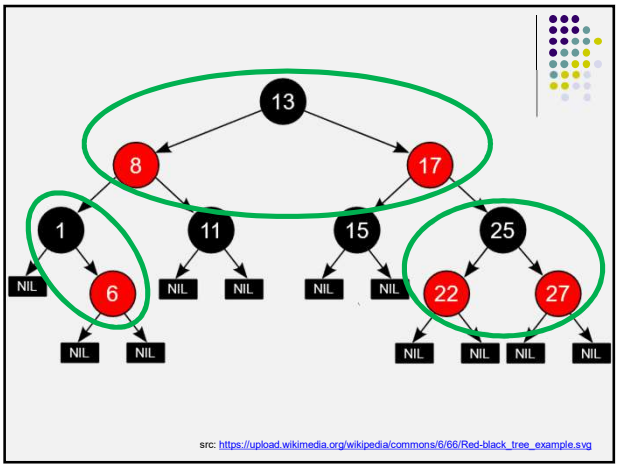
- Note that tree remains balanced *even when items are inserted in sorted order*.
- Height of tree: between $\log_4 n$ and $\log_2 n$
- 2-3-4 also known as B-trees of order 4

B+-Trees

- https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html
- B+-trees: generalization of the 2,3,4 tree
- Nodes have many pointers:
  - Typically 256-512
  - Depth of tree is $\log_{(\text{very large number})} n$
- Used for storing large databases on disk, where accesses are very expensive.
- Only leaf contain data, internal nodes are only intervals
- Leaves may include pointer to next leaf, to speed up sequential access



src: https://upload.wikimedia.org/wikipedia/commons/6/66/Red-black_tree_example.svg



Red-Black Trees

- Red-black trees implement a 2-3-4 tree as a binary search tree, using rotation to keep balance
- Beyond the scope of this subject
- An excellent description is found in Sedgewick, Algorithms in C, Parts 1-4, Section 13.4.

https://www.kernel.org/doc/Documentation/rbtree.txt

Access probability

- What if some items are searched much more frequently than others?
- Static optimization: adjust tree structure to shorten the path to more frequently accessed items
- What if you don't know the access probabilities in advance?

Splay Trees

- A splay tree is a self-adjusting tree
- Insertion:
  - Insert as for BST
  - "Splay" new node to the root
- Splay: do a series of rotations, that bring the node closer to the root

Splay Trees

- Search:
  - Search as for BST
  - "Splay" the searched node to the root

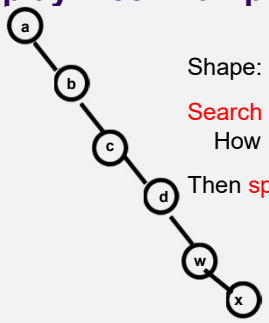
Note: might be O(n) search in a stick tree, but then splaying bushes out the tree

Splay Trees

Overall:
- A single search might take linear time.

BUT over time:
- The tree gets bushier.
- Highly accessed nodes are closer to the root

Splay Tree Example

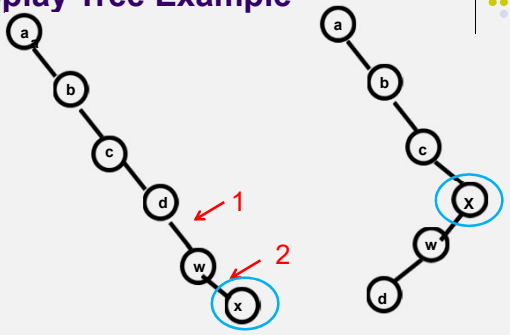


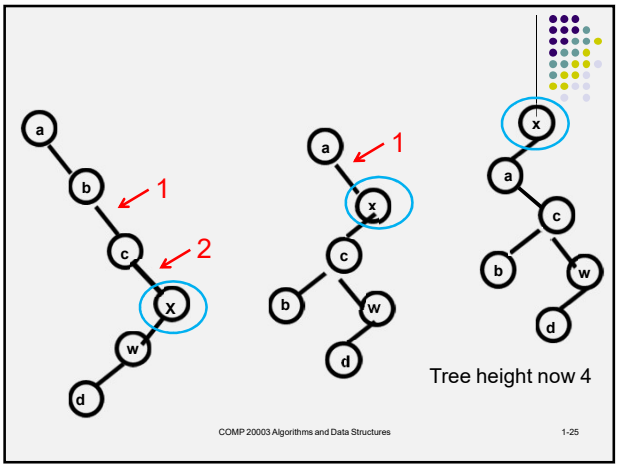
Shape: a stick, height 5

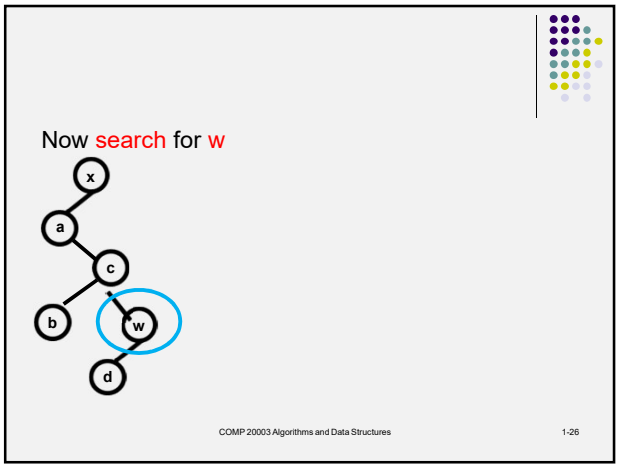
Search for node x:
How many comparisons?

Then splay x to root.

Splay Tree Example







Splay Trees

- Splay tree analysis:
  amortized over a series of searches
- Cope well with non-uniform access

Sleator and Tarjan, *Self-Adjusting Binary Search Trees*, *JACM* **32(3)**, 1985, 652-686.

Good things about balanced trees

Always relatively balanced for AVL, 2-3-4, B

On average balanced for splay trees

O(log n) search for AVL, 2-3-4, B.

Skip Lists: A Probabilistic Alternative to Balanced Trees

- Skip lists are lists pretending to be balanced trees
- They have excellent log n search behaviour,
- BUT, they are a probabilistic algorithm
  - There is an extremely high probability that a skip list search will complete in log n time
  - But there is always an infinitesimal probability of worst case linear behaviour

Skip Lists: A Probabilistic Alternative to Balanced Trees

More info and pseudocode:
- https://brilliant.org/wiki/skip-lists/
- http://ticki.github.io/blog/skip-lists-done-right/

Paper: Skip lists: a probabilistic alternative to balanced trees, W Pugh, Communications of the ACM 33 (6), 668-676, 1990