COMP20003
Algorithms and Data Structures
Balanced Trees

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

Binary search trees

- Good average case behavior – logn
- Bad worst case behavior – n
- So overall BST O(n).
  - Actual behaviour: trees usually are not linear
  - But they potentially can be linear
- Balanced trees: AVL, red-black; 2,3,4; B+tree.

Dictionaries: Summary

- We have looked at various underlying data structures for implementing dictionaries:

Dictionaries: Summary

- We have analyzed the computational complexity for these data structures:

Dictionaries: Summary

- So far the best we have done is log n search, where either:
  - Insertion is O(n); or
  - O(log n) average case but O(n) worst case.
- We can do better…

So far…

- Dictionary search with slow look-up or insertion:
  - Lists, sorted and unsorted
  - Array, unsorted
  - Sorted array has log n lookup, but n^2 build
- Binary search tree:
  - good average case, but very bad worst case.

Balanced trees

- Binary search tree:
  - Average case insertion and search: log n
  - Worst case for both: O(n)

Although simple, it's usually good enough, but not reliable

This section

- How to get a BST to stay balanced?
  - or almost balanced…
  - … no matter what order the data are inserted

Note: this material is not covered in Skiena.

It is essential knowledge for any computer scientist, however, and *is* examinable.

Balanced trees

- Idea: make BST perfectly (or almost perfectly) balanced
- In a balanced tree of *n* items, height is O(*log n*)
  - Perfectly balanced tree, height = log n, exactly
  - Balanced tree, height = O(log n).
- Therefore build a balanced tree is O(n log n)
  - Search is O(log n).

Balanced tree implementations

- AVL trees
- 2-3-4 trees
- B+ trees
- Red-black trees

Balanced Trees and Binary Search Trees

- In balanced trees, during insertion there are mechanisms for making sure the tree does not grow unbalanced
- At the same time, the BST ordering is preserved
- So, search in a balanced tree is exactly the same as binary tree
- The only difference is that it is O(log n)

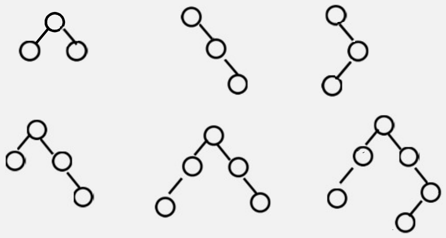
AVL Trees

The first balanced tree:

- Insert node + Keep track of height of subtrees of *every* node.
  - Balance node every time difference between subtree heights is >1.
  - Basic balancing operation: Rotation.

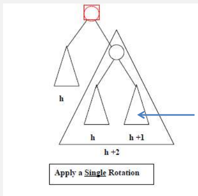
Adelson-Velskii, G.; E. M. Landis (1962). "An algorithm for the organization of information". Proceedings of the USSR Academy of Sciences **146**: 263–266. (Russian) English translation by Myron J. Ricci in Soviet Math. Doklady **3**:1259–1263, 1962.

Do these trees satisfy the AVL condition? Why / why not?

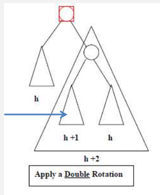


Non-AVL Trees caused by...

Outside insertion

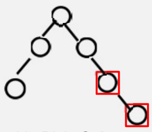


Inside insertion

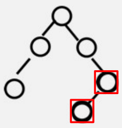


Symmetrical case is handled identically!

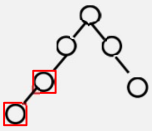
Unbalanced tree Categories



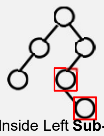
Outside Right **Sub-tree** Right **Child** (RR)



Inside Right **Sub-tree** Left **Child** (RL)

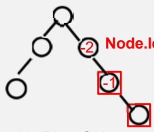


Outside Left **Sub-tree** Left **Child** (LL)



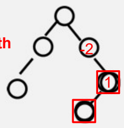
Inside Left **Sub-tree** Right **Child** (LR)

Unbalanced tree Categories

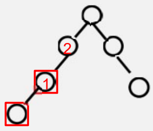


Outside Right **Sub-tree** Right **Child** (RR)

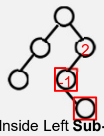
Counter = Node.left.depth – node.right.depth



Inside Right **Sub-tree** Left **Child** (RL)

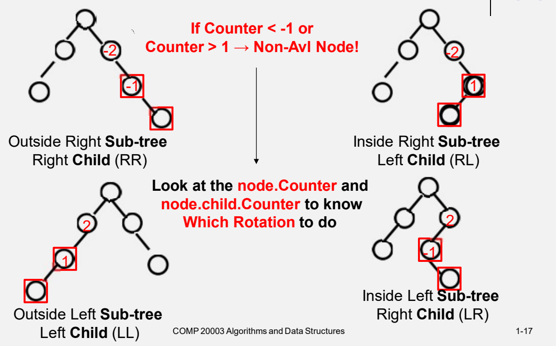


Outside Left **Sub-tree** Left **Child** (LL)

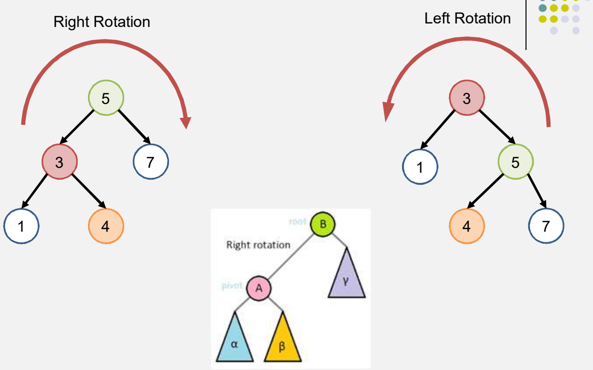


Inside Left **Sub-tree** Right **Child** (LR)

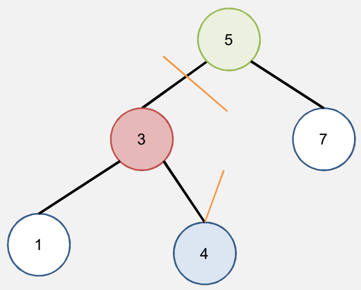
Unbalanced tree Categories



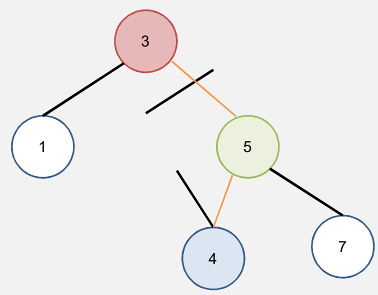
Rotation



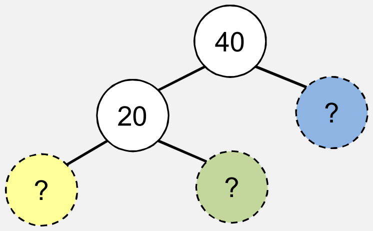
Right Rotation



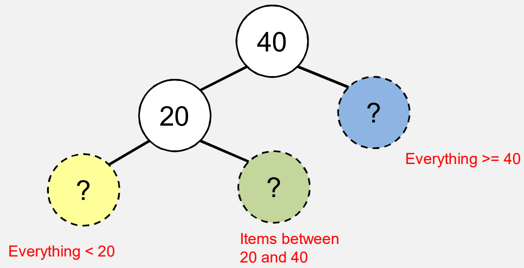
Left Rotation



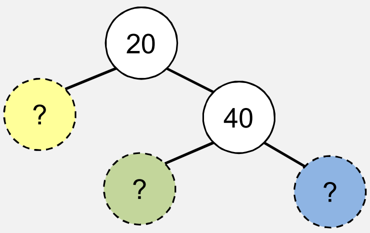
Preserving sorted order



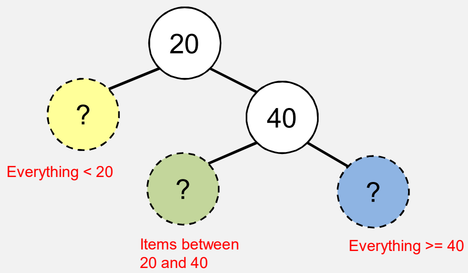
Preserving sorted order

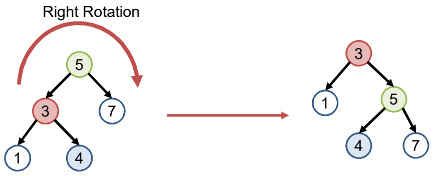


Preserving sorted order



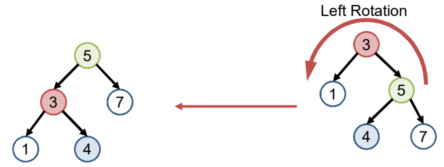
Preserving sorted order






```
RotateR(node)
{
    //Auxiliary variables
    left = node.Left;
    leftRight = left.Right;
    parent = node.Parent;

    //Operations
    left.Parent = parent;
    left.Right = node;
    node.Left = leftRight;
    node.Parent = left;
}
```

```
RotateR(5)
{
    //Auxiliary variables
    left = 3;
    leftRight = 4;
    parent = Null;

    //Operations
    3.Parent = Null;
    3.Right = 5;
    5.Left = 4;
    5.Parent = 3;
}
```

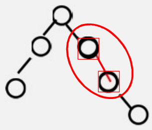




```
RotateL(node)
{
    //Auxiliary variables
    right = node.Right;
    rightLeft = right.Light;
    parent = node.Parent;

    //Operations
    right.Parent = parent;
    right.Left = node;
    node.Right = rightLeft;
    node.Parent = right;
}
```

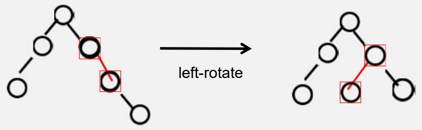
```
RotateL(3)
{
    //Auxiliary variables
    right = 5;
    rightLeft = 4;
    parent = Null;

    //Operations
    5.Parent = Null;
    5.Left = 3;
    3.Right = 4;
    3.Parent = 5;
}
```


How does rotation help balance a tree?

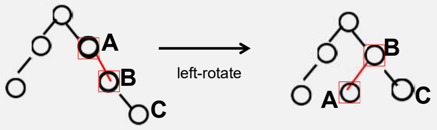
How does rotation help balance a tree?



How does rotation help balance a tree?

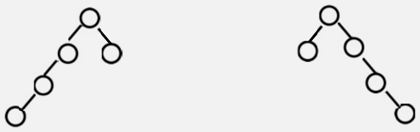


How does rotation help balance a tree?



- LeftLeft/RightRight-rotation (single) :
  - Take non-AVL node:
    - Rotate Child and node
    - Keep ordered subtree!

Which Rotation should we apply?

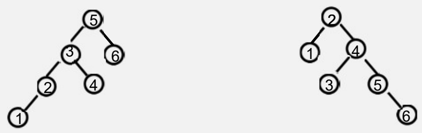


Excercise: Rotate? If so, do it...

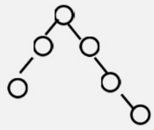
Quiz@piazza:

a) Tree1: rotate right and Tree2: rotate left
b) Tree1: rotate left and Tree2: rotate right
c) They are both already balanced

Excercise: Rotate? If so, do it…



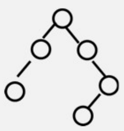
How does rotation help balance a tree?



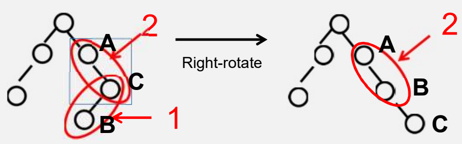
We have shown that: in these cases (LL/RR), Rotation rebalances the tree.

How does rotation help balance a tree?

What about in these cases (LR/RL)?



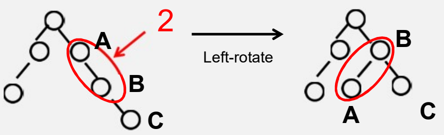
RL and LR: Double rotation



Right Left (RL) double Rotation:

- First rotation swaps Grandchild and child (Right Rotation)

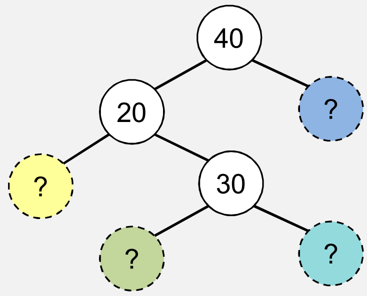
RL and LR: Double rotation



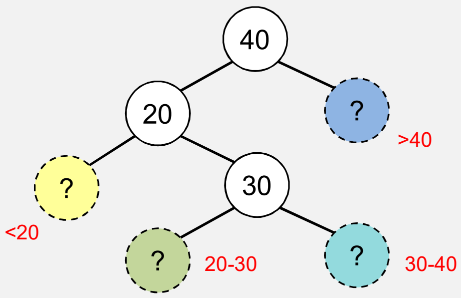
Right Left (RL) double Rotation:

- Second rotation swaps Parent and child (Left Rotation) , as before

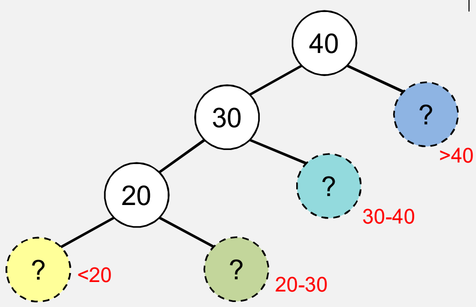
Preserving sorted order – double rotation



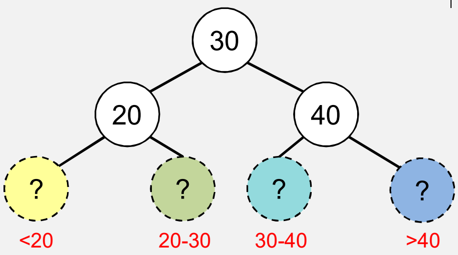
Preserving sorted order – double rotation

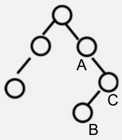


Preserving sorted order – double rotation

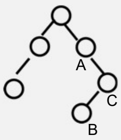


Preserving sorted order – double rotation

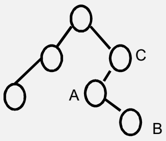




Why not just left rotate?



Why not just left rotate?



Excercise: Rotate? If so, do it...