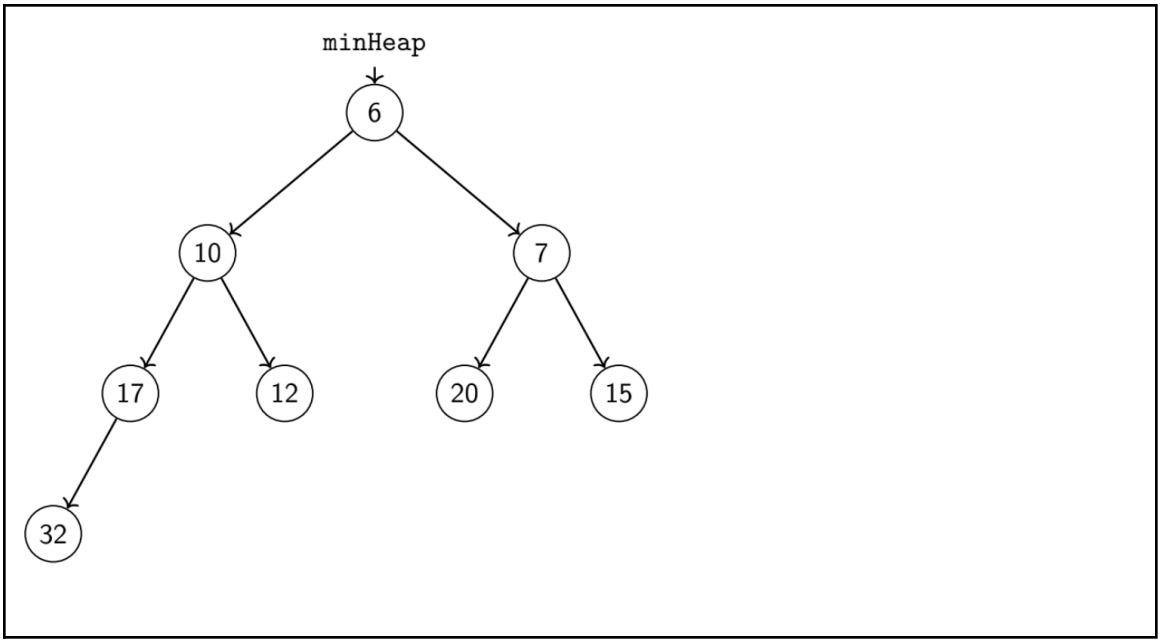
CSE 332: Data Structures and Parallelism

Section 3: Heaps + AVL Trees Solutions

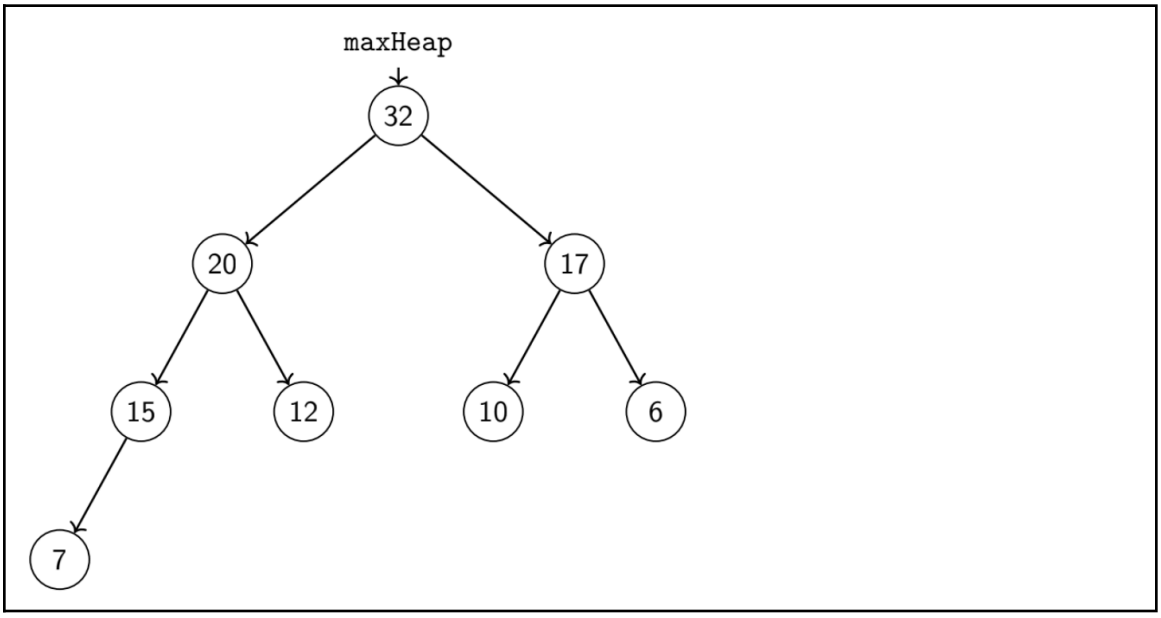
Part One: Heaps

0. Look Before You Heap

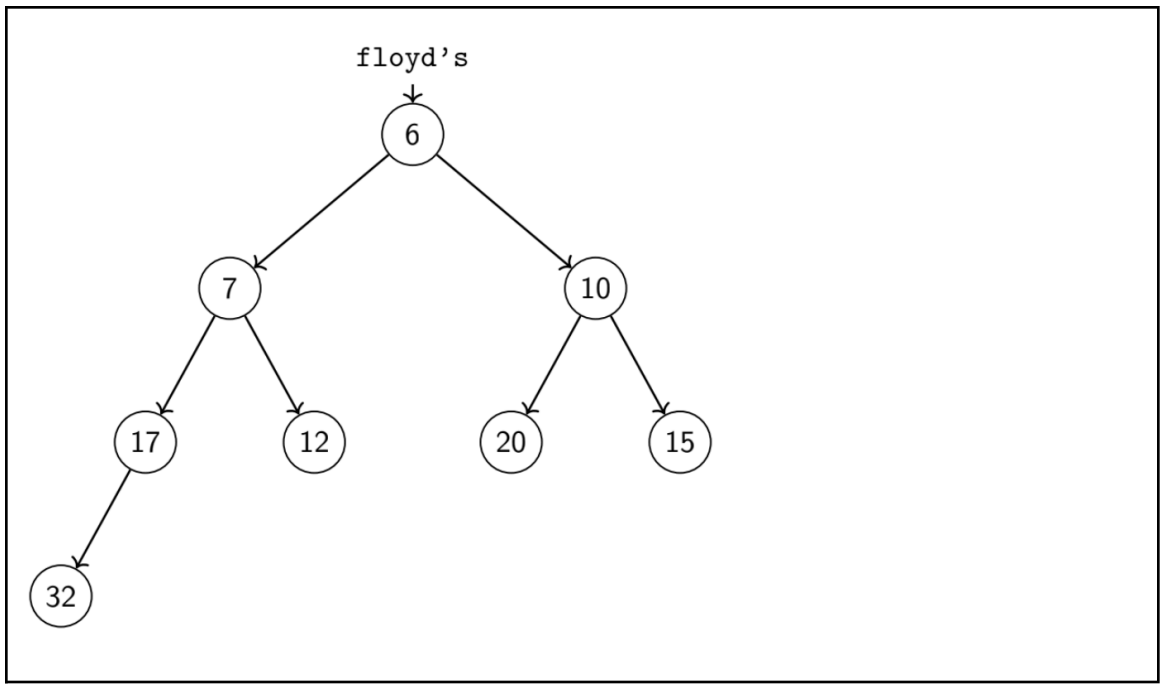
a) Insert 10, 7, 15, 17, 12, 20, 6, 32 into a *min* heap.



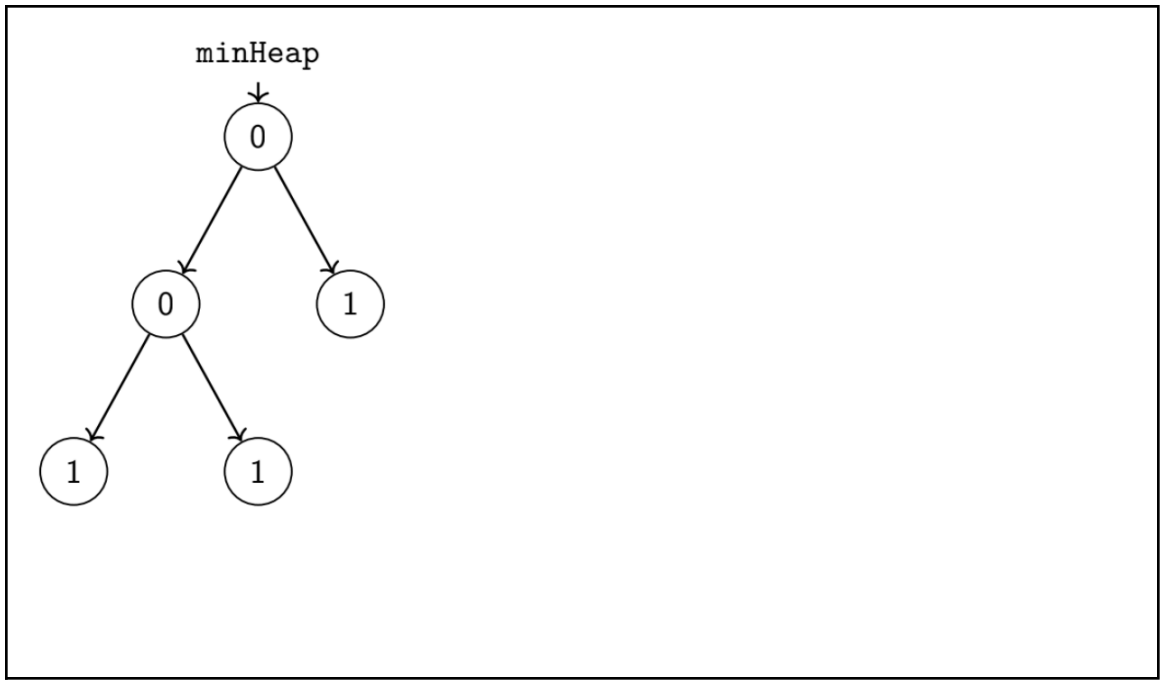
b) Now, insert the same values into a *max* heap.



c) Now, insert 10, 7, 15, 17, 12, 20, 6, 32 into a *min* heap, but use Floyd's `buildHeap` algorithm.



d) Insert 1, 0, 1, 1, 0 into a *min* heap.



Part Two: AVL Trees

0. The ABC's of AVL Trees

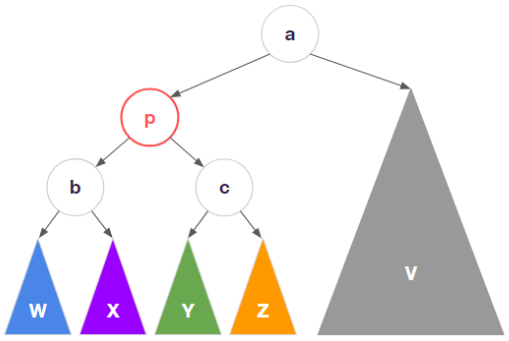
What are the constraints on the data types you can store in an AVL tree? When is an AVL tree preferred over another dictionary implementation, such as a HashMap?

AVL trees are similar to TreeMaps. The constraint is that they require that keys be comparable. The value type can be anything, just like any other dictionary.

A perk over HashMaps is that keys can be iterated over in sorted order. AVL trees are also preferred over BSTs when there's a possibility of sorted input because the balancing prevents the worst case of a degenerate tree.

1. Left Rotation References

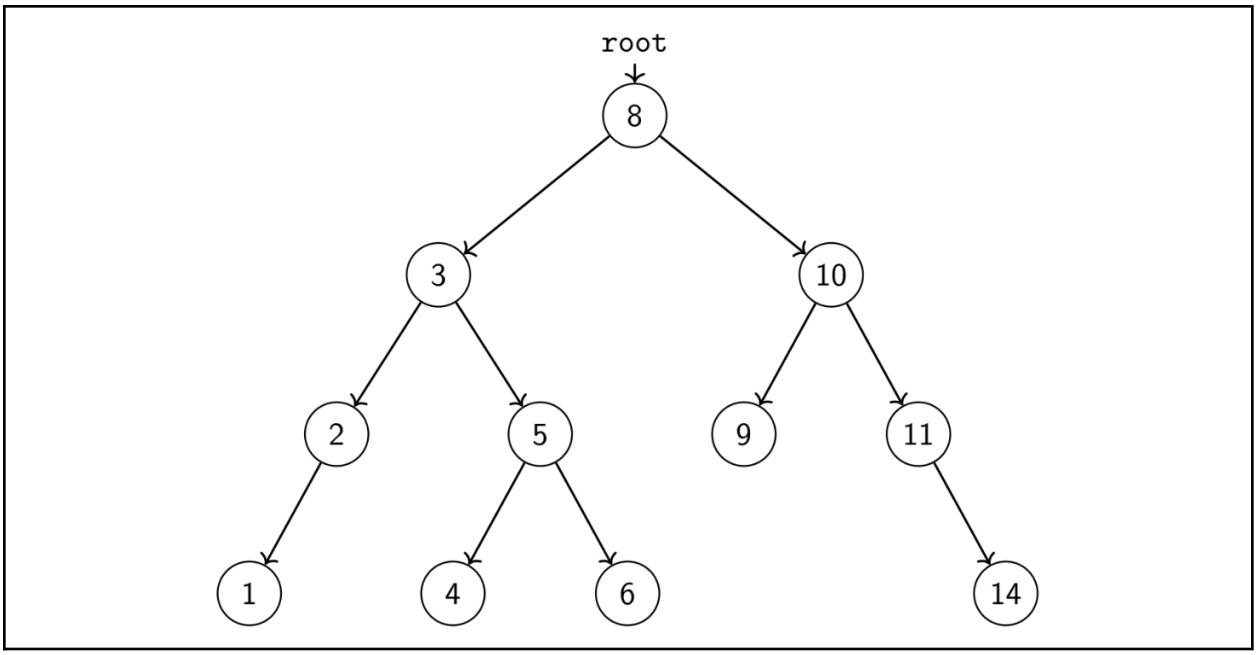
Suppose we do a left rotation on the tree below from node p. Write out what references need to change and what they need to change to.



a.left=c
p.right=c.left
c.left = p

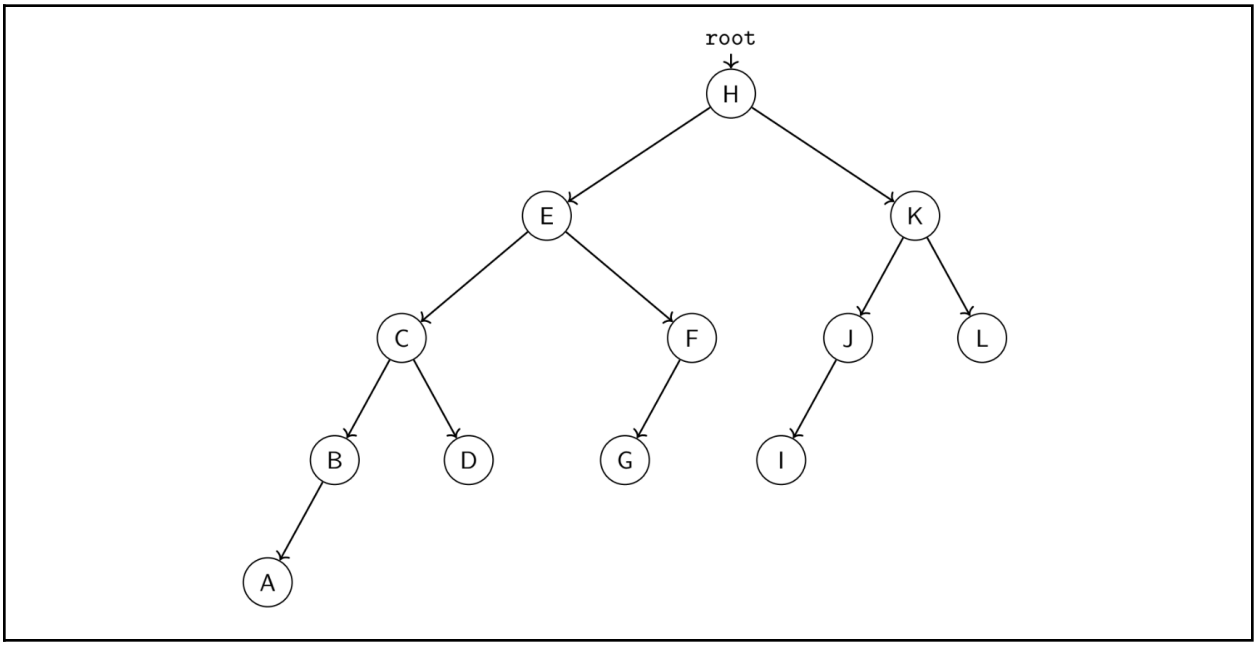
2. Let's Plant an AVL Tree

Insert `10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14` into an initially empty AVL Tree.



3. MinAVL Trees

Draw an AVL tree of height 4 that contains the minimum possible number of nodes.



4. AVL Trees

Insert `6, 5, 4, 3, 2, 1, 10, 9, 8, 7` into an initially empty AVL Tree.