CSE 100: AVL TREES, ROTATIONS AND TREAPS
Announcements

• PA2 is released
  • Checkpoint deadline next Monday 8pm
  • Due next Friday, 8pm
  • You can get started now, though you need the reading through Wednesday
Which of the two heap properties (refer to section 6.9) is maintained in a treap?

A. The value of each node is greater than or equal to the values stored in each of its children

B. The tree is perfectly balanced, and the leaves in the last level are all in the leftmost positions.

C. The tree is maintained as an array, where elements are placed at sequential locations representing the nodes from top to bottom and in each level from left to right.
What are the two elements associated with each node in a treap?

A. A level and a depth
B. A priority and a data key
C. A balance factor and a value
How are rotations used when inserting a node into a treap?

A. To rebalance the tree
B. To restore the heap property
C. To fix violations of the binary search tree ordering
Single rotation practice

Insert 50. Draw the resulting AVL tree. (Don’t peek)
Single rotation practice

After insertion
Single rotation practice

After rotation
Single rotation is not enough

What happens if we insert 66?
Single rotation is not enough

Why won’t a single rotation work? Try it.
Single rotation is not enough

![Diagram showing the problem with single rotation](image-url)
Single rotation is not enough

UH OH!
Single rotation is not enough

Reattaching 66 here will always work with respect to the BST properties, and we know that 66 will always fit here because 60 used to be 70s left child.

The problem is that this won’t fix the balance issue!
Double rotation to the rescue

Single rotations only work to balance the tree when involved nodes are “in a line”. That is, the balance factor of the node and its large subtree are both positive or both negative. This is not the case here. So we will first rotate left around 60, then we can rotate right around 70.
Double rotation to the rescue

Single rotations only work when involved nodes are “in a line”
So we will first rotate left around 60, then we can rotate right around 70.
Single rotations only work when involved nodes are “in a line”
So we will first rotate left around 60, then we can rotate right around 70.
Double rotation to the rescue

Single rotations only work when involved nodes are “in a line”
So we will first rotate left around 60, then we can rotate right around 70.
Double rotation to the rescue

Single rotations only work when involved nodes are “in a line”
So we will first rotate left around 60, then we can rotate right around 70.
Double rotation to the rescue

Single rotations only work when involved nodes are “in a line”
So we will first rotate left around 60, then we can rotate right around 70.

Where in the tree above should I cut to start the second rotation?
Double rotation to the rescue

Single rotations only work when involved nodes are “in a line”
So we will first rotate left around 60, then we can rotate right around 70.
http://www.cs.hmc.edu/~oneill/BST-Applet/avltree.html
It’s sometimes even more complicated

Insert 64… do we need a double or a single rotation?

A. Double
B. Single
C. No rotation needed
Rotate right around 66 to make a straight line
Rotate right around 66 to make a straight line
Rotate right around 66 to make a straight line

UH OH! Where do we put 64?? Are we stuck?
Rotate right around 66 to make a straight line

Will 64 always reattach there?

A. No, sometimes this doesn’t work
B. Yes, this will always work
Finishing the rotation to balance the tree

Study the book for the general case of AVL rotations
Trees + Heaps = Treaps!

Treaps maintain the following properties:
- Data keys are organized as in a binary search tree
  - All nodes in left subtree are less than the root of that subtree, all nodes in right are greater
- Priorities are organized as in a heap
  - The priority of a node is always greater than the priorities of its children.
Is the following statement true or false?

There is exactly one treap for a given set of key, priority pairs where all keys and priorities are unique.

A. True
B. False

(Hint: How many possible nodes could be at the root? Now recurse…)
Treaps are not necessarily balanced!

A bad match between keys and priorities can lead to a very imbalanced treap.

We will look at ways to ensure this doesn’t happen on average… when we discuss RSTs.