CSE 100: AVL TREES, ROTATIONS AND TREAPS
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.
Treap insert

Insert (F, 40)
Step 1: Insert using BST insert
Step 1: Insert using BST insert
Step 2: Fix heap ordering

Idea 1: “Bubble” F up until it’s in the right place…
Treap insert

Step 1: Insert using BST insert
Step 2: Fix heap ordering

Idea 1: “Bubble” F up until it’s in the right place…
Treap insert

Step 1: Insert using BST insert
Step 2: Fix heap ordering

Idea 1: “Bubble” F up until it’s in the right place… Does this work?
A. Yes
B. No
AVL rotation to the rescue!

Step 1: Insert using BST insert
Step 2: Fix heap ordering

Insert (F, 40)
AVL rotation to the rescue!

Step 1: Insert using BST insert
Step 2: Fix heap ordering
AVL rotation to the rescue!

Step 1: Insert using BST insert
Step 2: Fix heap ordering

Insert (F, 40)
AVL rotation to the rescue!

Do we need a double rotation now?
A. Yes, because C, F and E are not “in a line”
B. No, because we don’t care about the tree staying in balance and double rotations were used in AVL trees to fix balance issues
AVL rotation to the rescue!
Graphical depiction of the general case of a single rotation

Rotate left
AVL rotation to the rescue!
AVL rotation to the rescue!

rotate
AVL rotation to the rescue!

reattach
How would you delete in a treap? Rotate down!

Which node would I rotate C with to delete it?
A. Node E  
B. Node B  
C. Node F  
D. Node C
How would you delete in a treap? Rotate down!

Delete C
How would you delete in a treap? Rotate down!

Delete C
How would you delete in a treap? Rotate down!

Delete C
How would you delete in a treap? Rotate down!

Delete C
Why Treaps?

• Treaps are worth studying because...
  • they permit very easy implementations of split and join operations, as well as pretty simple implementations of insert, delete, and find
  • they are the basis of randomized search trees, which have performance comparable to balanced search trees but are simpler to implement
    • they also lend themselves well to more advanced tree concepts, such as weighted trees, interval trees, etc.

• We will look at the first two of these points
Tree splitting

- The tree splitting problem is this:
  - Given a tree and a key value K not in the tree, create two trees: One with keys less than K, and one with keys greater than K

Insert (K, Inf)
Try with K='K'

Based on tree property (K,00) will be root. This naturally splits the tree into the left subtree & right subtree of (K,00)
Tree splitting

- This is easy to solve with a treap, once the insert operation has been implemented:
  - Insert \((K, \text{INFINITY})\) in the treap
  - Since this has a higher priority than any node in the heap, it will become the root of the treap after insertion
  - Because of the BST ordering property, the left subtree of the root will be a treap with keys less than \(K\), and the right subtree of the root will be a treap with keys greater than \(K\). These subtrees then are the desired result of the split

- Since insert can be done in time \(O(H)\) where \(H\) is the height of the treap, splitting can also be done in time \(O(H)\)
- (yes, this same idea could be used in an ordinary BST as well...)

How could you do a join? For the answer, see [http://cseweb.ucsd.edu/users/kube/cls/100/Lectures/lec5/lec5.pdf](http://cseweb.ucsd.edu/users/kube/cls/100/Lectures/lec5/lec5.pdf)
Randomized Search Trees

- Randomized search trees were invented by Cecilia Aragon and Raimund Seidel, in early 1990’s
- RST’s are treaps in which priorities are assigned randomly by the insert algorithm when keys are inserted
- To implement a randomized search tree:
  - Adapt a treap implementation and its insert method that takes a (key,priority) pair as argument
  - To implement the RST insert method that takes a key as argument:
    - call a random number generator to generate a uniformly distributed random priority (a 32-bit random int is more than enough in typical applications; fewer bits can also be made to work well) that is independent of the key
    - call the treap insert method with the key value and that priority
- That’s all there is to it: none of the other treap operations need to be changed at all
- (The RST implementation should take care to hide the random priorities, however)
Analysis of RSTs

- How many steps are required, on average, to find that the key you’re looking for is in the tree? (Average case analysis of a “successful find”)

- You can read about this here: http://cseweb.ucsd.edu/users/kube/clsf100/Lectures/lec5/lec5.pdf

- Punch line: The average number of comparisons for a successful find in an RST is exactly the same as the average number of comparisons in a BST!

\[
D_{avg}(N) = \frac{2(N+1)}{N} \sum_{i=1}^{N} \frac{1}{i} - 3
\]

So what have we gained by using an RST??
BST Probabilistic Assumptions

Which of the following is/are the probabilistic assumptions we made in our average case successful find in a BST?

A. All keys are equally likely to be searched for
B. The tree is approximately balanced
C. All orders of data are equally likely to occur
D. A&B
E. A&C

This assumption is not needed for BSTs
RST Probabilistic Assumptions

• Which of the probabilistic assumptions from the BST is NOT included in the RST analysis?

A. All keys are equally likely to be searched for
B. All orders of data are equally likely to occur

Why not?
What second assumption is it replaced with? Why is that important?
Next time:

Go through average case find in RSTs (similar to what we did with BSTs)

- I won't ask you details about this proof on exams, but going through the proof might help you better understand the BST analysis...