HOMEWORK 2

- Theory is due tomorrow
- Flags clarifications
COMPUTER SCIENCE TREES

- An ordered directed acyclic graph where each node has at most 1 parent and zero or more children.
Search Trees

Question:
- Do search trees always provide $O(\log n)$ performance? Explain or provide counterexample
SEARCH TREES

Question:
- Do search trees always provide $O(\log n)$ performance? Explain or provide counterexample

Answer:
- No!
  - e.g., Inserting elements in order results in depth $O(n)$
- Search tree must be self-balanced
  - i.e., depth of $O(\log n)$
SEARCH TREES

Question:
- In what ways can we keep trees balanced?
SEARCH TREES

Question:
- In what ways can we keep trees balanced?

Answer:
- Transformations that modify the structure of the tree while preserving its nature as a search tree
- Different strategies
  - e.g., AVL trees, B-trees, Splay trees, Red-Black trees
  - Some require bookkeeping and additional memory
  - Some ensure $O(\log n)$ amortized rather than worst-case
  - Some are better adapted for non-uniform access
TREES HEAP TRIES

- Other Trees
  - B-Tree, 2-3-4, Trees
  - KD-tree, Range Tree

- 211 Trees
  - BST – Basic tree
  - AVL – Good only for theoretical purposes
  - Splay – Bad for everything other than search
  - RB – Good general purpose tree
  - Tries
  - Suffix, Prefix trees
AVL Trees
- They exist to force a search tree to be balanced
- AVL condition:
  - For any node $n$, $|\text{height}(n.\text{left}) - \text{height}(n.\text{right})| \leq 1$
  - This is not the easiest condition to maintain, but it guarantees a balanced tree
- Maintain the condition via rotations
  - These rotations are useful for other trees as well


**Trees**

- Rotation
  - Single rotation—this is the only one you have to remember!
  - Double rotation—it is just two single rotates in a specific order
  - Let’s do a few examples...

![Diagram showing single and double rotations in trees](image)
**Trees**

- These rotations will also be useful for:
  - Splay trees
  - Red-black trees if you ever use them
**Splay Trees**

- Self balanced tree
- Only one balancing operation: “splay element”
  - Repeated rotations move element towards the root
    - Efficient for repeated access to the same element
      - Like a cache
SPLAY TREES

- Let’s consider implementing find(k).
- The idea is simple, move k to the root.
- Let’s say we have a function to do something like this, called move-to-root(k).
MOVE-TO-ROOT

- Want to move-to-root(5)
- Just start at 5 and keep moving it up
MOVE-TO-ROOT

- Rotate about the 5-4 edge
**Move-to-root**

- Rotate about the 5-3 edge
MOVE-TO-ROOT

- Rotate about the 5-2 edge
MOVE-TO-ROOT

• Rotate about the 5-1 edge
MOVE-TO-ROOT

• Rotate about the 5-0 edge
**Move-to-root**

- Move-to-root is a “self-adjusting” heuristic.
- Suppose that every time we searched for a node $x$, we called $\text{move-to-root}(x)$.
**Move-to-root**

- Before we get ahead of ourselves, we have to remember the point of these structures.
- We want to get amortized $O(\log n)$ across the board for insert, find, and delete.
- Does it work?
- No, there are sequences of requests that cycle through all he vertices, which is $\Omega(n^2)$ for $n$ searches, giving $\Omega(n)$ per search.
Imagine accessing the elements in this order: 5, 4, 3, 2, 1, 0.

It would be $\Omega(n^2)$
**BOTTOM-UP SPLAY TREES**

- Key Idea: Instead of going up one step at a time, we should go up 2 steps at a time.
- Let’s create a new operation: splay($k$)
- As before, this operation will move $k$ to the root.
- We’ll consider three cases:
  - Zig
  - Zig-Zag
  - Zig-Zig
**Bottom-up Splay Trees: Splay(x)**

- Zig operation (used if path length is odd).
- Operation is same as before
  - Just move x to the root
**BOTTOM-UP SPLAY TREES: SPLAY(x)**

- Zig-Zag operation
- Operation is same as before
  - Move x to the root.
  - Then repeat.
**Bottom-up Splay Trees: Splay(x)**

- Zig-Zig operation
- Operation is different.
  - Rather than move x along the x-y edge and then again along the x-z edge.
  - We move y along the y-z edge and then move x along the x-y edge.
**Bottom-Up Splay Trees: Splay(x)**

- Zig-Zig operation
SPLAY TREE EXAMPLE

Splay(5)
Splay Tree Example

Splay(5)

Zig-Zig case
Splay Tree Example

Splay(5)

Zig-Zig case
SPLAY TREE EXAMPLE

Splay(5)

Zig case
SPLAY VS. MOVE-TO-ROOT

- Remember before, when we tried move-to-root(5), we got:

![Move-to-root](image1)

![Splay](image2)

VS
Splay Example

- Try this one on your own: Splay(4)
SPLAY EXAMPLE

- After a zig-zag, zig-zig, and a zig, we get:
SPLAY TREES

- We just talked about bottom-up splay trees, because we start from a node and splay it up.
- In reality, bottom-up splay trees requires 2 traversals to find and splay a node, and it also requires you to maintain parent pointers.
- Top-down splay trees splay as you traverse down the tree and are usually what’s implemented in practice.
- So how do we eliminate this extra traversal?
INSERTION IN SPLAY TREES

Question: How do we implement “insert (X)”?
**INSERTION IN SPLAY TREES**

- **Question:** How do we implement “insert (X)”?
- **Answer:**
  - Splay X
    - Splaying operates on an element or on the closest existing lower or higher value
  - If the root element is X, no need to re-add
  - Make X into the new root
    - Place the whole tree to the left or to the right of the new root
DELETION IN SPLAY TREES

- Question: How do we implement “delete (X)”?
DELETION IN SPLAY TREES

Question: How do we implement “delete (X)”?

Answer:
- Splay X
- If the root element is not X, abort
- If one child of root is null, remove root and making other child into root
- Splay largest key of left-branch (or smallest key of right-branch)
  - Can be done by splaying the non-existing element
  - New left-branch will have has empty right-branch
    - since we just splayed the largest value...
  - Set left-branch tree's missing right-branch to be the right-branch tree
Splay Tree

- From the examples we’ve done, it looks like splay trees perform better, but are they asymptotically $O(\log n)$ for insert, find, and delete?
SPLAY TREE

- From the examples we’ve done, it looks like splay trees perform better, but are they asymptotically $O(\log n)$ for insert, find, and delete?
- Before answering, let’s remember what we mean by asymptotically.
- We can see that some inserts, finds, and deletes, could require $O(n)$ steps, but the idea is that $n$ operations, don’t take $O(n^2)$ time, but $O(n \log n)$ time.
**Splay Tree**

- From the examples we’ve done, it looks like splay trees perform better, but are they asymptotically $O(\log n)$ for insert, find, and delete?
- Yes! Thereom: A sequence of $M$ splay operations on a tree of $N$ nodes takes time $O(M \log n)$. (Assuming $M > n$)
- Proof: Beyond the scope of this course =(
HAVE A GOOD (LONG) WEEKEND