1 Skip Lists

Unique, probability-based data structure

Structure
Series of linked lists, each on a different level
Bottom level contains all the data
Each node has 4 pointers:
(left/right for nodes like in a normal linked list)
(up/down to point to linked list above and below)
Each level is in ascending order
Each level starts with a phantom node containing -∞, and can end with a phantom node containing +∞
Head pointer points to upper left -∞ node.
The contents of each level are based on a coin flip.
Bottom: n nodes; 2^{nd} level: $n/2$ nodes; 3^{rd} level: $n/4$ nodes ...

Seaching
Start at head
On a given level:
  iterate through LinkedList
    if you find data 1, stop
    if you find node containing data > (data you are looking for), you know data you're looking for doesn't exist on that level.
    use the previous node's down pointer to the next level
    reiterate on this level.

Adding
Flip a coin repeatedly.
  if you get Heads, flip again and stop when you get Tails
  n heads = n of levels above the bottom level
  add empty levels to top if necassary (number of levels is capped at $\log n$)
  traverse the skip list to find where to add the node at bottom level
    add the node immediately before the 1^{st} node greater than the data
    after adding at the bottom, promote node to above levels based on the amount of heads.

Removing
Find the data to remove
If the data is found, remove from the level, drop down and repeat.

Efficiency

	average case	worst case
adding removing searching	$O(\log n)$	$O(n)$

Space complexity is $O(n)$ ($\sum_{i=0}^{\infty} \frac{1}{2^i}$) on average, and $O(n \log n)$ at worst.

2 AVL Trees

Binary trees that follow the order property and are height-balanced.

When a tree is height-balanced, all operations (add, remove, get) are $O(\log n)$.

This avoids the degenerate case for BSTs (in which they become basically a "linked list").

Properties

Nodes: 4 fields (left, right, height, balance factor)

Height: height of a node is max(height of left, height of right) + 1

Balance factor: height(left) $-$ height(right). A tree is balanced if $|bf| = 1$.

Pointer reinforcement (continued)

Traverse like you are looking for the onde, but children pointers:

```
node.left = add(node.left, data)
```

At each recursive call, return the corrected node.

In a linked list, for example:

```
public void removeFirstOccurrence(T data) {
    head = removeFirstOccurrence(head, data);
    //To return T data from the removed node, use a dummy node.
}

public Node<T> removeFirstOccurrence(Node<T> curr, T data) {
    if (curr == null) {
        throw new java.util.NoSuchElementException("Data not found");
    } else if (curr.data.equals(data)) {
        --size;
        return curr.next;
    } else {
        curr.next = removeFirstOccurrence(curr.next, data);
        return curr;
    }
}
```