

# Midterm 2 Studying

\*\*

## Singly-linked list

- $O(1)$  insertion time
- Corner case: delete at head
- Think about head and tail corner cases
- Vanilla case: delete after head
- Visit every node until some condition is met
- Linked list empty if both head and tail are null
- Careful of loops with node references
- Node data is different from node

## Doubly-linked list

- Each node has next and previous pointer
- Appending a node to a doubly-linked list
- A corner case of insertAfter (inserts node after provided node)

## Stack

- Last-in first-out
- LIFO
- Pushed or popped from top of stack
- Implemented using
- Linked lists
- Arrays
- ArrayLists
- Peek returns top item
- Methods return relevant item
- JVM has “call stack”

## Queue

- First-in first-out
- FIFO
- Only add to bottom
- Only remove from top

## Graphs



- Tree
- No loops/cycles
- Unlimited edges from any node
- Exactly 1 edge into any node
- Except the root node
- 0 edges into the root
- BINARY TREE IS A TYPE OF SEARCH TREE
- Leaf nodes have no children
- Internal nodes are not leaf nodes and not the root
- Have children but does not includes the root
- So must be a child and have a child
- Subtree at ...
- Depth-first search
- Start at root
- Explore as far as possible on each branch before backtracking
- Could keep track with list of "visited" nodes

### Tree integrity

- Exactly 1 root
- Exactly 1 edge into all nodes except root
- Count number of parents per node
- Count number of times that a node has been a child
- Corner case I
- Tree has node with multiple parents
- Tree has a cycle
- Collect all nodes
- Use hashset, only allows unique items

### Hash tables

- Unordered items stored by mapping each item's key to a location (bucket) in an array (or vector)
- A hash function maps the universe of keys to the slots in the hash table
- Takes  $O(1)$  time for search/insertion/removal
- Array or linked list takes  $O(n)$
- Bst takes  $O(\log n)$
- Collisions
- Chaining
- Array of linked lists
- Each list may store multiple items

- Open addressing
- Keep the hashtable as just an array of elements
- Hashing
- A fast efficient strategy for locating objects in memory
- $\text{hashCode()} = x \% 7$
- 6 slots from 0 to 6
- Remainder determines the slot
- Direct hashing
- Uses items key as bucket index
- Ex of key is 700, index is 700
- Table size equals largest key value plus 1
- Linear probing
- Linearly checks each bucket
- Inserts the item in the next empty bucket
- If reaches last bucket, then continues at bucket 0
- Insert algorithm returns true if inserted, false if all buckets occupied
- 
- 
- Removals using linear probing
- Key determines initial bucket
- Probes each bucket until matching item found
- Item removed and bucket marked empty after removal (ghost)
- Or if an empty-since-start bucket is found
- Return without removal
- Hash table resizing
- Commonly resized to the next prime number greater than or equal to  $N * 2$
- New array is allocated, everything is reinserted
- Time complexity
- $O(n)$  per resizing
- But amortized time complexity per insertion is only  $O(1)$
- When to resize

## • **items / # buckets**

- $(\# \text{ items} + \# \text{ ghosts}) / \# \text{ bucket}$
- Open-addressing


## • **of collisions during an insertion**

- Chaining
- Size of bucket's linked list
- Linear probing problem
- Primary clustering

- Long rungs of occupied slots tend to build up and these tend to grow
- Quadratic probing
- Double hashing
- Uses collisions
- $h_2(k)$  or whatever function multiplied by number of collisions
- So if only  $h_1(k)$  then no collisions
- Open addressing run time
- Worst case  $O(n)$  - probe sequence visits every full entry before finding an empty entry
- Avg case: at least 1 probe
- Running time of insert and search for open addressing
- $1 / (1 - \alpha)$
- A hashtable should be around half full
- Class hashset uses a hashtable
- Very fast add, remove, and contains
- Usually  $O(1)$
- Deep equality
- Hashcodes
  - ability to be managed by blackbelt collections (hash set/map)
  - ability to be sorted by Jedi collections (tree set/map)
- Shallow equality is `==` sign (true if same bit pattern, compares on the stack)
- Override equals otherwise you get behavior defined in object, which is `this == Object x`
- Equals hashCode contract
- Collisions should be rare

## Trees

- No loops/cycles
- Unlimited edges from any node
- Exactly one edge into any node
- Except the "root" node
- 0 edges into the root node
- Binary trees
- Each node has up to 2 children; left and right child
- Ancestors reach up till tree root
- Depth: # of edges on the path from root to node T
- If only root then depth 0
- Full if each node has 0 or two children
- Complete if
  - All levels except possibly the last level contain all possible nodes
  - And all node in the last level are as far left as possible

- Perfect if all internal nodes have 2 children and all leaf nodes are at the same level
- Tallest n-node binary tree is  $H = n - 1$
- Shortest n-node tree is  $H = \log(\text{base}2)N$
- Binary search tree
- Each node has 0 1 or 2 children
- Left subtree is less, right subtree is greater
- Smallest is leftmost, largest is rightmost
- Edge cases
- Loner (one node)
- The twig (all edges in one direction)
- Or edges in both directions
- No additional children either way
- Traversal
- In order
- Root -> smallest left node -> up one left -> down to the right one level -> smallest right node -> up one to the left -> end at root
- 
- Level order traversal
- Level by level, left to right
- Given sorted list
- A,B,C,D,E,F,G,H,I,J,K,L,...
- Node E is the successor of D
- Node E is the predecessor of F
- The successor of the last node (Z) is NIL
- The predecessor of the first node (A) is NIL
- Successor and predecessor must each have 1 or more children

## Quizzes

- Code runtime
- Break things down into individual operations
- Strip constants and identify highest-order term
- A time complexity is  $O(1)$  because the operations only happen once, and they do not depend on the size of the input as they run
- A normal for loop has a  $O(n)$  time complexity
- A normal nested for loop has a  $O(n^2)$  time complexity
- $O(n \log n)$  would be a normal outer loop, then an inner loop that keeps cutting in size every time it is run
- The range of the height of an n-Node binary tree is between  $O(\log n)$  and  $O(n)$
- Tree integrity (assume tree T has N nodes and m edges)

- No loop
- $M = N - 1$
- A singly linked list is a tree
- Binary search tree
- Empty BST, insert each of  $N$  numbers into  $T$  in any order: between  $O(N \log N)$  and  $O(N^2)$
- In-order traversal and then print in ascending order:  $O(N)$
- Given  $N$  numbers in the descending order, the tree sort will take  $O(N^2)$
- In the best case, the tree sort will take as little as  $O(N \log N)$

\*\*