Tree Cheat Sheet

General Trees

Purpose: represent hierarchical structures

Parent – Child, Container – Contained, Whole – Part, Generic – Specific, etc. relationships

A tree consists of a collection of nodes and edges

A node represents the real data

Tree node data structure: similar to linked list node data structure. Each node has: (real) data, references to children and/or parent (depending on the direction you want to traverse, you may need to add more references)

An edge represents the relation between 2 nodes

Trees are depicted upside down (compared to biological trees)

There is exactly one node at the top: root

A node can have zero, one, or more child nodes. A node, except root, has exactly one parent node

If a node has zero child node, it is called leaf node. Otherwise, it is called internal node

A path between 2 nodes is a sequence of edges (or links) starting from one node and ending at the other node

Path length: number of edges

Level of root: zero or one (depending on the definition)

Level of other nodes: path lengths from root to those nodes + 1

Tree height: highest level

Binary Search Trees (BST)

Binary trees: each node has at most 2 children

Binary tree traversal: depth-first (explore as far as you can) or breadth-first (nodes at the same level are explored consecutively)

In depth-first traversal, depending on the time when a parent node is processed, we have

- Pre-order: parent node is processed first
- In-order: parent node is processed between left subtree and right subtree
- Post-order: parent node is processed last

BST: is a binary tree in which: at every node, its key is greater than its left subtree's keys and smaller than its right subtree's keys

BST: allows O(lgN) search as well as O(lgN) insertion and deletion (if the tree is balanced)

Sorted array: O(lgN) search but O(N) insertion and deletion

Search on BST: compare with root. If not matched, go down to left subtree or right subtree depending on whether the outcome is greater than or smaller than

Insert to BST (assume there is no duplication): go from top to the node where you cannot go further, attach the new node there

Delete from BST: 3 cases

- Deleted node is a leaf node: just remove it
- Deleted node has just one subtree: root of its subtree will replace it
- Deleted node has two subtrees: select either the right-most node of its left subtree, or the left-most node of its right subtree, assign selected node's data to deleted node
    - Remove selected node: because selected node is either right-most or left-most, it has at most 1 subtree, so this case is the first or second case above

To ensure BST efficiency: make BST balanced, e.g. left subtree and right subtree have similar heights (at most 1-unit difference). Key operation: tree rotation.

Tree rotation: increase the height of one subtree and decrease the height of the other subtree: minimize the difference

Binary Heaps

Complete binary tree: nodes fill up one level before moving on to the next level; the order if node filling is left to right without any skipping

Level 0: 1 node; level 1: 2 nodes; level 2: 4 nodes, level 3: 8 nodes, etc. level n: 2^n nodes => the height of complete binary tree is proportionally to lgN

We can determine mathematically how far a node is from its parent or its left child or its right child => we can use an array to store node positions. This allows constant time access to nodes as well as saving space for not storing references

Binary heap: complete binary tree and the heap property: at every node, its value is greater than (max heap) or smaller than (min heap) all of its subtrees' values.

Heap operations:

Extract max (or extract min): remove and return the root

Insert: add an element to the end of the array representing the binary heap

Heapify: recursively exchange a node with its parent (or children) until reaching the root (or reaching a leaf node)