Endterm Review

Week 13, AY 19/20 Sem 2

Wang Zhi Jian
wzhijian@u.nus.edu

Endterm

BST/AVL

Heaps

UFDS

Graphs

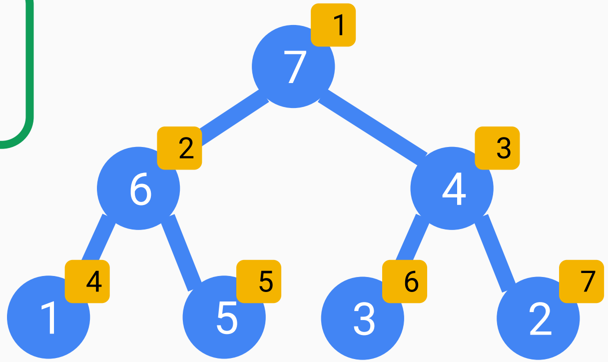
Priority Queue

Insert: O(log *n*)
Extract-Min: O(log *n*)
Peek: O(log *n*)

Heap Property
For every node, its left child and right child contain smaller values than itself.

Heapify
We can construct a priority queue from *any* array in O(*n*) time.

Array Representation
Use 1-indexed array. For every node at index *x*, left child at index 2*x*, right child at index 2*x* + 1, parent at index *x* / 2.



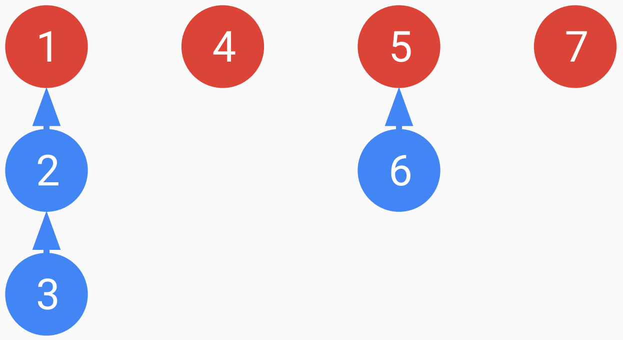
7	6	4	1	5	3	2
1	2	3	4	5	6	7

UFDS

Find: O(α(n))
Merge: O(α(n))

Path Compression. During a *find* operation, link nodes directly to representatives.
Union By Rank/Size. Merge 'shorter' or 'smaller' sets to bigger sets.

Representatives
Every set has a representative item. Sets are identified using their representatives.



Augmenting UFDS
Can store aggregate information for the entire set at the representative.
e.g. max of set, min of set, sum of set

BST/AVL

Insert, Find, Delete, Predecessor, Successor, Select, Rank: O(h)

Traversals: In-Order, Pre-Order, Post-Order
In-Order traversal allows you to obtain all the values in a BST/AVL tree in sorted order.

BST Property
For every node in a BST, all nodes in its left subtree are smaller, and all nodes in its right subtree are bigger.

AVL Trees
Balance Factor: Between -1 and 1
Uses **rotations** to maintain balance factor.

Rotations
After insert: O(1) rotations required
After delete: O(log n) rotations required

Graphs Representations

Adjacency Matrix
Use O(V^2) space
Check if two nodes are connected in O(1) time
Enumerate neighbours of node v in O(V) time
Good for dense graphs

Adjacency List
Use O($V + E$) space
Check if two nodes are connected in O(V) time
Enumerate neighbours of node v in O(deg(v)) time
Good for sparse graphs

Edge List
Use O(E) space
Check if two nodes are connected in O(E) time
Enumerate neighbours of node v in O(E) time
Generally only used for Kruskal's Algorithm

Graph Traversals and Common Problems

Traversals. DFS and BFS

Problem: Counting Connected Components
Given a graph, count how many connected components there are in a graph
Solution: Run DFS/BFS from every vertex. $O(V + E)$

Problem: Bipartite Check
Given a graph, check if it is bipartite
Solution: Run DFS/BFS and try to 2-colour the graph. $O(V + E)$

Problem: Connectivity
Given a graph, check if two nodes are connected via a path
Solution: Run DFS/BFS from one vertex and see if it can reach the other vertex. $O(V + E)$

Problem: Cycle Detection
Given a graph, check if it contains a cycle
Solution: Run DFS, check if back edge exists. $O(V + E)$.

Problem: Topological Sorting
Given a **DAG**, find a topological ordering of the graph.
Solution: Run DFS or Kahn's algorithm using BFS. $O(V + E)$

Minimum Spanning Tree

Cut Property
For any partitioning of nodes into two sets (“cut”), the edge with the smallest weight across the cut is in the MST.

Cycle Property
For every cycle in the graph, the edge with the maximum weight in the cycle is not in the MST.

Algorithms
Prim’s Algorithm: $O(E \log V)$
Kruskal’s Algorithm: $O(E \log V)$

Shortest Path Algorithms