# CS2040 final summary
AY 22/23 S1

Isaac Lai

December 1, 2023

## 1 Heaps

| **Operation** | **Time** |
|---|---|
| Insert | $O(\log n)$ |
| ExtractMax | $O(\log n)$ |
| Update | $O(\log n)$ |
| Delete | $O(\log n)$ |
| Create | $O(n)$ |

- ShiftUp/ShiftDown both $O(\log n)$. **Only heap creation using ShiftDown** is $O(n)$. Using ShiftUp is $O(n \log n)$
- Heap sort runs in $O(n \log n)$
- Partial sort of $k$ elements runs in $O(k \log n)$
- Select $k$th smallest/largest element (from any key) runs in $O(k \log n)$

## 2 UFDS

| **Operation** | **Time** |
|---|---|
| Initialise | $O(n)$ |
| UnionSet | $O(\alpha(n))$ |
| FindSet | $O(\alpha(n))$ |
| IsSameSet | $O(\alpha(n))$ |

- Above time complexities hold only when both union by rank and path compression are used. If only union by rank, time complexities of $O(\alpha(n))$ become $O(\log n)$. If neither, $O(n)$
- Rank is only an **upper bound** on the height of the subtree, not the actual height

## 3 BST and bBST/AVL Tree

| **Operation** | **Time** |
|---|---|
| Insert | $O(\log n)$ |
| Search | $O(\log n)$ |
| Delete | $O(\log n)$ |
| FindMax/FindMin | $O(\log n)$ |
| Predecssor/Succesor | $O(\log n)$ |
| Traversal | $O(n)$ |
| Floor | $O(\log n)$ |
| Ceiling | $O(\log n)$ |
| Rank | $O(\log n)$ |
| Select | $O(\log n)$ |

- For unbalanced BST, operations are $O(n)$ instead of $O(\log n)$ due to edge case when all elements are inserted in ascending/descending order to form one long chain
- All BST operations are dependent on $h$ except traversal. For bBST, $h = O(\log n)$
- AVL tree requires additional attributes `height` and `size`
- Insertion in AVL tree can only trigger a rebalancing case once. Deletion can trigger up to $h$ times
- Traversal visitation order: pre-order (current, left, right), in-order (left, current, right), post-order (left, right, current)

## 4 Graphs

### 4.1 Traversal

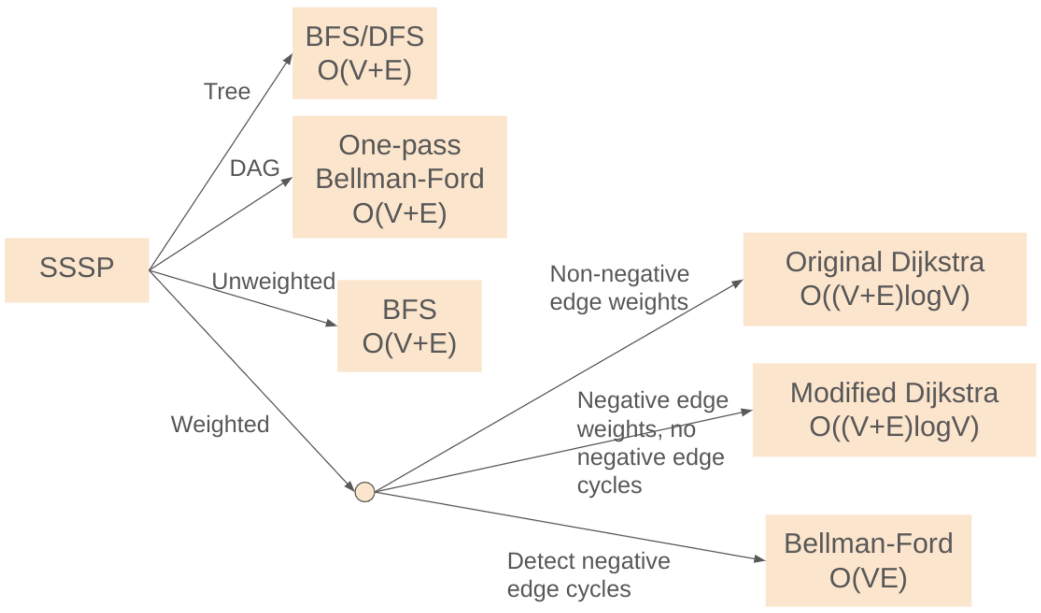
- BFS and DFS both $O(V + E)$
- BFS and DFS form a spanning tree
- Counting/labelling components – $O(V + E)$
- Check if bipartite – run BFS/DFS and use two colours to colour alternating vertices. No two adjacent vertices have the same colour $\implies$ bipartite
- Topological Sort – DFS or Kahn's Algorithm $O(V + E)$
- Counting SCCs – Kosaraju's Algorithm: toposort, reverse edges, DFS and count $O(V + E)$
- Cycle detection – DFS $O(V + E)$

## 4.2 MST

- General case – Prim's or Kruskal's $O(E \log V)$
- Dense graph – Prim's variant (iterate through neighbours instead of using PQ) $O(V^2)$
- Use for Minimax/Maximin problems
- **Cycle property** – in any cycle, the weight of the largest edge cannot be included in the MST
- **Cut property** – for any cut, if the weight of an edge in the cut set is strictly smaller than the weights of all other edges in the cut set, the edge belongs to all MSTs

## 4.3 SSSP

- One-pass Bellman-Ford – toposort then relax in topological order. $O(V + E)$ since toposort dominates the relaxation
- For modified Dijkstra, if $E < O(V)$, $O(E \log E) < O((V + E) \log V) = O(V \log V)$, so time complexity is $O(E \log E)$ instead of $O((V + E) \log V)$


SSSP
Tree
BFS/DFS O(V+E)
DAG
One-pass Bellman-Ford O(V+E)
Unweighted
BFS O(V+E)
Weighted
Non-negative edge weights
Original Dijkstra O((V+E)logV)
Negative edge weights, no negative edge cycles
Modified Dijkstra O((V+E)logV)
Detect negative edge cycles
Bellman-Ford O(VE)

## 4.4 APSP

- Dense graph – Floyd Warshall $O(V^3)$
- Sparse graph – can try running the two inner loops of Floyd Warshall on fewer vertices
- Solves transitive closure (reachability of vertex $i$ from vertex $j$, maximin/minimax from $i$ to $j$), cycle checking (if diagonal element $\neq \infty$)

## 5 Miscellaneous hacks

- For time complexities of graph algos, check if simplification can be done by expressing $E$ in terms of $V$
- Check if any bounds make the time complexity constant (21/22 S2 $O(1)$ Floyd-Warshall!)
- Check if DSes in standard algos can be replaced e.g. using constant number of queues instead of PQ in MST (21/22 S2 Q16), using iteration of neighbours
- To keep track of relationships between variables, use pairs, triples, or hashmap(s)
- MST – radix sort can be used in Kruskal (21/22 S2 Q16)
- Multi-source MST (T9 Q6)
- Run Dijkstra from both source (A) and destination (B) to get distance arrays $D_A$ and $D_B$, then modify shortest path with some edge having weight $w(u, v)$ by taking $D_A[u] + w(u, v) + D_B[v]$ (2040S 22/23 S1 Q19)
- Split vertices/edges into multiple vertices/edges
- **State-space graph** – if vertex has to be entered from multiple states (e.g. finite path length required so need to keep track of total path length), split into one vertex for each state (21/22 S2 Q19, 20/21 S2Q19)
- Logarithms – if path length is a product, convert all weights to log and do normal SSSP (T10 Q3)
- Special graphs – "Linked list", star graph, all nodes in a cycle, disconnected graph, single node, tree, two nodes with bidrectional edge between
- Replace predecssor array in Dijkstra with array of lists to form an adjacency list of the shortest paths spanning tree (2040S 22/23 S1 Q19, 20/21 S2 Q19)
- Create/merge vertices/edges to impose additional constraints (T9 Q6 create new vertex with weight 0 edges to link multiple vertices together)
- In general, adding/multiplying constants to the weights of edges does not produce the same shortest-path/MST/etc. Must check conditions given. Specific cases: multiplying all weights by positive constant gives same SSSP