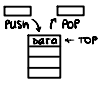
ADT : Abstract Data Type

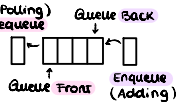
DS : Data Structure (implementation of ADT)

Dynamic Array : Access O(1)
Search O(n)
Insertion O(n)
Appending O(1)
Deletion O(n)

Linked List → Singly Linked List : pointer to the next node
↘ Doubly Linked List : pointer to previous and next node (can be traversed backwards)

Stack (LIFO) : 

used for DFS on graphs

Queue (FIFO) : 

used for BFS on graphs

Priority Queue : ADT implemented as binary Heaps which gives it the best possible time complexity
used in BFS and Prim's MST Algorithm

Heap : tree-based DS that satisfies Heap Invariant (parent ≤ child)

Complete Binary Heap : Insertion → upheap or Bubbling UP
Deletion (Polling) → swap radix and rightmost leaf, remove, then downheap new radix
Deletion (node other than radix) → swap node and rightmost leaf, remove, then downheap new node

Union Find : DS used by Kruskal's MST Algorithm

Kruskal's : 1. sorta gli archi in ordine crescente
2. scorri gli archi: se uniscono vertici NON già uniti, includilo nell' MST
3. L'algoritmo termina quando gli archi sono terminati / i vertici sono tutti uniti

Tree : undirected Graph which is connected and acyclic.

BST : Binary Tree that satisfies BST Invariant (left subtree has smaller elements, right subtree has larger elements)
Inserting, Deleting, Removing, Searching all cost O(n) in worst case and O(log n) in average

Hash Table : DS that provides a mapping from Keys (uniques) to values via hashing

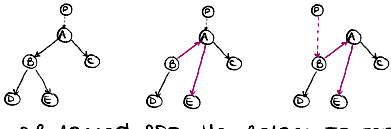
Hash Function : maps a key to a whole number in a fixed range

Collision Resolution : separate chaining : uses a linked list to hold all the different values which hashed to a particular value
open addressing : finds another place within hash Table

Balanced BST : has time cost O(log n) for all operations

TREE ROTATIONS

RIGHT ROTATION



AVL Tree : one type of Balanced BST. Has Balance Factor of each node ∈ [-1,0,1]