__/__/__

Time & Space complexities

⇒ **BFS :**

→ Time complexity = $O(V+E)$

V = vertices

E = edges

We visit each node once which takes const time c

$$T(V) = c \times V = O(V)$$

We explore edges denoted as $d(v)$

$$T(E) = \Sigma [d(v)] \quad \forall_v \in V = 2 * |E|$$
$$= O|E|$$

Time complexity ⇒ $T(n) = T(V) + T(E) = O|V| + O|E|$

$$= O[|V| + |E|]$$

→ **Space complexity**

$S(n)$ = space complexity

Queue : In worst case queue can contain all the vertices at last level. So,

$$S(n) = O|V|$$

⇒ Traversal Method : Level by level, visiting all neighbours at current depth, moves to next level

→ data structure : queue

⇒ $O(V+E)$ if adjacency list

⇒ $O(V^2)$ if " matrix

__/__/____

⇒ Sparse Graph: adjacency list is more efficient making $O(V+E)$ perform well

⇒ Dense graph: list is an option but leads to $O(V^2)$
⇒ less efficient

⟹ Depth First Search (DFS)

- Traversal Method:
  Explores a graph by traversing as deep as it can go before it hits a dead end. Then backtracks to the nearest node with a neighbour and repeats the process

Time: $O(V+E)$ if adj. list

Space: $O(V)$ due to ~~adj list~~ storing nodes in stack and visited array

⇒ Data Structure: Stack for managing visited nodes

⇒ Sparse Graph: $E << V^2$ $O(V+E)$ is largely dominated by $O(V)$ as E is comparatively small

⇒ Dense Graph: $O(V+E)$ complexity approaches $O(V^2)$ as E becomes the dominant factor of adj. matrix is used for representation