Time and space Complexities

BFS:

Time Complexity = $O(V+E)$ where,
V is nodes.
e is edges.

Space We visit each vertex or node once which take constant time C

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

We explore its edges denoted as $d(v)$

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

Time Complexity
$T(n) = T(V) + T(E)$
$= O|V| + O|E| = O[|V| + |E|]$

Space Complexity.

$S(n)$ • denote space Complexity.

Queue : In the worst case the queue can contain all vertices at the last level so

$S(n) = O|V|$

BFS (Breadth for search)

Traversal method:
Explores level by level, visiting all neighbours at the current depth before moving to next level (like ripples in water)

Data Structure:
uses Queue (FIFO) first in first out to manage nodes to visit

Time Complexity:

$O(V+E)$ if adjacency List
$O(V^2)$ if adjacency matrix

Space complexity:
$O(V)$ due to storing nodes in queue and a visited array.

Sparse Graph

* adjacency List is more efficient making $O(V+E)$ time complexity perform well

Dense Graph
* adjacency matric is an option but leads to $O(V^2)$ time complexity which can be less efficient than adjacency Lists for sparse graphs.

DFS

Transversal Method

* Explores a graph by traversing as deeply as possible along each branch before backtracking it visits a node then recursively visits one of its unvisited neighbors and continues until no unvisited neighbors are found at that point it backtracks to explore other branches

Time Complexity

$O(V+E)$ if adjacency list

Space Complexity

$O(V)$ due to storing nodes in stack and visited array

Data structure

Employs a stack (either explicity or implicity via recursion's call stack) for managing node visited nodes.

Sparse Graph

* $E << V^2$ The $O(V+E)$ complexity is largely dominated by $O(V)$ as E is compartively small

Dense graph

* The $O(V+E)$ complexity approaches $O(V^2)$ as E becomes the dominant factor of an adjacency matrix is used for representation both DFS & BFS inherently require $O(V^2)$