**BFS: -- Shortest Path**

Time complexity is O(|V|) where |V| is the number of nodes,you need to traverse all nodes.   
Space complecity is O(|V|) as well - since at worst case you need to hold all vertices in the queue.

**DFS: -- Check to see if something exist**

Time complexity is again O(|V|), you need to traverse all nodes.   
Space complexity - depends on the implementation, a recursive implementation can have a O(h)space complexity [worst case], where h is the maximal depth of your tree.   
Using an iterative solution with a stack is actually the same as BFS, just using a stack instead of a queue - so you get both O(|V|) time and space complexity.

(\*) Note that the space complexity and time complexity is a bit different for a tree then for a general graphs becase you do not need to maintain a visited set for a tree, and |E| = O(|V|), so the |E|factor is actually redundant.

//If you know a solution is not far from the root of the tree:

BFS

//If the tree is very deep and solutions are rare:

BFS - with some memory concern. DFS goes very deep which may not be efficient

//If the tree is very wide:

DFS. BFS will need to much memory

//If solutions are frequent but located deep in the tree:

DFS

//Determining whether a path exists between two nodes:

DFS

//Finding the shortest path:

~~DFS~~ BFS