Recall how we do a search in a binary tree:

A picture containing icon

Description automatically generated

We start with the root, E, then look into C then A then B. If not found, we return back up to C and visit D. What we are performing is known as a depth-first-traversal. We travel downwards from the root to a leaf then backtrack up the path. We would visit the nodes shown, in depth-first traversal as E,C,A,B,D,G,F,H (basically preorder). However, we could also visit the nodes in horizontal order, E,C,G,A,D,F,H,B. This is known as a breadth-first traversal.

The process is similar for a graph structure. Consider the following graph:

Diagram

Description automatically generated

A depth first search would visit the nodes in this order: 1, 2, 5, 9, 10, 6, 3, 4, 7, 11, 12, 8. Notice that the order is determined by traveling down the left most path as far as we can go, then backtracking until there is a right path of the immediate node, then tracking down the left path etc. In a depth-first traversal, we start with a node and then visit each individual path from that node to the end it its length. If cycles exist we stop when we reach a node that has already been visited. In breadth-first search, we visit all nodes within one hop of the start node, then two hops then three hops etc. So depending on what you are attempting to achieve, you will need to decide which may be faster, DFS or BFS. Consider determining the shortest path between two distinct nodes. O BFS will find the shortest path more quickly most of the time - the actual runtime depends on the shape of the graph. However a spanning algorithm (can a node be reached from another node) may be solvable in DFS more quickly, especially if there are a lot of outgoing edges. So breadth and depth (number of hops, or path length) will help dictate which approach to searching you use.