DFS –

DFS works with a stack

Initialize the stack with the head node. Starting element of traversal.

Empty visited list

visitedList = []

add the head node to the stack/queue

Loop until no element in the stack :

* Pop the element – this always take the element off of the top whether it is a queue or a stack.
* Check if the element is in the visited list
* If so we continue
* Check all the neighbors of the popped element:
* If not in the visited list the insert (at location 0) into the stack

print(visitedList)

BFS

The same implementation except instead of

a stack , we use a queue. That means we append the elements on to the queue

----- How to determine if a graph is connected – every node must have a path to the rest of the nodes

Have a nodeList – list of all the nodes in the graph

Assuming that we have a connected graph -

*connectedGraph*  = True

For node in nodeList :

Call DFS on node

If visitedList does not equal len(nodeList) then :

*connectedGraph =* False

break

if connectedGraph == False ---- We have a disconnected graph.

Recursive – DFS

Def DFS(n) :

Add n to the visited list

# Look at all the neighbors of n

For neighbors in adjacencyList[n] :

If cleared ( not in the visited list) :

DFS(neighbor)

Print(node)

* visitedList = []. # Set the visited list to empty
* iterate over the adjacencyList of the headNode :
* if the node is in visited continue
* call DFS(node) – recursive version.

Bipartite graph – there are only two labels for a node, and the label for a node’s node neighbor is the other label. Example when planting a garden with carrots and kale, find out if your graph configuration is so that there carrots touching each other and no kale touching each other.

Determing a bipartite graph

BFS works with a queue

Initialize the stack with the head node. Starting element of traversal.

Empty visited list

visitedList = []

add the head node to the stack – assign a color

Loop until no element in the queue :

* Pop the element – this always take the element off of the top whether it is a queue or a stack.
* Check if the element is in the visited list
* If this element is of the other label, then we are fine, otherwise we do not have a bipartite graph
* If so we continue
* Check all the neighbors of the popped element:
* If not in the visited append into the queue assigning the other label to the node