Challenge 1: Implement Breadth First Search

Now, we shall implement the BFS algorithm in Pythonic code.

We'll cover the following

Problem statement#

You have to implement the **Breadth First Search** traversal in Python. We have already covered the logic behind this algorithm. All that's left to do is to flesh it out in code.

To solve this problem, all the previously implemented data structures will be available to us.

> **Note:** Your solution should work for both connected and unconnected graphs.

Input#

A directed graph in the form of an adjacency list and an integer indicating the starting vertex number (`source`).

Output#

A string containing the vertices of the graph listed in the correct order of traversal.

Sample input#

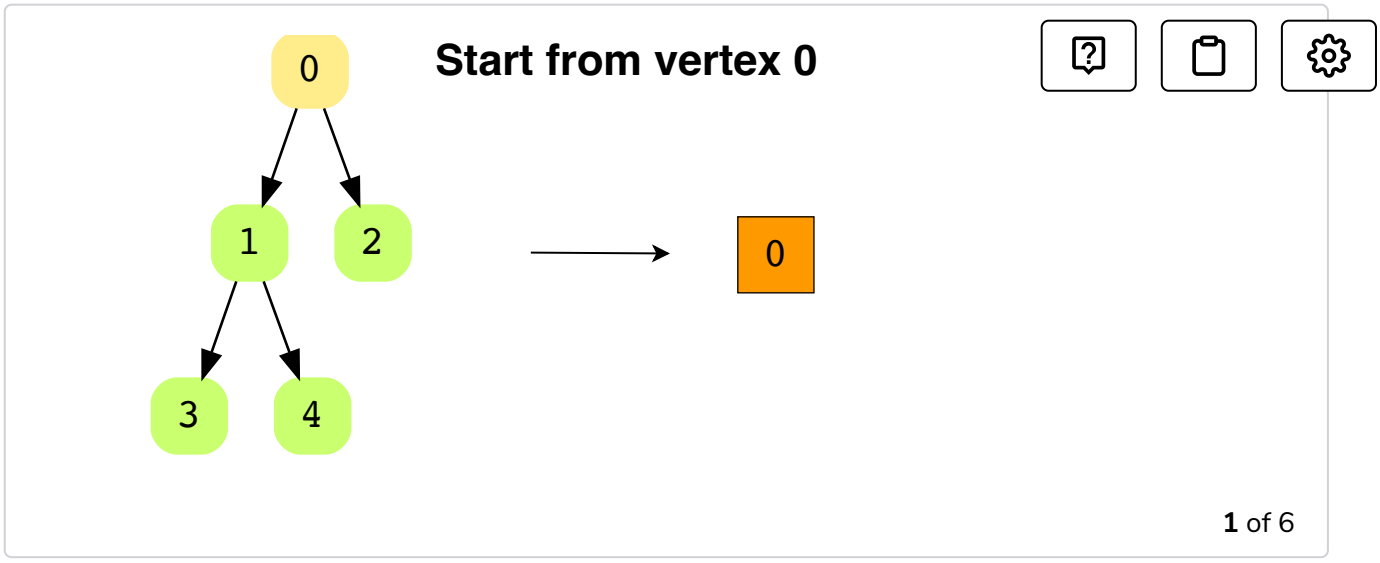
Graph:

Vertex	Connected to by an Edge
0	2, 1
1	4, 3
2	None
3	None
4	None

Source: 0

Sample output#

```
"02143" // Not the only possible output
```



The traversal above is one of several possibilities. It really depends on the implementation of the directed graph, but you do not need to worry about that.

Coding exercise#

Take a close look and design a step-by-step algorithm first before jumping on to the implementation. This problem is designed for your practice, so try to solve it on your own first. If you get stuck, you can always refer to the solution provided in the solution section.

Good luck!

main.py

Graph.py

LinkedList.py

Node.py

Queue.py

Stack.py

```
from Graph import Graph
from Queue import MyQueue
# You can check the input graph in console tab

# Create Queue => queue = MyQueue()
# Functions of Queue => enqueue(int), dequeue(), size(), front(), is_empty()
# Create Stack => stack = MyStack()
# Functions of Stack => push(int), pop(), top(), is_empty()
# Create Graph => graph = Graph(vertices)
# Create LinkedList => link_list = LinkedList()
# Functions of LinkedList => insert_at_head(Node), is_empty(), delete(),
#                            delete_at_head(list), search(), print_list()
# class Node => data, next_element
# Breadth First Traversal of Graph g from source vertex


def bfs_traversal(g, source):
    result = ""
    num_of_vertices = g.vertices
    # Write - Your - Code
    # For the above graph, it should return "01234" or "02143" etc
    return result
```