Graph Data Structure & Breadth First Search

```
# This tutorial is also on graphs. Previously I showed how
# use the Depth First Search to cycle through vertices in
# a graph. This time I'll us the Breadth First Search.

# With the Depth First Search we start and then move as far
# away as possible from the starting point. With Breadth First
# Search we 1st look at all vertices closest to the starting
# point and then move on. We use a Queue with BFSs instead of
# a Stack

# We visit all vertices next to the starting point while
# inserting each into the queue. When there are no more
# vertices connected to A we remove B from the queue
# and look for vertices connected to B. You continue to remove
# values from the queue as you find they don't have connecting
# vertices. This continues until there are no more vertices in
# the queue which signals that you are done.

# First we'll create a Queue
class Queue:
    def __init__(self, size):
        self.size = size
        self.my_queue = [0] * self.size
        self.front = 0
        self.rear = -1

    # This puts items at the rear of the queue
    def insert(self, val):
        # There are no values in the queue put val in index 0
        if self.rear == self.size - 1:
            self.rear = -1
        self.rear += 1
        self.my_queue[self.rear] = val

    # Remove value from front of the Queue
    def remove(self):
        temp = self.my_queue[self.front]
        self.front += 1
        if self.front == self.size:
            self.front = 0
        return temp

    def is_empty(self):
        return self.rear + 1 == self.front or self.front + self.size - 1 == self.rear

# Each vertex will have a name
class Vertex:
    def __init__(self, name):
        self.name = name
        # Used for searching
        self.visited = False
```

```
# Here I'll model a graph using a vertex array
class Graph:
    def __init__(self):
        self.max_vertices = 10
        self.vertex_list = [0]*10
        # Multidimensional list of zeroes
        # Use a generator to create a list of
        # elements defined by max and assigned 0
        self.adjacency_matrix = [[0] * self.max_vertices for i in range(self.max_vertices)]
        self.vertex_count = 0

        # NEW A Queue is used for Breadth First Searching (set size of stack to 20)
        self.the_queue = Queue(20)

    def add_vertex(self, name):
        self.vertex_list[self.vertex_count] = Vertex(name)
        self.vertex_count += 1

    # We will use an adjacency matrix that defines whether
    # an edge lies between 2 vertices
    # Each row &column represents a single vertex and
    # a 1 lies in a cell if vertices connect
    # Example when A connects to B & C but B & C don't
    #    A  B  C
    # A  0  1  1
    # B  1  0  0
    # C  1  0  0
    def add_edge(self, first, last):
        self.adjacency_matrix[first][last] = 1
        self.adjacency_matrix[last][first] = 1

    def print_vertices(self):
        for i in self.vertex_list:
            # If an instance of int print 0
            if isinstance(i, int):
                print(0)
            else:
                print(i.name)

    def print_edges(self):
        for row in self.adjacency_matrix:
            for elem in row:
                print(elem, end=' ')
            print()

    # This function returns the next unvisited vertex or -1
    def get_next_unvisited_vertex(self, curr_vertex):
        for i in range(0, self.vertex_count):
            if self.adjacency_matrix[curr_vertex][i] == 1 and self.vertex_list[i].visited is False:
                return i
        return -1
    # 5. NEW This is the Breadth First Search Function
    def bf_search(self):
        # Start searching at vertex in index 0
        self.vertex_list[0].visited = True

        # Print it
        print(self.vertex_list[0].name)

        # Insert 0 in the Queue
        self.the_queue.insert(0)

        while not self.the_queue.is_empty():

            # Remove vertex at head
            vert_1 = self.the_queue.remove()

            # Get the next unvisited vertex attached
            vert_2 = self.get_next_unvisited_vertex(vert_1)

            # While there are still attached vertices cycle
            while self.get_next_unvisited_vertex(vert_1) != -1:

                # Mark that I visited the vertex
                self.vertex_list[vert_2].visited = True

                # Print out visited vertex
                print(self.vertex_list[vert_2].name)

                # Insert the vertex in the queue
                self.the_queue.insert(vert_2)

                # Get the next attached vertex
                vert_2 = self.get_next_unvisited_vertex(vert_1)

        # Stack is empty so set all vertices back to unvisited
        for i in range(0, self.max_vertices):
            if isinstance(i, int):
                pass
            else:
                self.vertex_list[i].visited = False



# Test the Queue
queue = Queue(10)
queue.insert(1)
queue.insert(2)
queue.insert(3)

while not queue.is_empty():
    print(queue.remove())

# Test Breadth First Search
graph = Graph()
```

```
graph.add_vertex("A")
graph.add_vertex("B")
graph.add_vertex("C")
graph.add_vertex("D")
graph.add_vertex("E")
# A connects to B
graph.add_edge(0, 1)
# B connects to C
graph.add_edge(1, 2)
# A connects to D
graph.add_edge(0, 3)
# D connects to E
graph.add_edge(3, 4)
graph.print_vertices()
graph.print_edges()

# 6. Run Depth First Search
graph.bf_search()
```