Searching and Generating Graphs

This is a multi-stage project to implement a basic graph class and traversals.

Part 1: Graph Class

In the file `graph.py`, implement a `Graph` class that supports the API in the example below. In particular, this means there should be a field `vertices` that contains a dictionary mapping vertex labels to edges. For example:

```
{
    '0': {'1', '3'},
    '1': {'0'},
    '2': set(),
    '3': {'0'}
}
```

This represents a graph with four vertices and two total (bidirectional) edges. The vertex `'2'` has no edges, while `'0'` is connected to both `'1'` and `'3'`.

You should also create `add_vertex` and `add_edge` methods that add the specified entities to the graph. To test your implementation, instantiate an empty graph and then try to run the following:

```
graph = Graph()  # Instantiate your graph
graph.add_vertex('0')
graph.add_vertex('1')
graph.add_vertex('2')
graph.add_vertex('3')
graph.add_edge('0', '1')
graph.add_edge('0', '3')
print(graph.vertices)
```

You should see something like the first example. As a stretch goal, add checks to your graph to ensure that edges to nonexistent vertices are rejected.

```
# Continuing from previous example
graph.add_edge('0', '4')  # No '4' vertex, should raise an Exception!
```

Part 2: Implement Breadth-First Traversal

Write a function within your Graph class that takes takes a starting node as an argument, then performs BFT. Your function should print the resulting nodes in the order they were visited. Note that there are multiple valid paths that may be printed.

Part 3: Implement Depth-First Traversal with a Stack

Write a function within your Graph class that takes takes a starting node as an argument, then performs DFT. Your function should print the resulting nodes in the order they were visited. Note that there are multiple valid paths that may be printed.

Part 3.5: Implement Depth-First Traversal using Recursion

Write a function within your Graph class that takes takes a starting node as an argument, then performs DFT using recursion. Your function should print the resulting nodes in the order they were visited. Note that there are multiple valid paths that may be printed.

Part 4: Implement Breadth-First Search

Write a function within your Graph class that takes takes a starting node and a destination node as an argument, then performs BFS. Your function should return the shortest path from the start node to the destination node. Note that there are multiple valid paths.

Part 5: Implement Depth-First Search

Write a function within your Graph class that takes takes a starting node and a destination node as an argument, then performs DFS. Your function should return a valid path (not necessarily the shortest) from the start node to the destination node. Note that there are multiple valid paths.