CS256 Advanced Programming - Project 3

This is a project for design algorithms for the adjacency implementation of Section 14.2.5.

Problem

Design algorithms for the following operations for a Graph (which may be directed or undirected):

- remove_vertex(self, v): remove the vertex v and all its incident edges, and return the vertex been removed.
  - Parameter v is an instance of Vertex
  - Your algorithm should run in O(deg(v)) time
- remove_edge(self, e): remove the edge e from the adjacency map for each incident vertex, and return the edge removed.
  - Parameter e is an instance of Edge
  - Your algorithm should run in O(1) time.
- bfs_traversal(self): implement a Breadth-First Search method inside the class Graph, use a FIFO queue rather than a level-by-level formulation to manage vertices that have been discovered until the time when their neighbors are considered. Return a map of vertices and the edges that those vertices are discovered.
- print_graph(self): this method should print all vertices with their incident edges.

Requirements:

(1) The four algorithms should be implemented as member functions of class **Graph**.

(2) Create another python file **graph_test.py**, and place the test code in this file

(3) **Implement** the four methods as described, add **comments** for each method and control block. You could add additional methods to support your test.
Make sure that all methods support directed and undirected graphs.

(4) In the **test code**, you should test each method.
Make sure that:

- You test code provides **statement coverage** for the methods (refer to chapter 2: testing)
- **Catch** possible Exceptions, and print corresponding error messages.
- Your program running result is **readable**

For example, you may call the member functions in the following sequence:

a. Build a adjacency graph
   - refer to method **graph_from_edgelist(E, directed=False)** in graph_examples.py which could be downloaded from textbook's website

b. Call print_graph() to print the elements of all vertices and all edges

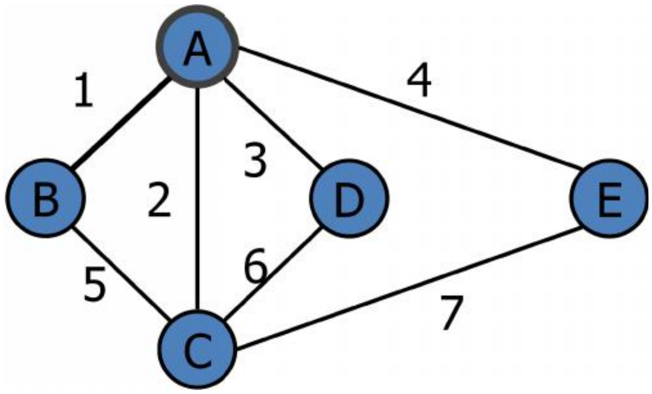
c. Call bfs_traversal (), and print the elements of vertices that are visited in the Breath-First Search

d. Call remove_vertex (v), and print the element in remove vertex

e. Call print_graph() to print the elements of all vertices and all edges

f. Call remove_edge(e), and print the edge
g. Call print_graph() to print the elements of all vertices and all edges

For example, given the following undirected **graph**:



Step (b) could print: (the order of the vertices and edges may be different)

```
The original graph:
A: {(A, B, 1), (A, C, 2), (A, D, 3), (A, E, 4)}
B: {(A, B, 1), (B, C, 5)}
C: {(A, C, 2), (B, C, 5), (D, C, 6), (E, C, 7)}
D: {(A, D, 3), (D, C, 6)}
E: {(A, E, 4), (E, C, 7)}
```

Step (c) prints:

```
BFS Traversal:
A       None
B       (A, B, 1)
C       (A, C, 2)
D       (A, D, 3)
E       (A, E, 4)
```

Step (d), if we remove the vertex A, you may print:

```
After remove vertex D:
```

Step (e) prints:

```
A: {(A, B, 1), (A, C, 2), (A, E, 4)}
B: {(A, B, 1), (B, C, 5)}
C: {(A, C, 2), (B, C, 5), (E, C, 7)}
E: {(A, E, 4), (E, C, 7)}
```

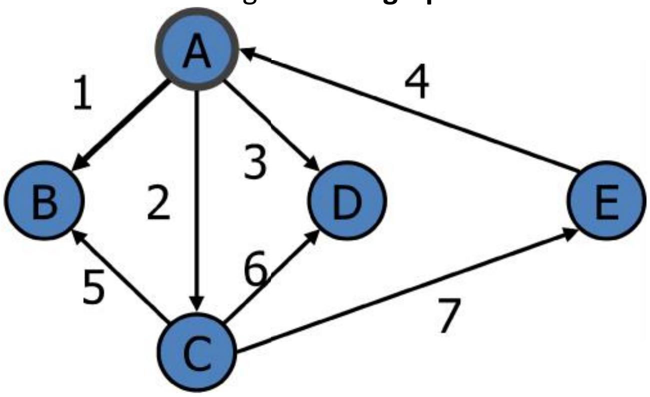
Step (f), we remove the edge 2, you may print:

```
After remove edge (B, C, 5):
```

Step (g) prints:

```
A: {(A, B, 1), (A, C, 2), (A, E, 4)}
B: {(A, B, 1)}
C: {(A, C, 2), (E, C, 7)}
E: {(A, E, 4), (E, C, 7)}
```

Given the following **directed graph**:



Step (b) could print: (the order of the vertices and edges may be different)

```
The original graph:
A: {(A, B, 1), (A, C, 2), (A, D, 3)}
B: { }
C: {(C, B, 5), (C, D, 6), (C, E, 7)}
D: { }
E: {(E, A, 4)}
```

Step (c) prints:

```
BFS Traversal:
A       None
B       (A, B, 1)
C       (A, C, 2)
D       (A, D, 3)
E       (C, E, 7)
```

Step (d), if we remove the vertex A, you may print:

```
After remove vertex D:
```

Step (e) prints:

```
A: {(A, B, 1), (A, C, 2)}
B: { }
C: {(C, B, 5), (C, E, 7)}
E: {(E, A, 4)}
```

Step (f), we remove the edge 5, you may print:

```
After remove edge (C, B, 5):
```

Step (g) prints:

```
A: {(A, B, 1), (A, C, 2)}
B: { }
C: {(C, E, 7)}
E: {(E, A, 4)}
```

Submission:

- **All Python files** that are needed for running your program **(75 points)**

    - **array_queue.py**
    - **graph.py** (40 points)
        - remove_vertex(self, v)
        - remove_edge(self, e)
        - bfs_traversal(self)
        - print_graph(self)
    - **graph_test.py** (25 points)
        - In this file, you could define functions graph_directed() and graph_undirected() to build a direct graph and an undirected graph.
        - (5 points) Place test code in the block **if __name__='__main__':** and make sure that all exceptions are catched and processed in test code
        - (20 points) Test code for graphs
            - Directed graph
            - Undirected graph
    - Well documented code and comments (10 points)
        - Refer to Chapter 2 : Coding Style and Documentation
- A **typescript** of running your program **(5 points)**
    - If you forget how to save the shell window content to a typescript, refer to Assign4 - Exercise 4.1 - Part 6.
- A **design document** of your algorithms(PDF file) **(20 points)**
    - Part 1 (5 points): Draw the **flow chart** (refer to Figure 1.6) for each of the first three methods
    - Part 2 (5 points): Draw the **graphs** you used in your test code.
    - Part 3 (5 points): State the **responsibility** of each group member: on which parts each group member is response for.
    - Part 4 (5 points): State the **problem**(s) you encountered in this project, state what you learn from your partner in your group.