**Notes:**

* You are required to upload your in–class implementations of problems 1, 2, 3 to canvas. This is due by 9:50 AM today.
* You are required to turn in a written report (Word or PDF file) for the homework part (problem 4) of the lab and upload implementations to canvas. These are due by 8:00 AM, November 8, 2017).

**Objectives:**

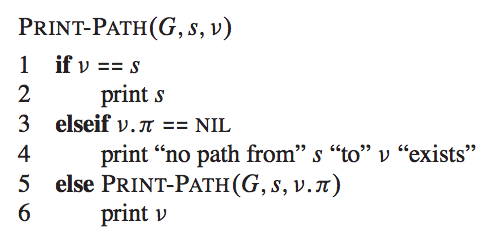
* Implement Breadth First Search algorithm on graphs.
* Compare the performances of BFS using adjacency lists and adjacency matrix

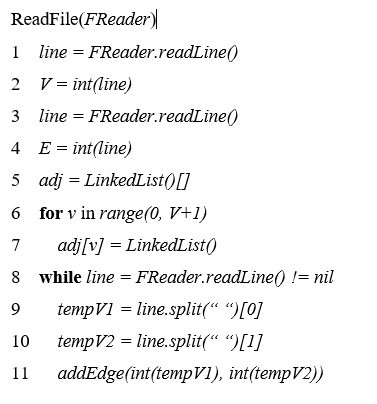
**Problems:**

1. The Graph class has been uploaded in Canvas for reference. It implements adjacency lists to store the neighbors of a vertex. Extend the graph class to make two new subclasses DirectedGraph and UndirectedGraph. Hint: Override the addEdge() method.
2. Write a driver program, which reads input files *mediumG.txt* as an undirected graph and reads an input file *tinyDG.txt* as a directed graph. This driver program should display the graphs in the form of adjacency lists.
3. Implement BFS algorithm on an undirected graph following the pseudo-code given in the next page. Print the BFS paths from a source to all the other nodes in the graph. Read the file mediumG.txt as the input graph.
4. Change the Graph implementation to have Adjacency matrix in place of adjacency lists and then compare the time to execute BFS from a source to all the nodes for both adjacency lists and adjacency matrix. Write a detailed report on the performance and the time complexities of the two variations of graph implementation for BFS. Report the performance for each of the three input graphs: largeG.txt, medium.txt, tinyDG.txt.

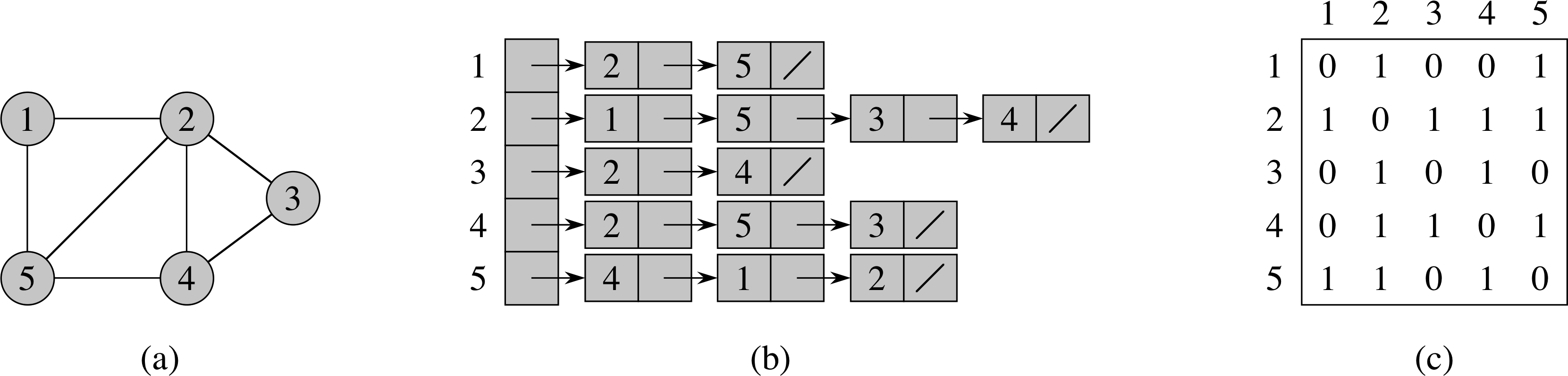
Hints: For problem 3, use the Queue interface from Java Collections Framework and instantiate a queue using a LinkedList type object. Enqueing can be done here by “add()” method while dequeing can be done by “remove()” method, you can use “element()” method to peek at the head of the queue. Import and use equivalent libraries for Python or other languages.

|  |  |
| --- | --- |
|  |  |





**Adjacency List Visualization**



**Adjacency Matrix Visualization**

