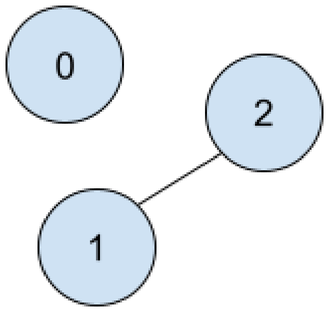
HW4: Graphs, Traversals

Reference

Example Undirected Graph

Visual	Adjacency List	Adjacency Matrix
	[[], [2], [1]]	[[0, 0, 0] [0, 0, 1] [0, 1, 0]]

Questions start on next page ⤵

Q1. Given an undirected graph with 10 vertices but 45 edges, which of the following will be true? Select all that apply.

A) Every node will have at least one neighbor
B) The graph will have duplicate labels
C) An Adjacency List would be more space-efficient to represent this graph vs. a Matrix
D) The graph is GUARANTEED to have a cycle in it
E) None of the above

Q2. There are N cities, with M roads, where each road connects a pair of cities. You are given city **x**. We want to see if every city has a direct road leading to **x**. Return true if this condition is met, and false otherwise.

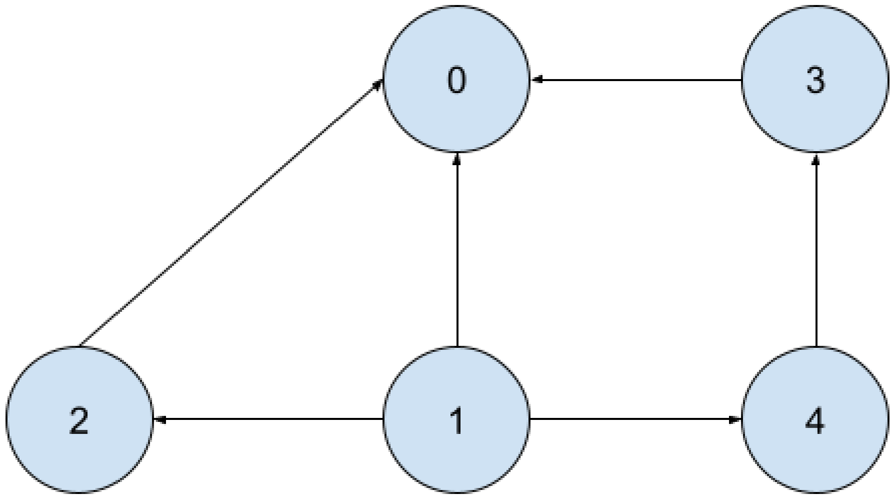
A) If we represent this as a graph problem, what are the nodes and edges?

B) Is this a directed or undirected graph? – Please consider the situation between two cities in the real world. The question is not about two places in a city.

C) Given an adjacency matrix for this graph, describe using words how you would find the answer to this problem. You do not have to write code.

D) Given an adjacency list for this graph, describe using words how you would find the answer to this problem. You do not have to write code.

Q3. Given the graph below, answer the following questions.



A) Represent this graph as an adjacency list.

B) Represent this graph as an adjacency matrix.

C) What is the ordering of nodes If we run Graph DFS starting on node 1? Assume we visit the smallest neighbour first.

D) Write your Java code(submit a .java file) to implement the DFS for graph traversal using the adjacency matrix (either recursive or iterative).
For the test case, you can directly use the above example. And you should call the DFS function several times with different starting points to show the different traversal orders.

```
DFS(graph,  0);   // one possible output likes   0
DFS(graph,  1);  //  one possible output likes   1  0  2 4 3
DFS(graph,  2);  // …
DFS(graph,  3);  // …
DFS(graph,  4);  // …
```