Exercise:

For each of the following real-world problems, try to formulate this as a problem about graphs.

What algorithmic problem about graphs do we need to solve in order to solve the following problems? (Note, you do not actually have to solve these problems, just transform them into graph problems that we might then be able to solve).

1. Among actors, a “Bacon number" is the number of degrees of separation from an actor to Kevin Bacon. For example, Kevin Bacon’s Bacon number is 0. If an actor works in a movie with Kevin Bacon, the actor’s Bacon number is 1. If an actor A works with an actor B who worked with Kevin Bacon in a movie, then actor A’s Bacon number is 2, and so forth.

(a) What is Samuel L. Jackson’s Bacon number?

(b) list all the people with Bacon number equal to 6.

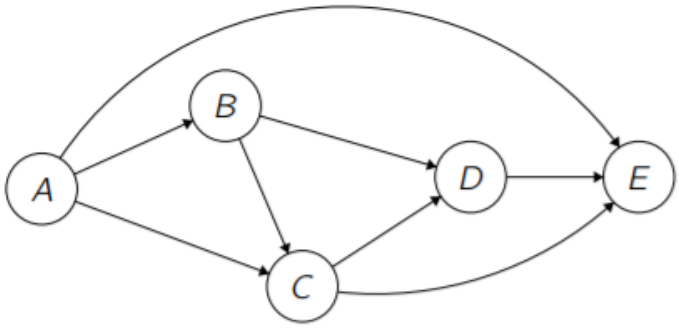
2. You need to take a bunch of classes at Stanford, and some of them depend on each other. For example, you must take CS 162 before taking CS 261. Given a set of classes you need to take, and information about which class is a pre-requisite for which other class, generate an order in which to take all of the classes. Assume you can only take one class at a time.
3. You are about to purchase a bunch of fish. You have two very large fish tanks. Unfortunately, some of these species of fish will fight if they are put in the same tank. For each pair of species, you know whether they will fight or whether they will peacefully co-exist. Find a way to separate the fish into two peaceful fish tanks if it exists

Python Notebook:

Python notebook has also been shared on this link. It contains implementation of Graphs, Breadth First Search Algorithm and Depth First Search Algorithm. Understand, execute and solve it.

Lab Questions:

1. Implement **Graphs** class in C++. For its implementation, a vertex or node class will be required. It is left upto you to think about what kind of information a vertex and graph class will have. You can also refer to Python notebook for more understanding.
    a. Execute DFS and BFS algorithms using a dummy graph. You may create graphs from class examples.
2. Consider the following directed acyclic graph (DAG):



In class, we saw how to use DFS to find a topological ordering of the the vertices; in the graph above, the unique topological ordering is A, B, C, D, E. We saw an example where we happened to start DFS from the first vertex in the topological order. In this exercise we'll see what happens when we start at a different vertex. Recall that when you run DFS, if it has reached everything it can but hasn't yet explored the graph, it will start again at an unexplored vertex.

(a) Run DFS starting at vertex C, breaking any ties by alphabetical order. *
   a. What do you get when you order the vertices by ascending start time?
   b. What do you get when you order the vertices by descending finish time?

(b) Run DFS starting at vertex C, breaking any ties by reverse alphabetical order.**
   a. What do you get when you order the vertices by ascending start time?
   b. What do you get when you order the vertices by descending finish time?

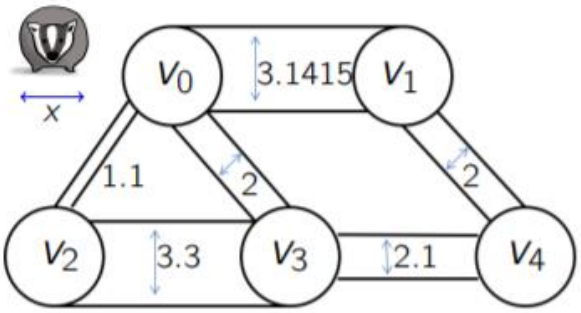
[**We are expecting:** For all four questions, an ordering of vertices. No justification is required.]

Note: *For example, if DFS has a choice between B or C, it will always choose B. This includes when DFS is starting a new tree in the DFS forest.

****When DFS has a choice between B or C, it will always choose C. This includes when DFS is starting a new tree in the DFS forest.

3. **Badger badger badger:** A family of badgers lives in a network of tunnels; the network is modeled by a connected, undirected graph G with n vertices and m edges (see below). Each of the tunnels have different widths, and a badger of width x can only pass through tunnels of width ≥ x.

For example, in the graph below, a badger with width $x = 2$ could get from v_0 to v_4 (either by $v_0 \to v_1 \to v_4$ or by $v_0 \to v_3 \to v_4$). However, a badger of width 3 could not get from v_0 to v_4.



The graph is stored in the adjacency-list format we discussed in class. More precisely, G has vertices $v_0, \ldots, v_n - 1$ and is stored as an array V of length n, so that $V[i]$ is a pointer to the head of a linked list N_i which stores integers. An integer $j \in \{0, \ldots, n-1\}$ is in N_i if and only if there is an edge between the vertices v_i and v_j in G. You have access to a function $tunnelWidth$ which runs in time $O(1)$ so that if $\{v_i, v_j\}$ is an edge in G, then $tunnelWidth(i, j)$ returns the width of the tunnel between v_i and v_j . (Notice that

$tunnelWidth(i,j) = tunnelWidth(j,i)$ since the graph G is undirected). If $\{v_i, v_j\}$ is not an edge in G, then you have no guarantee about what $tunnelWidth(i,j)$ returns.

(a) Design a deterministic algorithm which takes as input G in the format above, integers s, t ∈ {0, . . . , n − 1}, and a desired badger width x > 0; the algorithm should return **True** if there is a path from vs to vt that a badger of width x could fit through, or **False** if no such path exists.

(For example, in the example above we have $s = 0$ and $t = 4$. Your algorithm should return True if $0 < x \leq 2$ and False if $x > 2$).

Your algorithm should run in time $O(n + m)$. You may use any algorithm we have seen in class as a subroutine.

Note: In your pseudocode, make sure you use the adjacency-list format for G described above. For example, your pseudocode should not say something like "iterate over all edges in the graph." Instead, it should more explicitly show how to do that with the format described. (We will not be so pedantic about this in the future, but one point of this problem is to make sure you understand how the adjacency-list format works).

[**We are expecting:** Pseudocode AND an English description of your algorithm, and a short justification of the running time. You should make sure to use the adjacency-list representation of G described above in your pseudocode. You can use any algorithms we have seen from class as a subroutine, but if you significantly modify them make sure to be precise about how this interacts with the adjacencylist representation.]

(b) Design a deterministic algorithm which takes as input G in the format above and integers $s, t \in \{0, \ldots, n-1\}$; the algorithm should return the largest real number x so that there exists a path from vs to vt which accommodates a badger of width x. Your algorithm should run in time $O((n + m)\, log(m))$. You may use any algorithm we have seen in class as a subroutine. (**Hint**: use part (a)).

Note: Don't assume that you know anything about the tunnel widths ahead of time. (e.g., they are not necessarily bounded integers).

Note: The same note about pseudocode holds as in part (a).

[**We are expecting:** Pseudocode AND and English description of your algorithm, and a short justification of the running time. You should make sure to use the adjacency-list representation of G described above in your pseudocode. You can use any algorithms we have seen from class as a subroutine, but if you significantly modify them make sure to be precise about how this interacts with the adjacencylist representation.]

What to Submit:

- A report on implementation with questions of above answers is required.
- Evaluation also includes your capacity to convert your submission in lab