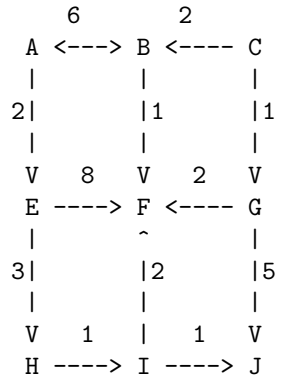
Section #7

Ashley Taylor and Shreya Shankar (Based on handouts by previous CS106B instructors and TAs)

Graphs

1. Graph Properties

Consider the following crudely-drawn graph, and answer the following questions about it:



1. Is the graph directed or undirected?
2. Is the graph weighted or unweighted?
3. Is the graph connected?
4. Is the graph cyclic?
5. What are the in-degrees and out-degrees of each vertex?
6. Write the order that a depth-first search (DFS) would visit vertexes if it were looking for a path from vertex A to vertex I. Assume that any "for-each" loop over neighbors returns them in ABC order. Write the vertex names in order, separated by commas.
7. What is the path that such a DFS would return?
8. Write the order that a breadth-first search (BFS) would visit vertexes if it were looking for a path from vertex C to vertex H. Assume that any "for-each" loop over neighbors returns them in ABC order. Write the vertex names in order, separated by commas.
9. What is the path that such a BFS would return?

2. Task Ordering

Suppose you have a list of tasks which need to be executed. Some of these tasks have dependencies which must be executed before they are. Given several lists of tasks, provide a valid ordering for each one.

1.
```
Input: [ A, B, C, D ]
A <- B, C
B <- C, D
D <- C
```

2.
```
Input: [ 0, 1, 2, 3, 4, 5 ]
0 <- 4, 5
1 <- 3, 4
2 <- 5
3 <- 2
```

3.
```
Input: [ A, B, C, D, E, F ]
B <- A
C <- B
D <- C, A
E <- C, D
```

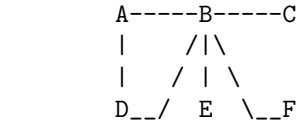
3. kth Level Friends

Imagine a graph of friends on social media, where users are vertexes and friendships are edges. Write a function named `kthLevelFriends` that accepts three parameters: a reference to an adjacency list (Map), the string name of a vertex to start from, and an integer K. Your function should return a set of strings representing the set of people who are exactly K hops away from the given vertex (and not fewer). For example, if K = 1, those are the person's direct friends; if K = 2, they are the person's friends-of-friends. If K = 0, return a set containing only the given vertex. You may assume that the parameter values passed are valid.

4. Minimum Vertex Cover

Write a function named `findMinimumVertexCover` that accepts two parameters: a reference to an adjacency list (Map), and a reference to a Set of edges. Your function should return a set of strings representing names of vertexes identifying a minimum vertex cover. A vertex cover is a subset of an undirected graph's vertexes such that each and every edge in the graph is incident to at least one vertex in the subset. A minimum vertex cover is a vertex cover of the smallest possible size.

For example, in the graph below, all of the following would be vertex covers: {"A", "C", "D", "E", "F"}; {"A", "B", "D"}; {"B", "D"}; {"A", "B"}. Each one is a vertex cover because each edge touches at least one vertex in the cover. The last two vertex covers, {"B", "D"} and {"A", "B"}, are minimum vertex covers, because there is no vertex cover with fewer vertexes.



Understand that because the graph is undirected, that means for every edge that leads from some vertex *v1* to *v2*, there will be an edge that leads from *v2* to *v1*. If there are two or more minimum vertex covers, then you can return any one of them. Think of this as a backtracking problem. The implementation of this function should consider every possible vertex subset, keeping track of the smallest one that covers the entire graph. Try all possible vertex combinations using a "choose-explore-unchoose" pattern and keep track of state along the way.

You may assume that the parameter values passed are valid.