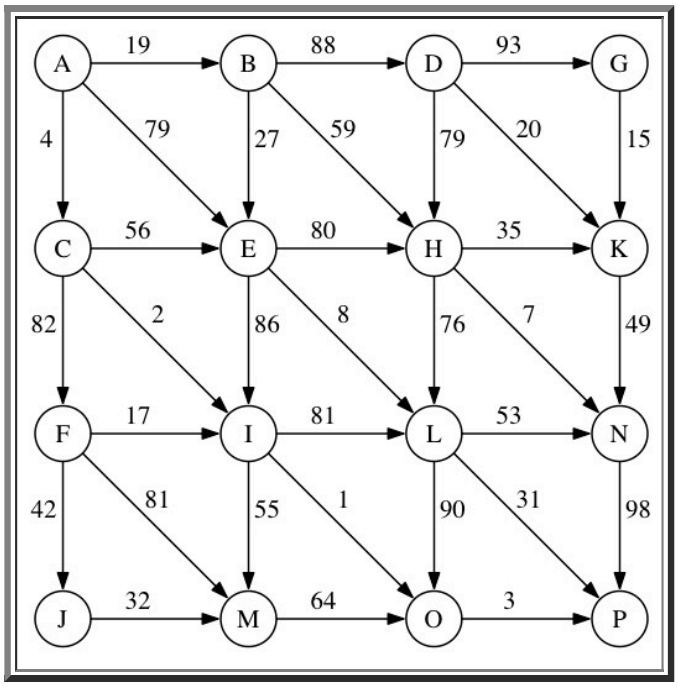
CS302 Final Exam, December 9, 2014 - James S. Plank

Question 2



Part 1: To the right is a directed acyclic graph. Suppose you run Dijkstra's shortest path algorithm on this graph to find the shortest path from A to P. When it completes, specify the nodes that are on the multimap, in the order in which they are on the multimap. This is a multiple choice question, so choose from the answers on the answer sheet.

Part 2: Because this is a directed, acyclic graph, you can use topological sort instead of Dijkstra's algorithm to find the shortest path from A to P. Tell me which one will be faster (circle the answer on the answer sheet).

Part 3: As discussed in class, there are times when Dijkstra's algorithm will be faster than topological sort for finding the shortest path through a graph, and there are times that topological sort will be faster. Explain to me in detail why this is so.

Question 3

A Hamiltonian Circuit is a simple cycle in a graph that touches each node exactly once. Let me define the problem HC-K to be the following: Given a weighted undirected graph and a number K, does the graph have a Hamiltonian Circuit whose edge weights sum to less than K?

Part A: Now, suppose I want to prove that HC-K is an NP-Complete problem. The first step is to prove that HC-K is in NP. Do that for me.

Part B: The second step involves a known NP Complete. Suppose I want to use 3-SAT as my known NP-Complete problem. Exactly what do I have to do to finish proving that HC-K is NP-Complete? *I don't want you to do the proof -- I just want you to tell me what it is that you have to prove*.

Question 4

```
class Graph {
  public:
    int N;
    vector <int> E1;
    vector <int> E2;
    vector <int> Components;
    void CompCon();
};
```

To the right is the definition of a class **Graph**. Each instance of one of these is an undirected graph with *n* nodes numbered 0 to *n-1*. Each node has exactly two edges on its adjacency list. For that reason, rather than having adjacency lists, the class has two vectors **E1** and **E2**. **E1[i]** and **E2[i]** specify the two edges for node *i*. Edges are specified simply by the node number to which the edge goes. So, if **E1[1] = 3** and **E2[1] = 5**, then the two edges coming from node 1 are to nodes 3 and 5.

Write the method **CompCon()**. This method determines the connected components of the graph. When it's done, the vector **Components** should be defined so that **Components[i]** is the component number for node *i*. If there are *c* connected components in the graph, then the component numbers should range from 0 to *c-1*.

If you want to define an additional method in **Graph** to help you write **CompCon()**, then please do so. Just implement it properly, and I'll assume that its prototype is defined in the Class.

BTW, don't worry about how a **Graph** is created. Just write **CompCon()**