NAME: ____

CS 146 (Taylor)	**Fall 2016 (Exam from Spring 2014)**
Data Structures & Algorithms	**Practice Midterm 2(3 pages)**

Cell phones, pagers, etc. should all be off! If such a device goes off, you may be asked to turn in your exam. No tools (calculators, computers, sliderules, screwdrivers, friends, books, or notes) should be used for the test, other than a pen/pencil, your wits and knowledge. Remember how tests are scored: a blank answer is worth 30% credit, which can be taken for up to half of the test. (You may also draw a large X through your scratchwork to indicate that you would prefer to take the 30%.) Spend your time working on questions which you think you can answer correctly. Good luck. **8 questions (or subquestions).**

Graph Algorithm, strongly connected component

1. (10 points) You run the strongly connected components algorithm on the following graph. Assume that all ties are broken alphabetically (in DFS, for instance). Show the components you get, in the order you get them, along with **all relevant work**. Show the final underlying component graph. The graph is given in adjacency matrix format.

	a	b	c	d	e	f	g	h
a	-					1		
b		-					1	
c			-	1	1	1		
d	1			-	1			
e				1	-			
f		1		1		-		1
g						1	-	
h								-

1. first DFS
2. Reverse the order
3. second DFS
4. find the final graph

Graph SSSP Algorithm

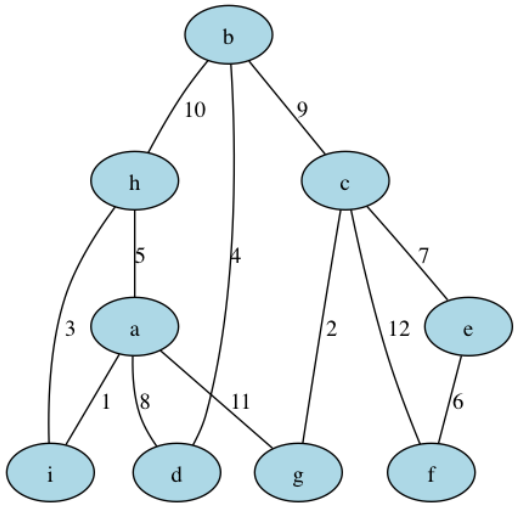
2. (10 points) Simulate the DAG Single Source Shortest Path algorithm on the graph below, using vertex s as a source. Use the non-DFS topological sort algorthm. Assume that initially, vertices are traversed alphabetically, and that within a vertex, its edge list is also traversed alphabetically. The graph is given as a list of outgoing edges for each vertex. Show all work, including the Queue, and all intermediate distances the algorithm temporarily stores. The graph is given in adjacency matrix format, with weights. (Blank means no edge.) Show work.

	a	b	c	d	e	f	g	s
a	-				10	6		
b		-					-4	-5
c			-	6		3		
d				-	1			
e					-			
f					6	-		
g	2		4				-	
s							-1	-

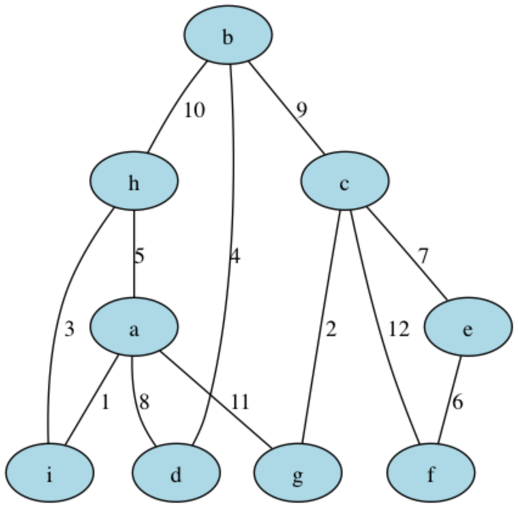
3. Consider the following graph for each of the two subproblems. Each subproblem can be answered (or left blank) independently of the other (subject to the 4 total blank for partial credit rule). The graph is drawn twice, so that you can use one for work for each subproblem. (Minimum Spanning Tree Algorithm)

   [a] (10 points) You are running Prim's algorithm on the graph below and left, starting with vertex c. You are about to take vertex i out of the min-heap, but have not done so yet. For that instant in time, directly label edges (every one of them) on the graph, using categories from the following list:

   (a) Already known to be in the graph.
   (b) The edge has not been seen yet.
   (c) The edge has been seen, and is known to not be in the graph.
   (d) The edge has been seen, and is stored with a vertex in the heap.
   (e) Other.



[a]



[b]

   [b] (10 points) You are in the middle of running Kruskal's Minimum Weight Spanning Tree algorithm, using the Disjoint Set data structure, with union by size and path compression. Use the graph above right. You are about to consider the edge with weight 9. Draw the corresponding disjoint set data structure for the graph, *before and after the edge with weight 9*. Assume that, for each edge of the graph, the edge is called with the alphabetically first vertex first, and that for sets of the same size, the first set joins to the second.

SSSP Algorithm

4. (10 points) You want to code Dijkstra's Single Source Shortest Path algorithm, using a min-heap to store path lengths. As discussed in class, you also want to extend it to work on graphs with negative edge weights, for those graph instances where it happens to work correctly. Thinking back to our heap lab, where is the natural check to see if something has gone wrong (that is, when would it be easiest to recognize that it is *not* working)? Specifically, how would you change your code to check for this, assuming that you had code which worked efficiently? (Assume that you have used your heap code with a graph vertex, rather than the "Student" class we used in the lab.)

NAME: ____

(Dynamic Programming)

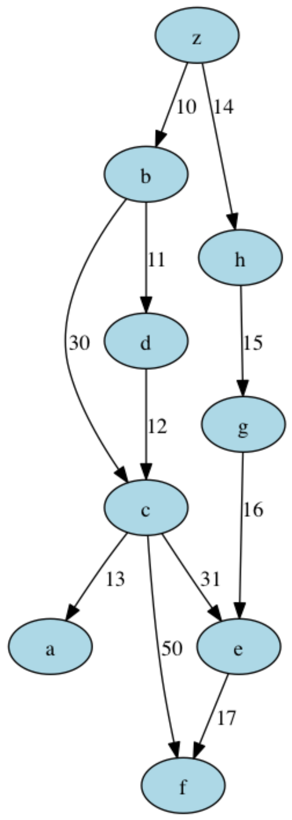
5. (10 points) You are in the middle of running the Floyd-Warshal algorithm. You have just completed calculating D^3 and Π^3. Unfortunately, at that instant in time, a lightning bold corrupts the data stored in your table. Unaware of this, your algorithm continues, correctly following the algorithm to calculate values for D^4 and Π^4. Fill in those values below. (You only need to do it once, I just give the table twice to help you make it legible if it comes out poorly once.)

difference of 3 tables

D^3	1	2	3	4		D^4	1	2	3	4		D^4	1	2	3	4
1	0	3	9	3		1						1				
2	4	6	6	2		2						2				
3	4	3	0	1		3						3				
4	1	3	-7	0		4						4				
Π^3	1	2	3	4		Π^4	1	2	3	4		Π^4	1	2	3	4
1	-	3	7	1		1						1				
2	4	-	6	-4		2						2				
3	17	3	-	2		3						3				
4	9	3	7	-		4						4				

6. (10 points) For the following graph, you are running the Bellman Ford algorithm, with source vertex z. Take all edges alphabetically, **including the ones from the source**. After passing through all edges **exactly 2 times**, what distance and π values are stored with each vertex? Show all work, including intermediate distance estimates.

SSSP Algorithm



7. (10 points) You are running the Bellman Ford algorithm on a graph with 364 vertices. By chance, the (sought after) single source shortest path happens to be a perfectly balanced tree, in which each (non-leaf) vertex has exactly 3 children. $(1 + 3 + 9 + 27 + 81 + 243 = 364)$ What is the largest possible number of vertices that will have their distance estimates change during the 3rd iteration of relaxing all edges? Explain your answer.

351