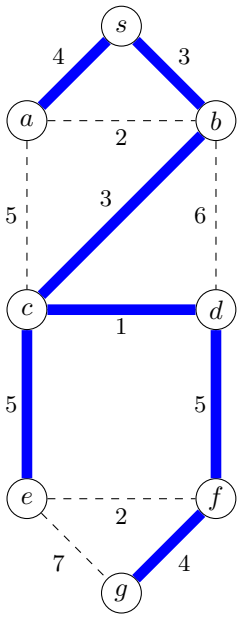
1. (a) (1 points) In the following graph, clearly fill in the edges that are in the single-source shortest path tree starting at s, such as one that would be output by Dijkstra's Algorithm. You do not need to show all of your work, but it is recommended that you keep your work reasonably well organized in the chart provided.

v	intree(v)	parent(v)	dist(v)
s		N/A	0
a		s	4
b		s	3
c		b	6
d		c	7
e		c	11
f		d	12
g		f	16



(b) (0.5 points) For the above graph, is the single-source shortest path tree rooted at s unique? Why or why not?

Yes; at no point in the run of Dijkstra's was an ambiguous choice made

(c) (0.5 points) What is the shortest path from s to g?
$s \to b \to c \to d \to f \to g$

2. (1.5 points) Let G be a simple, undirected graph with positive integer edge weights. Suppose we want to find the *maximum* spanning tree of G. That is, of all spanning trees of G, we want the one with the highest total edge weight. If there are multiple, any one of them is fine to find.

Describe a simple algorithm to accomplish this.

Many answers exist. One of them is to take every edge weight w_e and convert it to $1 - w_e$ then call Prim's Algorithm. Note that you should not describe how Prim's algorithm works on a test (unless asked to do so), but rather call it by name (or description if you can't remember the name "Prim" – "the MST algorithm from class" is fine here).

3. Suppose we have a Cuckoo Hash Table with each table having room for $m = 11$ entries each. Our hash functions are $h_0(x) = x\%11$ and $h_1(x) = (x/11)\%11$, where the / is integer division (floor of division; discard remainder). For example, $h_0(1289) = 2$ and $h_1(1289) = 7$. We insert the keys $12, 16, 27, 23, 31, 108, 103, 81$ and 48, in that order, into the table.

   (a) (1 point) After the first six keys are inserted, are they in the upper array (indexed by h_0) or the lower array (indexed by h_1) after all of first six keys have been inserted? Circle your choice on each clearly.

   (b) (0.5 point) Give a value that is not currently in the table (after we have inserted every key listed above) but *cannot* be inserted into the table without a rehash or resize. You need only give the two hash values for your answer, not the element's value itself. You may not select a value that has h_0 and h_1 both equal to the same value in the set being inserted.

Tables after part A:

0	1	2	3	4	5	6	7	8	9	10
	12				27				31	

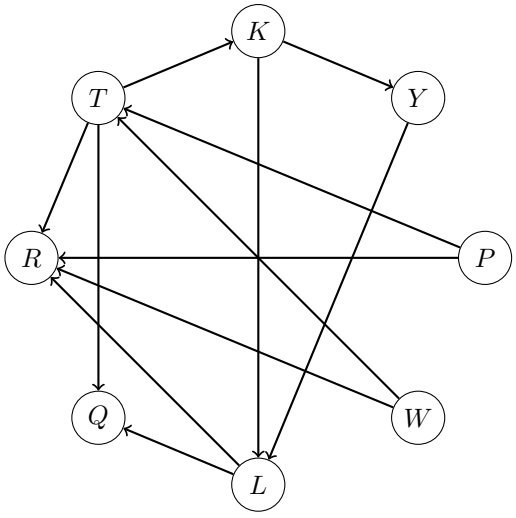
0	1	2	3	4	5	6	7	8	9	10
	16	23							108	

One possibility would be $h_0(x) = 9$ and $h_1(x) = 4$. There are many correct answers.

4. (1.5 points) Suppose we have a hash table of size $m = 13$, implemented with quadratic probing (defined here to mean that the ith choice is $(h(k) + i^2)\%m$, with the initial choice being $i = 0$) and with a hash function of $h(k) = k\%m$. The table stores unsigned values. Insert the following values into the hash table in the order they are given. Do not resize the hash table, regardless of load factor. The values to insert are: $38, 39, 26, 25, 16, 14, 22, 9$ (in the order listed).

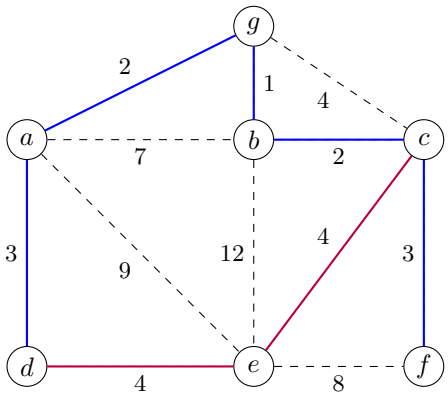
0	1	2	3	4	5	6	7	8	9	10	11	12
39	26	14	25	16					22	9		38

5. (1 point) Give a valid topological ordering of the following graph. Write only the eight letters on the response page (no arrows, commas, etc).



There are many correct answers; among them are WPTKYLRQ.

6. (1 point) Find the minimum spanning tree of the following graph:



MST is the blue edges, plus exactly one of the purple edges

7. (2 points) Suppose you are given an unweighted directed graph $G = (V, E)$ which may or may not have cycles. The vertices, however, are all designated as either blue or gold. This may or may not be a valid 2-coloring of the graph (and, for this problem, it does not matter if it is). The *secluded factor* of any given vertex is the **larger** of the distance to reach it from the nearest gold vertex or the nearest blue vertex. If we cannot get to any particular vertex from both a gold and a blue vertex, then that vertex is *infinitely* secluded.

   Given such a graph, describe how you would find a vertex with the *smallest* secluded factor. Give and briefly justify the running time of your approach. For full credit, your approach should have linear running time. You may make any reasonable assumption about how the graph is represented that you would like; if it affects your algorithm, write the assumption in the answer space.

Add two new vertices to the graph, b and g. Add an edge $b \to x$ for every blue vertex and an edge $g \to x$ for every gold vertex. Note that if we perform a BFS from b, every blue vertex will be in layer 1, everything that can be reached in one hop from a blue vertex is in layer two, anything where we need two hops from a blue vertex to reach is in layer three, and so on.

Run BFS twice: once from b and once from g. For each vertex, record the *larger* of the layer number in which it appeared in a BFS. Subtract one from this number for each vertex.

Now read through the list of vertices and their labels. Find the minimum among these. That vertex and that number is the value we want to return.

Adding the two vertices and their respective edges takes $\mathcal{O}(n)$ time. Running the BFS twice takes $\mathcal{O}(m+n)$ time (and observe that n now being two larger doesn't affect this asymptotic time). Reading the vertices to determine the minimum value takes $\mathcal{O}(n)$. Our total time is $\mathcal{O}(m+n)$.

A few notes. It is important that we use BFS and not Dijkstra's Algorithm; the latter is slower, but needed when edges have weights, which they don't here. It is also important that you do not waste time (yours or the reader's) describing BFS in your answer.