Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Total
6	4	10	4	4	4	12	15	6	15	20	100

CMPE 224 / 343 – Practice Exam 1 (Voluntary)

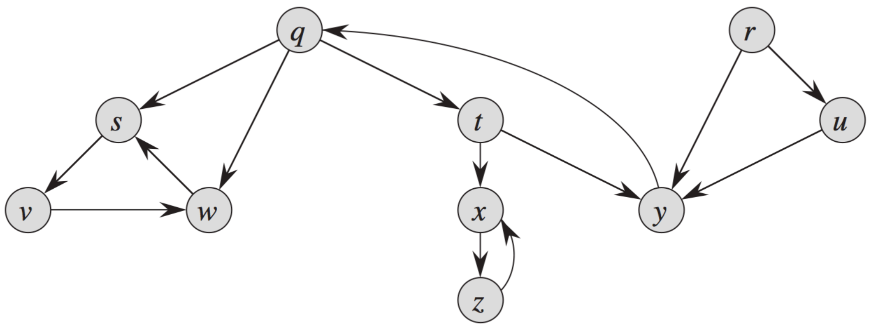
Date: 01.06.2023

Name:
Student ID:

You have 120 minutes for this practice exam. The exam is closed book, closed notes, except that you can use an **A4-size, double sided, handwritten** cheat sheet.

1. (6 points) Consider the following digraph. Assume the vertex adjacency lists are in sorted order: For example, when iterating through the edges pointing from *t*, consider the edge *t* → *x* before *t* → *y*.

   a. Run depth-first search on the digraph, starting from vertex ***q***. List the vertices in *postorder*.



   Vertices: ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

   b. Run breadth-first search on the digraph, starting from vertex ***q***. List the vertices in the order in which they are de-queued from the FIFO queue.

   Vertices: ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

2. (4 points) What is a collision in a hash table implementation of a symbol table? Check the best definition (check only one answer).

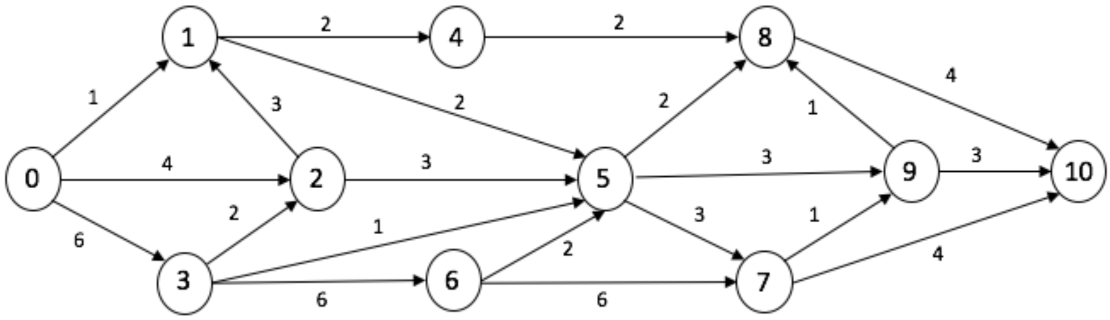
   ___ Two key-value pairs that have equal keys but different values.

   ___ Two key-value pairs that have different keys and hash to different indices.

   ___ Two key-value pairs that have different keys but hash to the same index.

   ___ Two key-value pairs that have equal keys but hash to different indices.

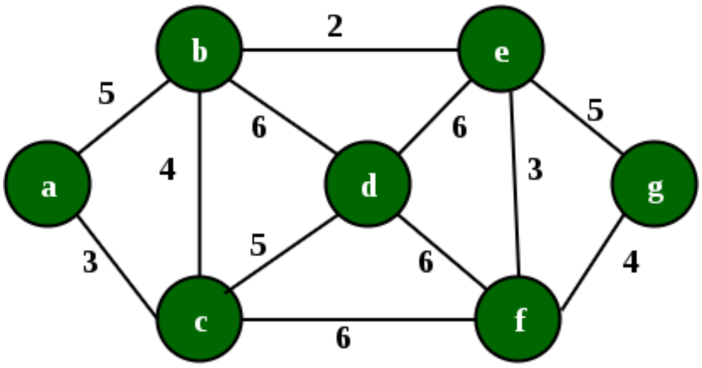
3. (10 points) Find the topological ordering of the graph below, using the *topological sort algorithm that we discussed in class*. Assume that each adjacency list is in order: for example, when iterating through the edges of vertex *5*, process the edges (5,7), (5,8), (5,9) in order.



Vertices: ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

4. (4 points) Which one of the following is **NOT** a possible sequence of edges added to the minimum spanning tree using Kruskal's algorithm? Mark all that apply.

(A) *(b,e), (e,f), (a,c), (b,c), (f,g), (c,d)*
(B) *(b,e), (e,f), (a,c), (f,g), (b,c), (c,d)*
(C) *(b,e), (a,c), (e,f), (b,c), (f,g), (d,e)*
(D) *(b,e), (e,f), (b,c), (a,c), (f,g), (c,d)*



5. (4 points) Let G be an undirected connected graph with distinct edge weights. Let e_{max} be the edge with maximum weight and e_{min} the edge with minimum weight. Which of the following statements are **true** and which are **false**? Mark T or F below.

(__) Every minimum spanning tree of G must contain e_{min}.

(__) If e_{max} is in a minimum spanning tree, then its removal must disconnect G

(__) No minimum spanning tree contains e_{max}

(__) G has a unique minimum spanning tree

6. (4 points) Let `e = v → w` be an edge with weight 17.0. Suppose that during the generic shortest paths algorithm, `distTo[v]` $= \infty$ and `distTo[w]` $= 15.0$. What will `distTo[w]` be after calling `relax(e)`?

7. (12 points) Suppose that you are running Dijkstra's algorithm on the edge-weighted digraph (Table A, below left), starting from a source vertex s. Table B gives the edgeTo[] and distTo[] values immediately after vertex 2 has been deleted from the priority queue and relaxed.

Table A

edge	*weight*	*edge*	*weight*
$0 \to 2$	6.0	$5 \to 1$	12.0
$0 \to 4$	6.0	$5 \to 2$	1.0
$0 \to 5$	17.0	$5 \to 4$	3.0
$1 \to 3$	17.0	$5 \to 7$	10.0
$2 \to 5$	11.0	$5 \to 8$	4.0
$2 \to 7$	6.0	$6 \to 0$	12.0
$3 \to 0$	1.0	$6 \to 1$	5.0
$3 \to 10$	3.0	$6 \to 2$	1.0
$3 \to 1$	25.0	$6 \to 4$	9.0
$3 \to 6$	13.0	$6 \to 9$	4.0
$3 \to 8$	9.0	$7 \to 1$	7.0
$4 \to 5$	3.0	$7 \to 5$	11.0
$4 \to 6$	4.0	$7 \to 9$	6.0
$4 \to 7$	3.0	$10 \to 1$	15.0
$4 \to 8$	1.0	$10 \to 5$	2.0
$4 \to 9$	15.0	$10 \to 8$	7.0

Table B

v	distTo[]	edgeTo[]
0	1.0	$3 \to 0$
1	17.0	$5 \to 1$
2	6.0	$5 \to 2$
3	0.0	*null*
4	7.0	$0 \to 4$
5	5.0	$10 \to 5$
6	13.0	$3 \to 6$
7	12.0	$2 \to 7$
8	9.0	$3 \to 8$
9	∞	*null*
10	3.0	$3 \to 10$

a) Give the order in which the first 5 vertices were deleted from the priority queue and relaxed.

				2

b) Modify Table B above to show the values of the edgeTo[] and distTo[] arrays immediately after the next vertex has been deleted from the priority queue and relaxed. Circle those values that changed.

8. (15 pts) Give a **static Java method** that gets as input a directed graph G and checks if there are any vertices *v* so that there is a path from *v* to **at most 10 vertices** in the graph. The program should print the id's of such vertices *v* that are found.

Use the following Digraph data structure. Assume that the graph is represented via adjacency lists. Hint: modify one of the *search* strategies for digraphs that we discussed in class.

`public class Digraph`

	`Digraph(int V)`	*create an empty digraph with V vertices*
	`Digraph(In in)`	*create a digraph from input stream*
`void`	`addEdge(int v, int w)`	*add a directed edge v→w*
`Iterable<Integer>`	`adj(int v)`	*vertices pointing from v*
`int`	`V()`	*number of vertices*
`int`	`E()`	*number of edges*

What is the computational complexity of your algorithm, in big-O notation? Explain your answer.

9. (6 points) Suppose that your hash function does not satisfy the uniform hashing assumption. Which of the following can result? Check all that apply.

___ Poor performance for insert.

___ Poor performance for search hit.

___ Poor performance for search miss.

___ Uneven distribution of lengths of chains in separate-chaining hash table.

___ Large clusters in linear-probing hash table.

___ Linear-probing hash table can become 100% full, causing a resize operation.

10. (15 points) Given three linked lists, write an efficient algorithm (**<u>in pseudocode</u>**) to find all common elements among the three unsorted linked lists. Your algorithm **<u>must run in O(N)</u>** time in the worst case (N: the total number of elements in the linked lists). First, explain your algorithm in sufficient detail; then, discuss the computational complexity of your algorithm. (Hint: you may want to use one of the data structures we covered in class, e.g. Binary Search Trees, 2-3 Trees, AVL Trees, Hash Tables, Red-Black Tries, etc. You may assume that its implementation is available, you do not need to implement it from scratch)

Examples:

```
Input :
   25 20 10 15 12
   10 13 12 15
   10 24 25 12 15 26
Output : 10 12 15
```

```
Input :
   2 1 3 4 5
   6 4 3 1 2 9 8
   1 10 4 5 2
Output : 1 2 4
```

11. (20 pts) Write a **static Java method** that will take an adjacency matrix for an undirected graph (which can be cyclic) and finds components. The function should display on the screen, one per line, a single representative vertex id of each connected component, and finally return the number of connected components.

The input matrix is passed as a parameter called **data** which is an *n x n* array of booleans where `data[i][j]` is `true` when there is an edge between vertices *i* and *j*, and `false` when there is no edge. Note that the matrix is full and symmetric (i.e. `data[i][j]=data[j][i]`).

You may implement an *iterative* or *recursive* solution and *may use helper functions*. Additionally, you may assume that stack and queue templates are available for your implementation. However, the primary method should have the following header:

```
public static int printRepresentatives(boolean[][] data, int n) {
```