COS 226 Written Exam 2 Fall 2016

There are ten questions on this exam, weighted as indicated at the bottom of this page. There is one question per lecture, numbered corresponding Lectures 12–21, *not in order of difficulty*. If a question seems difficult to you, skip it and come back to it.

Policies. The exam is closed book, though that you are allowed to use a single-page one-sided hand-written cheatsheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. You will have 80 minutes to complete the test. **This exam is preprocessed by computer.** If you use pencil (and eraser), write darkly. Blacken circles or squares *completely* when asked. Write all answers inside the designated rectangles. Do not write on the corner marks.

This page. *Print your name, login ID, and precept number on this page* (now), and write out and sign the Honor Code pledge.

Discussing this exam. As you know, discussing the contents of this exam before solutions have been posted is a serious violation of the Honor Code.

Name

Login

Precept

"I pledge my honor that I have not violated the Honor Code during this examination."

[*copy the pledge here*]

[*signature*]

Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	TOTAL
/7	/7	/7	/7	/7	/7	/7	/7	/7	/7	/70

Q12. Undirected graphs.

A. Consider the following implementation of a graph-processing class that is supposed to allow clients to test whether two vertices are connected:

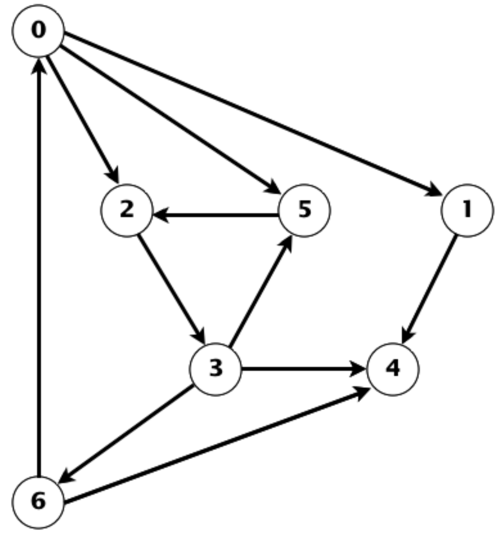
```
public class CC
{
   private int[] id;
   private int count = 1;

   public CC(Graph G)
   {
     id = new int[G.V()];;
     for (int s = 0; s < G.v(); s++)
         if (id[s] == 0)
            { dfs(G, s); count++; }
   }

   private void dfs(Graph G, int v)
   {
      /* MISSING LINE OF CODE */
      for (int w : G.adj(v))
         if (id[w] == 0)
            dfs(G, w);
   }

   public boolean connected(int v, int w)
   {  return id[v] == id[w];  }
}
```

In the box below, write the *one line of Java code* that is missing from the private `dfs()`.

B. Blacken a circle on each line to indicate whether each statement is **True** or **False**.

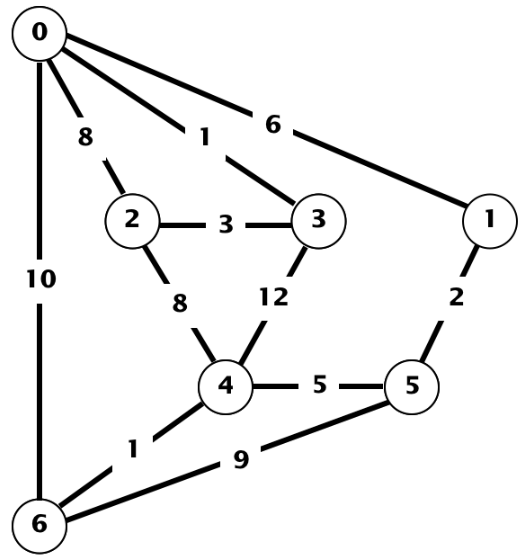
	True	**False**
DFS is not an appropriate strategy for determining whether a graph contains a cycle.	○	○
With DFS, we can support constant-time connectivity queries in an undirected graph, using linear space and linear preprocessing time.	○	○
Union-find is preferable to DFS for connectivity when edge insertion must be supported.	○	○

Q13. Digraphs. Consider the digraph drawn at right. Assume that, in the internal representation, all vertices appear in numerical order in each adjacency list. In the table below, indicate the order in which the vertices appear in *reverse postorder* for a depth-first search starting at **0**. Your answer must have exactly one blackened circle in each row and each column.



	first	*second*	*third*	*fourth*	*fifth*	*sixth*	*seventh*
0	●	○	○	○	○	○	○
1	○	○	○	○	○	○	○
2	○	○	○	○	○	○	○
3	○	○	○	○	○	○	○
4	○	○	○	○	○	○	○
5	○	○	○	○	○	○	○
6	○	○	○	○	○	○	○

Q14. MSTs. Consider the weighted graph drawn at right. Assume that, in the internal representation, all vertices appear in numerical order in each adjacency list. In the table below, indicate the order in which the vertices are added to the MST for *Prim's algorithm* starting at **0**. Your answer must have exactly one blackened circle in each row and each column.

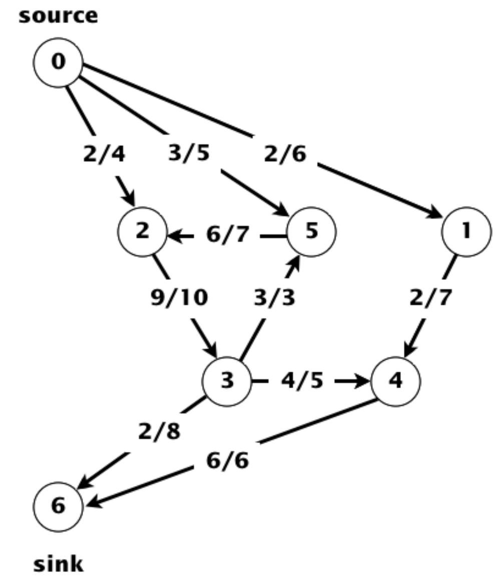


	first	*second*	*third*	*fourth*	*fifth*	*sixth*	*seventh*
0	●	○	○	○	○	○	○
1	○	○	○	○	○	○	○
2	○	○	○	○	○	○	○
3	○	○	○	○	○	○	○
4	○	○	○	○	○	○	○
5	○	○	○	○	○	○	○
6	○	○	○	○	○	○	○

Q15. Shortest paths. In each square at right, write the letter corresponding to the algorithm with the best *worst-case* running time to find a shortest paths tree for the corresponding type of graph, assuming that at least half of the vertices are reachable from the starting point. You may use each letter once, more than once, or not at all.

A	*Kosaraju-Sharir*		
B	*Bellman-Ford*	**digraph with positive edge weights**	☐
C	*brute force*	**DAG with positive edge weights**	☐
D	*Dijkstra*	**DAG with edge weights that could be negative**	☐
E	*topological sort*	**digraph with edge weights that could be negative**	☐
F	*Kruskal*	**digraph with edge weights that could be negative but no negative cycles**	☐
G	*Prim*	**digraph with negative cycles not reachable from the start**	☐
H	*none of the above*		

Q16. Maxflow. Consider the flow network drawn at right. As usual, each directed edge is labeled with its flow and its capacity, separated by a slash.



A. The table below lists a number of paths in the underlying undirected graph. Considering these paths individually, blacken the circle corresponding to the correct characterization of each path. Your answer must have one blackened circle in each row *except* if you think that one or more of the paths is both shortest and fattest.

	not an augmenting path	*a shortest augmenting path*	*a fattest augmenting path*
0–1–4–6	○	○	○
0–1–4–3–6	○	○	○
0–5–3–6	○	○	○
0–5–2–3–6	○	○	○
0–2–3–4–6	○	○	○

B. Blacken the one circle corresponding to the value of the maxflow in this network.

7	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
○	○	○	○	○	○	○	○	○

Q17. String sorts. The column on the left is an array of strings to be sorted. The column on the right is in sorted order. The other columns are the contents of the array at some intermediate step during one of the algorithms below. Write the letter corresponding to the correct algorithm under the corresponding column. You will need to use some letters more than once. *Hint*: Do not trace code—think about algorithm invariants.

input								*sorted result*
ghana	spain	aruba	aruba	aruba	chile	ghana	nepal	aruba
sudan	italy	benin	benin	burma	egypt	malta	sudan	benin
wales	ghana	burma	burma	benin	benin	china	qatar	burma
nepal	gabon	china	congo	china	congo	aruba	macau	chile
italy	libya	congo	chile	congo	burma	kenya	aruba	china
malta	macau	chile	china	chile	china	india	yemen	congo
niger	sudan	egypt	egypt	egypt	aruba	libya	niger	egypt
china	india	ghana	gabon	ghana	gabon	burma	wales	gabon
yemen	niger	gabon	ghana	gabon	ghana	zaire	congo	ghana
haiti	chile	haiti	haiti	haiti	haiti	chile	india	haiti
aruba	china	italy	kenya	italy	kenya	haiti	spain	india
kenya	zaire	india	spain	india	spain	nepal	benin	italy
spain	haiti	kenya	india	kenya	india	sudan	chile	kenya
india	wales	libya	libya	libya	libya	yemen	italy	libya
libya	malta	malta	yemen	malta	yemen	spain	burma	macau
burma	yemen	macau	macau	macau	macau	gabon	ghana	malta
macau	congo	nepal	niger	nepal	niger	benin	china	nepal
gabon	benin	niger	malta	niger	malta	congo	gabon	niger
congo	kenya	qatar	italy	qatar	italy	niger	egypt	qatar
benin	nepal	sudan	nepal	sudan	nepal	qatar	zaire	spain
egypt	burma	spain	zaire	spain	zaire	wales	malta	sudan
zaire	qatar	wales	wales	wales	wales	egypt	haiti	wales
chile	aruba	yemen	qatar	yemen	qatar	macau	kenya	yemen
qatar	egypt	zaire	sudan	zaire	sudan	italy	libya	zaire
A	☐	☐	☐	☐	☐	☐	☐	**E**

A. *input*

B. *LSD radix sort*

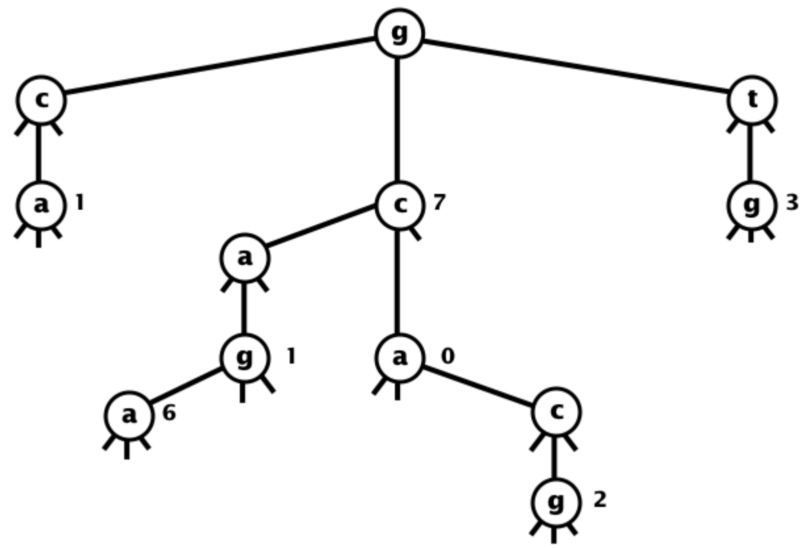
C. *MSD radix sort*

D. *3-way radix quicksort (no shuffle)*

E. *sorted result*

Q18. Tries.

A. Consider the TST drawn at right. In the table below, blacken each of the squares that correspond to the strings that were used to build it. You do not need to write the value—just blacken the square.



ca	ga	gc	gg	tg	gaa	gag	gca	gtg	gcag	gccg	gcacg	gcaga
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

B. Blacken a circle on each line to indicate whether each statement is **True** or **False**.

	True	**False**
The shape of a TST depends only on the set of keys that were used to build it, not the order in which they were inserted.	○	○
Search time in a TST built from randomly ordered keys is logarithmic in the number of keys on the average.	○	○
The height of a TST is dependent on the size of the alphabet	○	○

Q19. Substring search. Here is a trace of a Boyer-Moore algorithm (using only the mismatched character heuristic). Two of the characters in the pattern have been replaced with x and y.

i	*j*	*i+j*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*
			E	O	T	H	E	R	B	U	S	I	N	E	S	S	O	F	T	E	L	L	S	*y*	N	*x*	H	*y*	U	*x*	E	S
0	6	6	*y*	N	*x*	H	*y*	U	*x*																							
7	5	12								*y*	N	*x*	H	*y*	U	*x*																
8	6	14									*y*	N	*x*	H	*y*	U	*x*															
10	6	16											*y*	N	*x*	H	*y*	U	*x*													
17	5	22																		*y*	N	*x*	H	*y*	U	*x*						
21	0	21																						*y*	N	*x*	H	*y*	U	*x*		

A. Blacken the *one* circle corresponding to the character that *must* be represented by *y*.

B **E** **F** **H** **I** **L** **N** **O** **R** **S** **T** **U**

○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○

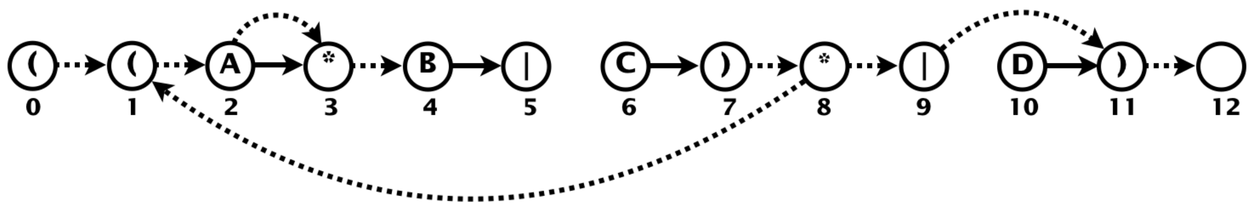
B. Blacken the *one* circle corresponding to the character that *must* be represented by *x*.

B **E** **F** **H** **I** **L** **N** **O** **R** **S** **T** **U**

○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○

Q20. REs.

A. Drawn below is an NFA (nondeterministic finite state automaton) that recognizes the same language that the regular expression `((A*B | C )* | D)` describes, except that it is missing four of its ϵ-transitions. In the table below the drawing, blacken the squares corresponding to the missing transitions.



0-10	1-4	1-6	1-8	1-10	3-2	4-1	4-3	5-6	5-7	9-10	10-1
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

B. It is easy to build an NFA (nondeterministic finite state automata) corresponding to a regular expression, and a basic theorem from automata theory says that we can convert every NFA to a DFA (deterministic finite state automata). Why do we not do so to implement RE pattern matching? Blacken one circle corresponding to the correct answer.

- The NFA might loop. ○
- The DFA might have an exponential number of states. ○
- There might exist a string that the NFA recognizes but the DFA does not. ○
- There might exist a string that the DFA recognizes but the NFA does not. ○
- The proof of the theorem is not constructive (does not tell us how to construct the DFA from the NFA). ○

Q21. Data compression.

A. Suppose that you receive the following message that was encoded using LZW compression:

`42    41    41    81    82    84    80`

Your job is to finish decoding the message, by writing one letter in each square:

B A ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

B. Which of the following best describes the length of the code produced by the LZW compression algorithm for a string consisting of N characters that are all the same? Blacken one circle corresponding to the correct answer.

- $\log N$ ○
- $(\log N)^2$ ○
- $N^{1/2}$ ○
- N ○
- $N \log N$ ○