COS 226 **Algorithms and Data Structures** **Fall 2014**

Final

This test has 14 questions worth a total of 100 points. You have 180 minutes. The exam is closed book, with the exception of a one page cheatsheet. No calculators or other electronic devices are permitted. **Write out and sign the Honor Code pledge just before turning in the test.**

This exam is preprocessed by computer. Please use a pen; if you use a pencil, be sure to write darkly. Do not write any answers outside of the designated frames. And do not write on the corners.

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

Name:

netID:

Room:

Precept: P01 ○ P02 ○ P03 ○ P03A ○ P04 ○ P04A ○

Problem	Score
0	
1	
2	
3	
4	
5	
6	
Sub 1	

Problem	Score
7	
8	
9	
10	
11	
12	
13	
Sub 2	

Total	

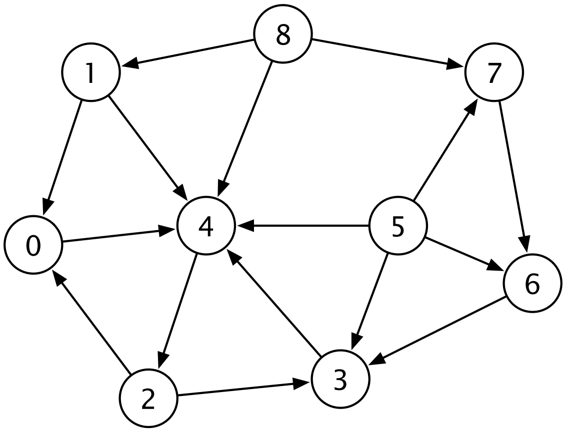
P01	F 9	Andy Guna
P02	F 10	Jérémie Lumbroso
P03	F 11	Josh Wetzel
P03A	F 11	Jérémie Lumbroso
P04	F 12:30	Robert MacDavid
P04A	F 13:30	Shivam Agarwal

0. **Initialization** (2 points)

In the space provided on the front of the exam, write your name and Princeton netID; fill in your precept number; write the name of the room in which you are taking the exam; and write and sign the honor code.

1. **Digraph Traversal** (6 points)

Consider the following digraph. Assume the adjacency lists are in sorted order: for example, when iterating through the edges pointing from vertex 5, consider the edge 5 → 3 before the others.



(a) Starting from vertex 0, run a depth-first search of the digraph, and list the vertices in *reverse postorder*.

(b) Starting from vertex 0, run a depth-first search of the digraph, and list the vertices in *preorder*.

2. **Analysis of Algorithms** (5 points)

For each code fragment on the left, check the best matching order of growth of the running time. You may use an answer more than once or not at all.

N	$\log N$	$N \log N$	$R + N$	RN	$N + R^2$	$(N + R) \log N$	$N(N + R)$

```
int x = 1, i;
for(i = 0; i < N; i++)
  x++;
```

○ ○ ○ ○ ○ ○ ○ ○

```
public static int f2(int N) {
  int x = 1;
  while(x < N)
    x = x * 2;
  return x;
}
```

○ ○ ○ ○ ○ ○ ○ ○

```
int x = 0, i;
for(i = 0; i < N; i++)
  x += f2(N);
```

○ ○ ○ ○ ○ ○ ○ ○

```
int x = 1, i, j;
for(i = 0; i < N; i++)
  for(j = 1; j < R; j++)
    x = x * j;
```

○ ○ ○ ○ ○ ○ ○ ○

```
int x = 0, i, j;
for(i = 1; i <= N; i++)
  for(j = 1; j <= N+R; j+=i)
    x += j;
```

○ ○ ○ ○ ○ ○ ○ ○

3. **String Sorting Algorithms** (7 points)

The column on the left is the original input of 24 strings to be sorted; the column on the right are the strings in sorted order; the other 7 columns are the contents at some intermediate step during one of the 3 radix sorting algorithms listed below.

Match up each column with the corresponding sorting algorithm. You may use a number more than once.

Hint: think about algorithm invariants; do not trace code.

`leaf`	`cost`	`hash`	`edge`	`rank`	`load`	`find`	`cost`	`cost`
`size`	`edge`	`edge`	`cost`	`hash`	`leaf`	`load`	`edge`	`edge`
`null`	`flow`	`cost`	`fifo`	`edge`	`heap`	`size`	`find`	`fifo`
`type`	`find`	`heap`	`flow`	`leaf`	`swap`	`type`	`fifo`	`find`
`cost`	`fifo`	`fifo`	`find`	`heap`	`node`	`trie`	`flow`	`flow`
`sink`	`heap`	`flow`	`heap`	`less`	`fifo`	`node`	`heap`	`hash`
`heap`	`hash`	`find`	`hash`	`next`	`edge`	`edge`	`hash`	`heap`
`trie`	`leaf`	`leaf`	`leaf`	`fifo`	`trie`	`time`	`leaf`	`leaf`
`loop`	`loop`	`loop`	`loop`	`time`	`swim`	`leaf`	`loop`	`less`
`flow`	`less`	`load`	`load`	`find`	`null`	`push`	`less`	`list`
`less`	`load`	`less`	`less`	`sink`	`time`	`hash`	`load`	`load`
`node`	`list`	`list`	`list`	`list`	`find`	`sink`	`list`	`loop`
`find`	`null`	`next`	`next`	`size`	`sink`	`rank`	`null`	`next`
`next`	`node`	`node`	`node`	`flow`	`rank`	`null`	`node`	`node`
`fifo`	`next`	`push`	`push`	`load`	`loop`	`swim`	`next`	`null`
`push`	`push`	`rank`	`rank`	`node`	`flow`	`fifo`	`push`	`push`
`rank`	`rank`	`trie`	`trie`	`loop`	`type`	`heap`	`rank`	`rank`
`load`	`size`	`sink`	`sink`	`cost`	`push`	`loop`	`size`	`sink`
`edge`	`sink`	`type`	`type`	`trie`	`hash`	`swap`	`sink`	`size`
`hash`	`swap`	`time`	`time`	`null`	`less`	`less`	`swap`	`swap`
`time`	`swim`	`swap`	`swap`	`push`	`cost`	`cost`	`swim`	`swim`
`swap`	`type`	`null`	`null`	`swap`	`list`	`next`	`type`	`time`
`list`	`trie`	`swim`	`swim`	`swim`	`next`	`list`	`trie`	`trie`
`swim`	`time`	`size`	`size`	`type`	`size`	`flow`	`time`	`type`

0 ☐ 4

(0) Original input

(1) LSD radix sort

(2) MSD radix sort

(3) 3-way radix quicksort (no shuffle)

(4) Sorted

4. **Substring Search** (8 points)

(a) Consider the Knuth-Morris-Pratt DFA for the following string of length 8:

`C A C A C B C B`

Complete the first row of the table.

	0	1	2	3	4	5	6	7
A								
B	0	0	0	0	0	6	0	8
C	1	1	3	1	5	1	7	1

(b) Suppose that you run the Boyer-Moore algorithm (the basic version considered in the textbook and lecture) to search for the pattern

`M Y F A T H E`

in the text

`Y B R O T H E R T H A T F A T H E R W A S M Y F A T H E R T`

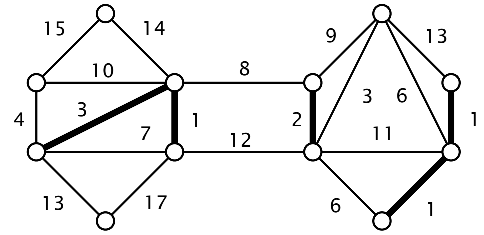
Give the trace of the algorithm in the grid below, circling the characters in the pattern that get compared with characters in the text.

Y	B	R	O	T	H	E	R	T	H	A	T	F	A	T	H	E	R	W	A	S	M	Y	F	A	T	H	E	R	T
M	Y	F	Ⓐ	Ⓣ	Ⓗ	Ⓔ																							

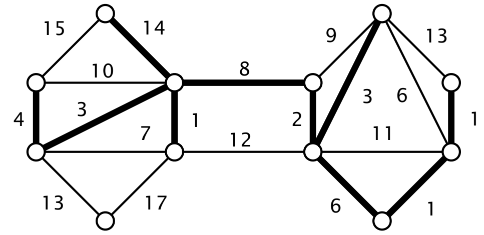
5. **Minimum Spanning Tree Algorithms** (6 points)

Each of the figures below represents a partial spanning tree. Determine whether it could possibly be obtained from (a prematurely stopped) Prim's algorithm, (a prematurely stopped) Kruskal's algorithm, both or neither.

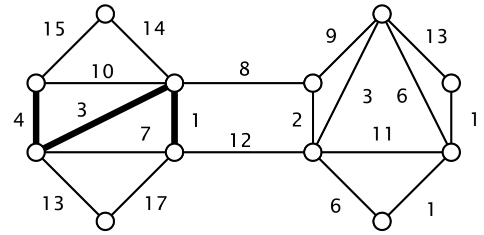
Prim Kruskal Both Neither



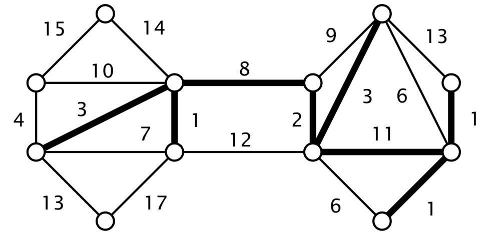
○ ○ ○ ○



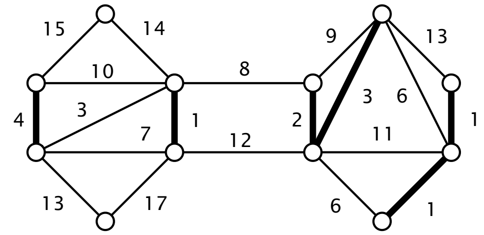
○ ○ ○ ○



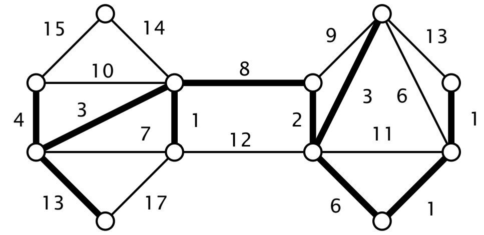
○ ○ ○ ○



○ ○ ○ ○



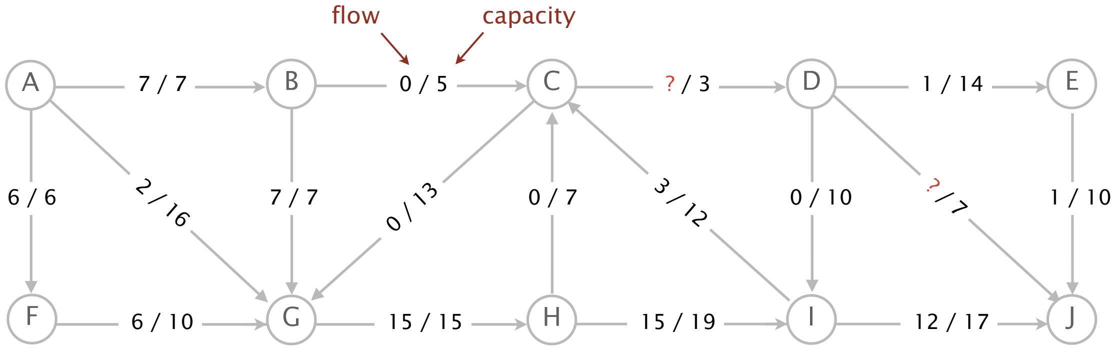
○ ○ ○ ○



○ ○ ○ ○

6. **Maximum Flow** (7 points)

Consider the following flow network and feasible flow f from from the source vertex A to the sink vertex J.



(a) Check the value of the flow on edge $C \to D$?

0	1	2	3	4
○	○	○	○	○

(b) Check the value of the flow f.

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	30
○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

(c) Starting from the flow f, perform one iteration of the Ford-Fulkerson algorithm. List the sequence of vertices on the augmenting path.

(d) Check the value of the maximum flow?

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	30
○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

(e) Check the vertices on the source side of a minimum cut.

A	B	C	D	E	F	G	H	I	J
○	○	○	○	○	○	○	○	○	○

7. **Properties of Algorithms** (9 points)

Check whether each of the following statements are True or False.

(a) **Shortest paths.** Consider an edge-weighted digraph G with distinct and positive edge weights, a source vertex s, and a destination vertex t. Assume that G contains at least 3 vertices, has no parallel edges or self loops, and that every vertex is reachable from s.

	True	False
Any shortest $s \to t$ path must include the lightest edge.	○	○
Any shortest $s \to t$ path must include the second lightest edge.	○	○
Any shortest $s \to t$ path must exclude the heaviest edge.	○	○
The shortest $s \to t$ path is unique.	○	○

(b) **Minimum spanning trees.** Consider an edge-weighted graph G with distinct and positive edge weights. Assume that G contains at least 3 vertices, has no parallel edges or self loops, and is connected.

	True	False
Any MST must include the lightest edge.	○	○
Any MST must include the second lightest edge.	○	○
Any MST must exclude the heaviest edge.	○	○
The MST is unique.	○	○

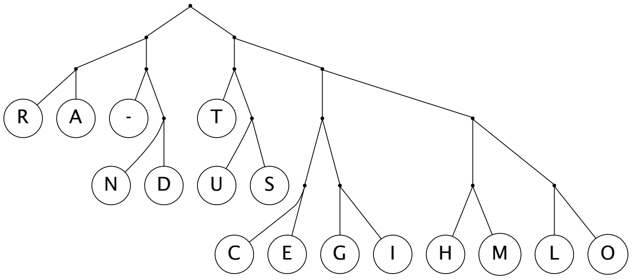
(c) **Burrows-Wheeler transform.**

	True	False
Any input x consisting of an integer (between 0 and $N-1$) followed by N characters is the Burrows-Wheeler transform of some string s of length N.	○	○
If the Burrows-Wheeler transforms of s and t are equal, then $s = t$.	○	○
If the Burrows-Wheeler inverse transforms of x and y are equal, then $x = y$.	○	○
In practice, applying the Burrows-Wheeler transform is significantly faster than applying the Burrows-Wheeler inverse transform.	○	○

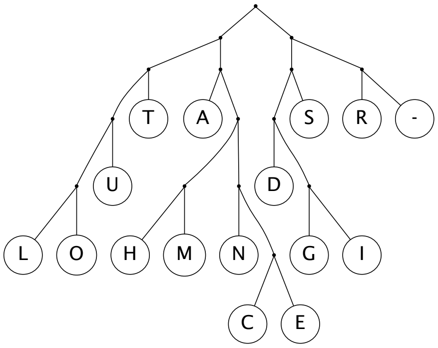
8. **Huffman Trees** (4 points)

Consider the string "`DATA-STRUCTURES-AND-ALGORITHMS`": which of the following trees is an optimal prefix-free code for this input string?

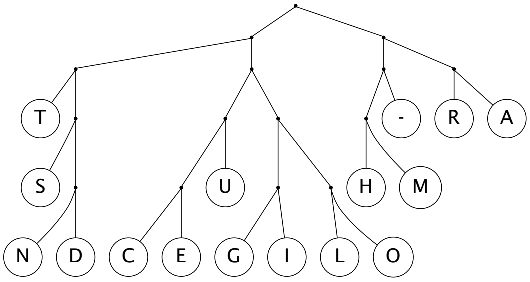
Optimal Prefix-Free Code   Not Optimal Prefix-Free Code



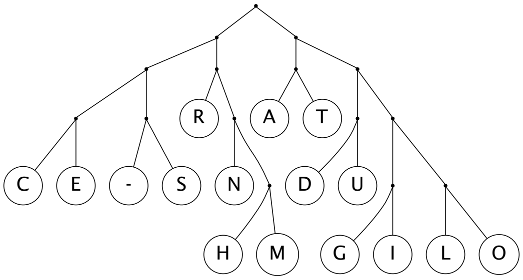
○ ○



○ ○



○ ○



○ ○

9. **LZW Compression** (5 points)

What is the result of compressing the following string of length 15 using LZW compression?

`B B B B B B C A B B C B B B C`

Assume the original encoding table consists of all 7-bit ASCII characters and uses 8-bit codewords. Recall that codeword `80` is reserved to signify end of file.

42								80

For reference, below is the hexadecimal-to-ASCII conversion table from the textbook:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	‘	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

10. **Burrows-Wheeler Transform** (6 points)

(a) What is the Burrows-Wheeler transform of the following?

`A D D B D B C A`

(b) What is the Burrows-Wheeler inverse transform of the following?

```
2
C    A    D    D    A    B    C    C
```

Feel free to use both of these grids for scratch work.

11. **Algorithm and Data Structure Design** (13 points)

Design a data type to store a collection of gene fragments over the DNA alphabet $\{A, C, T, G\}$, according to the following API:

`public class FragmentCollection`		
`public`	`FragmentCollection()`	*create an empty collection of DNA fragments*
`public void`	`add(String fragment)`	*add the DNA fragment to the collection*
`public int`	`prefixCount(String p)`	*number of DNA fragments that start with prefix p*

Here is an example:

```
FragmentCollection fc = new FragmentCollection();
fc.add("AC");
fc.add("TACG");
fc.add("TCGAA");
fc.add("CGA");
fc.add("AGCT");
fc.add("TCGG");
fc.add("TCGG");          // added twice, will be counted twice
fc.prefixCount("");      // returns 7 (number of adds)
fc.prefixCount("T");     // returns 4 (TACG, TCGAA, TCGG, TCGG)
fc.prefixCount("TC");    // returns 3 (TCGAA, TCGG, TCGG)
fc.prefixCount("G");     // returns 0
```

Give a crisp and concise English description of your data structure. Your answer will be graded on correctness, efficiency, and clarity.

(a) Declare the instance variables for your `FragmentCollection` data type. You may use nested data types.

```
public class FragmentCollection {
```

```
}
```

(b) Briefly describe how to implement each of the operations, using either prose or code.

- `public void add(String fragment)`:

- `public int prefixCount(String p)`:

(c) What is the order of growth of `prefixCount(p)` as a function of the number N of keys added, the length W of the prefix `p`, the alphabet size R, and the number M of fragments that match the given prefix `p`?

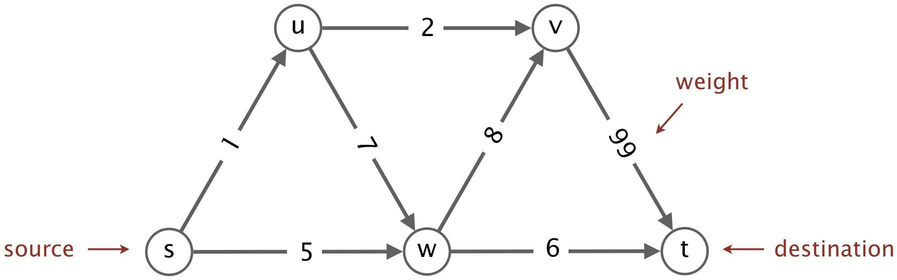
1	$\log N$	N	W	$W + \log N$	$W + M$	$N + W \log R$	WR	$\frac{N^R}{WR}$
○	○	○	○	○	○	○	○	○

12. **Reductions** (13 points)

Consider the following two graph-processing problems:

- SHORTEST-PATH. Given an edge-weighted digraph G with nonnegative edge weights, a source vertex s and a destination vertex t, find a shortest path from s to t.
- SHORTEST-TELEPORT-PATH. Given an edge-weighted digraph G with nonnegative edge weights, a source vertex s and a destination vertex t, find a shortest path from s to t where you are permitted to *teleport* across one edge for free. That is, the weight of a path is the sum of the weights of all of the edges in the path, excluding the largest one.

For example, in the edge-weighted digraph below, the shortest path from s to t is $s \to w \to t$ (with weight 11) but the the shortest teleport path is $s \to u \to v \to t$ (with weight 3).



(a) Design a linear-time reduction from SHORTEST-PATH to SHORTEST-TELEPORT-PATH. To demonstrate your reduction, draw the edge-weighted digraph (and label the source and destination vertices) that you would construct to solve the SHORTEST-PATH problem on the digraph above. You may additionally explain your construction with a few concise sentences.

(b) Design a linear-time reduction from Shortest-Teleport-Path to Shortest-Path. To demonstrate your reduction, draw the edge-weighted digraph (and label the source and destination vertices) that you would construct to solve the Shortest-Teleport-Path problem on the digraph given in the previous page. You may additionally explain your construction with a few concise sentences.

(c) Determine whether each of following statements can be infered from the fact that Shortest-Path and Shortest-Teleport-Path linear-time reduces to one another. For simplicity, assume $E \geq V$.

	Yes	No
If there exists an $E \log\log E$ algorithm for Shortest-Teleport-Path, then there exists an $E \log\log E$ algorithm for Shortest-Path.	○	○
If there exists an $E \log\log E$ algorithm for Shortest-Path, then there exists an $E \log\log E$ algorithm for Shortest-Teleport-Path.	○	○
If there does not exist a linear-time algorithm for Shortest-Path, then there does not exists a linear-time algorithm for Shortest-Teleport-Path.	○	○
If there does not exist a linear-time algorithm for Shortest-Teleport-Path, then there does not exists a linear-time algorithm for Shortest-Path.	○	○

13. **Problem Identification** (9 points)

You are applying for a job at a new software technology company. Your interviewer asks you to identify the following tasks as either *possible* (with algorithms and data structures introduced in this course), *impossible*, or an *open research problem.*

	Possible	Impossible	Open
Given a digraph, find a directed cycle that is simple (if one exists) in time proportional to $E + V$. A *simple* cycle is a cycle that has no repeated vertices other than the requisite repetition of the first and last vertex.	○	○	○
Given an edge-weighted digraph in which all edge weights are either 1 or 2 and two vertices s and t, find a shortest path from s to t in time proportional to $E + V$.	○	○	○
Given an edge-weighted DAG with positive edge weights and two vertices s and t, find a *path* from s to t that *maximizes the product* of the weights of the edges participating in the path in time proportional to $E + V$.	○	○	○
Given an edge-weighted graph with positive edge weights, find a *spanning tree* that *maximizes the product* of the weights of the edges participating in the spanning tree in time proportional to $E + V$.	○	○	○
Given an edge-weighted graph with positive edge weights and two distinguished vertices s and t, find a *simple path* (no repeated vertices) between s and t that *maximizes the sum* of the weights of the edges participating in the path in time proportional to EV.	○	○	○
Given a flow network and a mincut in that flow network, find a maxflow in time proportional to $E + V$.	○	○	○
Given an array of N strings over the DNA alphabet $\{A, C, T, G\}$, determine whether all N strings are distinct in time linear in the number of characters in the input.	○	○	○
Given an array a of N 64-bit integers, determine whether there are two indices i and j such that $a_i + a_j = 0$ in time proportional to N.	○	○	○
Given an array of N integers between 0 and $R^2 - 1$, *stably sort* them in time proportional to $N + R$.	○	○	○