COS 226	Algorithms and Data Structures	Fall 2015

Final Exam

You have 180 minutes for this exam. The exam is closed book, except that you are allowed to use one page of notes (8.5-by-11, one side, in your own handwriting). No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. You may use the back of each page for scratch space, or to continue long answers.

Name:

NetID:

Precept:

P01	9:00	Andy Guna
P02	10:00	Andy Guna
P02A	10:00	Elena Sizikova
P03	11:00	Maia Ginsburg
P03A	11:00	Nora Coler
P04	12:30	Maia Ginsburg
P04A	12:30	Miles Carlsten
P05	1:30	Tom Wu

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

Grading note: To ensure that guessing on true/false and multiple-choice questions does not affect your expected score, grading on these questions will be as follows:

True / False:	$+1$ point if correct,	-1 point if incorrect,	0 points if left unanswered.
Multiple choice:	$+2$ points if correct,	-0.4 points if incorrect,	0 points if left unanswered.

Problem	Score
0	
1	
2	
3	
4	
5	
6	
Sub 1	

Problem	Score
7	
8	
9	
10	
11	
12	
13	
Sub 2	

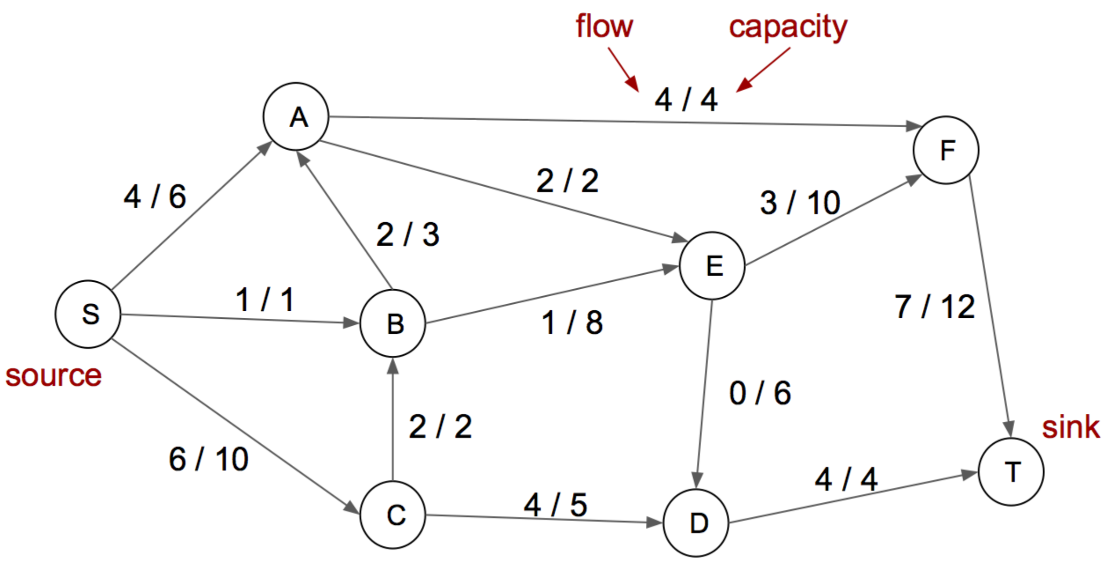
Total:

0. Init. (1 point)

In the space provided on the front of the exam, ***write*** your name, Princeton netID, and precept number, and ***write and sign*** the honor code.

1. Flow. (10 points)

Consider the following flow network and feasible flow f from the source vertex S to the sink vertex T.



(a) What is the value of the flow f? ***Circle*** the correct answer.

3 5 7 11 13 17

(b) Starting from the flow given above, perform one iteration of the Ford-Fulkerson algorithm. ***List*** the sequence of vertices on the augmenting path, in order from S to T.

(c) What is the value of the maximum flow? ***Circle*** the correct answer.

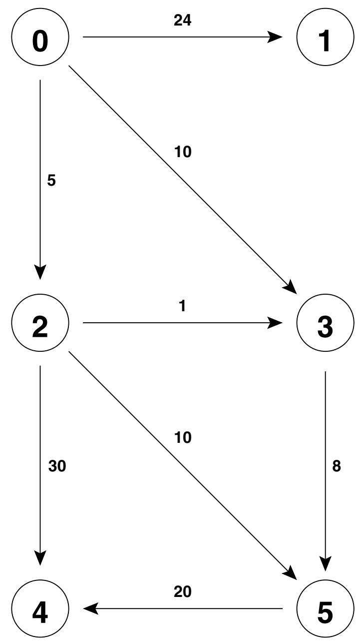
3 5 7 11 13 17

(d) ***Circle*** all vertices on the sink (T) side of the minimum cut.

S A B C D E F T

2. SPT. (12 points)

Simulate Dijkstra's algorithm on the edge-weighted digraph below, starting from vertex 0.



(a) ***Fill in*** the following table:

	`distTo[]`	`edgeTo[]`
0		
1		
2		
3		
4		
5		

(b) What is the maximum number of items in the priority queue? ***Circle*** the correct answer.

1 2 3 4 5 6

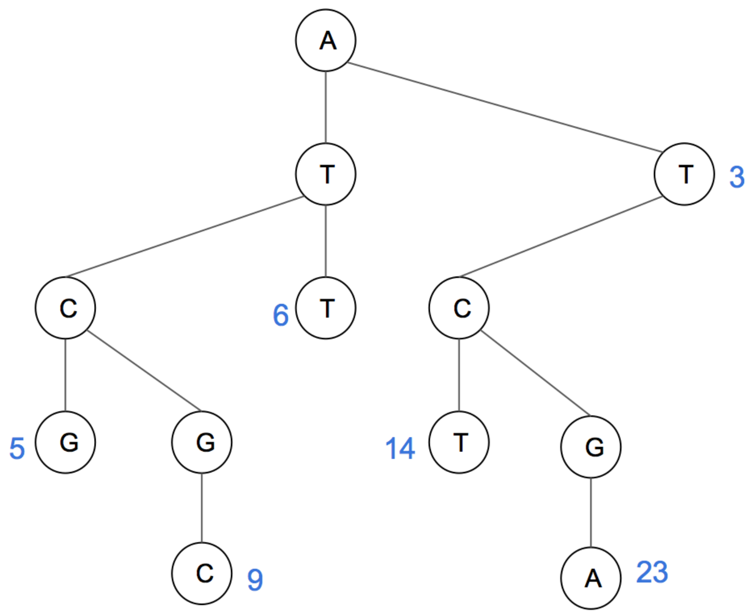
(c) What is the last vertex popped from the priority queue? ***Circle*** the correct answer.

0 1 2 3 4 5

(d) What letter is spelled out by the edges of the shortest-paths tree (SPT) computed by Dijkstra's algorithm?

3. TST. (13 points)

Consider the following Ternary Search Trie (TST), where the values are shown next to the nodes of the corresponding string keys.



(a) We would like to construct the above TST by inserting six strings into an empty TST. ***Circle*** the sequences below that ***can*** produce the above TST. There may be multiple correct answers.

Sequence 1: `ATT ACG T CT AGC GA`

Sequence 2: `ATT T CT ACG GA AGC`

Sequence 3: `ATT T GA ACG CT AGC`

Sequence 4: `ATT T AGC ACG CT GA`

Sequence 5: `ATT ACG AGC T CT GA`

Sequence 6: `ATT T AGC GA ACG CT`

Sequence 7: `ATT ACG T CT GA AGC`

(b) Insert the three strings `CA`, `AGA`, and `GAC` into the TST with the associated values 0, 18, and 29, respectively. ***Update the figure above*** to reflect the changes.

4. KMP DFA. (13 points)

(a) Below is a partially-completed Knuth-Morris-Pratt DFA for a string s of length 6 over the alphabet $\{A, B\}$. State 6 is the accept state. ***Fill in*** all the missing spots in the table.

j	0	1	2	3	4	5
`pat.charAt(j)`						
A	1	1				
B			3			3

(b) Given the following KMP DFA:

j	0	1	2	3	4	5	6
A	1	1	3	1	5	1	5
B	0	2	0	4	0	6	7

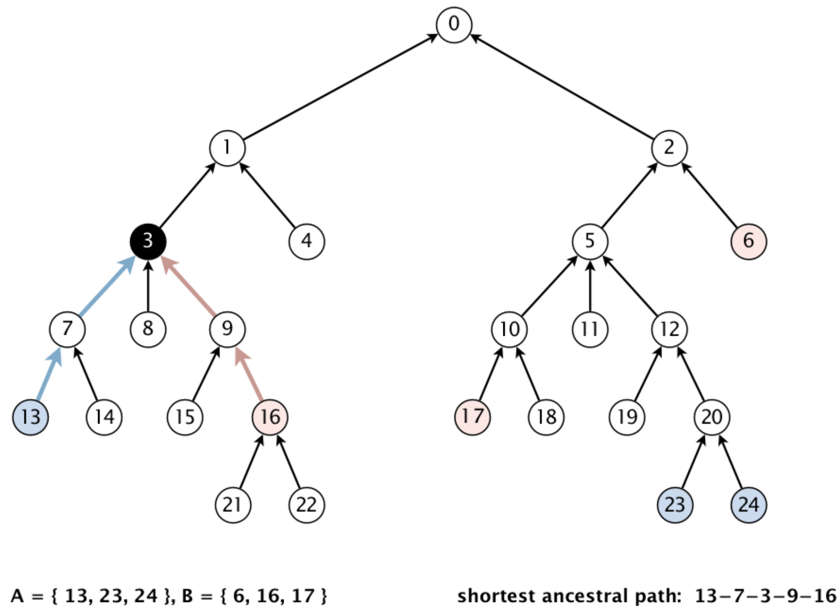
List the string that this DFA searches for.

(c) Given each of the following strings as input, what state would the DFA in (b) end in? ***Circle*** the correct answer for each string.

BABBAA:	0	1	2	3	4	5	6	accept
ABABABA:	0	1	2	3	4	5	6	accept
BABABABA:	0	1	2	3	4	5	6	accept
BBAABBABAB:	0	1	2	3	4	5	6	accept

5. DAG. (10 points)

Consider the following directed graph.



(a) You wish to find the shortest common ancestor (SCA) of the two given sets, using BFS. ***List the first six*** vertices added to the queue by running `BreadthFirstDirectedPaths.java` on an iterator with the sources from set $A = \{13, 23, 24\}$?

(b) ***How many*** vertices in all will `BreadthFirstDirectedPaths.java` visit when passed an iterator with the sources from set $B = \{6, 16, 17\}$?

(c) Using BFS to find the SCA can take running time proportional to $V + E$. Suppose you wished to use DFS instead. What would be the order-of-growth running time? ***Circle*** the correct answer.

constant $V + E$ $E \log V$ $V \log E$ $(V + E)^2$ *exponential*

(d) True or false: any pair of vertices in a ***rooted*** directed acyclic graph (DAG) has at least one shortest common ancestor.

True False

(e) True or false: any pair of vertices in ***any*** DAG for which a topological sort exists has at least one shortest common ancestor.

True False

6. Regex. (11 points)

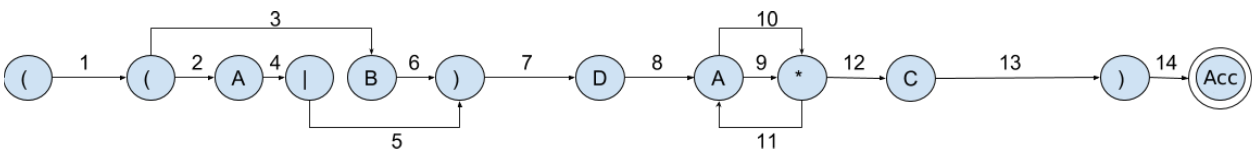
(a) Consider the regular expression

`((A|B)DA*C)`

Circle all words matched by this regular expression.

ABDAC ADAAC ABDACA BDC BDAC AACA

(b) The following NFA matches the regular expression in (a):



Which of the labeled edges correspond to ε transitions (as opposed to *match* transitions)? ***Circle the numbers*** of only the ε transitions:

1 2 3 4 5 6 7 8 9 10 11 12 13 14

(c) Which of the following (if any) are true reasons why we usually prefer NFAs for matching a regular expression (RE), as opposed to DFAs? ***Circle*** the correct answer in each case.

The size of the NFA is linear in the size of the RE, while the size of the DFA might be as bad as quadratic.

True False

The size of the NFA is linear in the size of the RE, while the size of the DFA might be as bad as exponential.

True False

The running time to simulate the NFA is linear in the size of the RE, while the running time for the DFA might be as bad as quadratic.

True False

The running time to simulate the NFA is linear in the size of the RE, while the running time for the DFA might be as bad as exponential.

True False

The NFA only has two kinds of transitions (match and ε), while the DFA requires determining the correct transition for each possible input character.

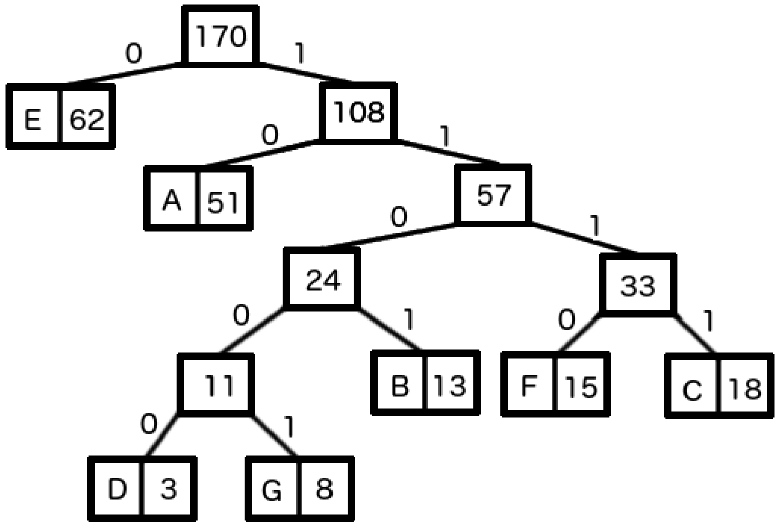
True False

The DFA might require backing up in the input stream, while the NFA does not.

True False

7. Huffman. (10 points)

Consider the following Huffman tree:



(a) Decode the following 24-bit bitstring: 111000110001100001011000

(b) What is the compression ratio (compressed size / uncompressed size) for the above bitstring? Assume that characters were represented by 8 bits before compression.

(c) What is the ***best*** compression ratio achievable on ***any*** string using this Huffman tree?

(d) Suppose you added another character, H, with a count of 1. After re-creating the new Huffman code, ***circle all*** the letters that acquire a ***different*** codeword.

A B C D E F G

(e) Using the Huffman code from (d), what is the ***worst*** compression ratio achievable on ***any*** string?

8. Graph T/F. (10 points)

(a) The adjacency matrix representation is usually preferred over adjacency lists, especially for storing sparse graphs compactly.

True False

(b) Given the data structures produced by depth-first search (DFS), one can check whether a given vertex is connected to the source in constant time.

True False

(c) Breadth-first search (BFS) will visit every vertex in a directed graph, in nondecreasing order from the source.

True False

(d) BFS and DFS are interchangeable and equally practical for all applications of graph search.

True False

(e) Kruskal's algorithm computes the minimum spanning tree (MST) in time proportional to $E \log E$ (in the worst case).

True False

(f) Given any directed graph, there is always a shortest-paths tree (SPT) containing every vertex reachable from a source vertex s.

True False

(g) Dijkstra's algorithm can find shortest paths in a directed graph with negative weights, but no negative cycles.

True False

(h) An st-cut in a graph is any partition of vertices into two disjoint sets, such that vertices s and t wind up in different sets.

True False

(i) A graph flow is a max flow if and only if there exists no cut with the same capacity as the flow's value.

True False

(j) The choice of which augmenting paths to consider first in the Ford-Fulkerson algorithm doesn't impact the number of paths that need to be considered.

True False

9. Sort. (14 points)

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 number*** of each algorithm under the corresponding column. You may use each number more than once.

```
mink   bear   bear   calf   crow   myna   crab   bear   bear
moth   calf   calf   lamb   lamb   crab   toad   crow   calf
crow   crow   crow   hare   deer   lamb   swan   calf   crab
myna   crab   crab   wasp   crab   toad   bear   crab   crow
swan   deer   hare   hawk   hare   mule   deer   deer   deer
wolf   hare   kiwi   ibex   bear   hare   ibex   hare   hare
mule   hawk   deer   bear   kiwi   sole   hoki   hawk   hawk
slug   hoki   hawk   deer   calf   wolf   mule   hoki   hoki
hare   ibex   ibex   mink   hawk   calf   sole   ibex   ibex
bear   kiwi   hoki   lion   ibex   slug   wolf   kiwi   kiwi
kiwi   lion   lion   kiwi   hoki   moth   calf   lion   lamb
calf   lynx   lynx   slug   lion   kiwi   lamb   lynx   lion
hawk   lamb   lamb   toad   lynx   hoki   myna   lamb   lynx
ibex   mink   mink   hoki   mink   mink   mink   mink   mink
oryx   moth   mule   sole   mule   hawk   lynx   moth   moth
lion   myna   myna   wolf   myna   swan   lion   myna   mule
sole   mule   moth   moth   moth   lion   crow   mule   myna
wasp   oryx   wasp   crab   wasp   wasp   hare   oryx   oryx
lynx   swan   sole   crow   sole   bear   wasp   swan   slug
hoki   slug   oryx   oryx   oryx   deer   moth   slug   sole
crab   sole   slug   mule   slug   crow   slug   sole   swan
deer   toad   wolf   swan   wolf   ibex   kiwi   toad   toad
lamb   wolf   toad   myna   toad   oryx   hawk   wolf   wasp
toad   wasp   swan   lynx   swan   lynx   oryx   wasp   wolf
----   ----   ----   ----   ----   ----   ----   ----   ----
0                                                          4
```

(0) Original input

(1) 3-way radix quicksort
(no shuffle)

(2) LSD radix sort

(3) MSD radix sort

(4) Sorted

10. $\mathbf{G^2}$. (10 points)

The *square* of a digraph G consisting of vertices V and edges E is a digraph G^2 such that:

- the vertices in G^2 are the same as the vertices in G, and
- two vertices in G^2 are connected by an edge (u,v) if and only if G contains edges (u,w) and (w,v), for some vertex w.

That is, vertices u and v are connected by an edge in G^2 whenever G contains a path with exactly ***two*** edges from u to v.

Describe an algorithm for computing the square of a digraph (represented using adjacency lists). For full credit, your solution should run in $O(VE)$ time. To simplify the problem, you need not remove duplicates from the adjacency lists in G^2.

11. ST Analysis. (10 points)

You are deciding between symbol table implementations to store L-character strings, consisting of characters from the extended-ASCII ($R = 256$) character set. Analyze the ***worst-case*** order-of-growth running time required by the `get()` operation (with the key present in the symbol table — i.e., a search hit) for the following implementations, assuming that N strings are already in the symbol table. ***Circle*** the correct answer in each case.

(a) A Left-Leaning Red-Black BST of strings.

Worst-case number of ***character comparisons***:

$$\lg N \qquad \lg^2 N \qquad L + \lg^2 N \qquad L \lg N \qquad (\lg N)(\log_R L) \qquad NL$$

(b) An M-entry hash table with separate chaining. Assume the hash table has been resized such that the average chain length N/M is bounded: $2 \le N/M \le 8$. Do not include the time to compute the `hashCode`.

Worst-case number of ***character comparisons***:

$$M + L \qquad NL/M \qquad ML/N \qquad (N/M)\log_R L \qquad (\log_R L)\left(1 + \frac{1}{1 - N/M}\right) \qquad NL$$

(c) An M-entry hash table with linear probing. Assume the hash table has been resized such that the average occupancy N/M is bounded: $1/8 \le N/M \le 1/2$. Do not include the time to compute the `hashCode`.

Worst-case number of ***character comparisons***:

$$M + L \qquad NL/M \qquad ML/N \qquad (N/M)\log_R L \qquad (\log_R L)\left(1 + \frac{1}{1 - N/M}\right) \qquad NL$$

(d) An R-way trie.

Worst-case number of ***array accesses***:

$$R \qquad L \qquad R + L \qquad RL \qquad \log_R N \qquad L + \log_R N$$

(e) A ternary search trie (TST).

Worst-case number of ***character comparisons***:

$$R \qquad L \qquad R + L \qquad RL \qquad \log_R N \qquad L + \log_R N$$

12. Reduction. (10 points)

(a) The FIND-42ND problem is to find the 42nd smallest item in an (initially unsorted) array. You can implement this easily by sorting the array in $O(N \log N)$ time and returning the item in the 42nd position. Given this, which of the following (if any) must be true? ***Circle*** the correct answer in each case.

FIND-42ND reduces to sorting.

True False

Sorting reduces to FIND-42ND.

True False

$O(N \log N)$ is a lower bound on FIND-42ND.

True False

$O(N \log N)$ is an upper bound on FIND-42ND.

True False

FIND-42ND must be NP-complete.

True False

FIND-42ND cannot be NP-complete unless P = NP.

True False

(b) Of course, it is also easy to implement FIND-42ND in $O(N)$ time, using $O(42)$ additional space. Furthermore, it is possible to show that linear time is the lower bound on FIND-42ND, since all elements must be examined. Given this algorithm and the reduction in (a), which of the following (if any) must be true:

$O(N)$ is a lower bound on sorting.

True False

$O(N)$ is an upper bound on sorting.

True False

Sorting is strictly harder than FIND-42ND, so can never be accomplished in $O(N)$ time.

True False

New developments in sorting might result in an asymptotically faster algorithm for FIND-42ND.

True False

13. MST. (16 points)

You are given an edge-weighted undirected graph, using the adjacency list representation, together with the list of edges in its minimum spanning tree (MST). Describe an efficient algorithm for updating the MST, when each of the following operations is performed on the graph. Assume that common graph operations (e.g., DFS, BFS, finding a cycle, etc.) are available to you, and don't describe how to re-implement them.

(a) Update the MST when the weight of an edge that ***was not*** part of the MST is ***decreased***.
Give the order-of-growth running time of your algorithm as a function of V and/or E.

(b) Update the MST when the weight of an edge that ***was*** part of the MST is ***decreased***.
Give the order-of-growth running time of your algorithm as a function of V and/or E.

(c) Update the MST when the weight of an edge that ***was not*** part of the MST is ***increased***.
Give the order-of-growth running time of your algorithm as a function of V and/or E.

(d) Update the MST when the weight of an edge that ***was*** part of the MST is ***increased***.
Give the order-of-growth running time of your algorithm as a function of V and/or E.