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.

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.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Sub 1</td>
<td></td>
<td>Sub 2</td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th></th>
<th>distTo[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

(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.

<table>
<thead>
<tr>
<th>$j$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>pat.charAt($j$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

(b) Given the following KMP DFA:

<table>
<thead>
<tr>
<th>$j$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

**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.

\[
\text{constant} \quad V + E \quad E \log V \quad V \log E \quad (V + E)^2 \quad \text{exponential}
\]

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

True \quad 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 \quad False
6. Regex. (11 points)

(a) Consider the regular expression

\(((A|B)D^*A^*)\)

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 \(\varepsilon\) transitions (as opposed to match transitions)? Circle the numbers of only the \(\varepsilon\) 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 \(\varepsilon\)), 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.

<table>
<thead>
<tr>
<th>mink</th>
<th>bear</th>
<th>bear</th>
<th>calf</th>
<th>crow</th>
<th>myna</th>
<th>crab</th>
<th>bear</th>
<th>bear</th>
</tr>
</thead>
<tbody>
<tr>
<td>moth</td>
<td>calf</td>
<td>calf</td>
<td>lamb</td>
<td>lamb</td>
<td>crab</td>
<td>toad</td>
<td>crow</td>
<td>calf</td>
</tr>
<tr>
<td>crow</td>
<td>crow</td>
<td>crow</td>
<td>hare</td>
<td>deer</td>
<td>lamb</td>
<td>swan</td>
<td>calf</td>
<td>crab</td>
</tr>
<tr>
<td>myna</td>
<td>crab</td>
<td>crab</td>
<td>wasp</td>
<td>crab</td>
<td>toad</td>
<td>bear</td>
<td>crab</td>
<td>crow</td>
</tr>
<tr>
<td>swan</td>
<td>deer</td>
<td>hare</td>
<td>hawk</td>
<td>hare</td>
<td>mule</td>
<td>deer</td>
<td>deer</td>
<td>deer</td>
</tr>
<tr>
<td>wolf</td>
<td>hare</td>
<td>kiwi</td>
<td>ibex</td>
<td>bear</td>
<td>hare</td>
<td>ibex</td>
<td>hare</td>
<td>hare</td>
</tr>
<tr>
<td>mule</td>
<td>hawk</td>
<td>deer</td>
<td>bear</td>
<td>kiwi</td>
<td>sole</td>
<td>hoki</td>
<td>hawk</td>
<td>hawk</td>
</tr>
<tr>
<td>slug</td>
<td>hoki</td>
<td>hawk</td>
<td>deer</td>
<td>calf</td>
<td>wolf</td>
<td>mule</td>
<td>hoki</td>
<td>hoki</td>
</tr>
<tr>
<td>hare</td>
<td>ibex</td>
<td>ibex</td>
<td>mink</td>
<td>hawk</td>
<td>calf</td>
<td>sole</td>
<td>ibex</td>
<td>ibex</td>
</tr>
<tr>
<td>bear</td>
<td>kiwi</td>
<td>hoki</td>
<td>lion</td>
<td>ibex</td>
<td>slug</td>
<td>wolf</td>
<td>kiwi</td>
<td>kiwi</td>
</tr>
<tr>
<td>kiwi</td>
<td>lion</td>
<td>lion</td>
<td>kiwi</td>
<td>hoki</td>
<td>moth</td>
<td>calf</td>
<td>lion</td>
<td>lamb</td>
</tr>
<tr>
<td>calf</td>
<td>lynx</td>
<td>lynx</td>
<td>slug</td>
<td>lion</td>
<td>kiwi</td>
<td>lamb</td>
<td>lynx</td>
<td>lion</td>
</tr>
<tr>
<td>hawk</td>
<td>lamb</td>
<td>lamb</td>
<td>toad</td>
<td>lynx</td>
<td>hoki</td>
<td>myna</td>
<td>lamb</td>
<td>lynx</td>
</tr>
<tr>
<td>ibex</td>
<td>mink</td>
<td>mink</td>
<td>hoki</td>
<td>mink</td>
<td>mink</td>
<td>mink</td>
<td>mink</td>
<td>mink</td>
</tr>
<tr>
<td>oryx</td>
<td>moth</td>
<td>mule</td>
<td>sole</td>
<td>mule</td>
<td>hawk</td>
<td>lynx</td>
<td>moth</td>
<td>moth</td>
</tr>
<tr>
<td>lion</td>
<td>myna</td>
<td>myna</td>
<td>wolf</td>
<td>myna</td>
<td>swan</td>
<td>lion</td>
<td>myna</td>
<td>mule</td>
</tr>
<tr>
<td>sole</td>
<td>mule</td>
<td>moth</td>
<td>moth</td>
<td>moth</td>
<td>lion</td>
<td>crow</td>
<td>mule</td>
<td>myna</td>
</tr>
<tr>
<td>wasp</td>
<td>oryx</td>
<td>wasp</td>
<td>crab</td>
<td>wasp</td>
<td>wasp</td>
<td>hare</td>
<td>oryx</td>
<td>oryx</td>
</tr>
<tr>
<td>lynx</td>
<td>swan</td>
<td>sole</td>
<td>crow</td>
<td>sole</td>
<td>bear</td>
<td>wasp</td>
<td>swan</td>
<td>slug</td>
</tr>
<tr>
<td>hoki</td>
<td>slug</td>
<td>oryx</td>
<td>oryx</td>
<td>oryx</td>
<td>deer</td>
<td>moth</td>
<td>slug</td>
<td>sole</td>
</tr>
<tr>
<td>crab</td>
<td>sole</td>
<td>slug</td>
<td>mule</td>
<td>slug</td>
<td>crow</td>
<td>slug</td>
<td>sole</td>
<td>swan</td>
</tr>
<tr>
<td>deer</td>
<td>toad</td>
<td>wolf</td>
<td>swan</td>
<td>wolf</td>
<td>ibex</td>
<td>kiwi</td>
<td>toad</td>
<td>toad</td>
</tr>
<tr>
<td>lamb</td>
<td>wolf</td>
<td>toad</td>
<td>myna</td>
<td>toad</td>
<td>oryx</td>
<td>hawk</td>
<td>wolf</td>
<td>wasp</td>
</tr>
<tr>
<td>toad</td>
<td>wasp</td>
<td>swan</td>
<td>lynx</td>
<td>swan</td>
<td>lynx</td>
<td>oryx</td>
<td>wasp</td>
<td>wolf</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(0) Original input  (1) 3-way radix quicksort  (2) LSD radix sort  (3) MSD radix sort  (4) Sorted
(no shuffle)
10. $G^2$. (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**:

$$\log N \quad \log^2 N \quad L + \log^2 N \quad L \log N \quad (\log N)(\log R L) \quad NL$$

(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: $2 \leq N/M \leq 8$. Do not include the time to compute the `hashCode`.

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

$$M + L \quad NL/M \quad ML/N \quad (N/M) \log R L \quad (\log R L) \left(1 + \frac{1}{1 - N/M}\right) \quad NL$$

(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: $1/8 \leq N/M \leq 1/2$. Do not include the time to compute the `hashCode`.

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

$$M + L \quad NL/M \quad ML/N \quad (N/M) \log R L \quad (\log R L) \left(1 + \frac{1}{1 - N/M}\right) \quad NL$$

(d) An $R$-way trie.

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

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

(e) A ternary search trie (TST).

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

$$R \quad L \quad R + L \quad RL \quad \log_R N \quad L + \log_R N$$
12. Reduction. (10 points)

(a) The \textsc{Find}-42\textsuperscript{nd} problem is to find the 42\textsuperscript{nd} 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 42\textsuperscript{nd} position. Given this, which of the following (if any) must be true? \textbf{Circle} the correct answer in each case.

\begin{itemize}
  \item \textsc{Find}-42\textsuperscript{nd} reduces to sorting. \hspace{1cm} \textbf{True} \hspace{1cm} \textbf{False}
  \item Sorting reduces to \textsc{Find}-42\textsuperscript{nd}. \hspace{1cm} \textbf{True} \hspace{1cm} \textbf{False}
  \item $O(N \log N)$ is a lower bound on \textsc{Find}-42\textsuperscript{nd}. \hspace{1cm} \textbf{True} \hspace{1cm} \textbf{False}
  \item $O(N \log N)$ is an upper bound on \textsc{Find}-42\textsuperscript{nd}. \hspace{1cm} \textbf{True} \hspace{1cm} \textbf{False}
  \item \textsc{Find}-42\textsuperscript{nd} must be NP-complete. \hspace{1cm} \textbf{True} \hspace{1cm} \textbf{False}
  \item \textsc{Find}-42\textsuperscript{nd} cannot be NP-complete unless P = NP. \hspace{1cm} \textbf{True} \hspace{1cm} \textbf{False}
\end{itemize}

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

\begin{itemize}
  \item $O(N)$ is a lower bound on sorting. \hspace{1cm} \textbf{True} \hspace{1cm} \textbf{False}
  \item $O(N)$ is an upper bound on sorting. \hspace{1cm} \textbf{True} \hspace{1cm} \textbf{False}
  \item Sorting is strictly harder than \textsc{Find}-42\textsuperscript{nd}, so can never be accomplished in $O(N)$ time. \hspace{1cm} \textbf{True} \hspace{1cm} \textbf{False}
  \item New developments in sorting might result in an asymptotically faster algorithm for \textsc{Find}-42\textsuperscript{nd}. \hspace{1cm} \textbf{True} \hspace{1cm} \textbf{False}
\end{itemize}
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 $E$. 