CSCE 221: Data Structures and Algorithms
Fall 2015
Exam 3

Name: __________________ Key: __________ UIN: __________________________ Section: __________

Instructions:

1. This is a closed book exam. Do not use any notes, books, or neighbors except your one page, two side, cheat sheet which MUST be turned in with your exam.

2. Show your work. Partial credit will be given. Grading will be based on correctness and clarity.

3. You have 120 minutes to complete the exam. Watch your time appropriately. You should take about 15 minutes per question section.

Integrity: The Aggie Honor Code is An Aggie does not lie, cheat, or steal or tolerate those who do. Upon accepting admission to Texas A&M University, a student immediately assumes a commitment to uphold the Honor Code, to accept responsibility for learning, and to follow the philosophy and rules of the Honor System. Students will be required to state their commitment on examinations, research papers, and other academic work. Ignorance of the rules does not exclude any member of the TAMU community from the requirements or the processes of the Honor System.

I agree to uphold this commitment and produce original work in this exam.

Signature: __________________________

DO NOT BEGIN THE EXAM UNTIL INSTRUCTED TO DO SO. GOOD LUCK!

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>received</td>
<td>possible</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
</tr>
</tbody>
</table>
1. (20 points, 2 points each) Provide the best answer you can for the following questions.

(a) Let \( G = (V, E) \) be an undirected graph, and let \( n = |V| \) and \( m = |E| \). An edge-list representation of \( G \) requires \( O(n + m) \) storage, an adjacency-list representation of \( G \) requires \( O(n + m) \) storage, and an adjacency-matrix representation of \( G \) requires \( O(n^2) \) storage.

(b) Let \( G = (V, E) \) be an undirected graph and assume it is stored in an adjacency-list representation. A breadth-first-search of \( G \) can be performed in \( O(n + m) \) time, which is asymptotically \( <\), \( =\), or \( >\) (circle one) the time required to perform a depth-first-search of \( G \).

(c) \( \boxed{True} \) or False: Let \( G = (V, E) \) be an undirected graph, and let \( n = |V| \) and \( m = |E| \). Then, \( m = O(n^2) \) and \( \log m = O(\log n) \).

(d) Consider the following weighted undirected graph. Number the edges in the order in which they would be added to the minimum spanning tree (MST) by the Prim-Jarnik MST algorithm starting from vertex \( c \) (recall that the Prim-Jarnik algorithm requires the evolving MST to be connected). This algorithm runs in time \( O((n + m) \log n) \).

![Graph](image)

(e) The single-source shortest path problem is only defined (i.e., makes sense) for weighted graphs which do not have \( \boxed{Negative \text{ weighted cycles}} \).

(f) If all edge weights are one, and the graph is undirected, then the single-source shortest path problem can be solved most efficiently by performing a \( \boxed{Breadth first search} \) starting from the source vertex \( s \).

(g) The Bellman-Ford algorithm solves the single-source shortest path problem by iteratively propagating path distances outward from a source vertex and runs in \( O(nm) \) time, while Dijkstra’s algorithm grows the shortest path tree iteratively and runs in \( O((n + m) \log n) \).

(h) The transitive closure of a directed graph \( G \) is a graph \( G' \) which contains an edge between two vertices \( u \) and \( v \) when there is a path between them in \( G \). The Floyd-Warshall algorithm can compute this in \( O(n^3) \) time when the graph is implemented with an adjacency matrix.
(i) Reflection on programming. Name two things you learned about programming this semester:
   i. ________________________________
   ii. ________________________________

(j) Reflection on lecture. Name two things you learned about computer science theory this semester:
   i. ________________________________
   ii. ________________________________
2. (20 points) **Graph Representation and Traversal.** Let $G = (V, E)$ be an undirected, weighted graph. Assume that $G$ has no self-loops and that multiple/parallel edges are not allowed. An isolated vertex in $G$ is defined to be a vertex that has no adjacent edges.

(a) (8 points) Describe an algorithm for determining if a graph $G = (V, E)$ has an isolated vertex. You should provide pseudo-code and also explain each step in words. You should use a graph ADT in your algorithm, but you should not assume any particular implementation of the ADT.

**Algorithm has_isolated(G)**

**Input:** Graph $G = (V, E)$

**Output:** Determination of $G$ containing an isolated vertex

for all $v \in V$ do
  if $v$.degree() = 0 then
    return true
  return false

Essentially, this algorithm visits each vertex one by one and if its degree is 0, implying there are no adjacent edges, it will return true. Otherwise, every vertex is not isolated and the algorithm returns true.
(b) (12 points, 3 points each) For the questions below, give the worst-case running time of your algorithm in terms of $n$ (number of vertices) and $m$ (number of edges). Justify each answer.

i. $G$ is represented by an edge list. Assume you do NOT have access to a $O(1)$ time method for determining a vertex’s degree.
$O(nm)$ time. Each vertex is visited by the outer for loop, i.e., $O(n)$ iterations, and for each vertex computing the degree requires traversing the entire edge list, which takes $O(m)$ time.

ii. $G$ is represented by an adjacency list. Assume you do NOT have access to a $O(1)$ time method for determining a vertex’s degree.
$O(n + m)$ time. Each vertex is visited by the outer for loop, i.e., $O(n)$ iterations, and for each vertex computing the degree requires traversing its adjacency list, which takes $O(deg(v))$ time. Summing up all of the degree’s of all vertices is $O(m)$ time.

iii. $G$ is represented by an adjacency matrix. Assume you do NOT have access to a $O(1)$ time method for determining a vertex’s degree.
$O(n^2)$ time. Each vertex is visited by the outer for loop, i.e., $O(n)$ iterations, and for each vertex computing the degree requires traversing a row of the adjacency matrix, which takes $O(n)$ time.

iv. $G$ now has access to a $O(1)$ time method for determining a vertex’s degree.
$O(n)$ time. Each vertex is visited by the outer for loop, i.e., $O(n)$ iterations, and for each vertex computing the degree requires accessing information in $O(1)$ time.
3. (20 points) **Spanning Trees.** Let $G = (V, E)$ be a connected, undirected, weighted graph. Consider the following algorithm for computing a spanning tree of a graph.

**Algorithm new_span(G)**

**Input:** Graph $G$

**Output:** Free spanning tree $T$

1. $T \leftarrow \emptyset$
2. Labeling $L$ noting which nodes are already in the tree. $L[s] \leftarrow true$ for the first vertex and $L[v] \leftarrow false$ for all $v \neq s$
3. **while** $|T| < n - 1$ **do**
4. $E_{out} \leftarrow \{(u, v) | L[u] = true \land L[v] = false\}$
5. Let $(x, y) \in E_{out}$ be the edge of maximum weight
6. $L[y] \leftarrow true$
7. $T \leftarrow T \cup \{(x, y)\}$
8. **return** $T$

Answer the following questions about the algorithm. Assume that the set $E_{out}$ is constructed by checking every edge of $G$ (i.e., $E_{out}$ is constructed from scratch in every iteration of the while loop).

(a) (3 points) Assume that $G = (V, E)$ is a connected, undirected, weighted graph. What type of spanning tree does algorithm New-Span compute? **Maximum spanning tree**

(b) (3 points) Which spanning tree algorithm is this most similar to? Why?

Prim-Jarnik’s Algorithm. The above algorithm iteratively adds a single edge to the tree iteration, which is in similar fashion to Prim-Jarnik’s algorithm.

(c) (7 points) What is the asymptotic complexity of this algorithm? Let $n = |V|$ and $m = |E|$. Justify your answer.

$O(nm)$. The outer while-loop will iterate $n$ times. To compute the set $E_{out}$ takes time $m$ to go through the edge list. Computing the maximum edge in $E_{out}$ takes time $O(|E_{out}|) = O(m)$. All other operations can be done in $O(1)$ time with proper data structures.

(d) (7 points) Assume $G = (V, E)$ is represented by an adjacency list. Describe how you could modify the algorithm to compute $E_{out}$ iteratively, as in how would you update $E_{out}$ efficiently instead of recomputing it.

In each iteration for the newest vertex $v$ added to the tree, add to $E_{out}$ all incident edges of $v$ going to an unvisited vertex, and remove from $E_{out}$ all incident edges of $v$ going to an already visited vertex.
4. (20 points) **Shortest Path Algorithms.**

(a) (5 points) Show the shortest path tree starting from vertex $c$ in the weighted, directed graph shown below. You may indicate this by circling the weights of the edges that are included in the shortest path tree.

(b) (10 points) Show the shortest path tree that would be found by Dijkstra’s SSSP algorithm starting from vertex $c$ in the weighted directed graph shown below. Number the vertices indicating the order in which they would be added to the tree and circle the weights of the edges that are included in the shortest path tree.

(c) (5 points) Give an example of a weighted directed graph $G$ with negative-weight edges, but no negative weight cycle, such that Dijkstra’s algorithm incorrectly computes the shortest path distances from some start vertex $s$. Indicate which vertex is the start vertex that causes Dijkstra’s algorithm to fail, and explain why your graph causes the algorithm to fail. Hint the graph only has to be 3 nodes.

Following Dijkstra’s algorithm, after processing $s$, $v$ has the lowest weight of 1, so it is added next. But $v$’s true shortest path is not $\{s, v\}$, but rather $\{s, u, v\}$.
5. (20 points) **Topological Ordering.** A *topological ordering* (or topological sort) of a directed acyclic graph (DAG) $G = (V, E)$ is an ordering of the vertices $v_1, v_2, \ldots, v_n$ such that all edges go from lower numbered vertices to higher numbered vertices. That is, for all edges $(v_i, v_j) \in E$, $i < j$. Consider the following algorithm for performing a topological sort. Let $n$ and $m$ be the number of vertices and edges in $G$, respectively.

**Algorithm** `NEW_TOPOLOGICAL_SORT(G)`

**Input:** Directed acyclic graph $G = (V, E)$

1: for all $v \in V$ do
2: \hspace{1em} `in_count[v]` $\leftarrow$ 0
3: Stack $S$ $\leftarrow \emptyset$
4: for all $x \in V$ do
5: \hspace{1em} for all vertex $y$ such that $(x, y)$ is in $E$ do
6: \hspace{2em} `in_count[y]` $\leftarrow$ `in_count[y]` + 1
7: for all $v \in V$ do
8: \hspace{1em} if `in_count[v]` = 0 then
9: \hspace{2em} `S.PUSH(v)`
10: while $\neg$`S.EMPTY()` do
11: \hspace{1em} $x$ $\leftarrow$ `S.POP()`
12: \hspace{1em} `OUTPUT(x)`
13: \hspace{1em} for all vertex $y$ such that $(x, y)$ is in $E$ do
14: \hspace{2em} `in_count[y]` $\leftarrow$ `in_count[y]` - 1
15: \hspace{1em} if `in_count[y]` = 0 then
16: \hspace{2em} `S.PUSH(y)`

(a) (5 points) Assume $G$ is represented by an adjacency list. State the worst-case running time for the above algorithm.

\[ O(n + m) \]

(b) (5 points) Assume $G$ is represented by an adjacency matrix. State the worst-case running time for the above algorithm.

\[ O(n^2) \]

(c) (5 points) What happens if this algorithm is run on a graph $G$ that has any directed cycles (i.e., if $G$ is not a DAG)?

Not every vertex is output.

(d) (5 points) Describe a simple modification to the algorithm so that it will detect if $G$ has a directed cycle (i.e., if $G$ is not a DAG).

Keep a count of the number of vertices output. If at the end of the algorithm not all vertices are output, then report the DAG has a cycle.