Exam #2 Review Packet – COP 3503 – Fall 2019

Exam Date: Tuesday, Nov. 5 (4:30 – 5:50 PM in HPA1-119)

Note: You are responsible for any materials posted in Webcourses so far throughout the semester, even if I have forgotten to mention them here. If you have any questions about what will or will not be on the exam, please feel free to ask!

1. (*Graph Terminology*) What is a vertex? What is a node? What is an edge? How do we denote the set of vertices in a graph? What about the set of edges? What does G = (V, E) mean? What is a path? What is a cycle? When is a cycle a path? Can a path be a cycle? What is a simple path? What is a simple cycle? What does it mean for two vertices to be "adjacent?" What does it mean for an edge to be "incident to" a vertex? What is a tree? What does X(G) represent? What is a bi-partite graph? What is the relationship between bi-partiteness and X(G)? What are the graphs K_n, C_n, and $K_{m,n}$? How do we denote the empty graph on *n* nodes (e.g., the graph with *n* nodes and zero edges)? How many nodes and edges are in each of those graphs (K_n, C_n, $K_{m,n}$, and the empty graph on *n* nodes)? What are their chromatic numbers? On the exam, be prepared to find the chromatic number of other graphs, too. (What *is* a chromatic number?) What does it mean for a graph to be *k*-colorable? What algorithm did we see for determining whether a graph is 2-colorable? What does $\overline{G}$ stand for? How do we find the complement of a graph? How many distinct graphs are there on *n* vertices? (*See Practice Exam Problem #6 from the lecture notes on Introduction to Graph Theory*.) What are the differences between the following flavors of graphs: weighted and unweighted, directed and undirected, connected and unconnected? What is a multigraph?

   References: *Introduction to Graph Theory (Oct. 10 lecture); Graph Coloring and Graph Traversals (Oct. 15 lecture); Traversal Applications and Topological Sort (Oct. 17 lecture; see Graph Counting Problems)*

2. (*Graph Representations*) What are the trade-offs between adjacency matrices and adjacency lists for representing graphs computationally? When would we choose to use one representation over the other? What *are* adjacency matrices and adjacency lists? What are the big-oh runtimes for operations such as inserting a new edge or an entirely new vertex? What about deleting edges? How can an adjacency matrix be used to represent an undirected, unweighted graph? What about a digraph (weighted and unweighted) or a multigraph (unweighted)? On the exam, could you write code to read a graph from a text file? Could you write code to perform operations on an existing adjacency matrix or adjacency list?

   Reference: *Introduction to Graph Theory (Oct. 10 lecture)*

3. (*Traversals*) Be able to perform DFS and BFS on arbitrary graphs. Be able to determine whether a particular ordering of vertices qualifies as a DFS or BFS traversal. Could you code these up if asked? Could you code up DFS iteratively? What are the runtimes of DFS and BFS in terms of |V| and/or |E|? What about breadth (*b*) and depth (*d*)? What are the runtimes strictly in terms of |V|? What applications have we seen for these algorithms? What is a connected component, and how can we count all the connected components in a graph?

   References: *Graph Coloring and Graph Traversals (Oct. 15 lecture);* Traversal Applications and Topological Sort *(Oct. 17 lecture); Hamiltonian and Eulerian Paths/Cycles (Oct. 31 lecture; see Researchy Interlude – tentative)*

4. (*Topological Sort*) What is a topological sort? Be prepared to give a topological sort for a graph, determine whether a given vertex ordering is a valid topological sort for a graph, or determine how many valid topological sorts exist for some arbitrary graph. What conditions cause a graph not to have a topological sort? Can a disconnected graph have a topological sort? What applications have we seen for topological sort? What is a dependency graph?

   Reference: Traversal Applications and Topological Sort *(Oct. 17 lecture)*

5. (*Minimum Spanning Trees*) What is a minimum spanning tree? What is a spanning tree? Be familiar with how Prim's and Kruskal's algorithms work. Be prepared to trace through either or both of them. What is the difference between these algorithms? What data structures do they rely upon? Could you create a graph with exactly one minimum spanning tree? What about a graph with multiple minimum spanning trees? What are some examples of why we might want to find a minimum spanning tree for a graph?

   Reference: *Minimum Spanning Trees (Week #7 recitation)*

6. (*Dijkstra's Algorithm*) What is the purpose of Dijkstra's algorithm? What requirement(s) must a graph satisfy in order for Dijkstra's algorithm to work? What is the runtime of Dijkstra's algorithm in terms of |V| and/or |E|? What is the runtime strictly in terms of |V|? Be able to trace through changes in the *dist*[] array at each iteration of Dijkstra's algorithm. (*See Practice Exam Problem #1 from the lecture notes on Dijkstra's Algorithm.*) How can you recover the full path from the source vertex to another vertex in the graph after Dijkstra's has run?

   References: *Dijkstra's Algorithm and Greedy Overview (Oct. 22 lecture); Bellman-Ford (Oct. 24 lecture)*

7. (*Dijkstra's Algorithm*) Suppose we have multiple vertices that can be reached from our source vertex with the same overall path cost. Does the order in which we process these vertices ever change the outcome of our final paths? If so, under what conditions does that happen? If not, why not? (*See Practice Exam Problem #3 from the lecture notes on Dijkstra's Algorithm.*)

   Reference: *Dijkstra's Algorithm and Greedy Overview (Oct. 22 lecture)*

8. (*Graph Algorithm Runtimes*) If a graph algorithm's runtime is O(|E|), can we necessarily say it is also $O(|V|^2)$? If an algorithm's runtime is $O(|V|^2)$, can we necessarily say it is O(|E|)? When is one of these notations preferable over the other? Somewhat related: Prove that $\log(n^c) = c \cdot \log(n)$. (That result is used to show that $\log(n^2)$ is O(log n), which I mentioned when analyzing the runtime of the specific implementation of Dijkstra's algorithm we coded up in class.)

   References: *Traversal Applications and Topological Sort (Oct. 17 lecture; see BFS and DFS: Runtime and Space Complexities); Dijkstra's Algorithm and Greedy Overview (Oct. 22 lecture; see Runtime Considerations); Bellman-Ford (Oct. 24 lecture: see Runtimes: Bellman-Ford and Dijkstra; see also: Supplemental Reading)*

9. (*Bellman-Ford Algorithm*) What is the purpose of the Bellman-Ford algorithm? How does it work? How does it differ from Dijkstra's algorithm, and when would we choose to use it instead of Dijkstra's algorithm? What requirement(s) must a graph satisfy in order for Bellman-Ford to work? (Under what conditions is it a bad idea to use Bellman-Ford?) Why does the main loop of Bellman-Ford go from $i = 1$ to |V| - 1 (inclusively) instead of $i = 1$ to |V|? How does Bellman-Ford check for negative cycles, and why does that method work? What is the runtime of Bellman-Ford in terms of |V| and/or |E|? What is the runtime strictly in terms of |V|? How is the runtime for Bellman-Ford impacted by whether we use an adjacency list or adjacency matrix?

   Reference: *Bellman-Ford (Oct. 24 lecture)*

10. (*Greedy Algorithms*) What greedy algorithms have we seen so far this semester? What properties must a problem have in order to lend itself to a greedy solution? What is the coin change problem, and is it possible to come up with a set of coin denominations in which the greedy approach does not give us an optimal solution to the problem? I might ask you to explain what it is about a particular algorithm that makes it qualify as greedy. I could also give you a novel problem and ask for a greedy solution (which would be giving you a big hint about how to solve it). For the exam, I will not expect you to regurgitate solutions to the single-room scheduling, multiple-room scheduling, or continuous knapsack problems from recitation. If you have a general understanding of what greedy means, you should be set for any questions I ask about greedy algorithms on the upcoming exam.

    References: *Dijkstra's Algorithm and Greedy Overview (Oct. 22 lecture); Greedy Algorithms (Week #8 recitation)*

11. (*Backtracking*) What algorithms have we seen so far this semester that involved backtracking? (What problems did we solve with backtracking, either in code or on paper?) Could you write a fairly small backtracking algorithm if required? Could you make minor tweaks to an existing backtracking algorithm to cause it to find all possible solutions to a problem instead of just the first solution (and vice versa), as we did with the n-queens code in class? What techniques did we discuss for preventing the generation of duplicate states while backtracking? Be sure you understand how my solutions to the n-queens and Hamiltonian cycle problems work, and be familiar with the basic anatomy of backtracking algorithms.

    References: *Backtracking (Oct. 29 lecture); Hamiltonian and Eulerian Paths/Cycles (Oct. 31 lecture)*

12. (*Hamiltonian Paths/Cycles*) What is a Hamiltonian path? How does this differ from a Hamiltonian cycle (or "circuit")? What is the degree of a vertex? What is the difference between in-degree and out-degree? Does every graph with a Hamiltonian path also have a Hamiltonian cycle (and vice versa)? Does a complete graph always have a Hamiltonian cycle? Can a disconnected graph have a Hamiltonian path and/or cycle? How many different Hamiltonian cycles are there in K_n (assuming K_n is undirected) if we do not count permutations of the same cycle as unique? (For example, we would say that K_3 only has one Hamiltonian cycle.) What if K_n is directed? What if we *do* count permutations of the same cycle as unique?

    References: *Hamiltonian and Eulerian Paths/Cycles (Oct. 31 lecture); Hamilton and Euler (Week #9 recitation)*

13. (*Euler Paths/Cycles*) What is an Eulerian path? What is an Eulerian cycle (or "circuit")? Be familiar with the properties that imply (bidirectionally) that an undirected or directed graph has an Eulerian cycle or path. How could you easily determine whether a very large graph has an Eulerian path or cycle? Does every graph with an Eulerian path also have an Eulerian cycle (and vice versa)? Is it possible for a graph to have a Hamiltonian cycle but not an Eulerian cycle (and vice versa)? Under what conditions does a disconnected graph have an Eulerian path? What about an Eulerian cycle? For the exam, be able to identify whether an arbitrary graph contains (a) a Hamiltonian path, (b) a Hamiltonian cycle, (c) an Euler path, and/or (d) an Euler cycle. You could also be asked to draw graphs that have one (or several) of those features.

    References: *Hamiltonian and Eulerian Paths/Cycles (July 1 lecture); Hamilton and Euler (Week #9 recitation)*

14. (*Problem Solving*) Given a novel graph theory problem, be prepared to identify which algorithms and/or data structures from this unit of the course would be best suited to solving that problem. You might also be asked to write a program that manipulates an existing adjacency matrix or adjacency list in order to solve such a problem. A problem of this nature could also require that you use a HashSet or HashMap in order to solve it efficiently.

Note: There will not be any questions about the complexity theory material covered in labs (P, NP, and NP-Complete), If that material is tested at all, it won't be until the final exam.