CSCI 2300 — Introduction to Algorithms
Exam 1 Prep and Sample Questions (document version 1.2)

- Exam 1 is scheduled for 6:00-7:50PM on Thursday 2/20 in West Hall Auditorium
- Be sure to bring your RPI ID to the exam
- Exam 1 is open book(s), open notes; given that you will be in a seat with a small fold-up "desk," be sure you prepare and consolidate your notes
- For extra-time accommodations, watch your RPI email for an earlier start time and location
- Make-up exams are given only with an official excused absence (`http://bit.ly/rpiabsence`)
- Exam 1 covers everything up to and including Tuesday 2/18 lecture
- Exam 1 covers: Homeworks 1, 2, and 3; Labs 1, 2, and 3; and Recitations 1 and 2
- Key topics are graph algorithms, runtime efficiency using $O()$, and greedy algorithms
- All work must be your own; do not even think of copying from others
- Be as concise as you can in your answers; long answers are difficult to grade

Sample problems

Work on the problems below as practice problems as you prepare for the exam. **Also work on the "Warm-up problems" and graded problems from our first three homeworks.**

Feel free to post your solutions in the Discussion Forum (except for graded Homework 3 problems); and reply to posts if you agree or disagree with the proposed approaches/solutions.

Some selected solutions will be posted on Wednesday 2/19 (for your lab/recitation sections).

1. Given an adjacency list representation, write an algorithm to reverse all edges of given directed graph G. Make sure your algorithm is correct and runs in linear time.

   Repeat the above with an adjacency matrix representation.

   SOLUTIONS: Create a new (empty) adjacency list for reverse graph G^R. Walk through the adjacency list of G and if in the list for u, we find node v, add/insert u into the list for v in G^R as the first element in the list. Note that insertion as the first position takes constant $O(1)$ time, so overall, the algorithm is linear time.

   Use a similar systematic approach for an adjacency matrix representation.

2. A binary tree is a tree with 0, 1, or 2 child nodes. Let B_n denote the number of binary trees possible with n nodes. Therefore, $B_1 = 1$ and $B_2 = 2$.

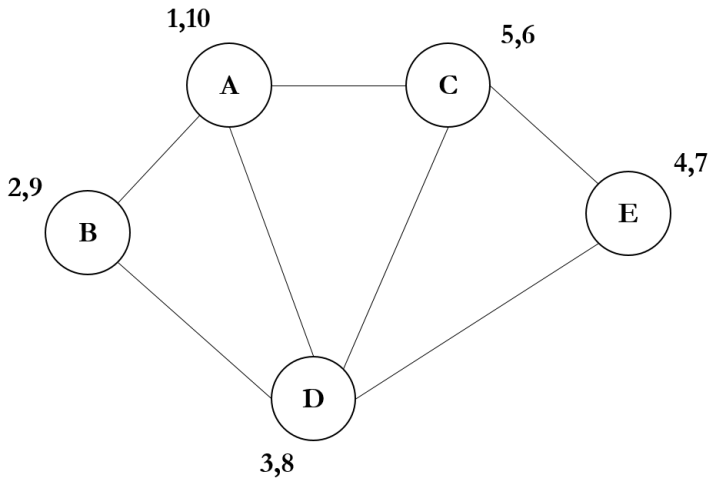
   (a) Determine B_3 and B_4 by drawing all possible binary trees in each case.

   (b) Generalize and define a recurrence relation for B_n. As a hint, consider how many nodes there are in the left subtree and therefore how many nodes there are in the right subtree.

   (c) Solve the recurrence relation and show using Big $O()$ notation.

3. Construct an undirected graph G with five nodes and seven edges such that the `pre` and `post` numbers (from the DFS algorithm) for all but one of the nodes differ by at least 3 (i.e., for each node u in G, $\texttt{post}(u) > \texttt{pre}(u) + 2$).

   SOLUTIONS: The graph shown below is one example solution.



4. Given directed acyclic graph G, write an efficient algorithm to identify whether there exists a node that can be reached by every other node in G. Show the runtime complexity of your algorithm.

   SOLUTIONS: To determine whether directed acyclic graph G contains a node reachable by every other node, we are looking to find a sink node with the constraint that G has only one sink node. To accomplish this, we can determine the outdegree of each node, counting the number of nodes with an outdegree of zero. If the resulting number of nodes with an outdegree of zero is exactly one, then we know that the sink node is reachable from all other nodes (remember, this is a DAG).

   The runtime is linear $O(|V| + |E|)$ since we "visit" all nodes and edges exactly once.

5. Write an algorithm to find a path that traverses all edges of directed graph G exactly once. You may visit nodes multiple times, if necessary. Show the runtime complexity of your algorithm.

   SOLUTIONS: This is asking to find what is called an Eulerian path in $G = (V, E)$. As a hint, start by taking a look at the indegree and outdegree values for all nodes.

   For all but two nodes in G, each node $u \in V$ must have indegree(u) equal to outdegree(u). For the two other nodes v and w, outdegree(v) is one larger than indegree(v) and indegree(w) is one larger than outdegree(w).

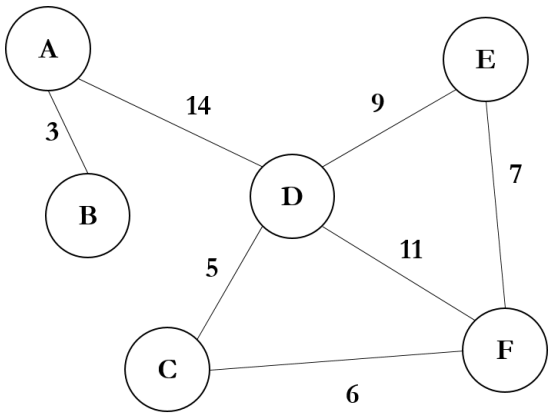
   (Try a few examples to see this.)

   Once the above properties are confirmed (which can be done in linear time), the path starts at v and ends at w. To determine the path, explore how you can find a Eulerian path from there....

6. Draw a connected weighted undirected graph with exactly six nodes that contains exactly three cycles; further, all edge weights must be distinct.

   Repeat the above, but this time a minimum spanning tree of the graph must contain the largest weighted edge.

   SOLUTIONS: The graph shown below is a connected weighted undirected graph with six nodes and three cycles. The cycles are $D-C-F-D$, $D-C-F-E-D$, and $D-F-E-D$. Further, the MST of this graph must contain largest weighted edge $A-D$. Overall, this MST contains edges $B-A$, $A-D$, $D-C$, $C-F$, and $F-E$.

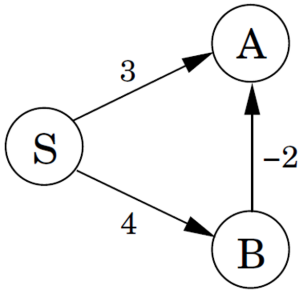


7. Design an algorithm that finds the largest weighted spanning tree of a given undirected graph. Show the runtime complexity of your algorithm.

   SOLUTIONS: Modify one of the MST algorithms such that at each iteration, the maximum-weight edge is selected instead of the minimum-weight edge. Alternatively, try multiplying all weights by -1, then applying one of the MST algorithms.

8. Give an example graph in which Dijkstra's algorithm fails to find the shortest path.

   SOLUTIONS: See Figure 4.12, copy-and-pasted here (and note the negative edge weight), as well as Section 4.6.1.



9. The *diameter* of a graph is the largest distance between any pair of nodes. Design an algorithm to find the diameter of a graph. Show the runtime complexity of your algorithm.

   SOLUTIONS: To find the diameter of a graph, first you need to find the shortest path between each pair of nodes. More specifically, apply Dijkstra's algorithm or the Bellman-Ford algorithm to each node. From this, the largest length is the diameter of the graph.

10. DPV Problem 3.13 (all parts)

SOLUTIONS:

(a) Consider the DFS tree of G starting at any node. If we remove leaf node v from this tree, we still have a tree that is a connected subgraph of the graph obtained by removing v. In other words, removing a leaf node does not "break" the properties of being a tree. Therefore, the graph remains connected even after removing v.

(b) A directed cycle. Removing any node from a cycle leaves a path that is not strongly connected.

(c) A graph consisting of two disjoint cycles. Each cycle is individually a strongly connected component; however, adding just one edge is not enough as it (at most) allows us to go from one component to another but not back.

11. DPV Problem 3.14 (all parts)

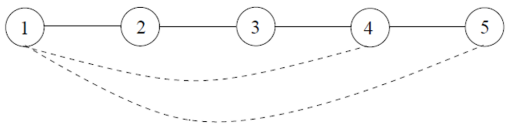
SOLUTIONS: The algorithm is to pick a source, delete it from the graph, then recurse on the resulting graph (i.e., repeat this process). Here, a source is just a node with indegree 0, which we can determine for each node in linear time. We can find all sources in the initial graph by performing a DFS and computing the indegree of all the nodes. We add all the nodes with indegree 0 to set S.

At each step, the algorithm then removes element X from S (i.e., a source node) and reduces the indegree of each of its neighbors by 1, which corresponds to deleting it from the graph. If this changes the indegree of any node to 0, we then add it S. Removed element X is assigned the next position in the ordering.

As noted above, computing all indegree values in the first step only requires a DFS, which takes linear time. Subsequently, we only look at each edge (u, v) at most once when we remove u from S. Therefore, the total time is linear.

12. DPV Problem 4.4

SOLUTIONS: The graph shown below is a counterexample, with nodes labeled with their level in the resulting DFS tree and back-edges shown as dashed lines. The shortest cycle consists of nodes $1 - 4 - 5$, but the cycle found by the algorithm is $1 - 2 - 3 - 4$. In general, the proposed strategy will fail if the shortest cycle contains more than one back edge.



13. DPV Problem 4.8

SOLUTIONS: The weighted graph shown below is a counterexample. In this graph, the shortest path between node B and node E is $B - C - F - E$ with a total weight of 1. The alternative (incorrect) path is $B - D - E$ with a total weight of 2.

According to the proposed algorithm, we could add say $+11$ to the weight of each edge, thereby making all edge weights positive. Then, the incorrect path $B - D - E$ would have a total weight of 24, which incorrectly is shorter than that of path $B - C - F - E$ with a total weight of 34.