CS3000: Algorithms & Data — Summer 2025 — Laney Strange

Exam 2 Practice Problems

- These practice problems are to help you prepare for Exam 2. We'll release the solutions on Tuesday (June 10) so you can go over them and ask questions in your recitation.
- Exam 2 takes places during lecture on June 12, 2025.
- The exam will be due at the end of lecture (1:20pm). You'll hand it in on paper, but we'll scan it later for grading on gradescope.
- You may bring one 8.5x11-inch paper as a cheat sheet, with anything written or typed on it (one side only). You will submit this cheat sheet along with your exam, and you will not be permitted to use any other materials or notes during the exams.
- If you have a DAS accommodation for exams, please arrange to take the exam at their center. Make sure you schedule that time ASAP if you haven't yet!

Problem 1. *Practice-BST*

Recall that, in a binary search tree, the data in every node is larger than its left subtree and smaller than its right subtree. For the BST drawn above, for any node n, we have the following properties:

- *n.key* (the integer contained in the node)
- n.left (the node's left child; NIL if n is a leaf)
- *n.right* (the node's right child; NIL if *n* is a leaf)
- *n.p* (the node's parent; NIL if *n* is a root)
- (a) Give complete pseudocode for a recursive post-order walk of a Binary Search Tree, i.e., recursively print the left and right subtrees before printing the root. Your function should take in one parameter, x, the root of the current subtree. You can assume you have access to a "print" function.
- (b) If you wanted to print out all the keys of a BST in sorted order, would you use a pre-order walk, in-order walk, or post-order walk?

Solution:

(c) Suppose we're concerned with a node x in a BST and we want to know its successor in sorted order. If x.right is not NIL, meaning that it has a right subtree, give pseudocode that would return x's successor. (Hint: you can use any BST functions we've seen in class and/or homework.)

Problem 2. Practice-BFS

This is a practice problem. A similar problem in an exam situation would also provide the original BFS pseudocode.

If we know BFS will be working on a binary tree instead of a general unweighted graph, we can do it more simply.

Give pseudocode for a new version of BFS that, given a binary tree, computes and returns the height of the tree (i.e., number of "levels" in the tree). It should use a queue like BFS, but without coloring, and without any vertex attributes (like d or π).

Your function should take in the root node of the tree, T, and return the height of the tree, an integer. All nodes in your tree have a *n.left* and a *n.right* attribute, either/both of which would be NIL if the node doesn't have a child on that side. *Hint:* You can assume that your queue has a *length* attribute, which can help you keep track of all the nodes at your current level.

Problem 3. Practice - Which Graph Algo

For each problem on the left, give the best matching algorithm that could solve the problem from the right.

- (1) The best way to lay out train tracks, if n cities need to have a station and you want to minimize the total amount of track used.
- (2) An order in which you can take classes with pre-requisites.
- (3) The fastest route to get from school to home.
- (4) How can an airline guarantee that they fly to all the places they say in their ads, but with as few flights as possible.
- (5) The fewest number of clicks to get from one web page to another.
- (6) The cheapest flight you can take, including possible stopovers, to get from Boston to Vancouver.

- (a) Breadth-First Search
- (b) Topological Sort
- (c) Minimum Spanning Tree
- (d) Dijkstra's Algorithm
- (e) APSP Algorithm

Problem 4. Practice - DFS

This problem is concerned with the directed graph below. In all subparts, assume that DFS is run on the graph starting with vertex 0, and that in case of a tie, the lower-valued vertex is visited first.



(a) Draw the DFS forest that would result after running the Depth-First Search algorithm on the graph.

Solution:

(b) Write out the vertices in the reverse-order in which they would be completed by the DFS algorithm.

Solution:

(c) The answer you reached in the previous section is not a valid topological sort of the graph. Why is this the case?

Problem 5. Practice - Greedy

Humpty Dumpty has fallen off the wall, and broken into n different pieces. All the king's horses are trying to put Humpty back together, but they need to make sure they have enough tape to repair him.

Humpty broke into n fragments of various sizes s_1, s_2, \ldots, s_n . Since bigger fragments are heavier, they need more tape to stick together. When we put two fragments with sizes s_i and s_j back together, we replace the two fragments with a single fragment of size $s_k = s_i + s_j$, and we use $s_i + s_j$ pieces of tape. We would like to completely put Humpty Dumpty back together, i.e. end with a single fragment, using the least amount of tape possible.

(a) Assume for this part that Humpty is size 28, and split into n = 4 pieces with sizes [4, 5, 7, 12]. What is the least amount of tape we can use to put Humpty together, and in what order do you connect the fragments?

Solution:

(b) For the general case, if you tackle this problem with a greedy approach, what would your greedy choice be at each step?

Solution:

(c) Give pseudocode for your greedy strategy. Your algorithm will take as input the value n > 0 and a min-heap of positive integers $[s_1, ..., s_n]$, with each integer representing the size of a fragment. Your algorithm should return the value of an optimal solution (i.e., the total amount of tape used for a Humpty of size n).

Problem 6. Practice - DP

The Fibonacci sequence follows a formula where the nth Fibonacci number is the sum of the two previous numbers. There are two base cases: the first and second Fibbonacci numbers are both 1.

(a) Give pseudocode for a bottom-up Dynamic Programming algorithm that computes and returns the nth Fibonacci number.

- (b) Show what your array/table looks like after calling your function with n = 6
- (c) Give a bound on the run-time of your algorithm

Problem 7. Practice - Greedy

You have an unlimited number of American coins with values: 50 cents, 25 cents, 10 cents, 5 cents, 1 cent. You want to determine the fewest number of coins needed to make some target.

(a) What's the fewest amount of coins you need to make 64 cents?

Solution:

(b) What's the fewest amount of coins you need to make 17 cents?

Solution:

(c) Write an algorithm that will find the fewest amount of coins needed to make n cents. You algorithm should take in n as well as c, an array of coin values with c = [50, 25, 10, 5, 1], and cv, the length of array c. It returns the value of an optimal solution.

Problem 8. MST- Kruskal

This is a practice problem. On an exam, we would provide the pseudocode for reference.

You are given the below graph. Assume you want to make a minimum spanning tree using Kruskal's algorithm. Draw what the MST would look like (solid lines for edges in the tree, dotted for those in the graph that are not in the tree) and label the order the edges would be chosen according to the algorithm



Problem 9. Practice - Binary Tree

A binary tree of depth k with $n = 2^{k+1} - 1$ vertices is given. Every vertex contains a hidden real number, all numbers are different. The vertex is called *maximal* if its number is greater than the numbers in all adjacent vertices. You can use the following operation: choose any vertex and get its number. Give an algorithm which finds a maximal vertex using O(k) such operations. Note that there could be several maximal vertices in the tree. You need to find only one of them, not all.

Problem 10. Practice - DP Shortest Paths

Let G(V, E) be a directed graph. Each edge $(u, v) \in E$ has an associated weight $w_{u,v}$ which could be positive, negative, or zero. Assuming G does not have a negative cycle (i.e., a cycle whose total weight is negative), we are interested in finding a path P from an origin node s to a destination node t with minimum total cost.

Because there are no negative cycles, you can assume that there is a simple path from s to t: if the minimum path takes n nodes, then you will use at most n - 1 edges.

Complete the recursive equation OPT(i, v) below. Here, OPT(i, v) represents the length of shortest path from a vertex v to destination t that uses $\leq i$ edges. (Feel free to write your answers below if they don't fit on the blank line.)

$$OPT(i, v) = \begin{cases} \infty & \text{if } i = 0 \text{ and } v \neq t \\ \hline & \text{if } i = 0 \text{ and } v = t \\ \hline & \text{if } i > 0 \end{cases}$$

Problem 11. Practice - Graph Properties

- (a) Show that every tree has at least one vertex with degree 1.
- (b) A bipartite graph is a graph G whose vertices can be divided into two sets, S and T, so that the only edges in G are edges between one vertex in S and one vertex in T. Give an algorithm to check if a graph is bipartite.
- (c) Show that a graph is bipartite if and only if it doesn't have a cycle of odd length (a cycle with 3 edges is an example of a cycle of odd length).

Problem 12. Practice - Greedy Graph Algorithms

Graph Coloring is an assignment of colors to vertices in a graph such that no two adjacent vertices share the same color. As an optimization problem, we try to color a graph with as few colors as possible.

(a) For the graph below, with vertices 1, 2, 3 and 4 sharing a color, and vertices 5, 6, 7, and 8 sharing a color, is the coloring an optimal solution?



Figure 1: Vertices numbered 1, 2, 3, 4 are all colored Red while those numbered 5, 6, 7, 8 are all colored Blue.

Solution:

(b) Suppose you attempt this optimization problem with a greedy approach: Your greedy algorithm takes in the graph G with adjacency-list representation and an array of colors c, with length |G.V|. Each vertex v would have an attribute v.color. Assuming the vertices are given in arbitrary order, visit each one of them and assign it the lowest-index color that is not assigned to a neighbor. Give a brief high-level description of how you would implement the algorithm. (Full pseudocode is not required here, but clear and descriptive bullet-points are.)

Solution:

(c) For your algorithm in Part B, what ordering of vertices would result in the coloring shown in Part A?

Solution:

(d) Suppose you are given 8 colors, and the same graph above. However, you get very unlucky in the initial ordering of vertices, the worst possible ordering your algorithm could get. In this situation, how many colors would your greedy algorithm end up using?