DSC 40B - Final Project

Due: Wednesday, December 9

About this "project": we have seen a lot of algorithms and data structures in DSC 40B, but we've studied most of them in isolation. In practice, the hardest part can be deciding which algorithms apply or how to model a problem in a way that makes it solvable. This "project" is an opportunity to see the "bigger picture" of how the DSC 40B content can be applied to real-world problems.

Unlike the homeworks, you will be graded on "correctness" in this final project. However, for most questions, there is no single correct answer – so it is more accurate to say that you'll be graded on how *reasonable* your answer is. Because of this, you should avoid one line answers and include a good amount of explanation and reasoning. We expect each problem to take at least a half a page of explanation (if not more!). Feel free to include drawings, diagrams, or other visualizations to help explain your reasoning.

Like any assignment these days, this final project is trivial for AI. We encourage you to think about each problem and come up with an organic answer. You can certainly ask an AI if your answer is reasonable and bounce ideas off of it, but you shouldn't just write down the output of the AI as your own answer. To dissuade you from doing this, **all answers must be handwritten and scanned in** (just like with the homeworks).

Note: slip days *are* permitted for the Final Project.

Problem 1.

In lecture, we said that finding a book in a large collection is easier when the collection is kept sorted. Find another real-world task that is easier with sorted structure. Explain 1) how you would sort in this situation if the items weren't already sorted and you had to do it by hand; and 2) how you would find an item in the sorted collection. Keep in mind that an algorithm that is fast for a computer may be cumbersome for a person (for example, a human probably wouldn't want to do a merge sort by hand to sort their books).

Problem 2.

You're building the leaderboard for a game with a million players, and you need to record the total score for each player. The operations that occur frequently are: adding a player, updating their score, removing a player, and determining a player's rank (i.e., "How many players have a higher score than me?"). You want to be able to answer rank queries quickly, even when scores change frequently.

Describe how you would store the players and their scores, and why you made that choice. What were alternatives that you considered, and how does your choice compare?

Problem 3.

You come home to your apartment after being gone for a week to find that someone stole your welcome mat. Good thing you have a doorbell camera that records video!

a) Describe two strategies for finding the point in the video where the welcome mat was stolen. Your first strategy should be "naive" – it is very simple, but will likely take you a long time. The second strategy should be more clever, and should allow you to find the point in the video much faster. Use concepts from DSC 40B wherever applicable.

b) Estimate how much time (in seconds, minutes, hours, etc.) it would take in the worst case to find the point in the video where the welcome mat was stolen using each of the two strategies you came up

with above. There's no single correct answer here, so be sure to state the assumptions that go into your estimate and explain your reasoning.

Problem 4.

Many fancy restaurants and event venues (especially in places that actually get cold) have *coat checks*, so that guests can check in their coats and pick them up after the event. If you think about it, a coat check is a data structure run by a person. Guests hand over coats (insertions) and later ask for them back (queries, followed by deletions), and the attendant wants both operations to be fast, even when hundreds of guests show up at once.

In this problem, you'll design a coat check process for a venue. You should make use of the concepts learned in DSC 40B, but you should also keep in mind that the coat check is run by a human, not a computer – and algorithms that may be fast for a computer to carry out might be cumbersome for a person. Again, there is no correct answer here, so focus on your reasoning and explanations.

a) Suppose you're in charge of designing a coat check process for a venue that only has 30 guests at a time. Describe a strategy for how the coat check should work, and analyze its performance. You should carefully explain: how coats are organized, how the attendant finds a good place for the coat, what information a guest uses to identify their coat (i.e., what information is on their ticket), how the attendant finds the coat when the guest comes back, and what happens if something goes wrong (e.g., a guest loses their ticket).

b) Now suppose you're in charge of designing a coat check process for a large venue. Describe a strategy that will work well even when hundreds of guests show up at once. Include the same details as in the previous part, and analyze its performance. You should also explain how your strategy is different from the one you described in the previous part. In particular, why won't your previous strategy work well for a large venue, and why isn't your new strategy ideal for a small venue?

Problem 5.

Many real situations are, or can be, represented by graphs. In each part, don't reuse an example that we used in class (e.g., a social network, a road map, or a graph of flights).

a) Describe a situation that you can model as an *undirected*, unweighted graph. What are the nodes, and when is there an edge between two of them? Why is the relationship symmetric (so the graph is undirected)?

b) In your example from the previous part, are the connected components of your graph meaningful? If not, come up with a different example where the connected components *are* meaningful. In either case, describe the connected components of your graph and explain what they mean in the context of your example.

c) Describe a situation that you can model as a *directed*, unweighted graph. What are the nodes, and when is there an edge from one to another? Why is the relationship asymmetric (so the graph is directed and not undirected)?

d) Describe a situation that you can model as a *weighted* graph with some positive and some negative edge weights. What are the nodes and what are the edges? What does a negative edge weight mean in your example? In your example, do the paths of a graph have a meaningful interpretation? If so, describe what they mean. What about the minimum spanning tree of your graph?

Problem 6.

In Lecture 13, we used a topological sort of the prerequisite graph to plan the order in which to take a set of courses. That plan assumes you take one course per quarter. Now let's generalize and assume that students may take up to three courses per quarter.

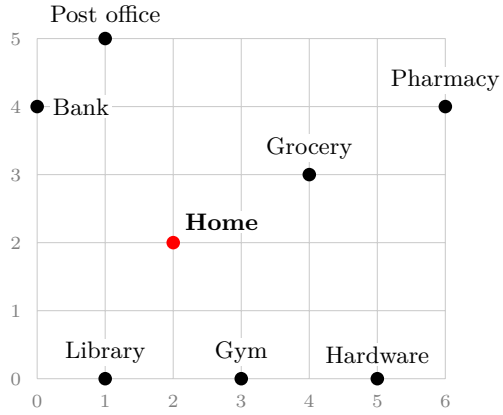
a) Design an algorithm that takes a prerequisite graph and produces a schedule: which courses to take in each quarter, with at most 3 courses per quarter, and every course taken in a later quarter than all of its prerequisites. Describe it in words or pseudocode, and demonstrate it on a small, non-trivial example of your own with at least 9 courses. It doesn't have to use DFS.

b) Is your algorithm optimal? That is, is it guaranteed to produce a schedule with the fewest possible quarters, for every prerequisite graph? If you think so, explain why. If not, give a prerequisite graph where it uses more quarters than necessary, and show a better schedule.

Problem 7.

You've just moved to Manhattan (which is famously laid out on a grid) and you have a list of errands to complete. Each errand takes you to a place to visit before coming back home. Finding the shortest circuit that visits every place and returns home is the *traveling salesperson problem*, which Lecture 18 said is hard (its decision version is NP-complete): nobody knows a polynomial-time algorithm for it. In this problem, you'll try a simple method that uses a minimum spanning tree (Lectures 16–17) instead, and see how good it is.

Each place's location is given in coordinates on the city grid. The distance between two places is the number of blocks you walk between them (left/right plus up/down). Home is at $(2, 2)$, the bank is at $(0, 4)$, the post office is at $(1, 5)$, the grocery store is at $(4, 3)$, the pharmacy is at $(6, 4)$, the hardware store is at $(5, 0)$, the library is at $(1, 0)$, and the gym is at $(3, 0)$.



Given those locations, the distances between them (in blocks) are:

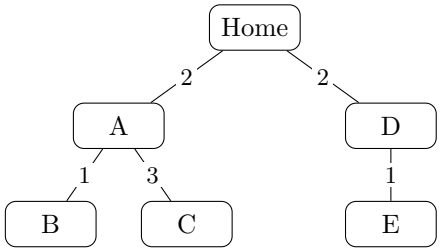
	Home	Bank	Post	Groc	Pharm	Hrdwr	Libr	Gym
Home	–	4	4	3	6	5	3	3
Bank	4	–	2	5	6	9	5	7
Post	4	2	–	5	6	9	5	7
Grocery	3	5	5	–	3	4	6	4
Pharmacy	6	6	6	3	–	5	9	7
Hardware	5	9	9	4	5	–	4	2
Library	3	5	5	6	9	4	–	2
Gym	3	7	7	4	7	2	2	–

a) Find a minimum spanning tree of this graph by hand, with Prim's or Kruskal's algorithm. Show the order in which edges are added, draw the tree, and give its total weight.

b) There is a natural way of turning a minimum spanning tree into a circuit that visits every place exactly once and returns home:

1. Start at home and do a depth-first search of the tree, keeping track of the order in which you first reach each place. In other words, walk around the spanning tree using only tree edges, like a depth-first search: go down a branch as far as you can, come back up until you reach a place with a branch you haven't walked yet, go down that one, and so on, until you're back home.
2. Write down the places in the order you *first* reach them.
3. Construct a circuit by visiting the places in that order, going straight from each one to the next (using the distances in the table, not the tree), and then go straight back home.

For example, suppose the MST were this tree, with the distances on its edges:



Walking the left branch first, the walk in step 1 would be Home, A, B, (back to A), C, (back to A, back to Home), D, E, (back to D, back to Home). It uses each tree edge twice, so it's $2 \cdot (2+1+3+2+1) = 18$ long. The places in the order they were first reached are Home, A, B, C, D, E, so the circuit is

$$\text{Home} \to \text{A} \to \text{B} \to \text{C} \to \text{D} \to \text{E} \to \text{Home}.$$

It takes three shortcuts: straight from B to C (skipping A), from C to D (skipping A and Home), and from E to Home (skipping D). Those aren't tree edges, so the tree doesn't give their lengths; in your problem, they'd come from the table of distances above. Here, suppose B to C is 3, C to D is 4, and E to Home is 3. Then the circuit's length is $2 + 1 + 3 + 4 + 1 + 3 = 14$.

What's the length of the circuit you get from your MST? Does the order in which you walk the branches matter? Try at least two orders.

c) Find the best circuit you can by hand, any way you like (by eye on a sketch of the grid, by trying changes to the circuit from part (b), …), and give its length. How does it compare with the circuit from part (b) and with the MST's weight? Was the MST circuit a good plan for a person, or would a person have done just as well by sketching the grid?

d) Explain why *every* circuit that visits all the places and returns home has length at least the MST's weight. (Hint: what do you get if you remove one edge from the circuit?)

Problem 8.

In the last couple of months, AI has helped solve several big open problems in computer science. Pick one of these results. Who or what organization published it? What was the result? Explain it in terms that a fellow DSC 40B student would understand and describe how it relates to DSC 40B. Explain whether the result was surprising (either to you or to computer scientists). Explain how important it is: will it change what people do in practice? There's no need to explain the details of the proof (or how the AI works); the important thing is to explain *what* was proved.