COMP 285 (NC A&T, Spr '22) Homework 7

Due. Friday, April 1st, 2022 @ 11:59 PM!

Homework Expectations: Please see Homework.

Exercises The following questions are exercises. We encourage you to work with a group and discuss solutions to make sure you understand the material.
Points This assignment is graded out of 50 points. However, you can get up to 60 points if you complete everything. These are not bonus points, but rather points to help make-up any parts you miss.

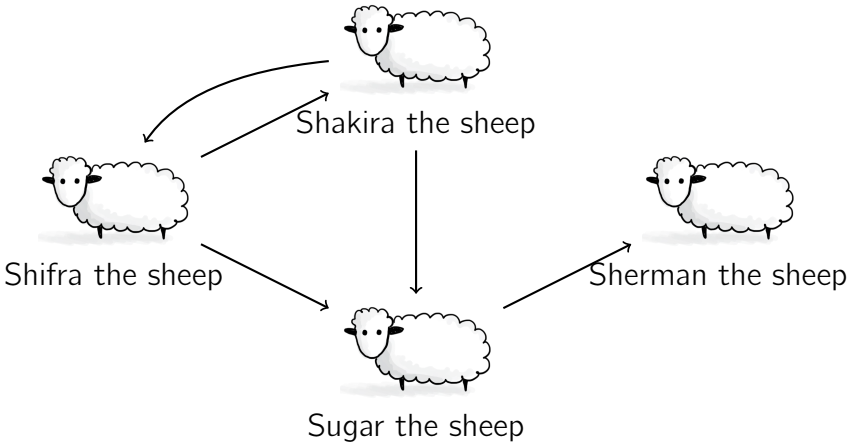
More Fun with Graphs and Dynamic Programming

Written Problems The following questions are to be submitted in written/typed form to gradescope.

1 Interview Practice: Wake up, Sheeple! (10 pt.)

You arrive on an island with n sheep. The sheep have developed a pretty sophisticated society, and have a social media platform called Baaaahtter (it's like Twitter but for sheep[1]). Some sheep follow other sheep on this platform. Being sheep, they believe and repeat anything that they hear. That is, they will re-post anything that any sheep they are following said. We can represent this by a graph, where $(a) \to (b)$ means that (b) will re-post anything that (a) posted. For example, if the social dynamics on the island were:

[1] Also my new start-up idea



then Sherman the Sheep follows Sugar the Sheep, and Sugar follows both Shakira and Shifra, and so on. This means that Sherman will re-post anything that Sugar posts, Sugar will re-post anything by Shifra and Shikira, and so on. (If there is a cycle then each sheep will only re-post a post once).

For the parts below, let G denote this directed, unweighted graph on the n sheep. Let m denote the number of edges in G.

1.1 The influencer circle (4 pt.)

Call a sheep an **influencer** if anything that they post eventually gets re-posted by every other sheep on the island. In the example above, both Shifra and Shakira are influencers.

Argue that, if there is at least one influencer, then all influencers are in the same strongly connected component of G, and every sheep in that component is an influencer. **[We are expecting:** A short argument explaning why the above is true.**]**

1.2 Who is the influencer? (6 pt.)

Suppose that there is at least one influencer. Give an algorithm that runs in time $O(n+m)$ and finds an influencer. You may use any algorithm we have seen in class as a subroutine.

[We are expecting: Pseudocode or a very clear English description of your algorithm, an informal justification that your algorithm is correct , an informal justification that the running time is $O(n+m)$.**]**

2 Interview Practice: Arbitrage in the Market (10 pt.)

Suppose the various economies of the world use a set of currencies $C_1, \cdots, C_n$ — think of these as dollars, pounds, bitcoins, etc. Your bank allows you to trade each currency C_i for any other currency C_j at an exchange rate r_{ij}, that is, you can exchange each unit of C_i for $r_{ij} > 0$ units of C_j. Due to fluctuations in the markets, it is occasionally possible to find a sequence of exchanges that lets you start with currency A, change into currencies, B, C, D, etc., and then end up changing back to currency A in such a way that you end up with more money than you started with. That is, there are currencies $C_{i_1}, \cdots, C_{i_k}$ such that

$$r_{i_1 i_2} \times r_{i_2 i_3} \times \cdots \times r_{i_{k-1} i_k} \times r_{i_k i_1} > 1$$

.

This is called an arbitrage opportunity, but to take advantage of it you need to be able to identify it quickly (before other investors leverage it and the exchange rates balance out again)! Devise an efficient algorithm to determine whether an arbitrage opportunity exists. Justify the correctness of your algorithm and its runtime.

[We are expecting: Pseudocode of your algorithm along with the running time and a short explanation.**]**

3 Concept Questions: A Clever Trick for Dijkstra's Algorithm (5 pt.)

Pepper and Plucky have been talking. They're working out whether they can modify Dijkstra's algorithm to deal with negative edge weights. Here's what they came up with:

Let $G = (V, E)$ be a weighted graph with negative edge weights, and let w^* be the smallest (most negative) weight that appears in G. Consider a graph $G' = (V, E')$ with the same vertices as G. Then to construct the edges E', we do the following: for every edge $e \in E$ with weight w, we add an edge $e' \in E'$ with weight $w - w^*$.

Now all of the weights in G' are non-negative, so we can apply Dijkstra's algorithm. The shortest path in G' will be the shortest path in G.

Does this suggestion work? (That is, does it always return a shortest path from s to t in G if it exists?) Either prove that it is correct (that is, prove that this algorithm correctly finds shortest paths in weighted directed graphs), or give a counter-example.

[We are expecting: Either a proof that this modification to Dijkstra's Algorithm works and lets us handle negative edge weights, or a single example of a graph for which this does not work.**]**

4 Pruning Trees (10 pt.)

Suppose you are given a **binary** tree with n nodes, and each node has an associated weight, which is a positive integer. You would like to remove nodes from this tree such that the resulting tree has at most k nodes, and you would like to maximize the sum of the weights of the remaining nodes. Additionally, at the end of the pruning, you must still have a tree rooted at the original root; in other words, if you remove a node, then all of that node's children/descendants must also be removed.

1. **(5 pt.)** For any node u of the original tree, and any positive integer $i \leq k$, let $A[u, i]$ denote the maximum weight of any subtree rooted at node u having at most i nodes. Letting r_u and ℓ_u denote the right and left children of node u (they are NULL if u does not have that child), and letting w_u be the weight of node u, formally describe the optimal structure by giving a recurrence that expresses $A[u, i]$ in terms of the quantities $A[r_u, 1], A[r_u, 2], \ldots$ and $A[\ell_u, 1], A[\ell_u, 2], \ldots$.

   [We are expecting: An expression for the recurrence relation for $A[u, i]$. If you find it convenient, you may additionally define $A[u, 0]$ and/or $A[NULL, i]$ to be an appropriate value, and use them in the recurrence.**]**

2. **(5 pt.)** Using the recurrence relation from the previous problem, define a dynamic programming algorithm that will efficiently solve the problem. (The algorithm only has to return the weight of the best tree. It doesn't have to return that best tree.) What is the runtime of your algorithm?

   [We are expecting: A brief description or pseudocode of your algorithm, no justification of correctness required. Additionally, the runtime and a brief description of how you arrived at your answer.**]**

Coding Problems The following questions are to be submitted as a ".zip" file on Gradescope.

5 Coding (25 pt.)

After completing the written portion of the assignment, you should submit it to Gradescope.

For the coding portion, get your starter C++ code or Python code.

Note that the starter code also include a few test cases you can run on repl.it. However, the full test suite is the one run on Gradescope.

Please reference the `README.md` included in your starter code for detailed instructions.

Submitting the Assignment

This assignment is a combination of written and programming questions. Both portions of the assignment should be submitted through Gradescope.

The "Homework 7: More Fun with Graphs and Dynamic Programming" assignment is the written portion, for which you should submit a **typed** response to the non-coding questions (questions 1-4). Each response should clearly be marked with its corresponding number. You are free to use the provided templates, print the questions and write your answers, or to simply type your responses on a blank document (whatever works for you).

The "Homework 7: Coding" is the programming portion of the assignment. For this portion, download the ".zip" file from replit and upload this ".zip" file as your answer to Gradescope. You can upload the assignment as many times as you want.