This homework has a total of 4 problems on 4 pages. Solutions should be submitted to T-square before 12:05pm on Monday Oct 17. Due to the modified due date, late submissions will accepted only until 12:05pm on Tuesday Oct 18.

The problem set is marked out of 15, you can earn up to $16 = 3 + 4 + 4 + 4$ points.

If you choose not to submit a typed write-up, please write neat and legibly. If the electronic version appears unclear, please also submit the originals, stapled, in addition to the T-square submission.

Collaboration is allowed/encouraged on problems, however each student must independently complete their own write-up, and list all collaborators. No credit will be given to solutions obtained verbatim from the Internet or other sources.

0. (1 point, only if all parts are completed)

(a) The answer to each question except question 0 should start on a new page.
(b) Please put your name and GT-ID on each of the answer sheets.
(c) Submit either a scan or photo of your homework in a single pdf file on T-square.
(d) Start the file name of your submission with your last name.

1. (3 points in 3 parts, moved from homework 2) Returning Negative Cycles

In this problem we will give a few more pointers about how the Bellman-Ford algorithm returns with a negative cycle. The question is mostly proof-based, so try to be as formal as you can.

The main idea is to keep a variable tracking where the last update came from. That is, as we update edges’s distances using

$$d[v] \leftarrow d[u] + l_{u \rightarrow v},$$

we also set

$$from[v] \leftarrow u.$$

(a) Show that if we have a negative cycle, the Bellman-Ford algorithm will still make updates at step $n$.

HINT: you may want to use proof by contradiction. A set of distances $d$ where no updates are possible satisfies

$$d[u] + l_{u \rightarrow v} \geq d[v]$$

1
(b) Show that at any point in the algorithm, we always have
\[ d[\text{from}[v]] + l_{\text{from}[v]\rightarrow v} \leq d[v] \]
for any vertex \( v \) whose \( \text{from}[v] \) value is not NULL. Note that \( d[\text{from}[v]] \) may have been updated after the update to \( d[v] \).

(c) Show that if we have a cycle of vertices \( u_1 \ldots u_k \) such that:
- \( \text{from}[u_{i+1}] = u_i \) for all \( 1 \leq i < k \),
- \( d[u_k] + l_{u_k\rightarrow u_1} < d[u_1] \).

Then this sequence of vertices gives a negative cycle.

**HINT:** add together inequalities in a way similar to part (a).

**NOTE:** A formal step that’s still missing is showing that a cycle in \( \text{from}[u] \) can be recovered if there has been \( n \) rounds of updates. This can be done by tracing back on the vertex that was updated at step \( n \).

2. (4 points) String Reduction

You are given a string \( x_1x_2\ldots x_m \) where each character \( x_i \) is drawn from an alphabet \( \Sigma \) of size \( n \).

For each pair of characters \( c_1, c_2 \in \Sigma \), there can be a conversion rule that replaces \( c_1c_2 \) by some character \( c \in \Sigma \). This can be thought of as an \( n \times n \) matrix \( W \) such that \( W(c_1, c_2) = c \) gives the conversion if it exists, or \( \text{NULL} \) if there isn’t a valid conversion from \( c_1c_2 \).

Your goal is to determine whether a string \( x_1x_2\ldots x_m \) can be reduced to a single character in \( \Sigma \) by a sequence of conversions, each involving two neighboring characters in the string (replacing \( x_ix_{i+1} \) with \( W(x_i, x_{i+1}) \)).

**Note that conversions can only involve neighboring characters.** For example, if \( W \) is such that \( W(a, b) = c \) and \( W(a, c) = d \) are the only two valid conversions, the string \( aaaaabbbbc \) can only be converted to \( aaacbbbc \), after which it can only become \( aadbbbc \).

(a) (1 point) Consider the following greedy algorithm: find the first \( i \) such that \( x_ix_{i+1} \) can be converted in the conversion list, convert it and repeat. Provide a short example (fewer than 8 characters) where this algorithm gives an incorrect answer.

(b) (3 points) Provide a dynamic programming algorithm which determines whether a string \( x_1x_2\ldots x_m \) can be converted to some (specified) character \( c \in \Sigma \) in \( O(m^3n^3) \) time.

**HINT:** See the chain matrix multiplication algorithm in DPV, chapter 6.5.
3. (Coyote Revolution)

Wile E. Coyote and The Road Runner are at it, yet again. Coyote, now enrolled at the Wild Wild West Institute of Technology, signed up for an ACME Student Prime membership in order to get a student discount (and more importantly, instant delivery) on his rocket purchases. Now, instead of just one rocket, Coyote can afford to purchase \( k \) rockets, where \( 0 \leq k \leq n \). The rocket doubles the Coyote’s speed between two nodes. That is, the Coyote can traverse an edge \( u \to v \) in time \( l_{u \to v} \) without using the rocket, and \( l_{u \to v}/2 \) with the rocket.

Your goal is to devise an efficient algorithm to find the quickest way to go from \( s \) to \( t \) while using up to \( k \) rockets.

(a) (1 point) A natural greedy approach is to first choose the shortest path, and then choose \( k \) longest edges on this path to use the rockets on. However, this is not always correct.

Give an example with 5 or fewer edges where this greedy approach doesn’t give the shortest route.

(b) (3 points) Using dynamic programming and shortest paths, give an \( O(n^{10}) \) time algorithm to this problem.

4. (4 points) Garage Sale

You are having a garage sale at your house and you are trying to carry items from your attic to the street as efficiently as possible. You are given a suitcase that can not carry more than a total weight \( W \) and total volume \( V \) and you want to maximize the price of the objects that you can carry with you in a single trip trip.

You are given a set of objects: \( a_1, a_2, ..., a_n \) of

- prices \( p \): \( p_1, p_2, ..., p_n \)
- weights \( w \): \( w_1, w_2, ..., w_n \)
- volumes \( v \): \( v_1, v_2, ..., v_n \)

respectively. You can use each object at most once. Find a way to maximize \( p \) while making sure the collective weights and volumes of the objects selected do not exceed \( W \) and \( V \).

(a) (1 point) Consider the greedy algorithm of repeatedly adding the object that maximizes

\[
p_i \cdot \left(1 - \frac{w_i}{W \text{ remaining}}\right) \cdot \left(1 - \frac{v_i}{V \text{ remaining}}\right),
\]

with ties leading to unspecified behavior. Give an example with at most 5 objects where this algorithm leads to set of objects with suboptimal price, and does not encounter ties (aka. unspecified behavior).
(b) (3 points) Provide an $O(nVW)$ time dynamic programming algorithm to find maximum price.

**HINT:** The knapsack problem in Chapter 6.4 of the textbook, and will be covered in the Oct 12 class is useful for solving this problem.