This homework has a total of 4 problems on 3 pages. Solutions should be submitted to GradeScope before 3:00pm on Wednesday Oct 11.

The problem set is marked out of 20, you can earn up to $21 = 1 + 5 + 4 + 5 + 6$ points.
If you choose not to submit a typed write-up, please write neat and legibly.
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) Submit your homework to Gradescope.

(b) Student id is the same as on T-square: it’s the one with alphabet + digits, NOT the 9 digit number from student card.

(c) Pages for each question are separated correctly.

(d) Words on the scan are clearly readable.

Updates to this homework since version 0 are:

- Question 1, changed in v1 to explicitly specify that you only need to find the length of the subsequence, not the subsequence itself.

1. (5 points) Longest alternating subsequence. Recall the longest increasing subsequence problem. Now instead we want to find the maximum length subsequence that alternates increasing/decreasing, starting with increasing. That is, we want to find a sequence of indices $i_1 < i_2 < \ldots < i_k$ such that

- For each $1 \leq j < k$ with $j$ odd, $A[i_j] < A[i_{j+1}]$, and
- For each $1 \leq j < k$ with $j$ even, $A[i_j] > A[i_{j+1}]$.

(a) (3 points) Given an $O(n^2)$ time algorithm for finding the length of the longest alternating subsequence using dynamic programming.

(b) (2 points) Show that the problem of finding the longest alternating subsequence can be formulated and solved as a shortest path on $2n$ vertices.
2. (4 points) Minimum Spanning Tree

Consider the graph below.

(a) (2 points) Give a cut that shows edge \( e = \{B, D\} \) is in an MST. Be sure to specify the cut via a set of vertices.

(b) (2 points) Show using the cut rule that if we increase the weights of some edges in \( G \) to form \( H \), then an edge \( e \) that
   - is in a minimum spanning tree of \( G \), and
   - has the same weight in \( G \) and \( H \),

is in a minimum spanning tree of \( H \).

For simplicity you may assume that \( G \) and \( H \) has all edge weights distinct, aka. a unique minimum spanning tree.

Note: this is a late edit to remove potential ambiguities regarding the case of non-unique MSTs / minimum weighted edges on cuts (more details can be found on Piazza). It restricts the problem to cases that fit both interpretations.

3. (5 points) Bellman-Ford algorithm for finding shortest paths.

In class, we discussed the following variant of the Bellman-Ford algorithm, whose pseudocode is below.

\[
\text{While true} \{ \\
\quad \text{For all edges } e = u \rightarrow v \{ \\
\quad\quad \text{dist}[v] = \min(\text{dist}[v], \text{dist}[u] + l(u \rightarrow v)); \\
\quad\} \\
\quad \text{if no vertex’s dist value is changed, stop} \\
\}\]

The text book’s version (in Section 4.6.) instead fixes the iteration count at \( \Theta(n) \). We also showed in class that if we do indeed get to such an iteration count, there must be a negative cycle.

In this question, we will discuss how many times the outer loop of Bellman-Ford algorithm actually needs to run. For such a discussion, it is important to fix an ordering by which we consider the edges, that is we view the edges as a fixed list, \( e_1 \ldots e_m \).
(a) (1 point) For any \( n \), exhibit a graph and an ordering of the edges where the Bellman-Ford algorithm takes more than \( n/2 \) steps.

(b) (2 points) Show that for any graph \( G \) without negative cycles and any \( s-t \) pair, there exists an ordering of edges so that \( \text{dist}[t] \) (as defined in the pseudocode above) becomes the value of the true \( s \) to \( t \) distance in 1 step.

(c) (2 points) Consider the \( k \times k \) square grid with two-way edges of positive lengths. We can label each vertex by its row/column number, with the top left corner being \((1, 1)\), which is also our source vertex.

Then each edge is a 4 tuple, \((r_1, c_1) \rightarrow (r_2, c_2)\), and we can produce edge ordering by sorting lexicographically via these tuples. For example, on a \( 2 \times 2 \) grid, this would order the edges as:

\[
(1, 1) \rightarrow (1, 2) \\
(1, 1) \rightarrow (2, 1) \\
(1, 2) \rightarrow (1, 1) \\
(1, 2) \rightarrow (2, 2) \\
(2, 1) \rightarrow (1, 1) \\
(2, 1) \rightarrow (2, 2) \\
(2, 2) \rightarrow (1, 2) \\
(2, 2) \rightarrow (2, 1) 
\]

Show that for any \( k \), there exists a set of positive weights for these edges of the \( k \times k \) square grid so that the Bellman-Ford algorithm with \((1, 1)\) as the starting point takes at least \( k/10 \) iterations (the 10 was chosen as an attempt to remove the need of getting constructions with good constants).

4. (6 points) Coloring a graph

The minimum coloring problem asks to label the vertices with the fewest number of distinct colors so that no edge has two endpoints having the same color.

(a) (2 points) Consider the following greedy algorithm: find a maximum independent set, color it with color 1, remove it from the graph, and repeat (using colors \( \geq 2 \)). Show that this algorithm is suboptimal by providing a graph of size at most 5 where it does not produce the optimal coloring.

(b) (3 points) Give an \( O(4^n) \) time algorithm for finding the fewest number of colors needed to color a graph. One way that this can be done is to define a state to be the minimum number of colors needed to color a subset of the vertices, and the transition can then be an entire subset of vertices that can be colored using the same color.
(c) (1 points): Improve the running time to $O(3^n)$. 