Write your solution using LATEX. Submit this homework using handin41. This is a partnered assignment. You must work with exactly one partner. It is still OK to discuss approaches with others at a high level, but most of your discussions (and all of your detailed discussions!) should be just with your lab partner.

Note: Make sure your homework includes the names of both lab partners. Only one partner should submit files. Make sure the files you submit are in the cs41/hw/xx directory before calling handin41.

Note: Make sure your homework gives credit when you and your partner did not completely solve the problem on your own. If you worked with other(s) during lab, say so. If you discussed problems at a high level with others, say so. If you accidentally stumbled on the solution or significant parts of a solution while surfing the web, say so. Not including attributions is a violation of the academic integrity policy.

Hint: For each of these problems, focus on the choice a user might make in trying to build up an optimal solution to the problem. E.g. in the Steel Rod Problem, we noted that the optimal solution must make the left-most cut at \( k \) feet for some \( k \) (unless we sell the \( n \)-foot rod as is)

1. Steel Rod Problem. In class we discussed the following greedy approaches to the Steel Rod Problem:

- (a) Always choose the cut \( k \) that maximizes revenue \( P[k] \). Sell that rod, then repeat with the remaining \((n-k)\)-foot rod.
- (b) Always choose the cut \( k \) that maximizes revenue per foot \( \frac{P[k]}{k} \). Sell that rod, then repeat with the remaining \((n-k)\)-foot rod.

Neither approach will solve the Steel Rod Problem on all instances. For each problem, come up with a counterexample demonstrating the greedy approach does not always return an optimal revenue.

2. Longest Common Substrings. Let \( \Sigma \) be a finite set called an alphabet. (For instance, \( \Sigma \) could be \( \{0,1\} \) or \( \{a,b,c,\ldots,y,z\} \).) Let \( x \in \Sigma^n \) and \( z \in \Sigma^k \). We say that \( z \) is a substring of \( x \) if there are indices \( 1 \leq i_1 < i_2 < \cdots < i_k \leq n \) such that \( z_j = x_{i_j} \) for all \( 1 \leq j \leq k \).

A common substring of strings \( x \) and \( y \in \Sigma^m \) is a string \( z \) that is both a substring of \( x \) and a substring of \( y \). For example, if \( x = aaabbb \) and \( y = abedbb \), then some common substrings of \( x \) and \( y \) include: \( z_1 = ab, z_2 = b, z_3 = abbb \).

- (a) Give a polynomial time algorithm that takes strings \( x \in \Sigma^n \) and \( y \in \Sigma^m \) and returns the length of the longest common substring.
- (b) Modify the algorithm above so it also returns the substring itself. (if there are more than one longest common substring, you are free to return whichever you want.)

---

\(^1\)This is a fancy way of saying that \( x \) is an \( n \)-letter “word”, where letters are taken from \( \Sigma \). Similarly \( z \) is a \( k \)-letter word.
3. **ICPC problem 6853 (Concert Tour).** Like most ICPC problems, this problem contains a certain kind of humor. However, the problem was for a regional competition where English was not the native language. Some part of the humor seems to be lost in translation :) While the writing isn’t superb, the problem is well defined.

To help with the input/output part of the problem, I provided some skeleton code for this in 6853.cpp. Feel free to use it (attributing of course) or start from scratch.

4. **Moving on a Checkerboard (CLRS problem 15-6).** Suppose you are given an \( n \times n \) checkerboard and a single checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule. At each step you may move the checker to one of three squares:

   (a) the square immediately above,

   (b) the square that is one up and one to the left (but only if the checker is not already in the leftmost position),

   (c) the square that is one up and one to the right (but only if the checker is not already in the rightmost position).

Each time you move from square \( x \) to square \( y \), you receive \( P(x, y) \) dollars. You are given \( P(x, y) \) for all pairs \( (x, y) \) for which a move from \( x \) to \( y \) is legal. \( P(x, y) \) may be negative.

Give a polynomial-time algorithm that figures out the set of moves that will move the checker from somewhere along the bottom edge to somewhere along the top edge while gathering as many dollars as possible. Your algorithm is free to pick any square along the bottom edge as a starting point, and any square along the top edge as a destination in order to maximize the number of dollars gathered along the way. What is the running time of your algorithm?

5. **Extra Credit.** Solve the following recurrence relation.

   \[
   T(n) = cn + \sum_{k=1}^{n-1} T(k)
   \]

   \[
   T(1) = 1
   \]