Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Shortest Paths with Dynamic Programming!

In this problem we will design a dynamic programming algorithm for finding the shortest $S - E$ path in a DAG like the one above.

(a) What are the subproblems you need to solve?

(b) Can you define the optimal solution to any subproblem as a function of the solutions to other subproblems?

(c) The dependency graph of a DP is a directed graph in which each subproblem becomes a vertex, and edge $(u, v)$ denotes that subproblem $u$ requires the solution to subproblem $v$ in order to be solved. What does the dependency graph look like for this problem? What property of the dependency graph allows us to solve the problem using DP?

(d) If we proceed iteratively, what would be the best order to solve these subproblems?

(e) How many subproblems are there, and how long does it take to solve each? What is the overall running time of this algorithm?

(f) Run your algorithm on the example graph. What is the shortest path?
2 String Shuffling

Let \( x \), \( y \), and \( z \) be strings. We want to know if \( z \) can be obtained only from \( x \) and \( y \) by interleaving the characters from \( x \) and \( y \) such that the characters in \( x \) appear in order and the characters in \( y \) appear in order. For example, if \( x = \text{efficient} \) and \( y = \text{ALGORITHM} \), then it is true for \( z = \text{effALGiOReiIenTHMt} \), but false for \( z = \text{efficientALGORITHMS} \) (extra characters), \( z = \text{effALGORITHMMicien} \) (missing the final \( t \)), and \( z = \text{effOALGRicieITHMnt} \) (out of order). How can we answer this query efficiently? Your answer must be able to efficiently deal with strings with lots of overlap, such as \( x = \text{aaaaaaab} \) and \( y = \text{aaaaaac} \).

1. Design an efficient algorithm to solve the above problem and state its runtime.

2. Consider an iterative implementation of our DP algorithm in part (a). Naively if we want to keep track of every solved sub-problem, this requires \( O(|x||y|) \) space (double check to see if you understand why this is the case). How can we reduce the amount of space our algorithm uses?

3 Egg Drop

You are given \( k \) identical eggs and an \( n \) story building. You need to figure out the maximum floor \( \ell \in \{1, \ldots, n\} \) that you can drop them from without breaking them. Each egg will break if dropped from a floor greater than or equal to \( \ell \), and will never break when dropped from a floor less then \( \ell \). Once an egg breaks, you cannot use it any more. Let \( f(n, k) \) be the minimum number of drops to find \( \ell \).

(a) Find \( f(1, k), f(0, k), f(n, 1) \) and \( f(n, 0) \).
(b) Find a recurrence relation for \( f(n, k) \).

4 Greedy Cards

Ning and Evan are playing a game, where there are \( n \) cards in a line. The cards are all face-up (so they can both see all cards in the line) and numbered 2–9. Ning and Evan take turns. Whoever’s turn it is can take one card from either the right end or the left end of the line. The goal for each player is to maximize the sum of the cards they’ve collected.

(a) Ning decides to use a greedy strategy: “on my turn, I will take the larger of the two cards available to me”. Show a small counterexample \( (n \leq 5) \) where Ning will lose if he plays this greedy strategy, assuming Ning goes first and Evan plays optimally, but he could have won if he had played optimally.
(b) Evan decides to use dynamic programming to find an algorithm to maximize his score, assuming he is playing against Ning and Ning is using the greedy strategy from part (a). Help Evan develop the dynamic programming solution by providing an algorithm with its runtime and space complexity.