Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Longest Common Subsequence

In lecture, we covered the longest increasing subsequence problem (LIS). Now, let us consider the longest common subsequence problem (LCS), which is a bit more involved. Given two arrays A and B of integers, you want to determine the length of their longest common subsequence. If they do not share any common elements, return 0.

For example, given $A = [1, 2, 3, 4, 5]$ and $B = [1, 3, 5, 7]$, their longest common subsequence is $[1, 3, 5]$ with length 3.

We will design an algorithm that solves this problem in $O(nm)$ time, where n is the length of A and m is the length of B.

(a) Define your subproblem.

Hint: looking at the Edit Distance subproblem may be helpful.

(b) Write your recurrence relation.

(c) In what order do we solve the subproblems?

(d) What is the runtime of this dynamic programming algorithm?

(e) What is the space complexity of your DP algorithm? Is it possible to optimize it?

2 Optimal Set Covering

Recall the set covering problem that you saw in lecture. Given a collection $S_1, S_2 \dots S_m$ where each S_i is a subsets of the universe $U = \{1, 2, \dots n\}$, we want to pick the minimum number of sets from the collection such that their union contains all of U. We saw in lecture that Greedy cannot do better than an approximation. In this problem we will use dynamic programming to come up with an exact solution.

(a) How long does a brute force solution of trying all the possibilities take? How large can m be compared to n in the worst case?

(b) As seen in lecture, we cannot hope for a polynomial time algorithm here but we can try to do better than exponential time in m by aiming for exponential time in n instead. Describe a dynamic programming algorithm that accomplishes this in $O(2^n \cdot nm)$ time.

(a) Define your subproblem.

(b) Write your recurrence relation.

(c) Describe the order in which we should solve the subproblems.

(d) Analyze the runtime of this dynamic programming algorithm.

(e) Analyze the space complexity.

3 Finding More Counterexamples

In this problem, we will explore why we had to resort to Dynamic Programming for some problems from lecture instead of using a fast greedy approach. Give counter examples for the following greedy algorithms.

(a) For the travelling salesman problem, first sort the adjacency list of each vertex by the edge weights. Then, run DFS N times starting at a different vertex $v = 1, 2, \ldots N$ for each run. Every time we encounter a back edge, check if it forms a cycle of length N, and if it does then record the weight of the cycle. Return the minimum weight of any such cycle found so far.

(b) For the Longest Increasing Subsequence problem, consider this algorithm: Start building the LIS with the smallest element in the array and go iterate through the elements to its right in order. Every time we enounter an element $A[i]$ that is larger than the current last element in our LIS, we add $A[i]$ to the LIS.

(c) Can you construct a counter example for the previous algorithm where the smallest element in the array is $A[0]$?

4 Balloon Popping Problem.

You are given a sequence of n-balloons with each one of a different size. If a balloon is popped, then it produces noise equal to $g_{left} \cdot g_{popped} \cdot g_{right}$, where g_{popped} is the size of the popped balloon and g_{left} and g_{right} are the sizes of the balloons to its left and to its right. If there are no balloons to the left, then we set $g_{left} = 1$. Similarly, if there are no balloons to the right then we set $g_{right} = 1$, while calculating the noise produced.

After popping a balloon, the balloons to its left and right become neighbors. (Note that the total noise produced depends on the order in which the balloons are popped.)

In this problem we will design a polynomial-time dynamic programming algorithm to compute the the maximum noise that can be generated by popping the balloons.

Example:

Input (Sizes of the balloons in a sequence): ④ ⑤ ⑦
Output (Total noise produced by the optimal order of popping): 175

Walkthrough of the example:

- **Current State** ④ ⑤ ⑦
  Pop Balloon ⑤
  Noise Produced $= 4 \cdot 5 \cdot 7$

- **Current State** ④ ⑦
  Pop Balloon ④
  Noise Produced $= 1 \cdot 4 \cdot 7$

- **Current State** ⑦
  Pop Balloon ⑦
  Noise Produced $= 1 \cdot 7 \cdot 1$

- **Total Noise Produced** $= 4 \cdot 5 \cdot 7 + 1 \cdot 4 \cdot 7 + 1 \cdot 7 \cdot 1$.

(a) Define your subproblem as follows:

$$C(i, j) = \text{maximum amount of noise produced by popping balloons in the sublist } i, i+1, \ldots, j \text{ first before the other balloons.}$$

What are the base cases? Are there any special cases to consider?

(b) Write down the recurrence relation for your subproblems $C(i, j)$.

Hint: suppose the k-th balloon (where $k \in [i, j]$) is the last balloon popped, after popping all other balloons in $[i, j]$. What is the total noise produced so far?

(c) What is the runtime of a dynamic programming algorithm using this subproblem? What is the space complexity?

Note: you may assume that all arithmetic operations take constant time.

(d) Is it possible to further optimize the space complexity? In either case, explain your reasoning.

5 LP Basics

Linear Program. A *linear program* is an optimization problem that seeks the optimal assignment for a linear objective over linear constraints. Let $x \in \mathbb{R}^n$ be the set of variables and $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n$. The canonical form of a linear program is

$$\begin{aligned} &\text{maximize } c^\top x \\ &\text{subject to } Ax \leq b \\ &\qquad\qquad x \geq 0 \end{aligned}$$

Any linear program can be written in canonical form.

Let's check this is the case:

(i) What if the objective is maximization?

(ii) What if you have a constraint $Ax \leq b$?

(iii) What about $Ax = b$?

(iv) What if the constraint is $x \leq 0$?

(v) What about unconstrained variables $x \in \mathbb{R}$?

Dual. The dual of the canonical LP is

$$\begin{aligned} &\text{minimize } y^\top b \\ &\text{subject to } y^\top A \geq c^\top \\ &\qquad\qquad y \geq 0 \end{aligned}$$

Weak duality: The objective value of any feasible primal $\leq$ objective value of any feasible dual

Strong duality: The *optimal* objective values of these two are equal.

Both are solvable in polynomial time by the Ellipsoid or Interior Point Method.

6 Taking a Dual

Consider the following linear program:

$$\begin{aligned} \max\ & 4x_1 + 7x_2 \\ x_1 + 2x_2 &\leq 10 \\ 3x_1 + x_2 &\leq 14 \\ 2x_1 + 3x_2 &\leq 11 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Construct the dual of the above linear program.

7 LP Meets Linear Regression

One of the most important problems in the field of *statistics* is the *linear regression problem*. Roughly speaking, this problem involves fitting a straight line to statistical data represented by points (x_1, y_1), $(x_2, y_2), \ldots, (x_n, y_n)$ on a graph. Denoting the line by $y = a + bx$, the objective is to choose the constants a and b to provide the "best" fit according to some criterion. The criterion usually used is the *method of least squares*, but there are other interesting criteria where linear programming can be used to solve for the optimal values of a and b.

Suppose instead we wish to minimize the sum of the absolute deviations of the data from the line:

$$\min \sum_{i=1}^{n} |y_i - (a + bx_i)|$$

Write a linear program with variables a, b to solve this problem.

Hint: Create new variables z_i and new constraints to help represent $|y_i - (a + bx_i)|$ in a linear program.