CS 170 Efficient Algorithms and Intractable Problems
Fall 2017 Prasad Raghavendra and Sanjam Garg

Homework 7

Instructions: You are welcome to form small groups (up to 4 people total) to work through the homework, but you **must** write up all solutions by yourself. List your study partners for homework on the first page, or "none" if you had no partners.

If using LaTeX (which we recommend), you may use the homework template linked on this Piazza post to get started.

Begin each problem on a new page. Clearly label where each problem and subproblem begin. The problems must be submitted in order (all of P1 must be before P2, etc). For questions asking you to give an algorithm, respond in what we will refer to as the *four-part format* for algorithms: main idea, pseudocode, proof of correctness, and running time analysis.

Read the Homework FAQ Piazza post on Piazza before doing the homework for more explanation on the four-part format and other clarifications for our homework expectations.

No late homeworks will be accepted. No exceptions. This is not out of a desire to be harsh, but rather out of fairness to all students in this large course. Out of a total of approximately 12 homework assignments, the lowest two scores will be dropped.

Special Questions:

- *Shortcut questions*: Short questions are usually easy questions that give you opportunities to practice basic materials. However, we understand that some of you are very familiar with the topics and do not want to spend too much time on easy questions. Therefore, we design shortcut questions for this purpose. A shortcut question usually has multiple parts that build upon each other and are ordered by their difficulty level. You can work on those in order or start from wherever you like. However you only need to submit the last part you are able to solve. For example, if a question has 5 parts (a, b, c, d, e), you are confident about part e, you should submit part e without any of the previous four parts. If you are confident about d but not sure about e, you should submit d for grading purposes. Please clearly indicate in your submission which part you are submitting.
- *Redemption questions*: It is important that you carefully read the posted solutions, even for problems you got right. To encourage this, you have the option of submitting a redemption file, a few paragraphs in which you explain, for each problem you choose to cover, what you did wrong and what the right idea was in your own words (not cutting and pasting from the solution!), and appending it to your homework. For example, suppose that as you review your solutions to HW1, you realize you had misunderstood question 3 and answered it incorrectly. You would write down what you just learned, and then submit it in your HW2 assignment the following week. Because these are mainly for your benefit, feel free to format them however is most useful for you.
- *Extra credit questions*: We might have some extra credit questions in the homework for people who really enjoy the materials. However, please note that you should do the extra credit problems only if you really enjoy working on these problems and want an extra challenge. It is likely not the most efficient manner in which to maximize your score.

Due Wednesday, October 25, at 4:59pm

This homework emphasizes dynamic programming problems. In your solutions, note the following:

- When you give the main idea for a dynamic programming solution, you need to explicitly write out a recurrence relation and explain its interpretation.
- When you give the pseudocode for a dynamic programming solution, it is not sufficient to simply state "solve using memoization" or the like; your pseudocode should explain how results are stored.
- When proving a dynamic programming solution, you need only justify the recurrence relation.

0. Who did you work with?

List all your collaborators on this homework. If you have no collaborators, please list "none".

1. (★ level) A HeLPful Introduction

Find necessary and sufficient conditions on real numbers a and b under which the linear program

$$\begin{aligned} \max\ & x+y \\ ax+by &\leq 1 \\ x,y &\geq 0 \end{aligned}$$

(a) Is infeasible.

(b) Is unbounded.

(c) Has a unique optimal solution.

2. (★★ level) TeaOne

TeaOne Cory is rolling out two new products to increase their sales. TeaOne uses sugarmilk (10 cents per unit), tea (20 cents per unit), and orange juice (1 cent per unit). There are two new products that TeaOne is creating: MilkyTea and ZestyJuice. One unit of ZestyJuice is made with 5 units of sugarmilk, 1 unit of tea and 8 units of orange juice. One unit of MilkyTea is made with 12 units of sugarmilk and 16 units of tea. TeaOne's servers can't produce more than 60 ZestyJuice units a day, and 40 MilkyTea units a day. TeaOne Cory has a budget of b\$ that can be spent on materials. MilkyTea sells for 5\$ per case, and ZestyJuice sells for 4.5\$ per case. The goal is to maximize your profit in one day.

(a) Create a linear program to represent this problem.

(b) Find the dual of the linear program.

(c) Find the optimal solution as a function of b (assume you can produce non-integer numbers of cases and can purchase items in non-integer quantities).

3. (★★ level) Mountain pass You are a traveler trying to cross a mountain range from west to east. You have access to a local elevation map of the mountains in the form of a matrix M of numbers with n columns and m rows, where the entry $M[i, j]$ represents the elevation at longitude i and latitude j (think of it as (x, y) coordinates in a Cartesian plane). From point (i, j), you can get to $(i+1, j+1)$, $(i+1, j)$, or $(i+1, j-1)$. That is you always move east, but you can move straight or diagonally up or down. Naturally, you want your path to be relatively flat. So, your aim is to find a path from any point on the first column to any point on the last column of the matrix which minimizes the height of the tallest mountain along that path.

(a) Give an efficient algorithm (four-part solution) that, for each element $(0, j), 0 \leq j \leq m$, finds the path to any element $(n, j), 0 \leq j \leq m$ that minimizes the height of the highest point visited.

(b) Suppose you are a rather forgetful traveler. So you want your algorithm to consume as little memory as possible. Describe how to use only $O(m)$ memory for part (a). You don't need to edit your algorithm in the previous part.

4. (★★★★ level) The Hungry Caterpillar

You are a caterpillar on a tree G. Your tree has n branch points (or vertices), where fruits are located. Since you live in a tree, these points are connected in a tree-like manner. Upon arriving at a branch point, you will either encounter two branches leading out from the point, or no branches at all.

As a hungry caterpillar, you'd like to start from your home v, visit k branch points to eat fruit (where $1 \leq k \leq n$), then return to your home v. For two connected points i, j, it costs $\ell(i, j)$ energy for you to travel from branch point i to branch point j. You may visit vertices or edges more than once, and you may travel forward or backward across each branch. Give an algorithm to find the shortest possible closed walk that visits k *distinct* branch points, from a start and end point v. A four-part algorithm is required for this problem.

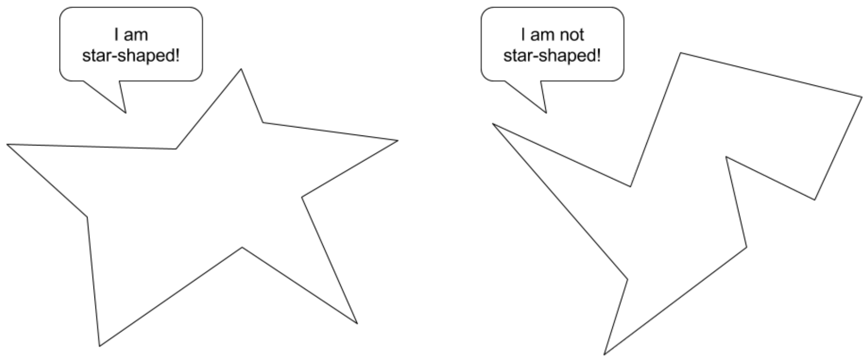
5. (★★★★ level) Star-shaped polygons in 2D

Consider a polygon in two dimensions. The polygon is specified by an ordered list $p_1 = (x_1, y_1), \dots, p_n = (x_n, y_n)$ where the n line segments between p_i and p_{i+1} for $i \in \{1, 2, \dots, n-1\}$ and p_n, p_0 do not intersect except at endpoints. The list is ordered in anti-clockwise order. In other words, if you draw each line segment in the order of the list, you will be able to draw the polygon without lifting your pencil (and the polygon will be drawn anti-clockwise).

Call a polygon *star-shaped* if there a point inside of the polygon from which all points on the boundary of the polygon can be seen. That is, the polygon is star-shaped iff there is a point x inside the polygon such that for every y on the boundary of the polygon, the segment xy only intersects the polygon at y.

Write a linear program that can be used to identify whether or not a polygon is star-shaped.

An example of two polygons is shown below:



6. (★★★★★ level) All Knight-er

Give an algorithm to find the number of ways you can place knights on an N by M chessboard such that no two knights can attack each other (there can be any number of knights on the board, including zero knights)? The runtime should be $O(2^{3M} \cdot N)$ (or symmetrically, switch the variables).

Hint: If you have a set of size M, how many subsets does that set have?

7. (??? level) (Optional) Redemption for Homework 6

Submit your *redemption file* for Homework 6 on Gradescope. If you looked at the solutions and took notes on what you learned or what you got wrong, include them in the redemption file.