CS 170 Fall 2017 | Efficient Algorithms and Intractable Problems
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.

0. Who did you work with?

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

Solution: N/A

1. (★ level) A HeLPful Introduction

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

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

(a) Is infeasible.

(b) Is unbounded.

(c) Has a unique optimal solution.

Solution:

(a) Note that for any possible real valued a and b, the point $x=0, y=0$ is always feasible (since $a\cdot 0+b\cdot 0=0\le 1$). Thus, the program is *not* infeasible for any choice of a and b.

(b) If the LP is unbounded, then the dual LP cannot be feasible (since, otherwise, by weak duality, the feasible solution for the dual will give an upper bound for the primal objective). Next, if an LP is bounded, then by the duality theorem, its (bounded) optimum is the same as that of its dual, which must therefore be feasible. Thus, this LP is unbounded if and only if its dual is infeasible.

We therefore write the dual of the original LP

$$\begin{aligned} \min & z \\ az &\ge 1 \\ bz &\ge 1 \\ z &\ge 0 \end{aligned}$$

Suppose $a>0$ and $b>0$. In this case, $z=\max(1/a,1/b)$ is a feasible solution for the dual. Thus, at least one of a,b must be non-positive for the dual to be infeasible. Suppose $b\le 0$. Then we have $bz\le 0<1$ for $z\ge 0$, so that the dual is infeasible. Similarly when $a\le 0$ the dual is infeasible. We conclude that the dual is infeasible if and only if $a\le 0$ or $b\le 0$. Thus, from the discussion above, the original LP is unbounded if and only if $\boxed{\text{at least one of } a \text{ and } b \text{ is non-positive}}$.

(c) From path (b), we see that the LP has a bounded optimum if and only if $a>0$ and $b>0$. In this case, the feasible reason is bounded by the lines $x=0$ and $y=0$ and $ax+by=1$, so that its vertices are $(0,0),(1/a,0)$ and $(0,1/b)$. Since $a,b>0$, the optimum is $\max(1/a,1/b)$, and is attained at one of the last two vertices.

Since every point inside the feasible region of an LP can be written down as a convex combination of vertices, it follows that the optimum is non-unique if and only if it is attained at at least two vertices. Since the only two vertices where the optimum can be attained in our LP are $(0,1/b)$ and $(1/a,0)$, we will have a unique optimum as long as the objective values at these points, $1/a$ and $1/b$, are different. We therefore get that the LP has a unique bounded optimum if and only if

$$a>0, b>0 \text{ and } a\ne b.$$

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).

Solution:

a) Without loss of generality, we can assume that no excess items are purchased (only materials that are used are purchased, as this is optimal).

Cost of each Zesty Juice case: $5 \cdot 0.1 + 1 \cdot 0.2 + 8 \cdot 0.01 = .78$; profit: $4.5 - .78 = 3.72$
Cost of each Sugar Water case: $12 \cdot 0.1 + 16 \cdot 0.2 = 4.4$; profit: $5 - 4.4 = .6$

LP:

$$\begin{aligned} \max \quad 3.72z + .6m \\ \text{subject to:} \quad z \leq 60 \\ m \leq 40 \\ .78z + 4.4m \leq b \\ z, m \geq 0 \end{aligned}$$

b) Multiplying by y_1, y_2, y_3 yields:

$$\begin{aligned} \max \quad 3.72z + .6m \\ \text{subject to:} \quad (y_1)z \leq (y_1)60 \\ (y_2)m \leq (y_2)40 \\ .78(y_3)z + 4.4(y_3)s \leq (y_3)b \\ z, s \geq 0 \end{aligned}$$

This yields the dual:

$$\begin{aligned} \min \quad & 60y_1 + 40y_2 + by_3 \\ \text{subject to:} \quad & y_1 + 3.4y_3 \geq 3.72 \\ & y_2 + 4.4y_3 \geq .6 \\ & y_1, y_2, y_3 \geq 0 \end{aligned}$$

c) We note that all other things equal, it is better to produce ZestyJuice if we can, as it costs less to make and earns a great profit per unit. Secondarily, we will want to make MilkyTea, as it does earn a profit, rather than making nothing. Finally, if our budget is so high that we can produce both MilkyTea and ZestyJuice to the maximum, we will then use the budget for nothing else, and earn no extra profit. Thus, our optimal value is as follows:

Buying solely ZestyJuice: $3.72\frac{b}{.78}$ for $b \leq 46.8$
Buying a mixture of both: $223.2 + .6 fracb - 46.84.4$ for $46.8 < b \leq 222.8$
Buying all of both: $223.2 + 24$ for $b > 222.8$

The corresponding solution, of form (z, m), is:
$(\frac{b}{.78}, 0)$ for $b \leq 46.8$
$(60, \frac{b-46.8}{4.4})$ for $46.8 < b \leq 222.8$
$(60, 40)$ for $b > 222.8$

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 every point on the first column to any point on the last column of the Cartesian plane 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 some element $(n, j), 0 \leq j \leq m$ that minimizes the height of the highest point visited.

(b) Consider the problem where we only want to return the max height of the mountain we'd have to cross if we are looking for a path that minimizes the height of the tallest mountain crossed (in other words, we want something similar to part 3a, but we need not reconstruct the path). It is possible to do this in $O(m)$ memory. Explain how in a few sentences. You don't need to edit your algorithm in the previous part.

Solution:

Main Idea:
Observe that from each point, you can only reach up to three others, all of which are in front of you. This means if we solve the problem from each grid point individually, we will have a DAG of subproblems and the nodes will have a constant degree. This fits the structure of a 2-D dynamic programming problem. Let's define a subproblem $H(i, j)$ to be the height of the highest mountain in the best path from point $M[i, j]$ to the east side of the mountain range, which is the end. Let's also say $N(i, j)$ stores the latitude (j-value) of the next point in the path, the point in column $i+1$. Observe that H follows the following recurrence relation:

$$H(i, j) = max\left[M[i, j], min\Big(H(i+1, j-1), H(i+1, j), H(i+1, j+1)\Big)\right]$$

Now we just need to solve for all values $H(0, j)$ and return the minimum, as well as the path from that point. Our base cases are that everything in the last column has it's own height as it's answer; that is: $\forall j, H(n, j) = M[n, j]$. We can iterate backward through the array one column at a time.

Pseudocode:

```
procedure OptimalTrek($M$)
    m, n := M.columnHeight, M.rowLength
    H, N ← $mxn$ arrays of ints                    ▷ Assume indexing out of bounds returns $+\infty$
    for $j = 1$ to $m$ do
        $H[n, j] := M[n, j]$
        $N[n, j] := -1$
    for $i = n - 1$ to 1 do
        for $j = 1$ to $m$ do
            $deps := [j-1, j, j+1]$
            $nextIndex := argmin_{k \in deps}(H[i+1, k])$
            $nextHeight := H[i+1, nextIndex]$
            $N[i, j], H[i, j] := nextIndex, max(M[i, j], nextHeight)$
```

```
bestStart := $argmin_{j \in 1 \dots m} H[1, j]$
$i, j, path := 1, bestStart, []$
while $j \neq -1$ do
    $path.append((i, j))$
    $j, i := N(i, j), i+1$
Output $H[1, bestStart], path$
```

Runtime: You are filling up a $m \times n$ matrix of subproblems. Each entry takes constant time to compute since there are 3 children for each node. So, this takes $O(mn)$ time in total.

Proof of Correctness: Our subproblems are defined such that the answer $H(i, j)$ is the answer to the problem we care about, which is the path to the east end that minimizes the highest peak. Therefore, some H values is our final answer; in particular, we are looking for the best H value on the west end of the mountain range, so we return the minimum of all of these. Additionally, we know that this value must be $M[i, j]$ for any grid point on the east end of the mountain pass, because the only point that path contains is the starting point. Therefore, our base cases are accurate, and tt remains to show that the recurrence relation is correct.

Since at any given point our options are to move to any of the three adjacent points in the next column, we consider these three our dependencies. Since all dependencies are in the following column, we can be sure our subproblems form a DAG, and that the leaves are the last column (which are our base cases). So, assume we are considering point (i, j), and we have filled out H and N for column $i+1$. To get to the end, we are required to reach one of these three spots. We can choose freely between them with no additional constraints, so our best choice is the one with the lowest H value. This is reflected in our pseudocode, and N will be populated with the index we choose in column $i+1$. H is the maximum height we see along the entire path, which is either the current height we are at, or the maximum height that occurs after this step, so we update it as such. Following these steps will allow us to solve every subproblem. Upon finishing, we can reconstruct the optimal path using the N values.

Part (b) Notice that all the subproblems in column i depend only on subproblems from column $i+1$. If we only care about the H value in the solution, then we do not need any information from column $i+1$ after we are done computing the solutions to the subproblems in column i. So, we can solve this in only $O(m)$ memory, by storing only two columns of subproblems at a time, the one we just finished and the one we are currently computing.

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

You are a caterpillar on a tree G. Your tree has n branch points, 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 $1 \le k \le n$), 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 tour that visits k distinct branch points, from a start and end point v. A four-part algorithm is required for this problem.

Solution: Main Idea:

Note that G is a full binary tree. We define our subproblem as $C[a,b]$, which is the shortest tour starting and ending at point a that visits exactly b distinct points, among a and its descendants. We now base our algorithm off of the following recurrence for non-leaf nodes, assuming a_1 and a_2 are the children of a:

Pseudocode:

```
procedure TreeTour($G = (V,E), v, k$)
    for $a$ in $V$ do
        $C[a][1] := 0$
    $v_1, \ldots, v_n$ := TopologicalSort(directed version of $G$ where nodes point to their children)
    for $a = n$ to 1 do
        for $b = 2$ to $k$ do
            if $v_a$.degree $= 1$ then
                $C[v_a][b] := \infty$
            else
                $a_1, a_2 := v_a$.children
                $C[v_a][b] := \min(\ell(a,a_1) + C[a_1][b-1] + \ell(a_1,a), \ell(a,a_2) + C[a_2][b-1]) + \ell(a,a_1)$
                for $i = 1$ to $b-2$ do
                    $C[v_a][b] := \min(C[v_a][b], \ell(a,a_1) + C[a_1][i] + \ell(a_1,a) + \ell(a,a_2) + C[a_2][k-i-1] + \ell(a_2,1))$
    Output $C[v][k]$
```

Note that we topologically sorted to ensure we have solved each vertex's children before trying to solve said vertex; an alternate approach would have been to use memoization and recursion.

Proof of Correctness:

Our base cases are that for each branch point a, $C[a][1] = 0$ because we don't need to go anywhere to explore 1 point, and $C[a][b] = \infty$ if $b > 1$ and the point has no children to visit.

For each other subproblem $C[a][b]$, we need to consider all possible ways of visiting b points starting from a; this means we need to visit $b-1$ of a's descendants. There are two types of ways. First, we could visit just one of a's immediate children, in which case we need to move along the branch between a and its child, find the best walk visiting $b-1$ points starting at the child (which we have already calculated, as we are iterating through the points in reverse topological order), and then move back from the child to a. Alternatively, we could we visit both of a's children, in which we case we explore i points along one branch and $b-1-i$ points along the other branch (again, both of which are subproblems that have been calculated). Thus, our algorithm considers all of these possible cases to find the shortest walk that goes through b vertices starting at a.

Runtime w/ Justification:

$O(nk^2)$. For each subproblem, we do $O(k)$ work, as we find the max over $O(k)$ terms. There are $O(nk)$ subproblems in total, so we do $O(nk^2)$ work in total.

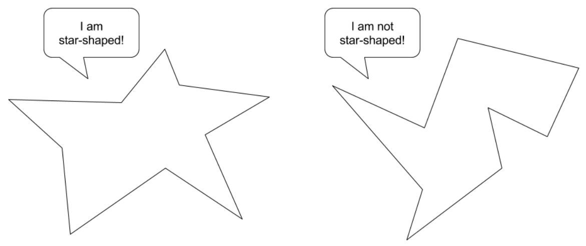
5. (★★★★ level) Star-shaped polygons in 2D

Consider a polygon in two dimensions. The polygon is specified by an ordered list $p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)$ where the n line segments between p_i and p_{i+1} for $i \in \{1, 2, \ldots, n-1\}$ and p_n,p_0 do not intersect except at endpoints. Since a polygon does not cross itself, the boundary of the polygon splits the plane into two regions. Each segment is adjacent to both regions and, accordingly, has two sides. Suppose that for each segment we are given as part of the input which side corresponds to the interior of the polygon.

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:



Solution:

The key insight is that we can treat the line segments themselves as the constraints in the LP, extending them infinitely as lines instead of just segments. Independent of all other line segments, a point can be "seen" if it lies anywhere on the side of the segment facing the interior of the polygon, even beyond the endpoints of the segment. (While it is true that sometimes a segment may block the view of another, this necessitates the existence of another segment connecting them for which the constraint would not be safisfied, so considering all the segments together works).

To construct our constraints, consider one segment of the polygon at a time. Notice that the interior of the polygon is adjacent to exactly one side of the segment and we are given this side as part of the input. For the line segment between points p_i and p_{i+1}, parameterize by the halfplane $a_i x + b_i y \geq c_i$ for some coefficients $a_i, b_i, c_i \in \mathbb{R}$. Call this halfplane H_i. (We can calculate these coefficients from the point values, which is detailed at the end).

Notice that if some point $p = (x, y)$ is not in H_i, then the line segment between p and p_i or between p and p_{i+1} must exit the polygon. In particular, p cannot affirm that the polygon is star-shaped. If p is in every H_i, then the path between p and any p_i must be contained inside of the polygon.

Therefore, a polygon is star-shaped if and only if there exists a point p in the intersection of all of the H_is. This is the same as asking whether or not the linear program

$$\max_{x,y} 0$$
$$\text{subject to } a_i x + b_i y \geq c_i \ \forall i \in \{1, 2, \ldots, n\}$$

is feasible.

Note: For $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, we can calculate the parameters of the line between them as

$$\begin{aligned} a &= (y_2 - y_1) \\ b &= (x_1 - x_2) \\ c &= (x_1 y_2 - x_2 y_1) \end{aligned}$$

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?

Solution: Wolog, we assume that $M < N$. We define $K(n,v,w)$ to be the number of ways we can put knights on an M by n chessboard such that the last M by 1 column has knights placed exactly in the positions specified in v, and the second-last M by 1 column has knights placed exactly in the positions specified in w (so v and w are length M bit vectors). We want to compute the sum of $K(N,v,w)$ over all length M bit vectors v and w.

By definition, $K(2,v,w) = 1$ if the M by 2 chessboard configuration defined by v and w is legitimate. Otherwise, $K(2,v,w) = 0$.

For $n > 2$, we have

$$K(n,v,w) = \sum_u K(n-1,w,u),$$

where we are summing over all possible configurations u for the third-last column of a chessboard whose last columns are specified by w and v.

We can precompute all valid M by 2 configurations and M by 3 configurations in $O(2^{3M}M)$ time because we can just check that each square with a knight is not being attacked by any other square with a knight. A square can only be under attack by at most 8 other squares so this check takes constant time.

We have $2^{2M}N$ subproblems $K(n,v,w)$ we wish to compute and each recurrence takes $O(2^M)$ time for a total running time of $O(2^{3M}N)$.

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.

Solution: N/A