CSE 421 Section 8

Linear Programming Practice
+ What Tool?
How to Determine Which Algo Paradigm to Try

Administrivia

Announcements & Reminders

- Midterm Exam

- HW6
  - Due yesterday Nov 15.

- HW7
  - Due on Nov 29 but no other section before then.

Linear Programming

Review of Key Concepts

Linear programming is the following problem:

$$\begin{aligned} &\textbf{max}\ c^T x \\ &\textbf{subject to}\ Ax \leq b \end{aligned}$$

In other words,

$$\begin{aligned} \textbf{max}\ & c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \\ \textbf{subject to}\ & a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n \leq b_1 \\ & a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n \leq b_2 \\ & \cdots \\ & a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n \leq b_m \\ & x \geq 0 \end{aligned}$$

Review of Key Concepts

Linear programming is the following problem:

$$\max c^T x$$
$$\text{subject to } Ax \leq b$$

Standard form is maximization with less than or equal to constraints

In other words,

$$\max c_1x_1 + c_2x_2 + \cdots + c_nx_n$$
$$\text{subject to } a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2$$
$$\cdots$$
$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m$$
$$x \geq 0$$

Non-negativity constraint

The Strategy

1. Read and understand the problem
2. Identify the constraints and the objective function
3. Transform the constraints to standard form
4. Correctness (and running time)

Problem 1 – Happy Eating

You are given a list of foods indexed $1, \ldots, n$, as well as the calories c_i, sugars (g) s_i, and vitamin D (mcg) d_i in each food. You're trying to maintain a healthy diet by eating exactly 2000 calories per day. You also heard that the American Heart Association recommends at most 30 grams of sugar per day. And because you just moved to Seattle from LA this year, it's your first winter and you need to eat at least 15 mcg of vitamin D to avoid SAD.

Along with the nutrition information, you also know that eating each food will give you happiness h_i. Find a way to compute a healthy diet that makes you happiest.

Problem 1.1 – Read the problem

Answer the usual quick-check questions:

- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

- What is the input type?

- What is the output type?

Work through this problem with the people around you, and then we'll go over it together!

Problem 1.2 – Identify constraints and objective

This sounds like a linear programming problem. From what you know so far, what would the constraints be? What is the objective function? What parts are not yet in standard form?

Work through this problem with the people around you, and then we'll go over it together!

Problem 1.3 – Transform to standard form

How can you transform the equations from the previous part to be into standard form?

Work through this problem with the people around you, and then we'll go over it together!

Problem 1.4 – Correctness and runtime

Explain why your algorithm is correct. For LP problems, the proof is usually just explaining how you've represented each part of the problem and relying on the correctness of the LP algorithm. (Runtime hasn't been covered in class yet, so don't worry about it for today.)

Work through this problem with the people around you, and then we'll go over it together!

What Tool?

In the real world...

When you come across a problem you'd like to solve (perhaps on an interview or on the final exam), you generally won't be told what kind of algorithm to write. You COULD just randomly start trying stuff, but that's going to be pretty inefficient, and you may waste a lot of time pursuing algorithmic paradigms that just won't work for the given problem.

Ideally, you want to be able to figure out what kind of algorithm to write in only a few tries. So how do you figure out what technique to use?

The Strategy

1. Read the Problem Carefully
2. Generate Example Inputs/Outputs
3. Ask Some Questions
4. Still Stuck?

Step 1: Read the Problem Carefully

- Are there any technical terms in the problem? Any words that look like normal words but really are technical terms?
- What is the input type?
- What is the output type?

If these questions look familiar, they should!

If you can’t answer these, there’s no way to figure out what technique to use—you don’t even know what problem you’re solving!

Step 2: Generate Example Inputs/Outputs

Spend a few minutes producing some examples. This will help make sure that you have a clear understanding of the problem. Like usual, it's helpful to think about a few different "typical" cases here, don't worry too much about edge cases yet.

Plus, you might stumble upon which technique to use here; for example, if you try to visualize an example input, and you start drawing a graph.

Step 3: Ask Some Questions

At this point, you should ask “is this really a graph modeling problem?” Some signs to look for:

- The problem mentions a graph or something graph-sounding (like “routes” or “maps”).
- There are “direct connections” between elements that could be edges.
- When you try to visualize an input example you end up drawing a graph.

Step 3: Ask Some Questions

If it doesn't feel like graph modeling, the next step to ask is probably "could I solve this problem recursively?" Try asking all of these:

- Is there a natural way to split things "in half" (or thirds, or...)?
- Could I make the problem a little bit smaller?
- What's "one step" toward the solution?

These might start leading you to either a divide and conquer (with the first bullet) or dynamic programming solution. In all cases, be sure you can see "how the recursion is helping".

Step 3: Ask Some Questions

Finally:

- Did an idea immediately jump to mind?
- Did you start a sentence with something like “Well, couldn’t I just...”?

Then maybe it’s time for a greedy algorithm. But really, it’s time for you to generate like 3 more examples and try them against your proposed algorithm. It’s not fun to write a bunch of code only to realize it doesn’t work! And greedy algorithms very often fail. Do some more checks before you jump into code writing.

Step 4: Still Stuck?

So, none of the earlier steps worked…

Take a deep breath, it’s going to be ok.

Step 4.1: Get a Baseline Algorithm

Figure out what “brute force” or any other baseline would be, and jot it down quickly on paper. And when you’re scared, look to your baseline like that hang-in-there-cat motivational poster. Worst-case, you’re going to use that one. And if you figured out that one, you could probably find another one.

Hang in there.

Step 4.2: Write a Few More Examples

And solve them. How are you, as a person, solving them? You're doing **some** process. See if you can reflect on it and realize what it looks like. That might inspire you toward an algorithm.

Step 4.3: Ask Yourself Some More Questions

- Does this remind you of any of the problems you've seen before? If so, a similar approach might work.
- Can you solve a simpler version of the problem? If there are two variables, make one of them a constant (a small constant, like 1) and ask "now what would I do?" Maybe you can generalize from there.
- Can you sort the input? Assume a graph is connected or topologically sorted? What if it's a tree? What if the array contains only positive elements? Any of these might give you inspiration for the general case.

2. Try it Yourself

Problem 2 – Try it Yourself!

For each of these problems, get far enough through these steps that you're able to guess what technique you might want to use. Then write down a sentence or two about what in the problem lead you toward that technique.

Work through the parts of this problem with the people around you, and then we'll go over it together!

Problem 2 – Try it Yourself!

a) There are a total of n courses you have to take, labeled from 1 to n. You are given a list prerequisites where $\text{prerequisites}[i] = (a_i, b_i)$ indicates that you must take course b_i first if you want to take a_i. Return true if you can finish all courses. Otherwise return false.

Problem 2 – Try it Yourself!

b) You are given a list of integers `coins` representing coins of different denominations and an integer `amount` representing a total amount of money. Return the fewest number of coins you need to make up that amount. If that amount of money cannot be made up by any combination of coins, return -1.

Problem 2 – Try it Yourself!

c) You are given an integer array `prices` where `prices`$[i]$ is the price of a given stock on the i^{th} day. On each day, you may decide to buy and/or sell the stock. You can hold at most one share of the stock at any time. However, you can buy it and then immediately sell it on the same day. Find and return the maximum profit you can achieve.

Problem 2 – Try it Yourself!

d) You are given an array of k linked-lists, where each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it.

Problem 2 – Try it Yourself!

e) Given an array of distinct integers `nums` and a target integer `target`, return the number of possible combinations that add up to `target`.

Problem 2 – Try it Yourself!

f) There are n cities. Some of them are connected, while some are not. If city a is connected directly with city b and city b is connected directly with city c, then city a is connected indirectly with city c. A province is a group of directly or indirectly connected cities and no other cities outside of the group. You are given an $n \times n$ matrix `isConnected` where `isConnected`$[i][j] = 1$ if the ith city and the jth city are directly connected, and `isConnected`$[i][j] = 0$ otherwise. Return the total number of provinces.

Problem 2 – Try it Yourself!

g) You are given an integer array `nums`. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position. Return true if you can reach the last index or false otherwise.

Problem 2 – Try it Yourself!

h) Write an efficient algorithm that searches for a value target in an $m \times n$ integer matrix. This matrix has the following properties:
   i. Integers in each row are sorted in ascending order from left to right.
   ii. Integers in each column are sorted in ascending order from top to bottom.

Problem 2 – Try it Yourself!

i) You are given an integer array `height` of length n. There are n vertical lines drawn such that the two endpoints of the ith line are $(i, 0)$ and (i,`height`$[i]$). Find two lines that together with the x-axis form a container, such that the container contains the most water. Return the maximum amount of water a container can store.

Problem 2 – Try it Yourself!

j) There is a group of n people labeled from 0 to $n-1$ where each person has a different amount of money and a different level of quietness. You are given an array `richer` where `richer`$[i] = [a_i, b_i]$ indicates that a_i has more money than b_i and an integer array `quiet` where `quiet`$[i]$ is the quietness of the ith person. All the given data in richer are logically correct (i.e., the data will not lead you to a situation where x is richer than y and y is richer than x at the same time). Return an integer array `answer` where `answer`$[x] = y$ if y is the least quiet person (that is, the person y with the smallest value of `quiet`$[y]$) among all people who definitely have equal to or more money than the person x.

Problem 2 – Try it Yourself!

k) Given an integer array `nums`, return an integer array `counts` where `counts`$[i]$ is the number of smaller elements to the right of `nums`$[i]$.

Problem 2 – Try it Yourself!

l) You are given several boxes with different colors represented by different positive numbers. You may experience several rounds to remove boxes until there is no box left. Each time you can choose some continuous boxes with the same color (i.e., composed of k boxes, $k \geq 1$), remove them and get $k * k$ points. Return the maximum points you can get.

Problem 2 – Try it Yourself!

m) There are n rooms labeled from 0 to $n - 1$ and all the rooms are locked except for room 0. Your goal is to visit all the rooms. However, you cannot enter a locked room without having its key. When you visit a room, you may find a set of distinct keys in it. Each key has a number on it, denoting which room it unlocks, and you can take all of them with you to unlock the other rooms. Given an array `rooms` where `rooms`$[i]$ is the set of keys that you can obtain if you visit room i, return true if you can visit all the rooms, or false otherwise.

Problem 2 – Try it Yourself!

n) Given an integer n, return the least number of perfect square numbers that sum to n. A perfect square is an integer that is the square of an integer, in other words, it is the product of some integer with itself. For example, 1, 4, and 9 are perfect squares while 3 and 11 are not.

Problem 2 – Try it Yourself!

o) You are given a network of n locations labeled from 1 to n. You are also given `times`, a list of travel times such that `times`$[i] = (u_i, v_i, w_i)$, where u_i is the source, v_i is the destination and w_i is the time it takes for a signal to travel from the source to the destination. We will send a signal from a given location k. Return the minimum time it takes for all n locations to receive the signal. If it is impossible for all n locations to receive the signal, return -1.

Problem 2 – Try it Yourself!

p) You are given two integers n and k and two integer arrays `speed` and `efficiency` both of length n. There are n engineers numbered from 1 to n. `speed`[i] and `efficiency`[i] represent the speed and efficiency of the ith engineer respectively. Choose at most k different engineers out of the n engineers to form a team with the maximum performance. The performance of a team is the sum of their engineers' speeds multiplied by the minimum efficiency among their engineers. Return the maximum performance of this team.

Problem 2 – Try it Yourself!

q) There are n piles of stones arranged in a row. The ith pile has `stones`[i] stones. A move consists of merging exactly k consecutive piles into one pile, and the cost of this move is equal to the total number of stones in these k piles. Return the minimum cost to merge all piles of stones into one pile.

Problem 2 – Try it Yourself!

r) A series of highways connect n cities numbered from 0 to $n-1$. You are given a 2D integer array highways where $\texttt{highways}[i] = [\texttt{city1}_i, \texttt{city2}_i, \texttt{toll}_i]$ indicates that there is a highway that connects $\texttt{city1}_i$ and $\texttt{city2}_i$, allowing a car to go from $\texttt{city1}_i$ to $\texttt{city2}_i$ and vice versa for a cost of $\texttt{toll}_i$.You are also given an integer discounts which represents the number of discounts you have. You can use a discount to travel across the ith highway for a cost of $\texttt{toll}_i$ /2 (integer division). Each discount may only be used once, and you can only use at most one discount per highway. Return the minimum total cost to go from city 0 to city $n-1$, or -1 if it is not possible to go from city 0 to city $n-1$.

Problem 2 – Try it Yourself!

s) A subsequence of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., ”ace” is a subsequence of ”abcde” while ”aec” is not). Given two strings source and target, return the minimum number of subsequences of source such that their concatenation equals target. If the task is impossible, return -1.

That's All, Folks!

Thanks for coming to section this week!
Any questions?