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 Egg Drop Revisited

Recall the Egg Drop problem from Homework 7:

You are given m identical eggs and an n story building. You need to figure out the highest floor $h \in \{0, 1, 2, \ldots n\}$ that you can drop an egg from without breaking it. Each egg will never break when dropped from floor h or lower, and always breaks if dropped from floor $h+1$ or higher. ($h = 0$ means the egg always breaks). Once an egg breaks, you cannot use it any more. However, if an egg does not break, you can reuse it.

Let $f(n, m)$ be the minimum number of egg drops that are needed to find h (regardless of the value of h).

Instead of solving for $f(n, m)$ directly, we define a new subproblem $M(x, m)$ to be the maximum number of floors for which we can always find h in at most x drops using m eggs.

For example, $M(2, 2) = 3$ because a 3-story building is the tallest building such that we can always find h in at most 2 egg drops using 2 eggs.

(a) Find a recurrence relation for $M(x, m)$ that can be computed in constant time given the previous subproblems. Briefly justify your recurrence.

Hint: As a starting point, what is the highest floor that we can drop the first egg from and still be guaranteed to solve the problem with the remaining $x - 1$ drops and $m - 1$ eggs if the egg breaks?

(b) Give an algorithm to compute $M(x, m)$ given x and m and analyze its runtime.

(c) Modify your algorithm from (b) to compute $f(n, m)$ given n and m.

Hint: If we can find h when there are more than n floors, we can also find h when there are n floors.

(d) Show that the runtime of the algorithm from part (c) is $O(nm)$. Compare this to the runtime you found in last week's homework.

(e) Show that we can implement the algorithm from part (c) to use only $O(m)$ space.

Solution:

(a)
$$M(x, m) = M(x-1, m-1) + M(x-1, m) + 1$$

We will first deduce i, the optimal floor from which we will drop the first egg given we have x drops and m eggs. If the egg breaks after being dropped from floor i, we have reduced the problem to floors 0 through $i-1$, and the strategy can solve this subproblem using $x-1$ drops and $m-1$ eggs. The optimal strategy for $x-1$ drops and $m-1$ eggs can distinguish between $M(x-1, m-1)+1$ floors, so we should choose $i = M(x-1, m-1)+1$.

On the other hand, if the egg doesn't break, we reduce the problem to floors i to n with $x-1$ drops and m eggs. The maximum number of floors we can distinguish using this many drops and eggs is $M(x-1, m)+1$, so we can solve this subproblem for n as large as $i + M(x-1, m) = M(x-1, m-1) + M(x-1, m) + 1$.

(b) For base cases, we'll take $M(0, m) = 0$ for any m and $M(x, 0) = 0$ for any x. Starting with $x = 1$, we compute $M(x, m)$ for all, $1 \le x \le m$, and do so again for increasing values of x, up until we compute $M(x, m)$ for all $1 \le x \le m$. We return $M(x, m)$.

We compute xm subproblems, each of which takes constant time, so the overall runtime if $\Theta(xm)$.

(c) Again, starting with $x = 1$, we compute $M(x, m)$ for all, $1 \le x \le m$, and do so for increasing values of x. This time, we stop the first time we find that $M(x, m) \ge n$, and return this value of x.

(d) Because there are only n floors, the optimal number of drops, x will always be at most n. From part (b), we know the runtime is $\Theta(xm)$, so if $x \le n$, we know the runtime must be $O(nm)$.

(This is a very loose bound, but it is still much better than the naive $O(n^2 m)$-time algorithm we'd get from using the recurrence on $f(n, m)$ directly.)

(e) While we're computing $M(x, m)$ for all $1 \leq x \leq m$, we only need to store $M(x-1, m)$ and $M(x, m)$ for all x, i.e. we only ever need to store $O(m)$ values. In particular, after computing $M(x, m)$ for all x, we can delete our stored values of $M(x-1, m)$.

2 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 minimizaiton?

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

(iii) What about $Ax = b$?

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

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

Solution:

(i) Take the negative of the objective.

(ii) Negate both sides of the inequality.

(iii) Write both $Ax \leq$ and $Ax \geq b$ into the constraint set.

(iv) Change of variable: replace every x by $-z$, and add constraint $z \geq 0$.

(v) Replace every x by $x^+ - x^-$, add constraints $x^+, x^- \geq 0$. Note that for every solution to the original LP, there is a solution to the transformed LP (with the same objective value). Similarly, if there is a feasible solution for the transformed problem, then there is a feasible solution for the original problem with the same objective value.

3 Huffman and LP

Consider the following Huffman code for characters a, b, c, d: $a = 0, b = 10, c = 110, d = 111$.

Let f_a, f_b, f_c, f_d denote the fraction of characters in a file (only containing these characters) that are a, b, c, d respectively. Write a linear program with variables f_a, f_b, f_c, f_d to solve the following problem: What values of f_a, f_b, f_c, f_d (that can generate this Huffman code) result in the Huffman code using the most bits per character?

Solution:

Our objective is to maximize the bits per character used:

$$\max f_a + 2f_b + 3f_c + 3f_d$$

We know the fractions must add to 1 and be non-negative:

$$f_a + f_b + f_c + f_d = 1, f_a, f_b, f_c, f_d \geq 0$$

We know the frequencies of the characters must satisfy $f_a \geq f_b \geq f_c, f_d$. We also know that $f_c + f_d \leq f_a$, since we chose to merge (c, d) with b instead of merging a. So we get the following constraints:

$$f_c \leq f_b, f_d \leq f_b, f_b \leq f_a, f_c + f_d \leq f_a$$

We can write this LP in canonical form as follows:

$$\max\ f_a + 2f_b + 3f_c + 3f_d$$
$$\text{subject to} \begin{cases} f_a + f_b + f_c + f_d \leq 1 \\ -f_a - f_b - f_c - f_d \leq -1 \\ f_c - f_b \leq 0 \\ f_d - f_b \leq 0 \\ f_b - f_a \leq 0 \\ f_c + f_d - f_a \leq 0 \\ f_a, f_b, f_c, f_d \geq 0 \end{cases}$$

Note that for the second-to-last constraint, we write $f_c + f_d - f_a \leq 0$ instead of just $f_c + f_d \leq f_a$ because canonical form requires all the variables to be on the left, and only constants on the right.

4 Simplex Practice

(a) Consider the following linear program:

$$\begin{aligned} \text{Maximize } & x + y \\ \text{subject to: } & -\frac{1}{2}x + y \leq 3 \\ & x + 2y \leq 12 \\ & y \leq 4 \\ & 3x + y \leq 21 \\ & x, y \geq 0 \end{aligned}$$

Draw the feasible region. Write out the sequence of vertices traversed by the simplex algorithm starting at $(0, 3)$.

(b) Consider the following linear program on three variables:

$$\begin{aligned} \text{Maximize } & f(x, y, z) \\ \text{subject to: } & x + 2y \leq 10 \\ & z \leq 3 \\ & x, y, z \geq 0 \end{aligned}$$

Is it possible for $p = \{(0, 0, 0), (0, 5, 0), (10, 0, 0), (10, 0, 3)\}$ to be a valid path that the simplex algorithm takes for some linear function f? If so, is there a linear function, f, such that p is

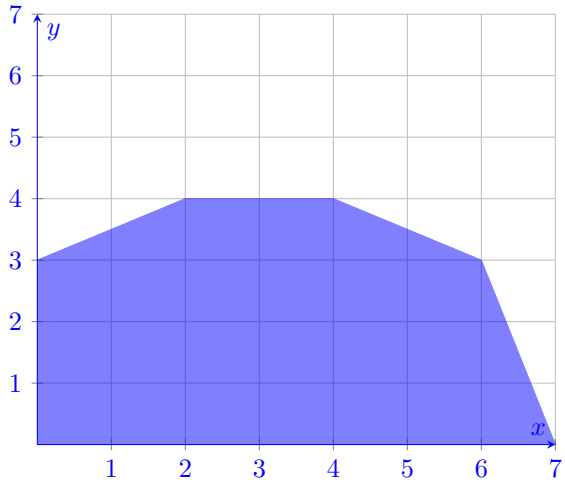
the *only* possible valid path that the simplex algorithm may take?

(c) Over all linear programs in two variables and n constraints (including $x_1, x_2 \geq 0$), determine the minimum and the maximum number of vertices that the simplex algorithm may need to traverse over including its starting and ending points. You may assume that we start already at some vertex on the feasible region, $\vec{x}$.

(d) In the simplex algorithm, we can choose *any* better vertex upon each iteration. What if we make the following modification: choose the *best* adjacent vertex. How does this change the answer to part (c)?

Solution:

(a)



After starting at $(0, 3)$, we next visit $(2, 4)$ then $(4, 4)$ and finally $(6, 3)$ before our algorithm terminates.

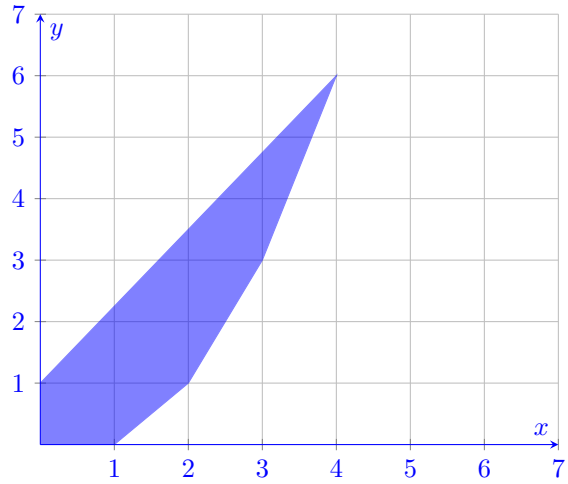
(b) First, it is possible for p to be a valid path! For example, consider $f(x, y, z) = x + y + z$. However, there is no such linear function such that p is the only valid path. Notice that $(0, 0, 0)$ is adjacent to both $(0, 5, 0)$ and $(10, 0, 0)$, but also that $(0, 0, 0)$ is adjacent than $(10, 0, 0)$. So, any time p is a valid path, then $(10, 0, 0)$ is also better than $(0, 0, 0)$, so the path $p' = \{(0, 0, 0), (10, 0, 0), (10, 0, 3)\}$ is also valid.

(c) The minimum number of vertices to iterate on is 1 as we may already be at the maximum. The maximum number of points we may travel over is n. One example uses the constraints in part (a) and an optimization function like $100x + y$. If we start from $(0, 0)$, we may travel to $(0, 3)$, then $(2, 4), (4, 4), (6, 3), (7, 0)$. We cannot traverse a vertex more than once because every iteration, we move to a vertex with strictly larger objective value.

(d) The minimum number of vertices is clearly still 1. As for the maximum, it is now $n - 1$.

We cannot travel over all n vertices. If that were the case, we'd travel to them in a cycle, and end at a vertex that is adjacent to the first vertex. So by our modified algorithm, we would simply travel to the optimal vertex first.

However, it might be the case that $n - 1$ vertices are traversed. Intuitively, this is because the worst vertex may be connected to the best vertex, so the simplex algorithm will have to go the other direction (starting at one of the "next-worst" vertices). For example, consider the following feasible region where we start at $(0, 0)$ and the optimization function is x:



In general, the following linear program would work for all n (a generalization of the example):

$$\begin{aligned}
&\text{Maximize } x \\
&\text{subject to: } (y-0) - 1(x-1) \geq 0 \\
&\qquad\qquad (y-1) - 2(x-2) \geq 0 \\
&\qquad\qquad (y-3) - 3(x-3) \geq 0 \\
&\qquad\qquad \vdots \\
&\qquad\qquad \left(y - \binom{n}{2}\right) - n(x-n) \geq 0 \\
&\qquad\qquad (y-1) + \frac{\binom{n}{2} - 1}{n} x \leq 0 \\
&\qquad\qquad x, y \geq 0
\end{aligned}$$

The simplex algorithm starting at $(0,0)$ would travel to $(1,0)$, then $(2,1)$, and so on, before reaching $\left(n, \binom{n}{2}\right)$.