The Transform and Conquer Algorithm Design Technique

In this part of the course we will look at some simple examples of another general technique for designing algorithms, namely "transform and conquer." Then we will study in depth one very important (for practical uses) such algorithm, the simplex method.

The idea of transform and conquer is simple: when faced with a problem, figure out how to transform it into a different form or different problem, such that a solution to the transformed problem gives a solution to the original problem.

An Example: the Element Uniqueness Problem

Consider this problem: given n integers a_1, a_2, ..., a_n, determine whether any two of them are the same.

The obvious algorithm for this problem is to go through all the numbers, in order, and for each a_k, see if it is the same as any of a_{k+1}, ..., a_n.

This algorithm has efficiency in $\Theta(n^2)$.

We can get a more efficient algorithm by applying the transform and conquer idea: first sort the numbers into, say, increasing order. Then easily go through the sorted list and determine whether any two are the same. The answer to the question for the sorted list is the same as the answer for the original list.

This algorithm has efficiency in $\Theta(n \log n)$, since the pre-sorting takes a number of comparisons in $\Theta(n \log n)$, and the scan to look for adjacent items that are the same is obviously in $\Theta(n)$, which is negligible compared to the work to pre-sort.

⇒ Project 28 [routine] Smallest Difference Problem

Consider this problem: given n real numbers a_1, a_2, ..., a_n, find the pair a_j and a_k for which $|a_j - a_k|$ is the smallest.

→ Design and describe (you don't have to actually program it, but you should write psuedo-code or something from which it would be trivial to implement it) the obvious algorithm for this problem based on the idea of computing the differences for all pairs and finding the smallest.

→ What is the efficiency category for this algorithm?

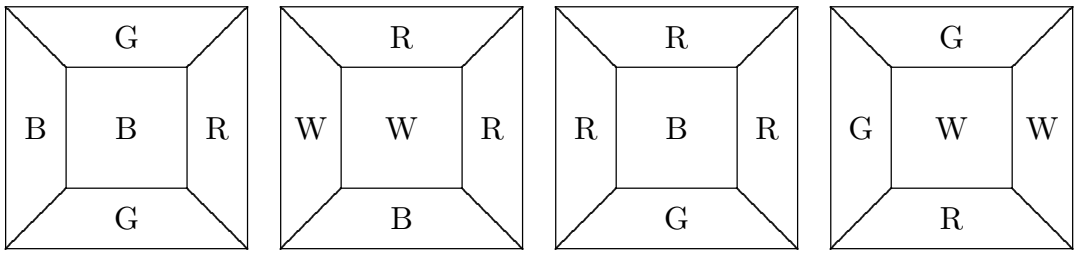
→ Use the transform and conquer idea to design a more efficient algorithm (again, write it in psuedo-code but you don't have to actually program it).

→ What is the efficiency category for this algorithm?

⇒ Project 29 [routine] Instant Insanity

For this Project work through the discussion and do the steps to solve a somewhat challenging puzzle (many of you tried to solve this puzzle—unsuccessfully—during the first class session) by cleverly transforming a puzzle into a graph where the solution to the puzzle is fairly obvious in the graph.

Consider the so-called Instant Insanity puzzle, which consists of these 4 cubes:



(W on bottom) (G on bottom) (W on bottom) (B on bottom)

> There is a file `insanity.pdf` in the `Documents` folder at the course web site that you could print, cut out, and tape together to get your own set of four cubes. For class time, just use one of the sets available.

The goal of the puzzle is to arrange the cubes in a row so that each row of four adjacent squares has each of the four colors appearing in it.

Our purpose for this example is to show how we can transform the puzzle to a different representation and solve a related problem, leading to solution of the original problem.

If we cleverly decide to represent the information using a graph (like with edges and vertices, not like a graph of a function), then we have to figure out what the vertices are and determine the edges. After some deep thought, and lots of playing with the cubes, we decide that we want a vertex for each color, and that whenever cube j has colors X and Y on opposite sides we'll draw an edge, labeled with j, between vertices X and Y.

When we create the graph like this, we obtain:

Now we solve the following transformed problem: find a tour of the graph that uses an edge for each of the cubes. One such tour is

$$R \xrightarrow{1} B \xrightarrow{3} W \xrightarrow{2} G \xrightarrow{4} R.$$

Now, if we arrange the cubes with cube 1 first, with red in front and blue in back, cube 3 with blue in front and white in back, cube 2 with white in front and green in back, and cube 4 with green in front and red in back, we see that we have solved part of the puzzle—the front and back rows have all four colors occurring.

Now we note that we can spin each of the cubes however we want, keeping the front and back fixed, getting whatever other pair of faces we want on the top and bottom rows. To see exactly how to do this, we go back to the graph and look for another tour, this time not using any of the edges that we have already used.

We see that the tour

$$R \xrightarrow{2} B \xrightarrow{1} W \xrightarrow{4} G \xrightarrow{3} R$$

meets these requirements.

If we spin the cubes to make the corresponding faces on the top and bottom—namely put cube 2 with red on top and blue on the bottom, cube 1 with blue on the top and white on the bottom, cube 4 with white on the top and green on the bottom, and cube 3 with green on the top and red on the bottom, then we have fully solved the puzzle.

To check your work I will simply verify that you have solved the puzzle!

This example is intended merely to suggest that changing the representation of a problem can lead to a situation where the solution is more apparent. No one is suggesting that this is not a very clever approach.

Another Example: Evaluating a Polynomial

Here is an apparently simple problem that you have been solving for years:
given a polynomial

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n,$$

compute its value at any desired value of x.

If a typical calculus student were asked to compute, say $P(9)$ for $P(x) = 3+7x-4x^2+5x^3$, they would probably compute 7 times 9, getting 63 and add that to 3, giving 66. Then they might square 9 to get 81, multiply by 4, and subtract that value from 66, and so on.

To analyze the efficiency of this algorithm, we note that the constant term requires 0 multiplications, the x^1 term requires 1, the x^2 requires 2, and so on, up to the x^n term requiring n, for a total of $0 + 1 + 2 + \cdots + n = n(n+1)/2 \in \Theta(n^2)$ multiplications. This algorithm is also irritating because you need to somehow store the products long enough to add them up.

If we apply the transform and conquer technique (this is all a little contrived—it's unlikely that Horner set out to design his algorithm by thinking "hmmm, can I transform a given polynomial into an equivalent form that is more efficient to evaluate?"), we might think to write the previous polynomial as

$$P(x) = (((5x - 4)x + 7)x + 3.$$

Then computing, say, $P(9)$ can be done with $\Theta(n)$ multiplications and no extra storage, by just pressing the keys

```
5 * 9 = - 4 = * 9 = + 7 = * 9 = + 3 =
```

A Super Famous Example: Solving a System of Linear Equations

Consider the familiar problem of solving a system of linear equations such as

$$\begin{aligned} 2x + 3y - z &= 5 \\ 4x + 7y - 5z &= 3 \\ -6x + 11y - 4z &= 4 \end{aligned}$$

You have actually taken an entire math course mostly about solving this problem (MTH 2140 or maybe MTH 3140). The idea, as we hope you recall, at least vaguely, is to first transform the problem by writing it as a matrix:

$$\begin{bmatrix} 2 & 3 & -1 & 5 \\ 4 & 7 & -5 & 3 \\ -6 & 11 & -4 & 4 \end{bmatrix}$$

(with the understanding that the columns correspond to the variables x, y, z, and the right-hand side).

Then we perform *row operations* on the matrix, namely multiplying a row by a non-zero number, adding a multiple of one row to another, and swapping two rows, until the matrix is transformed to a form where the solution can easily be obtained.

⇒ The instructor will perform this algorithm using a little program, available at the course web site in the folder `Code/ManualSimplex`, which conveniently performs row operations under interactive control.

Soon you will be using this program to perform the *simplex method* to solve *linear programming problems* (LP).

The Linear Programming Problem and the Simplex Method

Now we want to study a problem, known as "linear programming," and one algorithm that solves this problem, known as the "simplex method."

A lot of real world problems, from computer science, mathematics, various sciences, engineering, business, and economics, can be formulated as linear programming problems. In addition to its practical uses, our work in this section will give us an important case study in computational complexity in the last part of the course, when we study computational complexity and "P vs. NP."

Software packages that implement the simplex method have been well-developed for half a century, and are still in common use. When I taught MTH 3250 in the early 1980's, the text we used said that half the computer time being used on the planet was being devoted to performing the simplex method. Lists of the most important algorithms typically include the simplex method.

We have mostly been avoiding *numerical algorithms*, since they are covered in depth in MTH 4480–4490. Technically the simplex method is a numerical algorithm, subject to rounding error issues, but if it were executed with perfectly accurate arithmetic computations, it would behave like a non-numerical algorithm. And, it uses simple mathematical ideas in a profoundly powerful way. Finally, we will later see how to solve TSP by using the simplex method as a tool.

The Linear Programming Problem (LP)

The linear programming problem, known henceforth as LP, is to find the optimal value of a function, subject to certain constraints on the arguments, where the function and the constraints are linear, namely of the form of a sum of terms consisting of a scalar times a variable, such as $3x_1 + 2x_2$.

An instance of LP has the standard form

$$\begin{aligned} \max \quad & c^T x \quad \text{subject to} \\ & Ax = b \\ & x \geq 0 \end{aligned}$$

---

where standard matrix notation is used, n is the number of *decision variables*, m is the number of *equality constraints*, x and c are n by 1 matrices, A is m by n, b is m by 1, and for any matrix M, $M \geq 0$ means that all the components of M are non-negative. We also assume that $b \geq 0$.

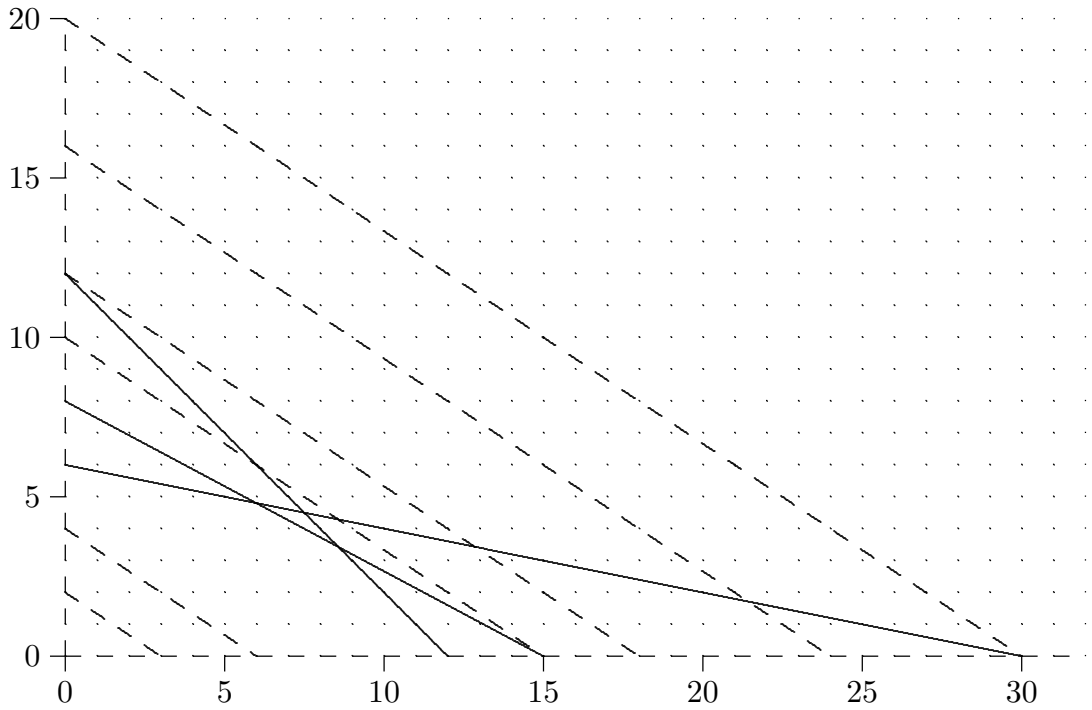
Later we will see how all sorts of problems that don't have the standard form can be transformed to an equivalent problem that fits this form. So, all our theory, and the algorithm we present for solving an LP, will be based on this form.

First Example of an LP Suppose a farmer has 1200 acres of farm land to be planted in either corn or wheat. Suppose further that the farmer has 12000 units of water to be devoted to these crops, and 18000 monetary units with which to buy seed for the crops. Each acre of corn requires 8 units of water for the season, and each acre of wheat requires 15. Seed for an acre of corn costs 6 monetary units, while an acre of wheat costs on 30 units per acre. If the expected produce from an acre of corn will sell for 20 monetary units and an acre of wheat will produce 30 monetary units, how many acres of corn and wheat should the farmer decide to plant?

To solve this word problem, we begin, like with many such problems, by identifying the variables whose values we need to determine. In the world of LP, these are known as *decision variables*. So, let x be the number of acres of corn to plant, and y be the number of acres of wheat to plant. With these variables, we easily translate the facts of the problem into precise mathematical quantities.

$$\begin{aligned} \max \quad & 20x + 30y \quad \text{subject to} \\ & x + y \leq 1200 \\ & 8x + 15y \leq 12000 \\ & 6x + 30y \leq 18000 \\ & x, y \geq 0 \end{aligned}$$

We can graph the boundary lines for the constraints, and some level sets for the objective function (sets on which all points have the same function value) for this problem:



(in units of 100)

⇒ We can solve this instance of LP by finding the feasible region—the region consisting of points (x, y) that satisfy all the constraints—and then finding the level set which crosses the feasible set and has the largest objective function value. We can do this graphically, or by solving pairs of linear equations to find candidates for the winning point.

Some Theory of LP and the Brute Force Approach

Consider an instance of LP, where the coefficient matrix A is m rows by n columns. To avoid bogging down in a lot of math, no matter how much fun that might be, we will simply state the crucial theoretical facts that we need to know in order to understand our two approaches to solving LP (brute force and the simplex method).

Big Fact: an optimal point can be obtained by selecting m of the n variables to be allowed to be positive, while holding the other $n - m$ variables at 0.

> The m variables that are allowed to be positive are known as *basic* variables, and the other $n - m$ are known, in a stroke of creative nomenclature, as *non-basic* variables.
>
> It is possible that there are no points that satisfy both the constraints and the restrictions. Such an instance of LP is said to be *infeasible*. The set of points that satisfies both $Ax = b$ and $x \geq 0$ is known as the *feasible set*.

The rest of what we need to know is based on a standard principle of matrix algebra: if we partition a matrix computation in any way that makes sense, then any fact that would be true if the blocks of matrices were numbers is still true.

Using this principle, let's partition x into m basic variables and $n-m$ non-basic variables, and purely for notational convenience pretend that the variables have been rearranged so that the basic variables come first, followed by the non-basics. Then we can partition $x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}$. We will similarly rearrange the columns of A, denoting the first m columns of A by B, and the remaining $n - m$ columns by N, so we have partitioned $A = [B\ N]$. And, we rearrange the components of c so that $c = \begin{bmatrix} c_B \\ c_N \end{bmatrix}$.

We also want to express the objective function in the same form as the constraints. To do this, we introduce a new variable z, and define $z = c^T x$. To put this in the same form as the constraints, we need a constant on the right-hand side, so we subtract $c^T x$ from both sides, obtaining $z - c^T x = 0$.

With all this notation, our LP becomes:

	z	x_B^T	x_N^T	
z	1	$-c_B^T$	$-c_N^T$	0
x_B	0 0 $\vdots$ 0	B	N	b

Note that the top row in this tableau consists of variable names, for convenience, showing us which column corresponds to which variables. Similarly, the leftmost column consists of the names, it will turn out, of the current basic variables.

The column for the z variable never changes (and in fact z doesn't appear in any equation except the objective function one), and is only there to remind us of what the objection function row means, namely

$$1z - c_B^T x_b - c_N^T x_N = 0.$$

Now we are ready to do some block row operations. We first multiply the "second row" of the partitioned matrix by B^{-1} (which of course only works if this matrix is non-singular), obtaining the equivalent set of equations

	z	x_B^T	x_N^T	
z	1	$-c_B^T$	$-c_N^T$	0
x_B	0 0 $\vdots$ 0	I	$B^{-1}N$	$B^{-1}b$

(noting that B^{-1} times the first block, which is a column of all 0's, gives a column of all 0's)

Then we add the correct multiple, namely c_B^T, of "row 2" to "row 1" to zero out "row 2" in "column 1," obtaining the tableau

	z	x_B^T	x_N^T	
z	1	0^T	$c_B^T B^{-1} N - c_N^T$	$c_B^T B^{-1} b$
x_B	$\begin{matrix} 0 \\ 0 \\ \vdots \\ 0 \end{matrix}$	I	$B^{-1}N$	$B^{-1}b$

Now, the "second row" says that

$$Ix_B + B^{-1}Nx_n = B^{-1}b,$$

so if we set $x_N = 0$, these equations say that $x_B = B^{-1}b$.

The "first row" says that

$$z + 0^T x_B + (c_B^T B^{-1} N - c_N^T)x_N = c_B^T B^{-1} b,$$

or, using $x_N = 0$, that

$$z = c_B^T B^{-1} b.$$

Thus, the first row of the rightmost column of the tableau gives the value of the objective function corresponding to the current choice of basic variables, and the elements below that in the rightmost column give the values of the basic variables.

Note: The document `lpsimplex.pdf` contains a bunch of details on linear programming and the simplex method, in case you want to read more in-depth on these topics. The following extended example, accompanied by some mini-lecturing, should tell you everything you need to know.

One Great Example of Everything

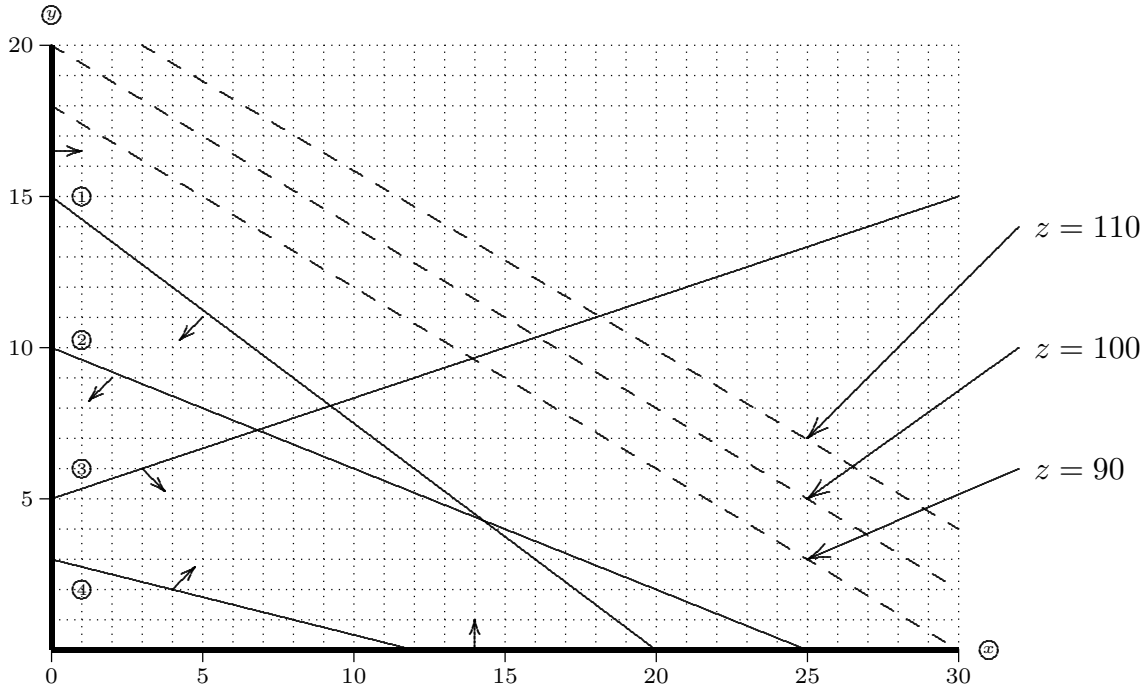
We will take a very pragmatic approach to learning the simplex method and just explain everything through the following example of pretty much everything, with embedded general discussion.

This material should be enough to enable you to succesffully complete the upcoming Projects, and later to understand how we use LP as a tool to solve the Euclidean traveling salesperson problem.

Consider this original problem:

$$\begin{aligned} \max \quad & 3x_1 + 5x_2 \quad \text{subject to} \\ & 3x_1 + 4x_2 \leq 60 \\ & 2x_1 + 5x_2 \leq 50 \\ & -x_1 + 3x_2 \leq 15 \\ & x_1 + 4x_2 \geq 12 \\ & x \geq 0 \end{aligned}$$

Because there are only two decision variables, we can picture this problem, where the arrows on the lines show which half-plane satisfies the corresponding inequality:



This problem is *not* an instance of LP, because it has inequality constraints (other than the restrictions). But, the fact that LP maintains nonnegativeness of all the variables allows us to convert this problem to an instance of LP by introducing a new variable, say s_1 ("s" for "slack") that is the amount by which $3x_1 + 4x_2$ is less than 60. So, we convert the first equation to $3x_1 + 4x_2 + s_1 = 60$, which is in the desired form for LP.

Formally, we note that

$$3x_1 + 4x_2 + s_1 = 60 \text{ and } s_1 \geq 0$$

if and only if

$$3x_1 + 4x_2 \leq 60.$$

We can convert the next two inequalities to equalities in the same way. For the last inequality we similarly add a new variable s_4 ("s" for "surplus") and note that

$$x_1 + 4x_2 \geq 12$$

if and only if

$$x_1 + 4x_2 - s_4 = 12 \text{ and } s_4 \geq 0.$$

Another little technique is to introduce a new variable z that is intended to be the value of the objective function. To do this, we add the equation

$$z = 3x_1 + 5x_2,$$

but in order to make this look like the other constraints, we write it in the obviously equivalent form

$$z - 3x_1 - 5x_2 = 0.$$

In the figure we have drawn three sample lines showing the set of all points (x_1, x_2) such that $z = 90$, 100, and 110. There is such a line for each possible value of z. We can visually solve the LP by imagining which of these parallel lines hits a point in the feasible set with z as large as possible.

These techniques give us this instance of LP:

$$\begin{aligned} \max \quad & z \quad \text{subject to} \\ & z - 3x_1 - 5x_2 = 0 \\ & 3x_1 + 4x_2 + s_1 = 60 \\ & 2x_1 + 5x_2 + s_2 = 50 \\ & -x_1 + 3x_2 + s_3 = 15 \\ & x_1 + 4x_2 - s_4 = 12 \\ & x \geq 0 \\ & s \geq 0 \end{aligned}$$

Note that really, to fit our definition of LP, we should have named s_1 as x_3, s_2 as x_4, and so on. We don't, because it is helpful to understand that the x_j variables are original "decision variables," while the s_k variables are slack or surplus variables added to the original problem to convert it to an LP.

> This example, though allegedly great, is somewhat misleading, because after this example we will never try to visualize what is going on again! The picture is merely intended to help us believe the theoretical claims we will make later (without, perhaps, carefully reading their proofs).

Important Theoretical Fact

It turns out that if we have an LP with A having m rows and n columns, then the optimal value can always be obtained by selecting the correct set of m variables to be *basic*, setting the other $n - m$ variables (known, perhaps not surprisingly, as "non-basic variables) to 0, and solving the system of equations by pivoting on the basic variable columns and noting that the values of the basic variables show up in the right-hand side.

Interpreting Rows of the Tableau

"Tableau" is the fancy name for the matrix we're doing row operations on.

Suppose we have a tableau like this (which we actually will later):

	z	x_1	x_2	s_1	s_2	s_3	s_4	rhs
z	1	−4.67	0	0	0	1.67	0	25
x_2	0	−0.33	1	0	0	0.33	0	5
s_2	0	3.67	0	0	1	−1.67	0	25
s_1	0	4.33	0	1	0	−1.33	0	40
s_4	0	−2.33	0	0	0	1.33	1	8

Here z is special and appears where it always does. The basic variables are x_2, s_2, s_1, and s_4, so the non-basic variables are x_1 and s_3. Thus, to understand what each row is saying, we need to remember that $x_1 = 0$ and $s_3 = 0$.

Then the first row—the so-called objective function row—says

$$z - 4.67x_1 + 1.67s_3 = 25,$$

but since x_1 and s_3 are 0, this row really just says $z = 25$. Thus, the current value of the objective function—z—is always written in the upper-right corner of the tableau.

The second row, which has x_2 as its basic variable, similarly says that

$$-0.33x_1 + x_2 + 0.33s_3 = 5,$$

or $x_2 = 5$, since x_1 and s_3 are non-basic.

Thus, the right-hand side gives the current values of all the basic variables.

The Brute Force Method

Now let's solve this LP by using `ManualSimplex` and the previous fact.

First, look at the data file named `great` to see how the software expects an instance to be input.

Next, go through all possible choices of basic variables (there are 6 variables—ignoring z which is special—giving us the $\binom{6}{4} = 15$ possibilities listed in the chart below). For each choice, note the resulting values of x_1 and x_2, note which original inequalities are satisfied exactly, whether or not the resulting point satisfies the restrictions (known as "feasible"), and if so, what is the objective function value. Then solve the LP by noting the choice of basic variables produces the biggest z value among all the choices that give all nonnegative variable values.

Basic Variables	(x_1, x_2)	On lines	Feasible?	z (if feasible)
x_1, x_2, s_1, s_2	$(-3.43, 3.86)$	3,4	no	
x_1, x_2, s_1, s_3	(,)			
x_1, x_2, s_1, s_4	(,)			
x_1, x_2, s_2, s_3	(,)			
x_1, x_2, s_2, s_4	(,)			
x_1, x_2, s_3, s_4	(,)			
x_1, s_1, s_2, s_3	(,)			
x_1, s_1, s_2, s_4	(,)			
x_1, s_1, s_3, s_4	(,)			
x_1, s_2, s_3, s_4	(,)			
x_2, s_1, s_2, s_3	(,)			
x_2, s_1, s_2, s_4	(,)			
x_2, s_2, s_3, s_4	(,)			
s_1, s_2, s_3, s_4	(,)			

The Simplex Method

We are now ready, in our example, to explain the simplex method. It is simply an improvement of the brute force method obtained by thinking about what the objective function row means.

Consider again our earlier sample tableau:

	z	x_1	x_2	s_1	s_2	s_3	s_4	rhs
z	1	−4.67	0	0	0	1.67	0	25
x_2	0	−0.33	1	0	0	0.33	0	5
s_2	0	3.67	0	0	1	−1.67	0	25
s_1	0	4.33	0	1	0	−1.33	0	40
s_4	0	−2.33	0	0	0	1.33	1	8

We use a very simple idea: what would happen if we allowed a non-basic variable to increase from 0? For example, suppose we let x_1 increase to some positive number α. Then the first row would say

$$z - 4.67\alpha = 25.$$

or

$$z = 25 + 4.67\alpha.$$

Thus, we would make z bigger by 4.67 units for every unit that we increase x_1.

But, as we increase x_1, how would the basic variables change (the other non-basics, in this case s_3, just stay at 0)?

Row 2 says

$$-0.33\alpha + x_2 = 5,$$

or

$$x_2 = 5 + 0.33\alpha.$$

This is great, because as we increase α, making z bigger, x_2 also gets bigger, continuing to satisfy the restriction that $x_2 \geq 0$. Note that x_2 doesn't appear in any other rows, so changes to x_2 don't affect the other equations.

Row 3, on the other hand, says that

$$3.67\alpha + s_2 = 25,$$

or

$$s_2 = 25 - 3.67\alpha.$$

Thus, as x_1 increases, s_2 decreases. In fact, at $\alpha = \frac{25}{3.67}$, s_2 hits 0. And, if x_1 were to go above $\frac{25}{3.67}$, s_2 would go negative, violating the restriction $s_2 \geq 0$.

Similarly, row 3 says that x_1 can't increase beyond $\frac{40}{4.33}$, or s_1 would go negative. And, as x_1 increases, s_4 also increases.

We have derived most of the simplex method. First, we look in the objective function row for a non-basic variable that has a negative coefficient. Then in the column for that non-basic variable we compute the minimum ratio of a right-hand side value in a row divided by the coefficient in the non-basic column—but we only do this for positive coefficients (negative or zero coefficients represent basic variables that get more positive). Having thus identified a column and then a row in the tableau, we pivot in the position, letting the formerly non-basic variable become the basic variable for the row, and forcing the previously basic variable for the row to hit 0 and become non-basic.

This step is repeated until there are no more non-basic variables with a negative coefficient in the objective function row.

Note that when doing this process, we can reach a place where a non-basic variable has no positive numbers in its column. This situation allows us to increase that non-basic variable to infinity, while keeping the constraints and restrictions satisfied. In this situation we say that the problem is *unbounded*.

A First Example of the Simplex Method

Using `ManualSimplex`, let's perform the simplex method on the example tableau, starting with x_1, s_1, s_2, and s_3 as the basic variables.

Getting an Initial Feasible Basis

This is all great, but the simplex method needs to start with a choice of basic variables that is feasible, so it can iteratively improve, but it turns out to be just as hard to find an initial feasible basis as it is to solve the problem in general.

One good strategy for getting an initial feasible basis is to add an *artificial variable* to each row that does not have a slack variable already providing a nice choice for a basic variable, and to temporarily use the objective function "minimize the sum of the artificial variables". This is known as "Phase 1." If the original constraints have a feasible point, all the artificial variables can be reduced to 0, giving an initial feasible basis without any artificial variables. Then we delete the artificial variable columns, change the objective function to be the actual objective function, and solve the original problem. This is known as "Phase 2."

At the start of both Phases we must pivot on all the basic variable spots to ensure that the objective function coefficients are correct. This is known as "pricing out."

Note that Phase 1 requires *minimizing* z instead of maximizing. To do this we simply multiply the objective function by -1 and maximize it, recognizing that the resulting objective function value will be the opposite of the optimal value of the original objective function.

> This technique is useful any time we want to minimize rather than maximizing.

Let's use the Phase1-Phase 2 technique to solve our example, starting with just slack variables and artificial variables as the basic variables at the beginning.

⇒ Project 30 [routine] Brute Force Method for LP

Demonstrate the brute force method, using the `ManualSimplex` application, on this LP (you will need to use a text editor to make a suitable input file for this problem, which is already in the required format for LP):

$$\begin{aligned} \max \quad & x_1 + 2x_2 + 3x_3 + 3x_4 + 2x_5 + x_6 \quad \text{subject to} \\ & 4x_1 + 8x_2 + 3x_3 + 6x_4 + 10x_5 - x_6 = 120 \\ & 8x_1 - 4x_2 - 6x_3 - 8x_4 + x_5 + 3x_6 = 24 \\ & 12x_1 + 5x_2 - 9x_3 + 6x_4 - 9x_5 + 8x_6 = 360 \\ & x \geq 0 \end{aligned}$$

List all different ways (order doesn't matter) to choose m basic variables, and for each, use the program to do the row operations, and mark as infeasible, or if it is feasible, write the objective function value. When all have been done, note the optimal choice, and state clearly the values of all the variables and the objection function value for this optimal choice.

⇒ Project 31 [routine] Solving Some LP Instances

For each of the 3 problems below, produce a careful mathematical description of the LP, ready to be put into a data file for use with `ManualSimplex`. Then use the application, noting the values of the basic variables and the objective function at the optimal tableau for both Phase 1 and Phase 2, as appropriate, and describe the final solution to each problem as infeasible, unbounded, or having an optimal feasible point.

a. Formulate and solve this LP:

$$\max x_1 + 2x_2 + 3x_3$$

subject to the constraints

$$-3x_1 + 15x_2 - 3x_3 \geq 3$$

$$6x_1 + 3x_2 + 6x_3 \leq 60$$

$$-6x_1 + 6x_2 + 3x_3 \leq 21$$

$$9x_1 + 5x_2 - x_3 \geq 21$$

$$-3x_1 + 5x_2 + 2x_3 \geq 3$$

$$6x_1 + 8x_2 - 4x_3 \leq 30$$

$$8x_2 - 4x_3 \leq 12$$

$$3x_1 + 3x_3 \geq 12$$

$$x_1, x_2, x_3 \geq 0$$

b. Formulate and solve the above LP, but with this constraint added:

$$2x_3 \leq 1$$

c. Formulate and solve the LP for part a, but with the first four constraints removed.

⇒ Project 32 [optional] Solving Continuous Collision Detection by LP

In this project you will explore using LP to solve a problem that comes up in game programming. Also, you'll get to practice performing the simplex method on a number of LP instances where you can verify the correctness of the results.

Suppose you have two polygonal objects in a 2D game, as follows. The first object has its center at the point $c = \begin{bmatrix} c_x \\ c_y \end{bmatrix}$, and has vectors from c to its three vertices (could use any number) a_1, a_2, and a_3. Then it turns out that every point in the first object can be expressed as

$$c + \mu_1 a_1 + \mu_2 a_2 + \mu_3 a_3,$$

where $\mu 1 + \mu 2 + \mu 3 = 1$ and all the μ_j values are nonnegative (this is a standard convex combination idea). In addition, let's say that this object is moving along a straight line with velocity vector $v = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$, so that at any time λ (with $\lambda \geq 0$, of course) the center is at $c + \lambda v$.

The second object is modeled similarly, with d in place of c, ν_k in place of μ_j, and w in place of v.

Then we can model the continuous collision detection problem (CCD) by saying that we want to find the earliest time λ at which a point in the moving first body touches a point in the moving second body. This problem is important to maintain physical realism in a game, since it lets us keep objects from overlapping (assuming they aren't already overlapping and are moving toward each other).

> A cheap alternative to continuous collision detection is to simply let everything move, check for *overlaps*, and somehow deal with them. This is common, but doesn't actually work in the sense that if objects are moving too fast for the thickness of other objects, they can move past each other in one time step so they aren't overlapping. This is called "tunneling."

In mathematical language, we want to find the minimum λ such that

$$c + \lambda v + \sum \mu_j a_j = d + \lambda w + \sum \nu_k b_k$$

with $\sum \mu_j = 1$, $\sum \nu_k = 1$, and all the μ_j's and ν_k's nonnegative.

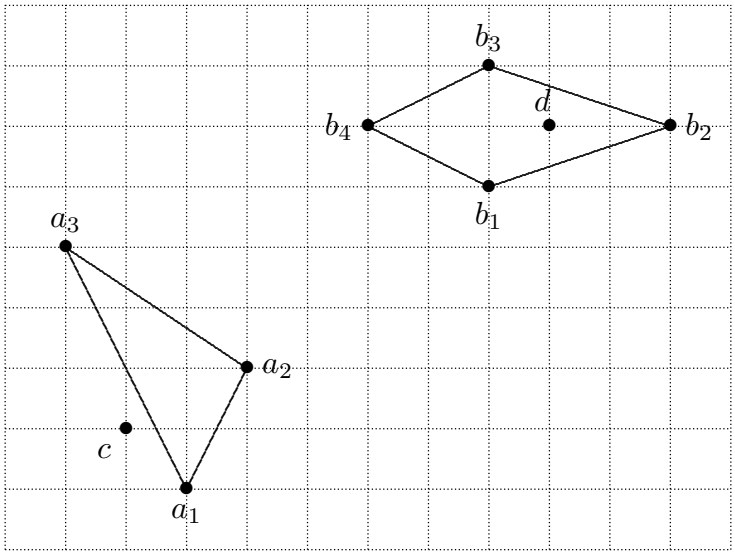
With some thought we realize that this is a linear programming problem, where the decision variables are λ, the μ_j's, and the ν_k's, all of which are restricted to be nonnegative, with four equality constraints. So, one way to do CCD would be to formulate and solve an LP instance.

Your job on this project is to write a small, somewhat trivial program that will ask the user to enter c, v, d, and w (as 8 `double` values—you might want to use `cx`, `cy`, etc. to hold these values), and will then produce an output file that contains data ready to be input into `ManualSimplex` to solve the Phase 1 LP corresponding to the CCD problem (once a feasible choice of basic variables is found in Phase 1, it's easy to put in the Phase 2 objective function, because it is just "minimize λ"), where for simplicity the vertex vectors are fixed.

In your program hard-code the vertex vectors as $a_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, $a_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $a_3 = \begin{bmatrix} -1 \\ 3 \end{bmatrix}$, $b_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$, $b_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, $b_3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$, and $b_4 = \begin{bmatrix} -3 \\ 0 \end{bmatrix}$.

> We could have these as inputs, but we won't because it doesn't add much conceptually, and hard-coding them keeps things a little simpler.

We want c and d to be input by the user, but for the first 4 problems we will use $c = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ and $d = \begin{bmatrix} 10 \\ 9 \end{bmatrix}$, giving this picture of what is going on:



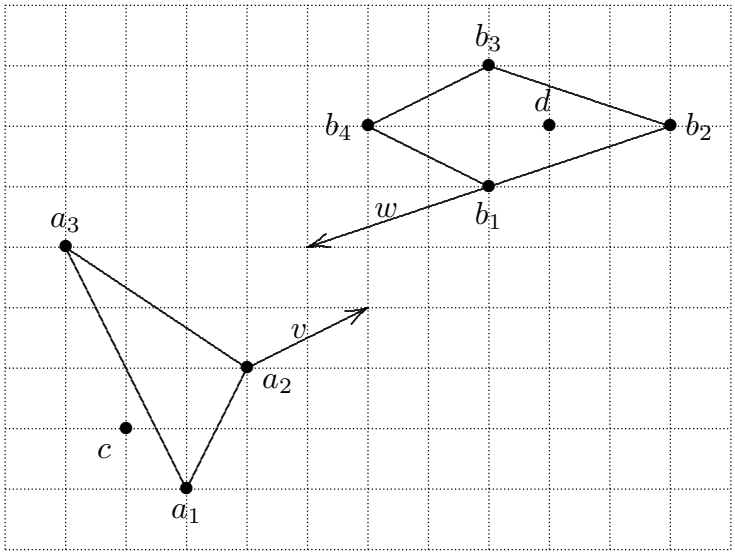
Note that the a_j's and b_k's are shown here as points, but they are actually vectors from c or d leading to the indicated points.

Be sure that your program generates a tableau with the right-hand side values all nonnegative.

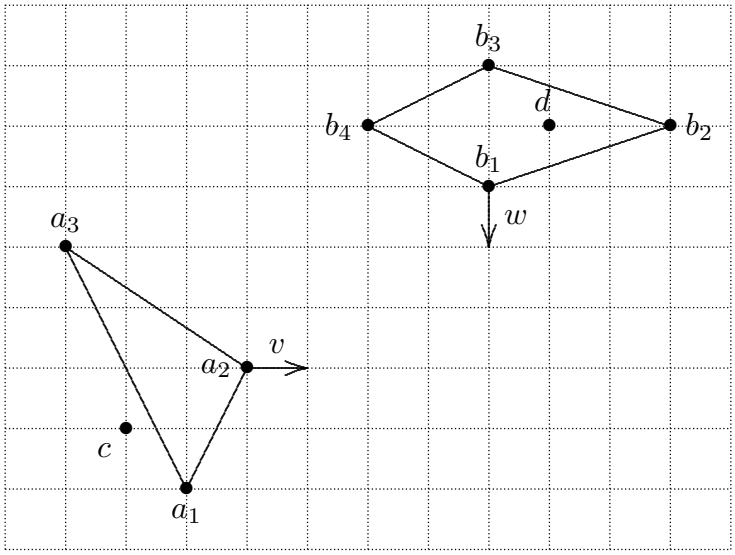
After you have written the little program, test it on the cases given below. For each test case, use your program to generate a tableau, use `ManualSimplex` to solve the LP instance, and interpret the results (the diagrams showing the velocity vectors should be helpful as you try to picture how the objects are moving). Note that the optimal value of λ tells the time at which the two objects first touch, and the values of the μ_j's and ν_k's tell precisely what points on the two objects are touching at that time.

When you do `ManualSimplex` on some of these problems a rounding error issue will arise. It is common when doing row operations to have values that should be exactly 0 be some tiny number, and those tiny numbers should be treated as 0, and even changed to 0 when possible (the software won't let you change the upper right corner value). It is crucial that you don't treat a tiny negative value in the objective function row as an indication that the non-basic variable for that column should become basic.

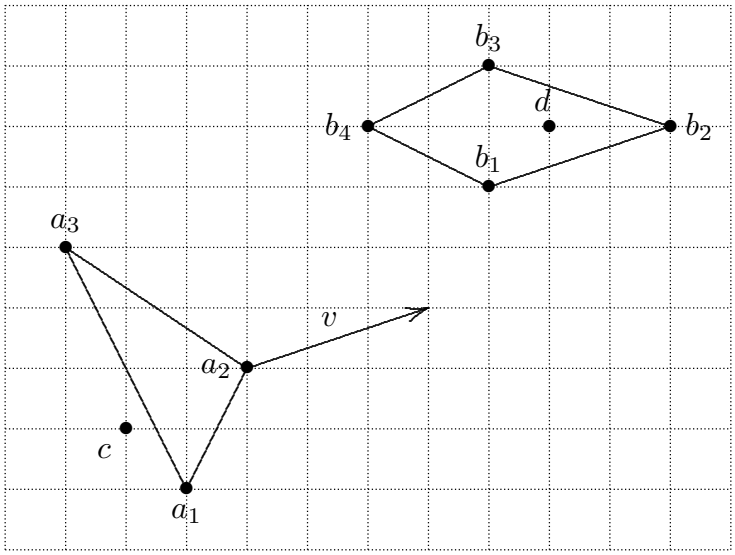
a. $c = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$, $v = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $d = \begin{bmatrix} 10 \\ 9 \end{bmatrix}$, $w = \begin{bmatrix} -3 \\ -1 \end{bmatrix}$:



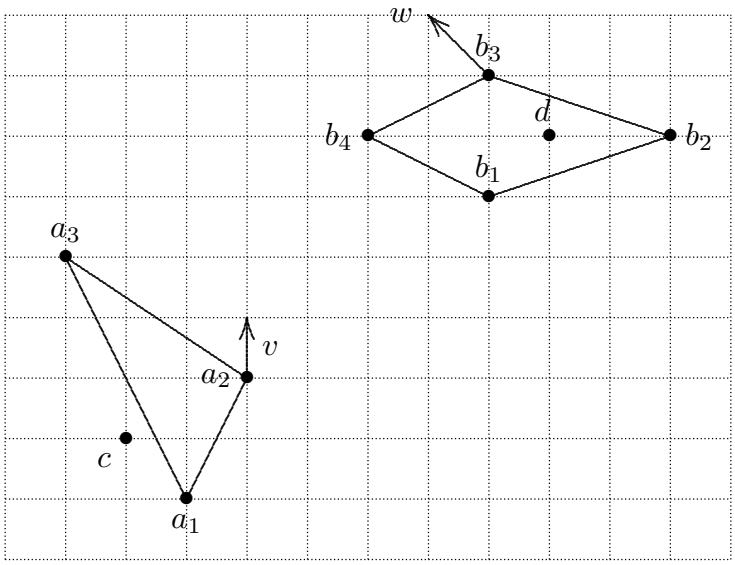
b. $c = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$, $v = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $d = \begin{bmatrix} 10 \\ 9 \end{bmatrix}$, $w = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$:



c. $c = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$, $v = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$, $d = \begin{bmatrix} 10 \\ 9 \end{bmatrix}$, $w = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$:

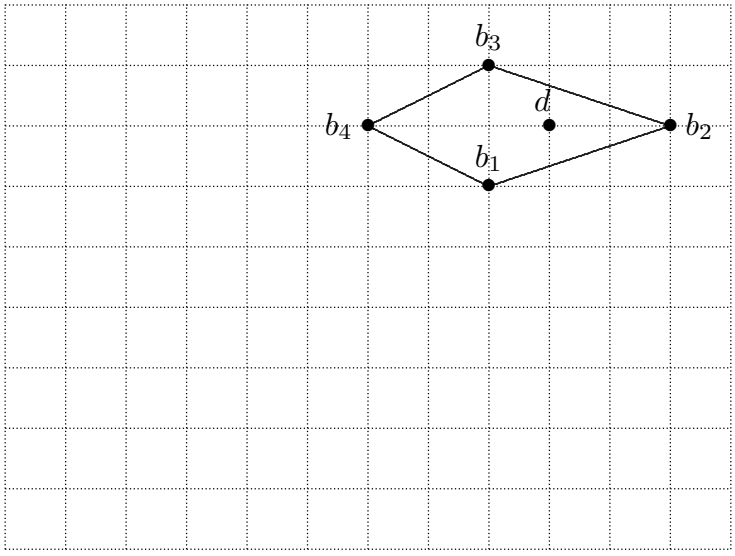


d. $c = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$, $v = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $d = \begin{bmatrix} 10 \\ 9 \end{bmatrix}$, $w = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$:



e. $c = \begin{bmatrix} 7 \\ 8 \end{bmatrix}$, $v = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $d = \begin{bmatrix} 10 \\ 9 \end{bmatrix}$, $w = \begin{bmatrix} -3 \\ -1 \end{bmatrix}$

(note that the first object has been moved from where it was in the first four problems—you'll need to sketch it in yourself):



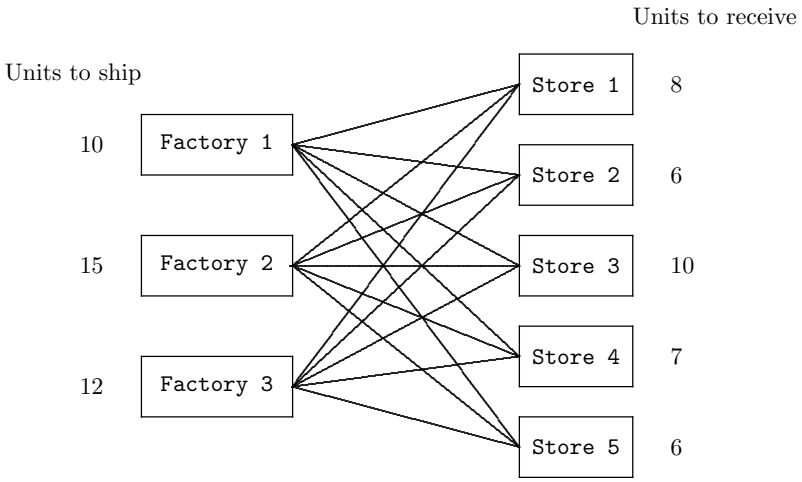
Turn in hand-written summaries of these 5 test cases showing your final results. Be sure to check geometrically that your results are correct.

You do not need to submit your program.

⇒ Project 33 [routine] The Transportation Problem

Before using LP to tackle TSP, let's look at a problem, known as the transportation problem, that can be solved by modeling it as an LP.

Suppose we have some factories, each of which produces some number of units of some product, and some stores, each of which wants to receive some number of units of the product, where the number of units produced is equal to the number of units desired (technically we are doing the *balanced* transportation problem, where the supply equals the demand).



Further, suppose there is a per-unit cost to ship a unit from any factory to any store. The "transportation problem" is to decide how many units to ship from each factory to each store in order to minimize the total shipping cost.

Because it is too cumbersome to write the unit shipping costs along each edge from a factory to a store, we typically display all those costs in a 2D chart, where the rows are the factories and the columns are the stores, along with the number of units to be shipped by each factory and the number of units to be received at each store:

	1	2	3	4	5	
1	3	7	11	4	2	10
2	5	9	4	2	8	15
3	6	1	9	4	7	12
	8	6	10	7	6	

(where the number in row j, column k is the cost of shipping one unit from factory j to store k)

To model this problem as an LP, we take as the decision variables

$$x_{jk} = \text{the number of units to be sent from factory } j \text{ to store } k.$$

Then we can simply write down all the facts of the problem using these variables, and realize that this problem is actually an LP.

Here is the mathematical form of this LP:

$$\begin{aligned}
\min \quad & 3x_{11} + 7x_{12} + 11x_{13} + 4x_{14} + 2x_{15} + \\
& 5x_{21} + 9x_{22} + 4x_{23} + 2x_{24} + 8x_{25} + \\
& 6x_{31} + 1x_{32} + 9x_{33} + 4x_{34} + 7x_{35} \quad \text{subject to} \\
& x_{11} + x_{12} + x_{13} + x_{14} + x_{15} = 10 \\
& x_{21} + x_{22} + x_{23} + x_{24} + x_{25} = 15 \\
& x_{31} + x_{32} + x_{33} + x_{34} + x_{35} = 12 \\
& x_{11} + x_{21} + x_{31} = 8 \\
& x_{12} + x_{22} + x_{32} = 6 \\
& x_{13} + x_{23} + x_{33} = 10 \\
& x_{14} + x_{24} + x_{34} = 7 \\
& x_{15} + x_{25} + x_{35} = 6 \\
& x \geq 0
\end{aligned}$$

The first three constraints say that the total units sent from each factory has to add up to the number of units available at the factory, while the last five constraints say that the total number of units arriving at each store has to be as desired.

→ Get the data file `project33` in the `Code/ManualSimplex` folder at the course web site and perform the simplex method to solve this transportation problem. Note that artificial variables have been added to give a feasible initial basis, so you will need to do Phase 1, then edit the objective function row to put in the actual objective function, and finally do Phase 2.

Record in the chart below the values of all the decision variables in the optimal solution, where the value of x_{jk} can conveniently be written in row j, column k. Just leave the non-basic ones blank to signify non-basic and 0. You can partially check your solution by multiplying all the non-zero values by the objective function coefficients for that value to verify that they add up to the value produced by the simplex method, and by summing each row and column to check that the constraints are satisfied.

	1	2	3	4	5	
1	3	7	11	4	2	10
2	5	9	4	2	8	15
3	6	1	9	4	7	12
	8	6	10	7	6	

Note that the eight constraints are redundant, because the total units produced at the factories equals the total units wanted at the stores, so if the first seven constraints are satisfied, the eighth one has to be. So, before creating the data file, we throw out the last constraint. If we don't do this, at the end of Phase 1 the last row says $a_8 = 0$, and has no non-zeros in the non-artificial columns, so it basically says nothing and just clutters up the tableau. If you don't understand this, you can do the simplex method on `rex32redun` to see it for yourself.

⇒ Project 34 [routine] Another Transportation Problem Instance

Formulate and solve the LP produced by the transportation problem with this data:

	1	2	3	4	5	6	
1	12	17	13	19	20	15	24
2	10	8	12	14	13	6	18
3	19	15	21	11	14	20	22
4	17	14	17	10	16	18	16
	13	12	10	14	15	16	

Put your optimal point in this chart and check that the objective function value matches the direct calculation and that the rows and columns add up to the targets.

	1	2	3	4	5	6	
1	12	17	13	19	20	15	24
2	10	8	12	14	13	6	18
3	19	15	21	11	14	20	22
4	17	14	17	10	16	18	16
	13	12	10	14	15	16	
