Chapter 7

Matrices

The word "matrix" has a variety of definitions.[1] Not surprisingly, we are interested in a definition used in mathematics:

Definition 36: Matrix *matrix*

A *matrix* is an n-dimensional collection of values, $n \in \mathbb{Z}^+$.

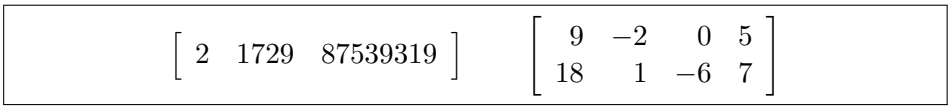
Figure 7.1 shows examples of small one–dimensional and two–dimensional matrices. We stop at two because comprehensible two–dimensional representations of three–dimensional matrices are hard to create. Happily, for this book, one– and two–dimensional matrices are all that we will need.

7.1 The Utility of Matrices

The ACM/IEEE Computer Society produces curriculum guidelines for Computer Science education, to help secondary and higher-education institutions ensure that they are graduating students that possess a common core of basic knowledge of the discipline. The section on Discrete Structures in the 2013 guidelines[2] does not mention the words 'matrix' or 'matrices' at all. So why, then, does this book devote a chapter to the topic?

[1] Such as: The area that produces the cells that become a fingernail is called the *matrix*. Makes you think of the Matrix movie franchise in a new way, doesn't it?

[2] ACM/IEEE-CS Joint Task Force on Computing Curricula, ACM Press and IEEE Computer Society Press, December 2013. DOI: http://dx.doi.org/10.1145/2534860



$$\begin{bmatrix} 2 & 1729 & 87539319 \end{bmatrix} \qquad \begin{bmatrix} 9 & -2 & 0 & 5 \\ 18 & 1 & -6 & 7 \end{bmatrix}$$

Figure 7.1: One– and Two–Dimensional Matrices of Integers

There are several reasons to provide the basics of matrices in a discrete structures book. For example:

- Chapter 8 covers a type of set known as a relation. There are multiple useful ways to represent relations, one of which is a matrix representation.

- Another representation of relations is a data structure known as a graph. Graphs, a generalization of tree data structures, are far more useful than that one application. Matrices are used to represent graphs within programs.

- Speaking of programs: If you are studying discrete structures, you have almost certainly written a program that used an array data structure. Many definitions of matrices define them in terms of arrays, which can be confusing because the two words are often assumed to be synonyms. An n–dimensional array and an n-dimensional matrix are essentially the same thing.

- In computer graphics, matrices are used for a variety of purposes, including 2D projections, affine transformations, and texturing. We will present one computer graphics example later in this chapter (see Section 7.5).

- Cryptography, the study of ways to encode messages such that only the intended recipient can easily decode them, relies heavily on matrices. The Advanced Encryption Standard (AES), for example, is based on a sequence of matrix operations.

In short, a good working knowledge of matrices and their common operations is essential to understand many topics in Computer Science, most immediately the concepts of relations and graphs covered later in this book.

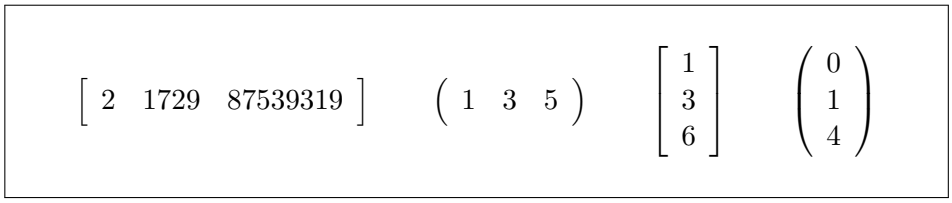


Figure 7.2: Examples of Row and Column Vectors with Brackets and Parentheses

7.2 Matrix Fundamentals

7.2.1 Matrix Notations and Sizes

Figure 7.1 demonstrates how one– and two-dimensional matrices are often drawn. The collections of values are bounded on the left and right with brackets (a.k.a. square brackets), with the tiny bracket 'tips' on the tops and bottoms of the symbols serving to limit the content vertically. Some people prefer to use parentheses (a.k.a. round brackets) instead of brackets.

One–dimensional matrices can be presented as *row matrices* (a.k.a. *row vectors*) or as *column matrices* (a.k.a. *column vectors*). The difference is more than just a matter of taste, as we will see later in this chapter. Either way, we still use brackets or parentheses to delimit their content, as shown in Figure 7.2.

Just as we labeled predicates, propositions and sets, we label matrices, usually with upper-case letters, as in:

$$T = \begin{bmatrix} 2 & 1729 & 87539319 \end{bmatrix}$$

To reference the values within a one–dimensional matrix, we use the lower-case version of the matrix's label and append a subscript, with the first value (left-most for row matrices or top-most for column matrices) indexed with one.[3] Thus, in T, $t_1 = 2, t_2 = 1729$, and $t_3 = 87539319$.

In two–dimensional matrices, we need two subscripts. By convention, the first index is the row index, the second is the column index, and element m_{11} is in the upper-left corner. In our two–dimensional matrix from Figure 7.1:

[3] I know, I know. You're a proud, card-carrying computer programmer, and know that, for reasons of efficiency, most programming languages use zero-based indexing for arrays and lists. Mathematics is just slightly older than computer science, so mathematicians got to define matrix indexing. To facilitate communication, everyone, even the proudest computer scientist, uses one-based indexing when discussing matrices.

$$M = \begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 9 & -2 & 0 & 5 \\ 18 & 1 & -6 & 7 \end{bmatrix} \end{array} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \end{bmatrix}$$

$m_{11} = 9$ and $m_{24} = 7$. Should a matrix have more than nine rows or columns, commas are added to separate the row and column values, as in $m_{32,15} = 7$.

We follow the same row–column ordering to define the size of a two–dimensional matrix. M has two rows and four columns, and so is a 2×4 (read: "two by four") matrix.

What about the size of a one–dimensional matrix? We can't only say that T, above, is a matrix of length three, because that alone doesn't distinguish between a row vector of length three and a column vector of length three. The problem is solved by thinking of a vector as either a rather flat, or a rather thin, two–dimensional matrix. Thus, T, a row vector, is a 1×3 matrix. A column vector of n values is described as having a size of $n \times 1$. Once the type of vector has been established, we can use a simplified notation for element references. That is, t_1 references the same element as does t_{11}, t_2 references the same element as does t_{12}, etc. Because we already know that T is a row vector, the single-subscript notation is all that we need.

7.2.2 Basic Matrix Definitions (and One Operation)

We need to present a few definitions (one of which is really an operation) before we encounter the numeric matrix operations that will be useful later in the book. The first one is easy:

square matrix

Definition 37: Square Matrix

A *square matrix* is a two–dimensional matrix in which the number of rows equals the number of columns.

Square matrices are frequently used in computing. For that reason, you will see many definitions of matrix representations and operations that require operands to be square. Note that the next definition is not one of them!

$$T = \begin{bmatrix} 2 & 1729 & 87539319 \end{bmatrix} \quad U = \begin{bmatrix} 2 \\ 1729 \\ 87539319 \end{bmatrix} \quad V = \begin{bmatrix} 12 & 23 \\ 34 & 45 \\ 56 & 67 \end{bmatrix}$$

$$W = \begin{bmatrix} 12 & 23 \\ 34 & 45 \end{bmatrix} \quad X = \begin{bmatrix} 12 & 23 \\ 43 & 45 \\ 56 & 67 \end{bmatrix} \quad Y = \begin{bmatrix} 2 \\ 1729 \\ 87539319 \end{bmatrix}$$

Figure 7.3: Six Integer Matrices for Example 123.

Definition 38: Matrix Equality *matrix equality*

Matrices A and B are *equal* (denoted $A = B$) iff they share the same dimensions **and** all pairs of corresponding elements are equal.

Example 123:

Consider the matrices in Figure 7.3. T and U have the same content, but they have different sizes. Thus, $T \neq U$.

V and W match in all four of their corresponding elements ($v_{11} = w_{11}$, $v_{12} = w_{12}$, etc.) but their sizes are not the same, so $V \neq W$.

V and X have the same size (3×2) but not all corresponding pairs of values match ($v_{21} \neq w_{21}$, for example), so $V \neq X$.

There are two matrices that are equal: $U = Y$. Both are 3×1 column vectors, and all three corresponding pairs of values are equal. Sure, the two columns of values are formatted differently, but that is not a problem.

Note that only W in Figure 7.3 is an example of a square matrix.

Does Definition 38 allow for us to say that a matrix is equal to itself? For example, referring again to Figure 7.3, can we say that $W = W$? Yes! W

is the same size as itself, and all of its corresponding pairs of values match. Thus, $W = W$.

Time for the operation masquerading as a matrix definition.

transposition

Definition 39: Transposition

The transposition of an $m \times n$ matrix A is the $n \times m$ matrix A^{T} in which the rows of A become the columns of A^{T}.

Example 124:

Consider this matrix S:

$$S = \begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{bmatrix}$$

To construct S^{T}, write the elements of the first row vertically. Do the same with the second row's values, writing them vertically just to the right of the column you formed from the first row's content. Do the same with the content of the third row. The result is the transpose of S:

$$S^{\mathrm{T}} = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \\ 13 & 14 & 15 \end{bmatrix}$$

Example 125:

Can vectors be transposed? Yes! Refer back to matrices T and U in Figure 7.3. $U = T^{\mathrm{T}}$ and $T = U^{\mathrm{T}}$. Because $U = Y$, we can also say that $Y = T^{\mathrm{T}}$ and $T = Y^{\mathrm{T}}$.

A generalization of Example 125 is worth remembering: Given a matrix A, the transpose of the transpose of A equals A. In notation, $\left(A^{\mathrm{T}}\right)^{\mathrm{T}} = A$.

Example 126:

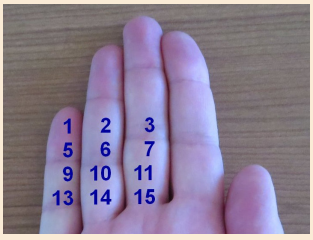
There's a well-known visualization of matrix transposition that is worth knowing because, once you learn it, you are unlikely to forget how to transpose a matrix.

Imagine a right human hand, held up so that you see the back of it with its fingers pointing to the left. If you're lacking in imagination, consult this photograph:

More imagination: Think of the little finger as the first row of the matrix to be transposed. The (in this case ringless) ring finger represents the second row, etc.

$$\begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{bmatrix} \quad \Longrightarrow$$

Now imagine the hand flipped over, with the palm facing you and the fingers pointing up, as in this picture:[4]



Now the little finger represents the first column of the matrix, with the ring finger the second column. That is, the flipped hand represents

the transposition of the matrix represented by the first view of the hand.

The next definition of this subsection makes use of all three of the preceding definitions.

matrix symmetry

Definition 40: Matrix Symmetry

Matrix A is *symmetric* iff $A = A^{\mathrm{T}}$.

Definition 40 clearly uses matrix equality and matrix transposition, but it doesn't say anything about A needing to be a square matrix. We can infer this detail from the definition. We know that equal matrices need to have matching quantities of rows and columns, and that transposition swaps rows and columns. Thus, for A and A^{T} to be equal, A's quantities of rows and columns must be the same. In other words, any symmetric matrix must be a square matrix.

Example 127:

Consider this matrix:

$$R = \begin{bmatrix} -2 & 4 & 1 & 9 \\ 4 & 13 & -6 & 21 \\ 1 & -6 & 0 & 4 \\ 9 & 21 & 4 & 10 \end{bmatrix}$$

Because R's first row and first column contain the same values, as do R's second row and second column, etc., R^{T} must equal R. By Definition 40, R is symmetric (and square!).

[4] Speaking of imagination: How did the owner of this hand not make a fortune as a hand model? Maybe this book will be the hand's big break. A hot-shot movie producer will be reading the book, trying to decide whether or not to option it as a major summer block-buster motion picture. ("Who should I cast as matrix S?") Suddenly, she sees this hand. "Forget the movie! I need this hand for a proctology commercial! The silly number tattoos don't matter; after all, we only need the index finger."

Example 128:

In the matrix of capital letters Q, below, some letters are missing. In order for the completed matrix Q' to be symmetric, which letters must be placed in the labeled locations?

$$Q = \begin{bmatrix} w & y & [1] & y & m \\ y & a & b & p & [2] \\ f & b & x & o & z \\ [3] & p & o & [4] & a \\ m & c & z & a & s \end{bmatrix}$$

Before rushing to examine the missing values, check the size of Q. If Q is not square, it can never equal its transpose, meaning that the missing values are irrelevant. Happily, Q is square — 5×5.

Location [1] is the third value of the first row. For Q to be symmetric, [1] must match the third value of the first column, which is f. We need to put an f in place of [1].

Do you prefer starting with columns? No problem! Location [2] is the second value of the fifth column. The second value of the fifth row is c, so we replace [2] with c to make the fifth column symmetric with the fifth row.

Let's use our indexing syntax to do [3]. [3] is located at q_{41}. To be symmetric, [3] must match q_{14}, which holds a y.

Finally, we have to replace [4]. It is located at q_{44}, which means that we have to replace [4] with ... itself! Because there's no letter at q_{44}, we can make Q' symmetric using any letter we wish to use; why not d?

Here's the resulting matrix Q':

$$Q' = \begin{bmatrix} w & y & f & y & m \\ y & a & b & p & c \\ f & b & x & o & z \\ y & p & o & d & a \\ m & c & z & a & s \end{bmatrix}$$

$Q' = (Q')^{\mathrm{T}}$; it is symmetric.

Before the final symmetry example, one more matrix definition.

main diagonal

Definition 41: Main Diagonal

The *main diagonal* of an $n \times n$ matrix A consists of the elements a_{11}, a_{22}, ..., a_{nn}.

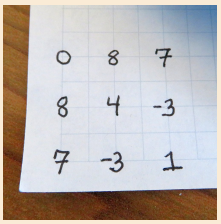
That is, the main diagonal runs from the upper–left corner to the lower–right corner of the matrix. Onto the example!

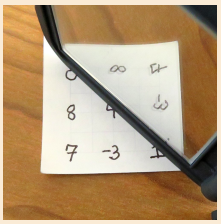
Example 129:

Did you enjoy the transposition–as–hand–flips idea? If so, you'll probably like this matrix symmetry visualization idea, too.

Imagine that you have a small rectangular mirror and a square matrix. Stand the mirror on one edge along the main diagonal so that you can see the values below the main diagonal reflected in the mirror. If the values in the matrix in the locations above the main diagonal match those in the reflection, the matrix is symmetric.

For instance, here's a 3×3 matrix that just happens to be a symmetric matrix, and the same matrix with a mirror positioned upon it as described above:





Yes, you have to realize that the infinity symbol is a reflected '8', that the accented 'w' is '-3', and the reflected '7' is ... well, whatever that symbol is, but apart from that, the reflected values match the existing values below the main diagonal.

Are you wondering about the values on the main diagonal? Don't worry about them! As we saw in Example 128, the main diagonal's values stay right where they are when the transposition is created. Because they only have to match themselves, main diagonal values cannot break symmetry.

7.3 Numeric Matrix Operations

Matrix transposition, covered in Section 7.2.2, is a matrix operation, but one that can be applied to any matrix, no matter the content. When the content of a matrix consists of real numbers, we can do a few more operations, the last of which is more than a bit of work to evaluate.

7.3.1 Matrix Addition and Subtraction

Matrix addition is a simple and commonly-used operation. If you have ever written a computer program that filled two arrays with data and added the content together, element by element, to fill a third array, you've performed matrix addition.

matrix addition

Definition 42: Matrix Addition

The sum of two numeric $n \times m$ matrices A and B is the $n \times m$ matrix C such that $c_{ij} = a_{ij} + b_{ij}$. (Also known as *Matrix Sum.*)

The definition of matrix subtraction is only slightly different.

matrix subtraction

Definition 43: Matrix Subtraction

The difference of two numeric $n \times m$ matrices A and B is the $n \times m$ matrix C such that $c_{ij} = a_{ij} - b_{ij}$. (Also known as *Matrix Difference.*)

Example 130:

Let:

$$O = \begin{bmatrix} 0 & 9 \\ -4 & 2 \\ 4 & 5 \\ -2 & 1 \end{bmatrix} \qquad P = \begin{bmatrix} 4 & -8 \\ 1 & 8 \\ -4 & 2 \\ 0 & 1 \end{bmatrix}$$

O and P are both 4×2 in size, and contain integers. Thus, we can

add their corresponding pairs of values to compute the matrix addition $O + P$:

$$O + P = \begin{bmatrix} 0+4 & 9+(-8) \\ -4+1 & 2+8 \\ 4+(-4) & 5+2 \\ -2+0 & 1+1 \end{bmatrix} = \begin{bmatrix} 4 & 1 \\ -3 & 10 \\ 7 & 0 \\ -2 & 2 \end{bmatrix}$$

Perhaps not surprisingly, the sum $P + O$ is the same:

$$P + O = \begin{bmatrix} 4+0 & -8+9 \\ 1+(-4) & 8+2 \\ -4+4 & 2+5 \\ 0+(-2) & 1+1 \end{bmatrix} = \begin{bmatrix} 4 & 1 \\ -3 & 10 \\ 7 & 0 \\ -2 & 2 \end{bmatrix}$$

Example 130 suggests (but does not prove!) that matrix addition is commutative, which it is. Matrix addition is commutative because each element of the resulting matrix is formed from the addition of the corresponding elements of the operand matrices, and addition of real values is commutative.

Example 131:

Matrix addition is commutative, but what about matrix difference? The answer to that question depends on the answer to this question: Is basic subtraction commutative? That is, does $a - b = b - a$ for all values of a and b?

Remember, to disprove a conjecture, we only need one counter-example. Let's try $a = o_{42} = 1$ and $b = p_{42} = 1$ from the matrices O and P in Example 130. $a - b = 1 - 1 = 0$ and $b - a = 1 - 1 = 0$. In this case $a - b = b - a$, which means that this is not a counter-example. But, this case is the specific situation in which $a = b$. We need a little more variety.

Let's try an $a \neq b$ example by using $a = o_{11} = 0$ and $b = p_{11} = 4$. $a - b = 0 - 4 = -4$, but $b - a = 4 - 0 = 4$. Because $a - b \neq b - a$, we have our counter-example. It follows that matrix difference is not commutative.

7.3.2 Scalar Multiplication

Having just covered matrix difference, we can reveal a minor secret: We did not need to cover it. Just as we can accomplish the subtraction of two real numbers by multiplying the second by -1 and adding ($a - b = a + (-1 \cdot b)$), we can perform matrix difference by performing a scalar multiplication on the second matrix and adding.

Before we can define scalar multiplication, we need to explain what a 'scalar' is.

Definition 44: Scalar *scalar*

A scalar is a real number.

Yes, very exciting. Also somewhat superficial, but good enough for our needs.[5]

Definition 45: Scalar Multiplication *scalar multiplication*

The multiplication of a scalar d and an $n \times m$ numeric matrix A, written dA, is the $n \times m$ matrix B such that $b_{ij} = d \cdot a_{ij}$.

In plain English: To perform a scalar multiplication, we multiply every element of the given matrix by the given scalar.

An alternate term for 'scalar multiplication' is 'scalar product,' but that term is more commonly used as an alternate name for 'dot product,' which is why we are not going to use the phrase 'scalar product' in this book.

Example 132:

Consider matrix O from Example 131, and the scalar 3:

[5] The term *scalar* has several meanings in math and science. For example, a scalar is, to mathematicians more generally, an element of any *field*, where a field is a set of values for which the operations of addition, subtraction, multiplication, and division are defined. For example, the set $\mathbb{Z}$ can be called the field of integer values. Another example: In computer programming, 'scalar' is an older term for a variable.

$$3\,O = 3 \cdot O = 3 \cdot \begin{bmatrix} 0 & 9 \\ -4 & 2 \\ 4 & 5 \\ -2 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 27 \\ -12 & 6 \\ 12 & 15 \\ -6 & 3 \end{bmatrix}$$

Example 133:

In Section 7.3.1, we explained, but did not demonstrate, matrix difference. Here's how to do matrix difference with scalar multiplication, using matrices O and P from Example 131:

$$\begin{aligned} O - P &= O + (-1\,P) \\ &= \begin{bmatrix} 0 & 9 \\ -4 & 2 \\ 4 & 5 \\ -2 & 1 \end{bmatrix} + \left(-1 \cdot \begin{bmatrix} 4 & -8 \\ 1 & 8 \\ -4 & 2 \\ 0 & 1 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0 & 9 \\ -4 & 2 \\ 4 & 5 \\ -2 & 1 \end{bmatrix} + \begin{bmatrix} -4 & 8 \\ -1 & -8 \\ 4 & -2 \\ 0 & -1 \end{bmatrix} \\ &= \begin{bmatrix} -4 & 17 \\ -5 & -6 \\ 8 & 3 \\ -2 & 0 \end{bmatrix} \end{aligned}$$

7.3.3 Matrix Multiplication (a.k.a. Matrix Product)

Matrix multiplication, the first time you see it, may appear to be someone's idea of a joke. It can be difficult to see, from the definition, how there can be a practical application of such a complex collection of seemingly arbitrary operations. So, let's start with a practical application.

Example 134:

Sports leagues often need to rank their teams for playoff seedings, or just bragging rights. The obvious starting point is each team's total number of victories, but how can we 'break' ties?

Consider the Letters League, with teams Alpha, Beta, Gamma, and Delta. Each team has played two games, with the following results. In the first pair of games, Alpha defeated Gamma and Beta defeated Delta. In the second pair, Alpha defeated Delta and Gamma defeated Beta. We can represent these results in a *domination matrix*, D. d_{rc} is '1' if team r defeated team c, and is '0' otherwise. D is a square matrix (4×4 in this case), with the rows and columns labeled identically:

$$D = \begin{array}{r} \\ \text{Alpha} \\ \text{Beta} \\ \text{Gamma} \\ \text{Delta} \end{array} \begin{array}{c} \begin{array}{cccc} \text{Alpha} & \text{Beta} & \text{Gamma} & \text{Delta} \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \rightarrow \begin{array}{c} \text{Wins} \\ \begin{bmatrix} 2 \\ 1 \\ 1 \\ 0 \end{bmatrix} \end{array}$$

Adding up the rows gives us the number of victories: Alpha has won two games, and Beta and Gamma one game each. (Adding the columns gives the number of losses.) Beta and Gamma are tied with one victory each. But are they equally talented? What additional information can we use to distinguish them?

Besides telling us direct wins and losses, the domination matrix can also tell us *transitive* results. That is, we know that Alpha defeated Gamma, and then Gamma defeated Beta. By transitivity, we could reason that Alpha is a better team than Beta, even through Alpha and Beta have not played directly against one another.

We can also represent these transitive results (which are called *two–step* results, with the direct results called *one–step*) using a similar matrix

which, for reasons of foreshadowing, we will name D^2:

$$D^2 = \begin{array}{r} \\ \text{Alpha} \\ \text{Beta} \\ \text{Gamma} \\ \text{Delta} \end{array} \begin{array}{c} \begin{array}{cccc} \text{Alpha} & \text{Beta} & \text{Gamma} & \text{Delta} \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \rightarrow \begin{array}{c} \text{Two-Step} \\ \text{Results} \\ \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \end{array}$$

The '1' in the top row is the example we just used: Alpha is two–step dominant over Beta because Alpha defeated Gamma and Gamma defeated Beta. The other '1' in D^2 comes from Gamma defeating Beta and Beta defeating Delta.

Adding the values across the rows of D^2 gives the numbers of two–step results. Adding the one–step totals to the two–step totals distinguishes Beta and Gamma: Gamma's total is two, while Beta's is still one:

$$\text{One-Step} + \text{Two-Step} = \begin{array}{c} \text{One-Step} \\ \text{Results} \\ \begin{bmatrix} 2 \\ 1 \\ 1 \\ 0 \end{bmatrix} \end{array} + \begin{array}{c} \text{Two-Step} \\ \text{Results} \\ \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \end{array} = \begin{array}{r} \\ \text{Alpha} \\ \text{Beta} \\ \text{Gamma} \\ \text{Delta} \end{array} \begin{array}{c} \text{Totals} \\ \begin{bmatrix} 3 \\ 1 \\ 2 \\ 0 \end{bmatrix} \end{array}$$

You may well be asking: "OK, great, but what does this have to do with *multiplying* matrices?" Filling D^2 with two–step results wasn't too hard to do by inspection, because we have just four teams and four games played. Any thoughts as to an operation that we might be able to use to automate that process to handle more teams and more games? We will graciously give you one guess.

(We will tell you if your guess is correct, we promise. We will revisit this example later in this chapter, in Example 148.)

Matrix addition and scalar multiplication were straight-forward; matrix multiplication is not.

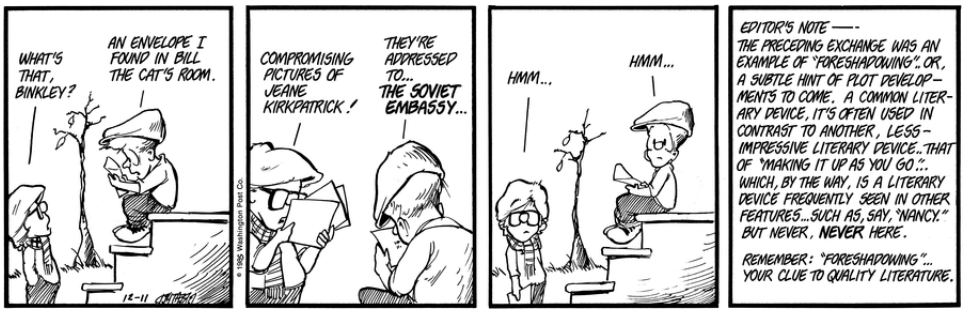


Figure 7.4: Foreshadowing — also your guide to quality textbook writing? Credit: Berkeley Breathed, "Bloom County", December 11, 1985.

Definition 46: Matrix Multiplication *matrix multiplication*

The product of an $m \times n$ matrix A and an $n \times o$ matrix B is an $m \times o$ matrix $C = A \cdot B = AB$ in which $c_{ij} = \sum_{k=1}^{n} (a_{ik} \cdot b_{kj})$. (Also known as *Matrix Product*.)

For a definition that is not even two lines long, we have a lot to explain.

First, take a close look at the sizes of the operand matrices A and B. They have a letter (n) in common, and that is not an accident. We can perform the product of matrices D and E, in that order (that is, with D on the left of E), only when D's quantity of columns equals E's quantity of rows.

Second, look at the size of the resulting matrix, C. Its quantity of rows (m) matches the quantity of rows of A and its quantity of columns (o) matches the quantity of columns of B. Again, this is not an accident.

Putting these two size observations together gives us the two preliminary steps of performing a matrix multiplication:

1. Verify that the quantity of columns of the first/left matrix equals the quantity of rows of the second/right matrix. If they do not match, a matrix multiplication cannot be performed on the matrices in this order.

2. The size of the resulting matrix will have the same quantity of rows as does the first/left matrix, and the same quantity of columns as does the second/right matrix.

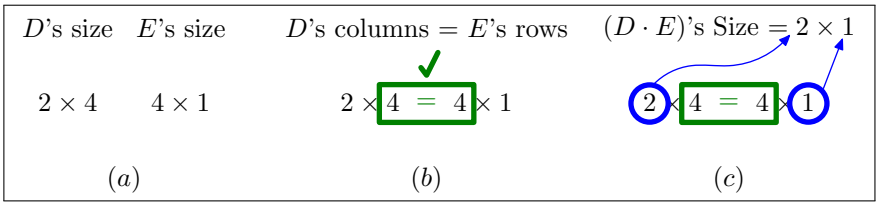


Figure 7.5: Checking Matrix Sizes before Matrix Multiplication

When written out, the steps are pretty dry. We can show these steps visually instead, as Example 135 and Figure 7.5 demonstrate.

Example 135:

Let D and E be the following matrices:

$$D = \begin{bmatrix} 3 & 5 & -1 & 0 \\ 0 & 2 & 3 & -2 \end{bmatrix} \qquad E = \begin{bmatrix} 1 \\ 0 \\ -4 \\ 2 \end{bmatrix}$$

D's size is 2×4, and E's is 4×1. By the definition of matrix multiplication, in order to perform $D \cdot E$, the number of columns of D (4) must equal the number of rows of E (4). The remaining two values, the rows of D (2) and the columns of E (1) give us the size of their product: 2×1.

Figure 7.5 shows a visualization that can help you remember the needed relationships. Begin, as shown in part (a), by writing down the sizes of the matrices D and E, side by side, with the size of the first/left matrix on the left. This places the number of columns of the left matrix next to the number of rows of the right matrix, making it easy to remember that this is the pair of values that must be equal (part (b)). The remaining pair of values provide the size of the resulting matrix, as shown in part (c).

Determining the that matrix product can be computed, and how large the resulting matrix will be, is most of the text of the Definition 46. Unfortunately, the hard part remains: Computing the product. The notation is short and

sweet in the definition ("$c_{ij} = \sum_{k=1}^{n} (a_{ik} \cdot b_{kj})$"), but what does that expression require us to do?

Start with c_{ij}. The subscripts tell us the location within the result matrix whose value we are trying to compute: row i, column j. The summation $\left(\sum_{k=1}^{n}\right)$ has the other two variables, n and k. We already know what n represents: The quantity of columns of the first/left matrix, A, as well as the quantity of rows of the second/right matrix, B. k starts at 1 and is incremented through n. For each value of k, we find the values at locations a_{ik} and b_{kj} and multiply them. Adding up those n products produces the value to be placed at c_{ij}. Yes, computing a matrix product can require a fair amount of arithmetic, but not difficult arithmetic, merely tedious and error-prone arithmetic.[6]

Example 136:

Let's finish what we started in Example 135. Here are D and E again, with place-holders for the values of the resulting $D \cdot E$ matrix, which we will call F:

$$D = \begin{bmatrix} 3 & 5 & -1 & 0 \\ 0 & 2 & 3 & -2 \end{bmatrix} \qquad E = \begin{bmatrix} 1 \\ 0 \\ -4 \\ 2 \end{bmatrix} \qquad D \cdot E = F = \begin{bmatrix} \square \\ \square \end{bmatrix}$$

Let's start with f_{11}. With $i = j = 1$, and D's quantity of columns being 4, the summation becomes $f_{11} = \sum_{k=1}^{4} (d_{1k} \cdot e_{k1})$. Expanding the

[6] We cannot understand how 'tedious' and 'error-prone' are not wildly popular names for children. They aren't even gender–specific! Yeah, it might be hard to know which name to give to your first child, but you can always swap names later.

summation makes our task plain:

$$\begin{aligned} f_{11} &= \sum_{k=1}^{4} (d_{1k} \cdot e_{k1}) \\ &= (d_{11} \cdot e_{11}) + (d_{12} \cdot e_{21}) + (d_{13} \cdot e_{31}) + (d_{14} \cdot e_{41}) \\ &= (3 \cdot 1) + (5 \cdot 0) + (-1 \cdot -4) + (0 \cdot 2) \\ &= 3 + 0 + 4 + 0 \\ &= 7 \end{aligned}$$

Notice that what we are doing is 'walking', left to right, across the first row of D while simultaneously 'sliding', top to bottom, down the first column of E. This is always what you will need to do to compute the value at row i and column j of the result matrix: Walk across the i-th row of the first/left matrix, and slide down the j-th column of the second/right matrix.

All that remains is for us to repeat this for the other location, f_{21}:

$$\begin{aligned} f_{21} &= \sum_{k=1}^{4} (d_{2k} \cdot e_{k1}) \\ &= (d_{21} \cdot e_{11}) + (d_{22} \cdot e_{21}) + (d_{23} \cdot e_{31}) + (d_{24} \cdot e_{41}) \\ &= (0 \cdot 1) + (2 \cdot 0) + (3 \cdot -4) + (-2 \cdot 2) \\ &= 0 + 0 + (-12) + (-4) \\ &= -16 \end{aligned}$$

And so:

$$\begin{bmatrix} 3 & 5 & -1 & 0 \\ 0 & 2 & 3 & -2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ -4 \\ 2 \end{bmatrix} = \begin{bmatrix} 7 \\ -16 \end{bmatrix}$$

Before we leave D and E, a final question: Can we compute the matrix product $E \cdot D$? E is 4×1 in size, but D has 2 rows. $1 \neq 2$, and so $E \cdot D$ cannot be computed. It follows that, in general, matrix multiplication is not commutative.

Example 137:

Are you worried that you will have trouble remembering to walk across the rows of the first matrix and slide down the columns of the second, rather than the reverse? Perhaps you're still a bit worried about checking the matrix sizes correctly? Positioning the operand matrices in an unusual way can help with both concerns.

Consider the matrices G and H and the eventual matrix product J, positioned as we've been positioning matrices:[7]

$$G = \begin{bmatrix} 3 & 2 \\ 0 & -2 \\ 1 & 0 \end{bmatrix} \qquad H = \begin{bmatrix} 1 & 3 \\ 3 & 4 \end{bmatrix} \qquad G \cdot H = J = \begin{bmatrix} \square & \square \\ \square & \square \\ \square & \square \end{bmatrix}$$

By moving H up, and shifting J to be next to G and beneath H, we produce:

$$H = \begin{bmatrix} 1 & 3 \\ 3 & 4 \end{bmatrix}$$

$$G = \begin{bmatrix} 3 & 2 \\ 0 & -2 \\ 1 & 0 \end{bmatrix} \quad \begin{bmatrix} \square & \square \\ \square & \square \\ \square & \square \end{bmatrix} = J = G \cdot H$$

This arrangement shows that J fits perfectly next to G and beneath H because it has the same quantity of rows as does G, and the same quantity of columns as does H — exactly our sizing relationships. Also notice that the rows of G and the columns of H, if you imagine extending them, intersect at the boxes within J. If you select a particular box from J, the row of G and the column of H that intersect at that box are the row/column that we will walk–across/slide–down to produce the value that belongs in that box. Be aware that we have to check separately that the quantity of columns of G equals the number of rows of H.

What's the content of J? We encourage you to work that out yourself!

[7] "Wait; G, H, ... J? What have you got against I?" Calm down; we like I just fine. A special family of matrices has the name I, so using it here to name a different matrix could be confusing. Consumed by curiosity about I? Jump ahead to Section 7.4.3.

7.4 Matrix Powers and the Identity Matrix

7.4.1 Matrix Powers

Consider multiplying a square matrix by another square matrix. The only way that this can be done is if both operand matrices are the same size (because the quantity of rows of the first/left square matrix must equal the quantity of columns of the second/right square matrix). The resulting matrix will be of the same size as the operand matrices.

Let's take this a bit further. If we want to multiply a matrix by itself, the matrix must be square, and the resulting matrix will be a square matrix of the same size. We can multiply the result by the original again, and repeat this as many times as we wish. It's similar to multiplying a real number by itself multiple times to raise the number to an integer power. For example, $2 \cdot 2 = 2^2 = 4$, $(2 \cdot 2) \cdot 2 = 2^2 \cdot 2 = 2^3 = 8$, etc. Doing repeated matrix multiplication with square numeric matrices forms *matrix powers* of the matrix.

matrix power

Definition 47: Matrix Power

The n^{th} matrix power of an $m \times m$ numeric matrix A, denoted A^n, is the result of $n-1$ successive matrix products of A.

Example 138:

What is K^2, if $K = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}$?

Because we already know how to perform matrix multiplication, computing K^2 is straight-forward.

$$\begin{aligned} K^2 &= K \cdot K \\ &= \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix} \end{aligned}$$

Example 139:

K^2 wasn't hard, because it didn't matter which copy of K was on the left — $K = K$! But to create K^3, do we compute $(K \cdot K) \cdot K$ or $K \cdot (K \cdot K)$?

The answer is: It doesn't matter! From Example 138, we know what K^2 contains. Let's compute both $K^2 \cdot K$ and $K \cdot K^2$:

$$K^2 \cdot K = \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 5 \\ 5 & 12 \end{bmatrix}$$

$$K \cdot K^2 = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 5 \\ 5 & 12 \end{bmatrix}$$

That might seem to be a little ... suspicious. Sure, this worked out, but K is symmetric; maybe that's a special case. We could try it again with a non-symmetric matrix, but perhaps that would be a matrix that we cooked up to make both orderings work. To convince people that the ordering of the matrix products really does not matter, what we need is more generality, such as we would use in a proof.

Example 140:

Problem: Prove that A^3, the third matrix power of an 2×2 numeric matrix A, is equal to both $A^2 \cdot A$ and $A \cdot A^2$.

Solution: Let's get something out of the way immediately: We demonstrated, in Section 7.3.3, that matrix multiplication is not generally com-

mutative. However, our current problem is not a general situation; rather, it is specific to matrix powers, in which there is only one given matrix. Further, Example 139 gives us some reason to believe that matrix multiplication might actually be commutative for matrix powers. Given that, attempting a proof is worth the trouble.

We know that the resulting matrix A^3 will be a 2×2 matrix. What we need to show is that the four values that comprise A^3 are the same for both $A^2 \cdot A$ and $A \cdot A^2$. Because the matrices we need to compute are all 2×2, and because the power is just 3, we can simply multiply all four values for each of the two products and verify that they are the same expressions. Hard? Not really. Tedious and error-prone? Definitely. Good thing we're going to do the dirty work for you!

Conjecture: If A is a 2×2 numeric matrix, then $A^3 = A^2 \cdot A = A \cdot A^2$.

Proof (Direct): We are given that A is a 2×2 numeric matrix.

Let $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, where $a, b, c, d \in \mathbb{R}$.

$$\begin{aligned} A^2 &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix} \\ &= \begin{bmatrix} a^2+bc & ab+bd \\ ac+cd & bc+d^2 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} A^2 \cdot A &= \begin{bmatrix} a^2+bc & ab+bd \\ ac+cd & bc+d^2 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix} \\ &= \begin{bmatrix} (a^3+abc)+(abc+bcd) & (a^2b+b^2c)+(abd+bd^2) \\ (a^2c+acd)+(bc^2+cd^2) & (abc+bcd)+(bcd+d^3) \end{bmatrix} \end{aligned}$$

$$\begin{aligned} A \cdot A^2 &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} a^2+bc & ab+bd \\ ac+cd & bc+d^2 \end{bmatrix} \\ &= \begin{bmatrix} (a^3+abc)+(abc+bcd) & (a^2b+abd)+(b^2c+bd^2) \\ (a^2c+bc^2)+(acd+cd^2) & (abc+bcd)+(bcd+d^3) \end{bmatrix} \end{aligned}$$

The expressions in all four locations of both results are equivalent (due to commutativity of addition of real numbers).

Therefore, $A^3 = A^2 \cdot A = A \cdot A^2$.

A note about the added parentheses: We added them to the result matrices to organize the expressions a bit; they aren't mathematically necessary.

In the proof, we explained what we were doing in terms of commutativity.

We can also look at it in terms of associativity: $A^3 = AAA = (AA)A = A(AA)$. Associativity of matrix multiplication holds for any three numeric matrices A (of size $i \times j$), $B(j \times k)$, and $C(k \times l)$, not just for matrix powers: $A(BC) = (AB)C$. For that matter, matrix multiplication is also distributive over matrix addition, when the sizes are appropriate: $D(E+F) = DE+DF$.

Are you wondering how the proof can be generalized to A^4, to A^5, and eventually to A^m, where m is any positive integer power? We can do that by applying the definition of matrix multiplication more directly. We can also do it inductively ... but we haven't covered inductive proofs yet. We soon will; stay tuned!

7.4.2 The Cost of Matrix Multiplication

How much effort is required to multiply two matrices? That's not an easy question to answer. Complicating factors include whether or not code is being parallelized, how memory is being managed, and which matrix multiplication algorithm is used.[8] We can ignore those complications and still learn something useful about how to multiply matrices using the definition's approach.

When analyzing the execution cost of an algorithm, individual operations are typically ignored in favor of an approximate category result. For example, sequentially searching a list is a linear operation, meaning that the work required is described as a linear function of n, the quantity of items in the list, rather than other function of n (e.g., logarithmic or quadratic). Different implementations of sequential search can have different linear functions that describe the number of operations performed, such as $3n + 2$ or $5n + 6$. Because those are linear functions, we say that sequential search belongs to the "linear" category of algorithm efficiencies.

In this section, we will be more detailed. We know that matrix multiplications require many real–number multiplications and additions. But how many? Let's find out.

The Cost of Multiplying Two Matrices

We defined two matrices D and E in Example 135:

[8] Yes, there are algorithms other than the one described by the matrix multiplication definition, all created to require less effort. Two examples: The definition's 'naive' approach can be parallelized. Strassen's algorithm exchanges multiplications for additions, but an advantage is gained only for matrices with hundreds of rows/columns.

$$D = \begin{bmatrix} 3 & 5 & -1 & 0 \\ 0 & 2 & 3 & -2 \end{bmatrix} \qquad E = \begin{bmatrix} 1 \\ 0 \\ -4 \\ 2 \end{bmatrix}$$

Because the size of $F = D \cdot E$ is 2×1, we need to compute two intermediate results, f_{11} and f_{21}. But, as we perform the same operations to compute each of them (albeit on different groups of values), we can figure the numbers of multiplications and additions for any one element f_{ij} and double it to get the total for the entire matrix product.

Consider f_{21}. Expanding the definition's summation expression, as we did in Example 136, shows us the required quantities of operations:

$$f_{21} = \sum_{k=1}^{4} (d_{2k} \cdot e_{k1}) = (d_{21} \cdot e_{11}) + (d_{22} \cdot e_{21}) + (d_{23} \cdot e_{31}) + (d_{24} \cdot e_{41})$$

Computing f_{21} requires four multiplications and three additions. Thus, computing the content of F requires eight multiplications and six additions. Hardly a lot of either, but of course the numbers will grow as as sizes of the matrices grow. We need to generalize our observations so that we can determine the work required for any pair of matrices.

To make what follows easier to understand, let's re–introduce the variables we used in the matrix product definition. Assume that we are computing $C = A \cdot B$. Let A be $m \times n$ and let B be $n \times o$, which means C is $m \times o$.

Both the quantities of multiplications and additions are determined by n, which is both the quantity of columns of A, and the number of rows of B. Specifically, for each element c_{ij}, we perform n multiplications of elements of A and B, and $n-1$ additions to sum those products. There are $m \cdot o$ values in C to compute. Thus, there are mno multiplications and $m(n-1)o$ additions performed in the computation of C.

Example 141:

Question: How many total multiplications and additions are required to compute AB using the definition when A is 10×12 and B is 12×6?

Answer: Using our m, n, and o variables, $m = 10$, $n = 12$, and $o = 6$.

There are $mno = 10 \cdot 12 \cdot 6 = 720$ multiplications and $m(n-1)o = 10 \cdot 11 \cdot 6 = 660$ additions, for a total of $720 + 660 = 1380$ operations.

Are you wondering how costly multiplications and additions of real numbers are to perform? There's no easy answer to that question, because different CPUs (Central Processing Units) implement addition and multiplication operations differently. Very roughly, floating–point multiplications are twice as expensive as are floating–point additions and subtractions.

The Cost of Multiplying Three Matrices

At the end of Section 7.4.1, we learned that matrix multiplication is associative; that is, $ABC = (AB)C = A(BC)$. This means that we can compute the matrix product ABC either by computing AB and then multiplying (on the right) by C, or by computing BC and multiplying (on the left) by A. At first glance, you might believe that the choice doesn't matter — both give you the correct answer. But do they both require the same amount of effort to produce that answer? Let's find out.

Example 142:

We will again assume, as we did in Example 141, that A is 10×12 and B is 12×6. Further, assume that our new third matrix, C, is 6×8.

From Example 141, we know what it costs to compute the 10×6 matrix AB. Computing the 10×8 matrix $(AB)C$ requires an additional $10 \cdot 6 \cdot 8 = 480$ multiplications and $10 \cdot 5 \cdot 8 = 400$ additions, which is 880 operations. Add in the 1380 operations needed to compute AB, and the grand total for $(AB)C$ is 2260 operations.

We need to deal with $A(BC)$ from scratch. BC requires $12 \cdot 6 \cdot 8 = 576$ multiplications and $12 \cdot 5 \cdot 8 = 480$ additions. $A(BC)$ requires $10 \cdot 12 \cdot 8 = 960$ multiplications and $10 \cdot 11 \cdot 8 = 880$ additions. The grand total is 2896.

The difference is a (probably) surprising 636 (28%) more operations to compute $A(BC)$ than are needed to compute $(AB)C$. Keep in mind that this result is for matrices of these three sizes. The math will work out differently for other matrix sizes.

The take–away message: If you need to compute a sequence of matrix multiplications just once, the associativity you choose is likely to make a

difference, but probably not enough of a difference to be a concern. However, if you need to perform the same product repeatedly (for example, because the contents of the matrices change), figuring out the most efficient product sequence is likely to be worth the effort.[9]

Example 143:

Huge matrices are not uncommon. The SuiteSpace Matrix Collection,[10] as of this writing, includes a dataset of U.S. patent citations covering just utility patents granted during the 37 years 1963 through 1999. The matrix representing it is $3,774,768 \times 3,774,768$. Using the matrix multiplication definition's approach, computing its second matrix power would require 53,786,191,549,544,312,832 multiplications and a mere 53,786,177,300,670,859,008 additions, for a total of more than 10^{20} operations.

7.4.3 Multiplicative Identity and the Identity Matrix

You might remember learning about additive and multiplicative identities in other classes. A value a is the *additive identity* of a set of values if adding a to another value of the set does not change the value. That is, $a + x = x$. We hope you know that $a = 0$ is the additive identity for the set of real numbers. Similarly, a value m is the *multiplicative identity* of a set when $m \cdot x = x$. Considering the set of reals again, its multiplicative identity is $m = 1$.

Because addition and multiplication are defined for matrices, it makes sense for us to think about additive and multiplicative identities for numeric matrices. The $m \times n$ additive identity matrix only contains zeroes. Of course, we have to choose m and n to allow the matrix addition to be performed. Practically, additive identity matrices are not very interesting. Multiplicative identity matrices, on the other hand, are rather useful.

A multiplicative identity matrix (always called I) must be sized so that the matrix multiplication with another matrix A, $A \cdot I$, is possible, and must have content such that the result equals A.

[9] Fun fact: The number of ways to associate an n matrix multiplication sequence is the $(n-1)^{\text{st}}$ Catalan number. For example, there are $C_3 = 5$ ways to associate a multiplication sequence of $n = 4$ matrices: $W(X(YZ)), W((XY)Z), (W(XY))Z, ((WX)Y)Z$, and $(WX)(YZ)$.

[10] Look for "SNAP/cit–Patents" at https://sparse.tamu.edu/

Let's consider size first. If A is $m \times n$ in size, to compute $A \cdot I$, I must have n rows. For the result to equal A, I must also have n columns. Thus, I is a square matrix of size $n \times n$. Notationally, we represent this matrix as I_n.

But what about $I \cdot A$? We want that to equal A, too. No problem; if we want I on the left of A, we need to use I_m to make the matrix multiplication work. That is: When the size of A is $m \times n$, multiplying on the left of A with I_m, or multiplying on the right with I_n, will make the sizes work out correctly.

Now for the content: What must I_m and I_n contain so that the products $I_m \cdot A$ and $A \cdot I_n$ both produce the original matrix A's content? We can figure this out for ourselves by thinking about how a matrix product is produced. Let A be 2×2, and consider $A \cdot I_2 = A$. Let's expand the matrices and stack them to make the products easier to see:

$$I_2 = \begin{bmatrix} i_{11} & i_{12} \\ i_{21} & i_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \begin{bmatrix} a'_{11} & a'_{12} \\ a'_{21} & a'_{22} \end{bmatrix} = A \cdot I_2 = A$$

Consider just the upper–left value of the product (a'_{11}). From the definition of matrix multiplication, we know that $a'_{11} = (a_{11} \cdot i_{11}) + (a_{12} \cdot i_{21})$. To end up with $a'_{11} = a_{11}$, We need to eliminate all of the other values on the right–hand side. Using multiplicative identity, i_{11} must be 1. To eliminate a_{12}, we need to multiply by zero, so $i_{21} = 0$.[11]

To complete the first row, a'_{12} must equal $(a_{11} \cdot i_{12}) + (a_{12} \cdot i_{22})$. By similar reasoning, $i_{12} = 0$ and $i_{22} = 1$, giving us $I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. We need to retain the values of the second row of A in exactly the same way. Happily, this content for I_2 will do that job, too. You can check for yourself that this content also works for $I_2 \cdot A$, as well as for other sizes of A that have more rows of A (when computing $A \cdot I_2$) or more columns of A (for $I_2 \cdot A$).

[11] The property $a \cdot 0 = 0 \cdot a = 0$ has at least two names, both of them quite dull: The Multiplication Property of Zero, and the Zero Property of Multiplication. We propose that it be renamed "Multiplicative Oblivion", which is significantly more exciting. "Multiplicative Domination" would work, too, but asking logical equivalences to share "Domination" with multiplication doesn't feel right.

Figure 7.6: If only Merlin had thought to consign Morgana to Mathematical Oblivion instead. Credits: John Boorman's "Excalibur"; https://memegenerator.net.

We can make I any size we need. In general:

$$I_n = \begin{array}{c} \\ 1 \\ 2 \\ \vdots \\ n \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \end{array}$$

Having provided so much explanation, the actual definition is old news:

Definition 48: Identity Matrices *identity matrices*

Identity Matrices (denoted I_n) are $n \times n$ matrices populated with 1 along the main diagonal and 0 elsewhere.

Time to talk about the utility of the identity matrix, starting with matrix powers. We now know that $A^3 = A \cdot A \cdot A$, and $A^2 = A \cdot A$. It follows that $A^1 = A$. But what, if anything, does A^0 represent? Right: I!

Here are two reasons that this makes sense. First, when r is a real number, we know that $r^1 = r$ and $r^0 = 1$. $A^1 = A$ and $A^0 = I$ parallels these basic

powers of real numbers. Second, think about writing a few lines of code to compute a running product. Traditionally, we would declare a variable to hold the product, and would use a loop to multiply values into this variable. But what should be the initial value of this variable? It can't be zero, because of "Mathematical Oblivion",[12] but one works nicely. The first time through the loop, we multiply the variable's value of one by the first value that forms the running product, and the result is that value. With that starting point, multiplying in the rest of the values will work as desired.[13] Similarly, when needing to compute a running product of matrices, we can initialize our running product matrix to hold I and multiply the first matrix from the collection into it.

Why would we need to have a running product of a collection of matrices? Computing a matrix power (see Section 7.4.1) is one reason. A second appears when performing object transformations in computer graphics, as explained in Section 7.5.

7.5 Matrix Operations in Orthographic Projections

7.5.1 Our Problem: Design a Shed . . . or Maybe a Dog House

Imagine that you want to build, in the back left corner of your yard, a storage shed that is 12 feet wide and 12 feet deep, with walls 9 feet high and a gable roof that rises to 12 feet at the ridgeline. To help yourself visualize the shed, you sketch out a diagram, with the origin at the corner of your yard, as shown in Figure 7.7 (a).

This diagram is a two-dimensional representation of a three-dimensional object. In drafting, this kind of representation is known as an *orthographic projection*.[14]

Having drawn your shed to show three sides, next you decide to draw only the front view of the shed; that is, how the front of the shed would look if you stood in the middle of the road, directly in front of the shed, and looked

[12] You read the previous footnote, didn't you?

[13] Yes, we could initialize the product variable to the first value of our collection and start the 'product–ing' with the second value, but that wouldn't help us make our point, and nothing is more important than whatever point we are currently trying to make . . . whether or not we can remember what it was ten seconds from now.

[14] Specifically, an axonometric orthographic projection. We're pretty sure. It sounds impressive, anyway. We are confident that it is not a *perspective projection*, because there are no vanishing points.

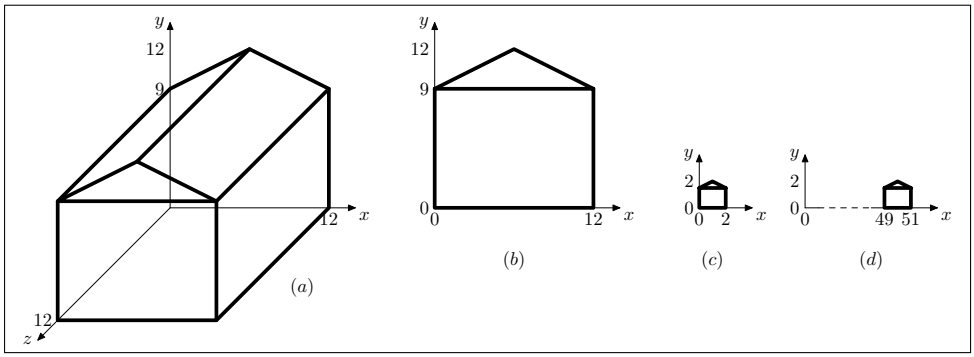


Figure 7.7: Turning a Shed into a Dog House in Three Easy Steps.

toward your back yard. You would see only the front of the shed, not the sides, the roof, or the back. This is shown in Figure 7.7 (b). You can think of this as 'flattening' the shed against an imaginary wall along your back lot line, much as you might flatten an aluminum can by standing it upright on the ground and stomping on it with your foot. This is also an example of an orthographic projection, but is more straight–forward because the projection plane — the imaginary wall against which we are flattening the shed — is the same as one of the three coordinate planes (in this case, the xy plane).[15]

Time for a plot twist: You start pricing materials and realize that your vision greatly exceeds your budget. What will you do? Quit and look like an incompetent fool to your friends and neighbors? No chance! You regroup and get to work on the design of a dog house for your faithful companion. A shed design quickly becomes a dog house design with just a little *scaling* (a.k.a. resizing) of the dimensions. After measuring your puzzled dog, you decide to make the dog house one-sixth the width of the shed, and one-fourth as tall and as deep. Figure 7.7 (c) shows the scaled front view.

Your intellectual reputation has been saved! But is the back left corner of the yard the right place for the dog house? If you win the lottery[16] and revive your shed dreams, you'll still want it in the back corner. With that in mind, you decide to place the dog house in the middle of the back of the yard. Your lot is 100 feet wide, so the middle of the house will be 50 feet from the

[15] We are intentionally not giving precise definitions of these projections, because we want to get back to matrices and matrix operations before the sun flames out. Please forgive us, American Design Drafting Association!

[16] Reality check: You are highly unlikely to win the lottery's grand prize. How extremely highly unlikely? We will cover that in a later chapter. In the meantime, just take our word for it: Ridiculously extremely highly unlikely.

back corners. This *translation* of the dog house from one location to another is depicted in Figure 7.7 (d).

7.5.2 Combining The Operations

These three steps ('flattening' the shed, scaling it, and translating it) can be done in separate steps, but they can be combined into one step. Let (x, y, z) be a point on our original shed diagram (Figure 7.7 (a)). To create Figure 7.7 (b), the orthographic projection ('flattening') simply moves all of the points to the xy–plane, which can be done by simply setting the z components of the points to zero. Scaling, to create Figure 7.7 (c), is done by multiplying the x and y components by their scaling factors s_x and s_y. In this case, $s_x = 1/6$ and $s_y = 1/4$. The final step, the translation to create Figure 7.7 (d), is accomplished with addition (or subtraction) of translation amounts t_x and t_y. Here, we want to slide along the x–axis to the right by 49 feet but leave all of the y components unchanged. Thus, our translation amounts are $t_x = 49$ and $t_y = 0$. Combined, we produce formulae we can use to transform every (x, y, z) point that we used to describe the shed into points (x', y', z') that describe the dog house's front view:

$$\begin{aligned} x' &= s_x \cdot x + t_x = (1/6)x + 49 \\ y' &= s_y \cdot y + t_y = (1/4)y \\ z' &= 0 \end{aligned}$$

For example, consider the upper front right corner of the shed diagram. It is described by the point $(12, 9, 12)$. Applying the expressions above, we find that the corresponding corner of the dog house is located at $(51, 2\,{}^1\!/_4, 0)$.

That was pretty easy ... and not a matrix in sight! Unfortunately, this matrix–free approach has limitations as the applications become more complex. First, that was the transformation of one point; we have to translate all of the points that describe the object. Second, the shed / dog house is just one object; what if we have to work with several different objects, as in an architectural drawing of an office building? Third, perhaps we need to transform several objects many times per second to achieve smooth simulation motion, as in a real–time video game.

When speed is critical, we can try to reduce the number of operations that need to be performed. One way to do that is to combine transformation operations into one or more matrices. Modern computers go a step further and off–load those matrix operations to one or more graphics processing units — GPUs — often found on graphics cards added to general–purpose computers.

To turn our projection–scaling–translation operation sequence into matrix operations, we need to start by representing a point as a matrix. We could use a row vector or a column vector. Traditionally, a column vector is used, so we'll follow that convention. Thus:

$$(x, y, z) \Longrightarrow \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Time to consider the operations. We will start with the orthographic projection. In our example, the projection retained the x and y components of each point and replaced the z components with zeroes. We can perform this with matrix multiplication and a 3×3 matrix that is almost an identity matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

A lot of work, isn't it, just to copy x and y and replace z with zero? This can't be worth the trouble! It wouldn't be worth the trouble if this were all that we needed to do . . . but it isn't.

Scaling requires multiplying x and y by s_x and s_y, respectively. We can do that with the same almost–identity matrix, by replacing the ones with s_x and s_y:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{1}{6}x \\ \frac{1}{4}y \\ 0 \end{bmatrix}$$

Notice that we are performing both the 'flattening' and the scaling with this one matrix — two transformations in one! This works because the product of the two matrices is just this scaling matrix:

$$\begin{bmatrix} \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The final operation, the translation, cannot be done by modifying this combined transformation matrix. But, we can do with matrix addition:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ 0 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} 49 \\ 0 \\ 0 \end{bmatrix}$$

Working out these matrix operations will produce the expressions for x', y', and z' from earlier: $x' = s_x \cdot x + t_x = (1/6)x + 49$, $y' = s_y \cdot y + t_y = (1/4)y$, and $z' = 0$.

Needing to perform the translation with a matrix addition is unsatisfying, after having been able to combine the 'flattening' with the scaling into a single matrix. It is possible to combine all three operations into a single matrix multiplication, but not with 3×3 matrices — 4×4 matrices, combined with *homogeneous coordinates*, are necessary. If you'd like to know how homogeneous coordinates work, consult an introductory computer graphics textbook. With the matrix background you now possess, homogeneous coordinates, not to mention other types of transformations, will be easier to understand.

7.6 Logical Matrices

Some of the matrices that we have used in this chapter, identity matrices in particular, contain only two values: Zero and one. Does having just two values to work with make you think about false, true, and logical operations? Whether it did or not, that's where this section will take us.

Logical matrices, a.k.a. "$(0, 1)$–matrices", are so named because their content is drawn from the domain consisting of just the two symbols '0' and '1'. We do not interpret the symbols as integers; instead, we treat '0' as 'false' and '1' as 'true'. Such matrices have many applications. We will present one here, along with three operations that are reminiscent of matrix operations that we introduced earlier in this chapter. Related applications will appear in a later chapter.

7.6.1 Using Logical Matrices to Represent Subsets of Cartesian Products

Quick review: A Cartesian Product of two sets A and B is the set of all possible ordered pairs (a, b) of the elements of A and B.

Example 144:

Let $A = B = \{a, b\}$. Then $A \times B = \{(a, a), (a, b), (b, a), (b, b)\}$.

We know that Cartesian Products are sets, which is why we have been representing Cartesian Products with set notation. Logical matrices are another possible representation. Location M_{ab} in the matrix is set to 1 when the ordered pair (a, b) is an element of $A \times B$.

Example 145:

Continuing from Example 144, because $A \times B$ contains all possible ordered pairs, every location of M contains a 1:

$$M = \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \end{array}$$

Each '1' tells us that a particular ordered pair exists in the set. For example, the lower–left '1' means that the ordered pair (b, a) is in the set.

Now consider two subsets of $A \times B$. Let $S = \{(a, b), (b, b)\}$ and $T = \{(a, b), (b, a)\}$. Represented as logical matrices with the same labels:

$$S = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \qquad T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

7.6.2 'Meet' and 'Join'

In Section 7.3.1 we introduced matrix addition and matrix subtraction, which applied addition and subtraction to corresponding pairs of matrix elements. Similarly, with logical matrices, we can apply logical AND ('$\wedge$') and inclusive OR ('$\vee$') to pairs of same-sized $(0, 1)$–matrices. The names of these operations are 'meet' and 'join', respectively.[17]

[17] Don't blame us for these names! We didn't come up with them. 'Meet' and 'join' are also used as names for related but more complex operations on lattices, but we have yet to learn whom to blame for the names.

'meet'

Definition 49: 'Meet'

The 'meet' of two logical $n \times m$ matrices A and B is the $n \times m$ matrix C such that $c_{ij} = a_{ij} \wedge b_{ij}$.

'join'

Definition 50: 'Join'

The 'join' of two logical $n \times m$ matrices A and B is the $n \times m$ matrix C such that $c_{ij} = a_{ij} \vee b_{ij}$.

Performing 'meet' and 'join' is no more complex than is performing matrix addition. What can be complex is remembering which operation performs ANDing and which ORing. Here's the memory aid we use: The words 'or' and 'join' both contain the letter 'o'.[18]

Believe it or not, 'meet' and 'join' do have practical value, as the next example demonstrates.

Example 146:

Continuing from Example 145, the 'meet' and 'join' of the logical matrices S and T are:

$$S \text{ 'meet' } T = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \text{ 'meet' } \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$S \text{ 'join' } T = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \text{ 'join' } \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

The corresponding set representations of these results are $\{(a, b)\}$ and $\{(a, b), (b, a), (b, b)\}$. So what? Consider this: The intersection of the sets S and T from Example 145 is the set $\{(a, b)\}$ and their union is the set $\{(a, b), (b, a), (b, b)\}$. That is, $S \cap T$ on subsets of Cartesian Products, and S 'meet' T on logical matrices representing the same subsets, are actually

[18] Yeah, we know; it's not a great aid, but it works for us.

performing the same operation. The same is true of $S \cup T$ and S 'join' T.

With a little reflection, this shouldn't be very surprising — we already know about the connections between logical operators and set operators from the previous chapter. We will learn much more about subsets of Cartesian Products when we cover relations in Chapter 8.

7.6.3 Logical Matrix Product

Do you remember matrix multiplication from Section 7.3.3? We hope so, because if you do, *logical matrix products* (a.k.a. *boolean products*) will be a lot easier to understand! Their definitions are very similar; we merely replace additions with inclusive ORs and multiplications with logical ANDs, and introduce new symbols $\odot$ (LaTeX: `\odot`) and $\bigvee$ (LaTeX: `\bigvee`).

logical matrix product

Definition 51: Logical Matrix Product

The logical matrix product of an $m \times n$ logical matrix A and an $n \times o$ logical matrix B is an $m \times o$ logical matrix $C = A \odot B$ in which $c_{ij} = \bigvee_{k=1}^{n} (a_{ik} \wedge b_{kj})$. (Also known as *Boolean Product*.)

The $\bigvee$ symbol tells us that we are to OR together all of the conjunctions we produce for each application of the expression. That is, just as $\sum$ tells us to add up all of the terms of the given sequence, $\bigvee$ says to OR up all of the terms of the given sequence. For a logical matrix product, we have a sequence of conjunctions to OR together.

Example 147:

Let L be a 3×3 logical matrix and M a 3×2 logical matrix. Before computing the result of $N = L \odot M$, let's start by stacking the matrices

as we have done for matrix multiplication:

$$M = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \square & \square \\ \square & \square \\ \square & \square \end{bmatrix} = L \odot M = N$$

Consider n_{11}. If we were computing $L \cdot M$, the expression would be $n_{11} = (0 \cdot 1) + (1 \cdot 1) + (0 \cdot 0)$. To produce $L \odot M$ instead, we replace the additions with inclusive ORs and the multiplications with ANDs:

$$\begin{aligned} n_{11} &= (l_{11} \wedge m_{11}) \vee (l_{12} \wedge m_{21}) \vee (l_{13} \wedge m_{31}) \\ &= (0 \wedge 1) \vee (1 \wedge 1) \vee (0 \wedge 0) \\ &= 0 \vee 1 \vee 0 \\ &= 1 \end{aligned}$$

Because most people are less comfortable with logical operators than they are with arithmetic operators, silly mistakes are common when computing these products manually. Please compute the rest of N's content on your own. If you list the resulting values row by row, pretend that the six values are bits of a binary value, and convert it to octal, you should get 53_8.[19] For example, following this conversion process, L becomes $010110101_2 = 265_8$.

Example 148:

Finally, it is time to complete Example 134! At the end of that example, we strongly hinted that matrix multiplication was the way to compute the D^2 matrix from which we computed the two–step column vector, and it is. Thanks to what we have covered between these examples, we now know that "D^2" isn't just a name; it represents the second power of D, the matrix product $D \cdot D$.

[19] Forgotten how to convert from Base 2 to Base 8? See Section A.11 in Appendix A. Yeah, we could just give you the answer, but would you work it out for yourself if we did? That's what we thought; see Figure 7.8.

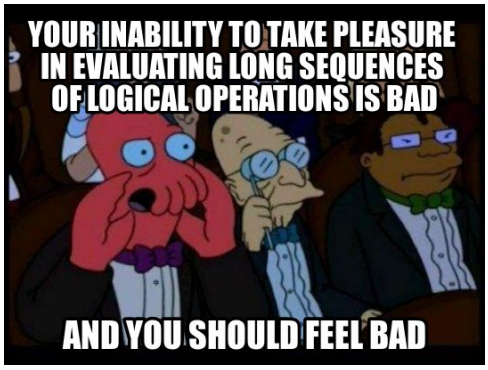


Figure 7.8: Honestly, if you enjoyed it, we'd be concerned. Credits: "Futurama" episode 4ACV18 ("The Devil's Hands Are Idle Playthings"); https://memedad.com.

A deeper question: How does that matrix multiplication produce the two-step results? A follow–up question: Can we use a logical matrix product to compute the same result?

We will answer both questions in this example. But first, for convenience, here are D and D^2, but with the actual (lower-case) Greek letters instead of their English names, to save space and to help you learn to recognize them:

$$D = \begin{array}{c} \\ \alpha \\ \beta \\ \gamma \\ \delta \end{array} \begin{array}{c} \begin{array}{cccc} \alpha & \beta & \gamma & \delta \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \qquad D^2 = \begin{array}{c} \\ \alpha \\ \beta \\ \gamma \\ \delta \end{array} \begin{array}{c} \begin{array}{cccc} \alpha & \beta & \gamma & \delta \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Consider the 1 at row α and column β of D^2, and recall that it represents the transitive result that team Alpha (α) could be considered to be 'better' than team Beta (β) because, looking at D, Alpha beat Gamma and Gamma beat Beta.

In Example 134, that 1 value in D^2 was discovered inspecting the content of D. We want to learn why it can also be produced by matrix multiplication. For clarity, here is the expanded expression for the location row α column β of D^2, and its evaluation:

$$\begin{aligned} d^2_{\alpha\beta} &= (d_{\alpha\alpha} \cdot d_{\alpha\beta}) + (d_{\alpha\beta} \cdot d_{\beta\beta}) + (d_{\alpha\gamma} \cdot d_{\gamma\beta}) + (d_{\alpha\delta} \cdot d_{\delta\beta}) \\ &= (0 \cdot 0) + (0 \cdot 0) + (1 \cdot 1) + (1 \cdot 0) \\ &= 0 + 0 + 1 + 0 \\ &= 1 \end{aligned}$$

Notice which of the four products produced the 1: $d_{\alpha\gamma} \cdot d_{\gamma\beta}$. Computing these four products requires us to examine all four ways of 'connecting' Alpha to Beta through the four teams. Looked at another way, each value d_{ij} represents a win (1) or a loss (0) when i played j. The only way that $d^2_{\alpha\beta}$ can be 1 is for both $d_{\alpha x}$ and $d_{x\beta}$ to be 1 for at least one team x. In this case, $x = \gamma$. Because the matrix product is examining all ways that Alpha could be transitively (a.k.a. two–steps) better than Beta, if a way exists, the matrix product will find it. Here there was one way: Alpha beat Gamma, and Gamma beat Beta.

The complete matrix power D^2 contains the two–step results for all possible pairings of teams. As we know, each result is due to four products, but some of those products will always be zero in this situation. For example, consider the term $d_{\alpha\alpha} \cdot d_{\alpha\beta}$ from above. No intrasquad games (that is, a game in which a team plays itself) will count in the standings, meaning that $d_{\alpha\alpha}$ will always be zero. In the full four-product expression, only the last two terms could possibly be non–zero, because there are only two teams to consider other than Alpha and Beta.

This has an important implication: Depending on the game outcomes, with four teams playing each other at most once, it is possible for $d^2_{\alpha\beta}$ to evaluate to zero, one, or two. Using the simple "one–step plus two–step" tie-breaking approach shown in Example 134, the two–step results could contribute a significant amount to the sum, perhaps more than you feel the two–step data is worth as a way to distinguish teams. One way to limit the two–step contribution is to cap the sum; for example, if the two–step amount is greater than one, you could just add one.

7.6.4 Logical Matrix Powers

Because the definitions of matrix multiplication and logical matrix multiplication are so similar in construction, the fact that we have a concept known

as *logical matrix powers* (a.k.a. *boolean powers*) that corresponds to matrix powers should not be much of a surprise.

Definition 52: Logical Matrix Powers *logical matrix powers*

The n^{th} logical matrix power of an $m \times m$ matrix L, denoted $L^{[n]}$, is the $m \times m$ logical matrix resulting from $n-1$ successive logical matrix products. (Also known as *Boolean Powers*.)

Notice that we add square brackets around the exponent, to distinguish the notations for matrix powers from logical matrix powers. Another detail that we need to mention, for completeness: $L^{[0]} = I$.

Example 149:

At the end of Example 148, we suggested capping the magnitude of the two–step results as a way to keep them from contributing too much to the "one–step plus two–step" sum. Computing a logical matrix power instead of a matrix power will do this automatically.

Let's extend the game example by another set of games: Alpha continues its domination of the league by defeating Beta, and Gamma defeats Delta to keep Delta winless. D is now:

$$D = \begin{array}{c} \\ \alpha \\ \beta \\ \gamma \\ \delta \end{array} \begin{array}{c} \begin{array}{cccc} \alpha & \beta & \gamma & \delta \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

So that we can compare them, here are D^2 and $D^{[2]}$. The difference should be easy to identify:

$$D^2 = \begin{array}{c} \\ \alpha \\ \beta \\ \gamma \\ \delta \end{array} \begin{array}{c} \begin{array}{cccc} \alpha & \beta & \gamma & \delta \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \qquad D^{[2]} = \begin{array}{c} \\ \alpha \\ \beta \\ \gamma \\ \delta \end{array} \begin{array}{c} \begin{array}{cccc} \alpha & \beta & \gamma & \delta \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

D^2 reveals that Alpha is transitively 'better' than Delta in two ways: Alpha defeating Beta and Beta defeating Delta is the first, and Alpha defeating Gamma and Gamma defeating Delta is the second. Because D^2 is computed with addition, we get a sum of two:

$$\begin{aligned} d^2_{\alpha\delta} &= (d_{\alpha\alpha} \cdot d_{\alpha\delta}) + (d_{\alpha\beta} \cdot d_{\beta\delta}) + (d_{\alpha\gamma} \cdot d_{\gamma\delta}) + (d_{\alpha\delta} \cdot d_{\delta\delta}) \\ &= (0 \cdot 1) + (1 \cdot 1) + (1 \cdot 1) + (1 \cdot 0) \\ &= 0 + 1 + 1 + 0 \\ &= 2 \end{aligned}$$

The content of a logical matrix power is naturally capped at one because, now matter how many times we inclusively-OR with 'true', the result is nothing more than 'true':[20]

$$\begin{aligned} d^{[2]}_{\alpha\delta} &= (d_{\alpha\alpha} \wedge d_{\alpha\delta}) \vee (d_{\alpha\beta} \wedge d_{\beta\delta}) \vee (d_{\alpha\gamma} \wedge d_{\gamma\delta}) \vee (d_{\alpha\delta} \wedge d_{\delta\delta}) \\ &= (0 \wedge 1) \vee (1 \wedge 1) \vee (1 \wedge 1) \vee (1 \wedge 0) \\ &= 0 \vee 1 \vee 1 \vee 0 \\ &= 1 \end{aligned}$$

In Example 149, you may have noticed that, with this extra pair of games in D, there are no ties — each team has a unique number of victories — making the computation of the two–step wins unnecessary . . . if all we needed them for was to break ties. Looking beyond ties, ranking teams and estimating

[20] Logically, nothing is more true than 'true', but one might look beyond logic: "Fairy tales are more than true: not because they tell us that dragons exist, but because they tell us that dragons can be beaten." Neil Gaiman, in *Coraline*, attributed the quote to G. K. Chesterton's *Tremendous Trifles*, but Chesterton never wrote it.

differences in measures of skill in various sports are important for reasons such as tournament seeding and sports wagering. Some of the ranking schemes created for these purposes include transitive results as contributing data.

