Embedding Cycles within Adjacency Matrices to Represent Rational Generating Functions

Date: 5/4/2021
Author: Yonah Berwaldt
Email: yberwaldt01@gmail.com
Dedicated to Peter Herreshoff for his outstanding teaching of high school math.
Special thanks to: George Andrews, Ariel Berwaldt, Elie Feder, and Ken Ono
MSC Subject Classification 2020: Primary 15B99; Secondary 15-04, 40-08.

Keywords: algorithm, adjacency matrix, graph, integer partition number, logarithmic runtime, memoization, Molien series, rational generating function

Abstract

This paper explores populating adjacency matrices with connected cycles whose final outputs represent the coefficients of rational generating functions (RGFs). An RGF takes the form of: $p(x)/q(x) + r(x)$. The denominator, $q(x)$, takes the form of: Constant $\cdot(1-c_1x^{x_1})(1-c_2x^{x_2})...(1-c_nx^{x_n})$ where the c_i are complex numbers and where factors can possibly have multiplicities greater than one. It is well known that a closed form solution exists for computing coefficients of RGFs. Also, one can write the linear recurrence relation associated with every RGF into a matrix format. Using matrices, one can compute coefficients for an RGF, such as Molien series for finite groups, in logarithmic time.

What has not yet been shown (or is not yet commonly discussed) is that one can conceptualize an RGF as a system of connected cycles within an overarching adjacency matrix. For example, a single cycle of length two would have vertex A connect to vertex B which itself connects back to vertex A with a directed arrow of weight c_i. In this conceptualization, each coefficient of an RGF can be reproduced by taking a suitable adjacency matrix to an integer power. Nothing essential is lost by taking this perspective. Due to the self-similar nature of the matrix, we devise an algorithm that can calculate coefficients of RGFs in constant time. Using memoization, a technique for caching intermediate results, calculating coefficients of RGFs can also be done in logarithmic time.

One observation is that, depending on the situation (i.e. what $q(x)$ is), there may be a computational benefit to taking the cyclical perspective. For example, for certain $q(x)$, the traditional matrix has cells containing positive and negative values whereas

the cyclical approach has cells containing only positive values. The computational benefit is probably irrelevant for computers; however, it may be important for restrictive systems, such as biological systems / neural networks that may have a tight operating envelope.

We make a final observation that each cyclical matrix representation can be thought of as a graph which is an epsilon away from being strongly connected. Studying the behavior of these matrices may yield insight into the behavior of a broader class of function. In essence, the study of sequences modeled by RGFs can be converted to the study of connected cyclical graphs that model the RGFs or vice versa.

1 Introduction

We will demonstrate how to create adjacency matrices composed of connected cycles whose output corresponds to the sequence produced by a general class of rational generating functions.[16] Let $R(x) = P(x)/q(x)$ where $P(x)$ and $q(x)$ are polynomials in x and where the coefficients of $R(x)$ are complex numbers. If the numerator $P(x)$ is of equal or higher degree than the denominator $q(x)$, we can use polynomial division to rewrite it as:

$$R(x) = p(x)/q(x) + r(x) \tag{1}$$

Here $p(x)$ has degree less than $q(x)$ and $r(x)$ is the remainder. Let $p(x)$, $q(x)$, and $r(x)$ be of the following form:

$$p(x) = p_0 + p_1x + p_2x^2 + \ldots + p_nx^n \tag{2}$$

$$q(x) = (1 - c_1x^{x_1}) \ldots (1 - c_ix^{x_i}) \ldots (1 - c_jx^{x_j}) \ldots (1 - c_nx^{x_n}) \tag{3}$$

$$r(x) = r_0 + r_1x + r_2x^2 + \ldots + r_nx^n \tag{4}$$

In terms of a generating function, $r(x)$ has a one time impact insofar as it modifies the coefficients in front of the x_k terms by adding r_k to the otherwise infinite sequence produced by $p(x)/q(x)$. We thus focus the analysis on finding an algorithm to populate a matrix whose powers generate the infinite sequence produced by expanding $p(x)/q(x)$.

For the purpose of the following discussion, we assume $q(x)$ is of a form where the powers x_i and x_j can be, but do not have to be, equal. In other words, terms in the expansion can have multiplicities greater than one. Also, we assume every c_i is a real or a complex number. Without loss of generality, let $x_1 \leq x_2 \leq x_3 \ldots \leq x_n$. To produce the sequence corresponding to $p(x)/q(x)$, we expand $p(x)$ as in 2. Once we have the underlying infinite sequence produced by the denominator, we can shift it by x_i and multiply the result by p_i. We can do this for every x_i and p_i in seriatim then sum the total. Thus to find $p(x)/q(x)$ we really only need the infinite sequence produced by the denominator because any amount of shifting and multiplying by constants is straightforward.

We touch upon a closed form solution to generating coefficients of an RGF. We also briefly describe a typical approach to casting an RGF linear recurrence relation in the context of matrices. However, we will attempt a different approach involving using an adjacency matrix to represent cyclical graphs that are connected to each other. Once we have found an adjacency matrix representation, we will find a way to encode matrix multiplication at a faster than (side length)^3 runtime via updating a graph that is almost strongly connected.[13][24] Let K_p, K_q, and K_r be the degrees

of $p(x)$, $q(x)$, and $r(x)$ respectively. The algorithm finds the coefficients in front of x^N in a runtime of $\Omega(K_p K_q N + K_r)$. We will also make use of a technique from dynamic programming, memoization, to find coefficients in front of x^N in a runtime of $\Omega((K_p K_q^3) \log_2(N) + K_r)$.[19] Due to the nature of the encoding, one can transfer from the matrix to the graph form without loss of information but one cannot do the reverse. Depending on K_p, K_q and N one method may be superior to the other (or an entirely different method from the literature, such as an exact solution, may be preferable.)

This paper contains examples of matrices that represent various RGFs, including a representation for integer partition numbers restricted to four parts. Also, an example of memoization for the alternating group A_4 is included.

The paper ends with several open questions about using the adjacency matrix representation to study the underlying RGFs.

2 Expressing an Arbitrary Polynomial in the form of 3

The Fundamental Theorem of Algebra states that every non-zero, single-variable, degree n polynomial $a_n x^n + a_{n-1} x^{n-1} + ... + a_0$ with complex coefficients has, counted with multiplicity, exactly n complex roots.[15] We will show this polynomial can be expressed in the form of 3.

Let the roots of the arbitrary polynomial be $t_1, t_2, \ldots t_n$. Then for the factor $(1 - c_i x^{x_i})$ of $q(x)$ we can choose $c_i = 1/t_i$ and the power $x_i = 1$. When $x = t_i$ we have $(1 - c_i x^{x_i}) = 0$ and thus $q(x)$ also equals 0 and thus t_i is a root of $q(x)$. We can multiply $q(x)$ by the constant a_0 to bring the constant term of $q(x)$, when written in its expanded form rather than its factored form, into alignment with the constant term of the arbitrary polynomial. This means $a_0 q(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$. Finally, the factor of a_0 in the expression $a_0 q(x)$ can be moved into the numerator of 1 as $p(x)/a_0$ where the form of $p(x)$ does not matter. The important thing is that $q(x)$ has the form of 3, and after this processing it does. Once $q(x)$ has the form of (3), we can proceed with constructing a matrix representation for finding the coefficient in front of x^N in 1.

Incidentally, there is no theoretical reason why c_i should be restricted to an element of the complex numbers. The c_i could be a function of x, such as $c_i(x) = e^x$. However, the technique discussed in this paper does not apply in a straightforward way to such $c_i(x)$, in particular if the $c_i(x)$ have an infinite Taylor series expansion. In case $c_i(x)$ has a finite Taylor series, then $c_i(x)$ is akin to a finite polynomial of degree N, in which case $(1 - c_i x^{x_i})$ can be re-factored in such a way that the form of $q(x)$ is still preserved. However, unless an infinite Taylor series can be factored like in 3, then the algorithm described in this paper will not immediately apply. The contribution to the coefficient in front of x^N in equation 1 would need to be found by a standard approach. Therefore, theoretically c_i could be a function of x, but in practice it appears best to restrict every c_i to a complex number.

3 Exact / Closed Form Solution for Rational Generating Functions

There is an explicit formula for finding the coefficients of RGFs. Pantone, in [11], describes a procedure for finding an exact solution. The first step is that partial fraction decomposition can be applied to $q(x)$.[21] We can then obtain a form for

$q(x)$ that looks like: $q(x) = d(x) + b_1(x–a_1)^{(-r_1)} + \cdots + b_n(x–a_n)^{(-r_n)}$ where the a_i and b_i are complex numbers and the r_i are positive integers. We can then apply Newton's Generalized Binomial Theorem to each $(x–a_i)^{(-r_i)}$.[14] We then sum the results together to obtain the final explicit result for the coefficient of x^N of $q(x)$.

Though $q(x)$ can be factored over the complex numbers into its most basic parts, many times it is easier to think of $q(x)$ in the form of 3 where the x_i are positive integers. For example, rather than break $(1 - x^4)$ into $(1 - x)(1 + x)(1 - ix)(1 + ix)$ and then apply partial fraction decomposition, we can leave it unfactored. As will soon be described, we can then build a matrix representing $q(x)$ that often has large swaths of zeros interspersed with occasional ones and the c_i.

The matrix representation brings a change in perspective that may be useful in its own right. Additionally, it is possible that certain cases occur in which computing with the matrix form may be quicker than obtaining and then using the closed form approach.

4 Traditional Method to use an RGF Linear Recurrence Relation to Populate a Matrix

The typical method to produce a matrix that represents an RGF is to make use of the expanded form of $q(x)$ by writing a linear recurrence relationship.[3] We write $q(x)$ as $1 + q_1x + q_2x^2 + \cdots + q_nx^n$. We then rewrite it in the form of a recurrence $a_n = f(a_{n-1}, ..., a_0)$ by noticing that $a_n = -(q_1a_{n-1} + q_2a_{n-2} + ... + q_na_0)$. This can be converted into a matrix. Note that the "1's" below the main diagonal transition the position occupied by a_{n-i} to that occupied by a_{n-i-1}.

$$\begin{bmatrix} -q_1 & -q_2 & \dots & -q_n \\ 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \cdot \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_1 \end{bmatrix} \tag{5}$$

Depending on the RGF, the typical method based on a linear recurrence may involve cells with negative numbers whereas the alternative method based on connected cycles may contain only positive entries. (The derivation of the second method will be discussed in detail later in this paper. Just as the traditional method follows a simple procedure, so too does the alternative method.) An example of an RGF where this situation occurs is $q(x) = (1 - x)(1 - 2x^2) = 1 - x - 2x^2 + 2x^3$ which corresponds to a recurrence of $a_n = a_{n-1} + 2a_{n-2} - 2a_{n-3}$. The corresponding typical matrix is on the left hand side whereas the alternative cycle matrix is on the right hand side. After taking the matrix to the proper power, the uppermost cell of the resultant column vector contains the value of the coefficient of the RGF.

$$\begin{bmatrix} 1 & 2 & -2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^N \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ versus } \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 2 & 0 \end{bmatrix}^{N+1} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

While mathematically both matrices produce the same output (in the uppermost cell of the resultant column vector), the matrix on the left contains a negative value. It is possible that certain use cases may arise when having a matrix with strictly positive numbers is preferable. For example, it is conceivable that physical constraints in biological systems, such as perhaps a neural network, would preferentially select for

a relationship that involves strictly positive weights. [1] For the remainder of the paper, we focus primarily on describing the adjacency matrix basod on connected cycles.

5 A New Convention for Labeling Cycles and Vertices within an Adjacency Matrix that Represents an RGF

We will soon exhibit an adjacency matrix that represents the sequence produced by an RGF. This matrix, when interpreted as a graph, is composed of multiple connected cycles and each cycle is itself composed of sequentially connected vertices. It will be helpful to be able to uniquely identify a specific cycle, to pinpoint a vertex within that particular cycle, and to identify the connections that vertex has with other vertices. We develop a new labeling convention which fulfills the above requirements without needing to make recourse to complicated subscripts which would be needed if we were using the traditional a_{ij} notation that refers to the i^{th} row and j^{th} column within a matrix.

We label the horizontal and vertical axes of matrix M as in figure 1: Matrix Labeling Convention. In order to not confuse the traditional (i,j) notation with the new convention, we insert an X in front of the "i" to signify we are referring to a cycle per the new notation. We thus read "(X_i, j)" as meaning the unique row (or column) whose primary label is X_i and whose secondary label is j. The i refers to the i^{th} cycle and the j refers to the j^{th} vertex within that cycle. The i^{th} cycle does not necessarily have a length of i. Rather, the i^{th} cycle has a length of Y_i. We begin counting from vertex 0 up through vertex $Y_i - 1$. Every cycle has a vertex labeled 0 and the length is at least 1.

Consider the unique value of c_i near the upper middle of figure 1. If we scroll our eyes verticaly upward from the cell in which c_i resides, we find the label "X_i $Y_i - 1$" i.e. (X_i, $Y_i - 1$). This translates to "cycle number i vertex number $Y_i - 1$." If instead we scroll our eyes horizontally leftward from the cell in which c_i resides, we find the label "X_i 0" i.e. (X_i, 0). This translates to "cycle number i vertex 0" where "vertex 0" means the formative vertex that "begins" cycle i. Based on the interpretation of an adjacentry matrix as a graph, we can interpret this value of c_i located at this specific place of the overarching matrix, as meaning, "Within the i^{th} cycle, whose length happens to be Y_i, the ultimate (Y_i-1) vertex has a directed arrow of weight c_i toward the formative (0^{th}) vertex."

As an additional example, consider figure 2. There is a single entry near the middle of the matrix whose value is "3." Looking upward, we find the label "3 1" i.e. (3,1). Looking leftward, we find the label "3 0" i.e. (3, 0). The interpretation of this cell is thus, "The vertex labeled 1 within the third cycle has a directed arrow of weight "3" toward the vertex labeled 0 within that selfsame cycle." The column under the header "3 0" i.e. (3,0) has three separate 1's. The interpretation of this column is thus, "The vertex labeled 0 within the third cycle has directed arrows each of weight "1" toward a) the single vertex labeled 0 in the first cycle, b) the single vertex labled 0 in the second cycle, c) the vertex labeled 1 within the third cycle."

To refer to a specific cell in a matrix, we can substitute the new notation for the a_{ij} notation. We can triangulate a position in the matrix by finding the intersection of the horizontal label (X_i, j) and the vertical label (X_k, l) We can refer to the cell

[1] The author is not a neurobiologist and thus this in pure speculation, but it illustrates the point that some real life situations might occur for which one representation may be superior to the other even though the output of the system is the same.

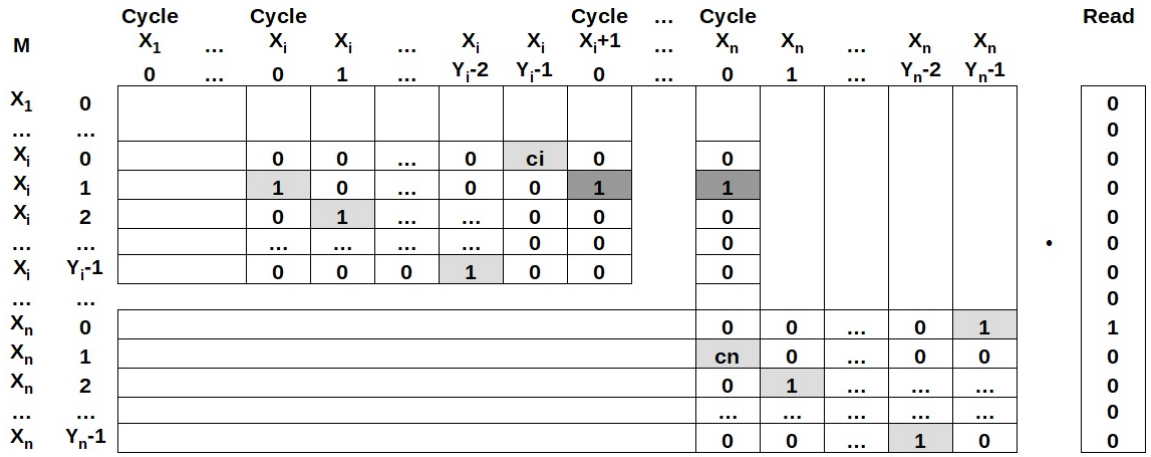

Fig. 1: Matrix Labeling Convention
Light gray refers to the cyclical components. Gray refers to the "creators" of smaller cycles. [a]

[a] There is a slight ambiguity in the labeling in that, for example, X_1 could equal X_2 if the cycles are of the same length. However, in the one situtation when this arises (see Figure 2) it is not ambiguous because the values of the cell's contents differ.

corresponding to the intersection of "row" (X_i, j) and "column" (X_k, l) without ambiguity. For example, in figure 2, we can identify the value of "2" in the sixth row and fifith column, traditionally denoted as the value of cell i = 6, j = 5 in the a_{ij} notation, with the alternative notation as the cell corresponding to row (4, 1) and column (4, 0). We sometimes write this notation as row 4 subsection 1 and column 4 subsection 0.

6 Matrix Form for Rational Generating Functions

In this section we construct a matrix that represents the sequence produced by rational generating functions. To most easily understand the construction, it is important to arrange the cycles from smallest to largest such that the x_i in 3 satisfy $x_1 \leq x_2 \leq x_3 \ldots \leq x_n$.[2] [3] We can translate between the terminology of figure 1 and the expansion of $q(x)$ by letting the length of cycle i, which is Y_i, equal x_i. Note the uppercase Y's correspond to the lower case x's.

We can interpret the adjacency matrix in terms of cycles because if we view the adjacency matrix information as a graph then vertex / node $(X_i, 0)$ links to node $(X_i, 1)$, and $(X_i, 1)$ links to $(X_i, 2)$, etc. In general, node (X_i, k) links to node $(X_i, (k+1) \bmod Y_i)$. Consequently, $(X_i, Y_i - 1)$ links back to $(X_i, 0)$ which had initiated the cycle. The cyclical feature being built into the matrix is reminiscent of the cyclical nature of the sequence produced by an RGF of the form $(1 - c_i x^{x_i})^{-1}$ with $c_i = 1$. Letting $Y_i = x_i$, we have the sequence produced by this RGF as: 1, $x_i - 1$ zeros, 1, $x_i - 1$ zeros ....

For cycles of length greater than 1, if $i < n$ and there is a constant c_i associated with cycle i, we place the c_i into the cell corresponding with row X_i subsection 0 and column X_i subsection $Y_i - 1$. If there are multiple cycles multiplying each other, we place a "1" in the cell corresponding to row X_i subsection 1 column $X_i + 1$ subsection 0. However, if $i = n$, we place c_n in the cell corresponding to row X_n subsection 1 and column X_n subsection 0.

For cycles of length 1, we also have two cases. If the cycle of length 1 corresponds to row $(X_i, 0)$ and column $(X_i, 0)$ and if $i < n$, then the cell corresponding to row $(X_i, 0)$ has a c_i and the entire column above is also populated with c_i's. In other words, every cell in column X_i subsection 0 corresponding to row X_1 subsection 0 through row X_i subsection 0 has a value of c_i. Otherwise, if $i = n$, then only the cell corresponding to row $(X_i, 0)$ has a c_i and all the cells above it in column $(X_i, 0)$ have a value of 1.

To replicate the sequence produced by $q(x)$, we require a method to choose the cell of the matrix that corresponds to tabulating the sequence of coefficients of $q(x)$. Let v_1 be a row vector of length equal to the side length of the square matrix M with a value of 1 in cell i = 1, j =1 and zeros elsewhere. Let v_2 be the Read column vector which has 0's everywhere except for a single 1 corresponding to row X_n subsection 0. Call $M^N \cdot v_2$ the N^{th} Result vector. The coefficient of x^N when $q(x)$ is expanded will be generated in the uppermost cell of the N^{th} Result vector. In other words, we have:

$$\text{coefficient of } x^N = v_1 \cdot M^N \cdot v_2 \tag{6}$$

[2] Technically, because $q(x)$ factors over the complex numbers, $q(x)$ can be broken into a series of cycles of length 1. Nonetheless, it is helpful to think of cycles of length x_i, which is the convention adopted in this paper.

[3] The inequalities are not strictly necessary because one can use any other adjacency matrix that preserves the connections.

M		1	2	3	3	4	4	4		Read Result	
		0	0	0	1	0	1	2			
1	0	1	2	1	0	1	0	0		0	0
2	0	0	2	1	0	1	0	0		0	0
3	0	0	0	0	3	0	0	0	•	0	0
3	1	0	0	1	0	1	0	0		0	0
4	0	0	0	0	0	0	0	1		0	0
4	1	0	0	0	0	2	0	0		1	0
4	2	0	0	0	0	0	1	0		0	1

Fig. 2: Matrix for $x^2((1-x)(1-2x)(1-3x^2)(1-2x^3))^{-1}$. Light gray refers to the cyclical components. Gray refers to the "creators" of smaller cycles.

The process described in 6 is equivalent to picking out the cell in the matrix corresponding to row $(1,0)$ and column $(X_n,0)$. We call this cell the tabulator of the sequence. The original sequence $q(x)$ is produced when k, the time delay, equals 0. A time delay of k is equivalent to $x^k \cdot q(x)$. A time delay of $0 < k < n$ corresponds to picking a column to the right of column $(X_n,0)$ by changing the Read column vector so that the 1 is in the row corresponding to $(X_n, n-k)$ and the rest of the vector is populated with zeros.

Figure 2 illustrates several of the features discussed in the preceding paragraphs. The sequence begins as: 0, 0, 1, 3, 10, 26, 67, 155, 362, ... Note that the two initial zeros correspond to the x^2 term in the RGF and they correspond to the Read vector picking out the sixth column (as opposed to the fifth column) in the matrix representation.

We now explore how to create the initial cycle corresponding to $(1-c_1x^{x_1})^{-1}$. If $x_1 = 1$, then the matrix corresponding to $(1-c_1x)^{-1}$ is a single c_1 along the diagonal of the matrix. The matrix $M = \begin{bmatrix} c_1 \end{bmatrix}$ to the N^{th} power is c_1^N. We have the special case that $v_1 = v_2 = \begin{bmatrix} 1 \end{bmatrix}$ and hence $v_1 \cdot M^N \cdot v_2 = c_1^N$, which is the sequence we want produced. Stepping forward 1 step at a time produces a sequence of $1, c_1, c_1^2, \ldots$ which is the sequence of coefficients in the expansion of $(1-c_1x)^{-1}$.

On the other hand, if $x_1 > 1$, then the matrix corresponds to a cycle of size 2 or more. This can be arranged by introducing a 0 in cell i=1, j=1 then building a cycle directly to the right and down. An additional 1 in cell (i=1, j=2) acts as the tabulator, which is the creator of the sequence that the matrix produces. There are other ways to build a cyclical matrix and to tabulate the produced sequence but this method aligns nicely with the proof that will follow. After the proof, we will describe a slightly modified version of the matrix that is equivalent and nicer to use but slightly harder to understand.

For example, the matrix corresponding to the generating function $(1-2x^2)^{-1}$ is shown in Figure 3. The read vector begins with a 1 corresponding horizontally to row (2,0). The sequence produced by reading the uppermost cell of the Result vector under consecutive matrix powers is 1,0,2,0,4,0,8... which is as expected.

We will soon prove the construction outlined above populates a matrix whose powers, dotted with the read vector, produce a sequence corresponding to a generic $q(x)$ in the uppermost cell of the result vector. As an illustration, Figure 4 shows the

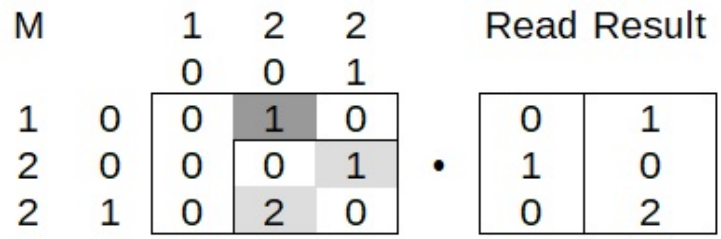

Fig. 3: Matrix for $(1-2x^2)^{-1}$. Light gray refers to the cyclical components. Gray refers to the "creators" of smaller cycles.

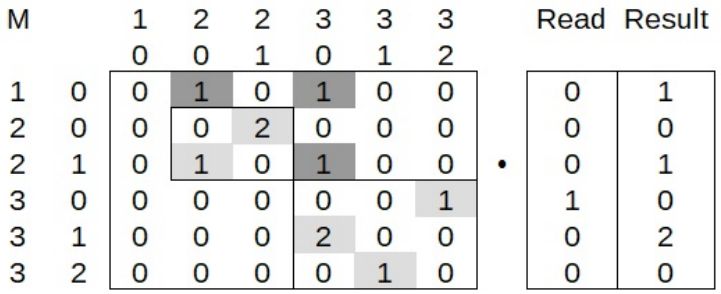

Fig. 4: Matrix for $((1-2x^2)(1-2x^3))^{-1}$ Version A. Light gray refers to the cyclical components. Gray refers to the "creators" of smaller cycles.

matrix for $(1+2x^2/(1-2x^2)) \cdot 1/(1-2x^3) = ((1-2x^2)(1-2x^3))^{-1}$ whose sequence[4] is 1, 0, 2, 2, 4, 4, 12, 8, 24, 24, 48, 48 .... Note the Read vector has a 1 corresponding horizontally to row (3,0) i.e. it picks out the fourth column. If we wanted a time delay of 1 for the entire sequence, (that is, the generating function times x) then we would have picked the sixth column, which is labeled (3,2).

7 Proving the Adjacency Matrix Form for Rational Generating Functions

We now use strong induction to prove that Figure 1 and the method by which it is populated corresponds to finding the sequence produced by $q(x)$.

Base Case: Stepping forward by one step is equivalent to multiplying by M. We have already demonstrated how to create any single cycle with a time delay less than or equal to the cycle length. We will now show how to multiply cycles together in matrix form. Let us assume that for $0 < i < n$ we have 7 and for $i = n$ we have 8. To avoid double subscripts, read Mx_i as M_{x_i} for the remainder of this paper.

$$Mx_i = c_i((1-c_1x^{x_1})(1-c_2x^{x_2})...(1-c_ix^{x_i}))^{-1} \tag{7}$$

$$Mx_n = ((1-c_1x^{x_1})(1-c_2x^{x_2})...(1-c_ix^{x_i}))^{-1} \tag{8}$$

[4] If we had chosen $((1-x^2)(1-x^3))^{-1}$ we would have had OEIS sequence A103221.[9]

We will use a suitable choice of 0's, 1's, and c_{n+1}'s to populate the matrix corresponding to Mx_{n+1} according to the construction outlined in Figure 1. We will show the construction will produce the sequence corresponding to $Mx_n \cdot (1 - c_{n+1}x^{x_{n+1}})^{-1}$.

Inductive Step: For matrix Mx_{n+1} extend and then label the horizontal and vertical axis as $(X_{n+1}, 0), (X_{n+1}, 1), \ldots (X_{n+1}, Y_{n+1} - 1)$. This labeling is the same as that used in Figure 1 and practically the same as that used in Figures 2 through 6. Next, populate 1's down the column labeled $(X_{n+1}, 0)$ in every space corresponding horizontally to the rows $(X_i, 1)$ for $0 < i \leq n$. Note that a 1 is also entered at the top of the $(X_{n+1}, 0)$ column corresponding horizontally to the row labeled $(X_1$,0). This cell acts as the new tabulator. These 1's correspond to multiplying the sub matrices Mx_i by 1 and adding them together. Note that each sub matrix has a time delay of $x_i - 1$ by virtue of the positioning of the 1's down the new column. Additionally, due to the creation of the $n+1^{st}$ cycle, each previous cycle will be delayed by an additional 1 step. Therefore, the composite time delay for each sub matrix will be x_i. By time delay, we mean the matrix cells affected by the Mx_i will not impact the tabulator for a period of x_i steps, after which point their impact will appear to pop into realization in the tabulator cell. The impact of the Mx_i will first be felt in the tabulator when $N = x_i + 1$.

We have a 1 (from the top of the $(X_{n+1}, 0)$ column that is realized now, plus all Mx_i have a composite time delay of x_i. To find out what Mx_{n+1} is, we observe the equivalence:

$$Mx_{n+1} = 1 + x^{x_1}Mx_1 + x^{x_2}Mx_2 + \cdots + x^{x_{n-1}}Mx_{n-1} + c_n x^{x_n}Mx_n + c_{n+1}x^{x_{n+1}}Mx_{n+1} \tag{9}$$

The reason Mx_{n+1} is equivalent to itself delayed x_{n+1} steps times c_{n+1} (plus the various time delayed Mx_k) is because it is a known cycle of length x_{n+1}, which consequently recapitulates itself after x_{n+1} steps. Also, based on the position of c_{n+1} in cycle $n + 1$, the recapitulation occurs with a factor of c_{n+1}. The x^{x_i} in front of each Mx_i come from the time delay discussion in the previous paragraph. In other words, the left hand side of the equation refers to the $n + 1^{st}$ cycle beginning immediately and the right hand side refers to the subsequences produced by column X_{n+1} with their appropriate time delays plus a single delayed revolution of cycle $n + 1$ which has a factor of c_{n+1} in front of it.

The c_n in front of Mx_n arises because the c_n switches positions in the matrix when the next cycle is appended as can be seen in Figure 1 by contrasting the positioning of c_i with c_n within their respective cycles. The meaning of the switching of the position of c_n is that we must arrange the matrix so that c_n impacts the tabulator on the $N + 1^{st}$ step. This is in line with the sequence produced by $(1 - c_n x^n)^{-1}$ beginning as: 1, $n-1$ zeros, c_n, ... which shows that the impact of a constant c_n associated with a cycle of length n is not realized until $N = n + 1$. Consequently, if X_n is a cycle of length two or more, the c_n switches places from corresponding to row $(X_n, 1)$ column $(X_n, 0)$ with the value of 1 sitting in row $(X_n, 0)$ column $(X_n, Y_n - 1)$. In case X_n is a cycle of length 1, once the $n+1^{st}$ cycle is appended, then the entire column above the cell corresponding to row $(X_n, 0)$ column $(X_n, 0)$, including that cell itself, gets filled with a value of c_n.

Rearranging the terms in 9 and then factoring out Mx_{n+1} from the left side leads to the equality:

$$Mx_{n+1}(1 - c_{n+1}x^{x_{n+1}}) = 1 + x^{x_1}Mx_1 + x^{x_2}Mx_2 + \cdots + x^{x_{n-1}}Mx_{n-1} + c_n x^{x_n}Mx_n \tag{10}$$

Now the right hand side telescopes after we used strong induction to replace each

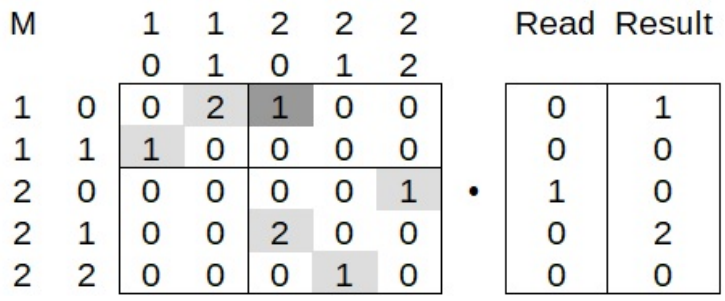

Fig. 5: Matrix for $((1-2x^2)(1-2x^3))^{-1}$ Version B. Light gray refers to the cyclical components. Gray refers to the "creators" of smaller cycles.

Mx_i with $c_i((1-c_1x^{x_1})(1-c_2x^{x_2})...(1-c_ix^{x_i}))^{-1}$ as described in 7. For example, $1+x^{x_1}Mx_1 = 1+c_1x^{x_1}/(1-c_1x^{x_1}) = (1-c_1x^{x_1})^{-1}$. Then adding the next term allows us to factor out the $(1-c_1x^{x_1})^{-1}$ from $((1-c_1x^{x_1})^{-1}+x^{x_2}Mx_2)$ to get $(1-c_1x^{x_1})^{-1}(1+c_2x^{x_2}/(1-c_2x^{x_2}))$ which equals $(1-c_1x^{x_1})^{-1}\cdot 1/(1-c_2x^{x_2})$ which equals $((1-c_1x^{x_1})(1-c_2x^{x_2}))^{-1}$.

For the X_i^{th} term, we have: $((1-c_1x^{x_1})\dots(1-c_{i-1}x^{x_{i-1}}))^{-1}+c_ix^{x_i}Mx_i$ which translates to:

$$((1-c_1x^{x_1})\dots(1-c_{i-1}x^{x_{i-1}}))^{-1}(1+c_ix^{x_i}(1-c_ix^{x_i})^{-1}) = ((1-c_1x^{x_1})\dots(1-c_ix^{x_i}))^{-1} \tag{11}$$

We now recall $c_nx^{x_n}Mx_n$, which is c_n times (8), has the same form as 7. Thus the sum can continue to telescope through term Mx_n. Eventually we have $Mx_{n+1}(1-c_{n+1}x^{x_{n+1}}) = ((1-c_1x^{x_1})(1-c_2x^{x_2})\dots(1-c_nx^{x_n}))^{-1}$. Then we move $(1-c_{n+1}x^{x_{n+1}})$ to the right hand side and we have:

$$Mx_{n+1} = ((1-c_1x^{x_1})(1-c_2x^{x_2})\dots(1-c_{n+1}x^{x_{n+1}}))^{-1} \tag{12}$$

Finally, we note that the position of c_{n+1} in the matrix is under column X_{n+1} subsection 0, which was the "spot" previously held by c_n on the previous inductive step and that 12 has the form of 8. This completes the induction.

We have thus proven that the constructed matrix as described in Figure 1 forms the sequence corresponding to $q(x)^{-1}$ in the new tabulator cell corresponding to row $(X_1, 0)$ and column $(X_{n+1}, 0)$.

8 Alternative Matrix Form for Rational Generating Functions

Let us investigate a slightly condensed version of a matrix whose least cycle is greater than 1. We can modify the algorithm for populating the matrix by removing the very first 1 in cell (1,2) shifting all other 1's in the first row downward by 1 cell, and then removing the first column and first row. The Read vector gets modified by removing the uppermost cell. Everything else remains the same. The matrix whose output corresponds to $((1-2x^2)(1-2x^3))^{-1}$ is shown as Figure 5 Version B, which can be contrasted with Figure 4, Version A.

The reason this modified version of the matrix works is because we have condensed the first two terms $(1+x^{x_1}Mx_1) = (1+c_1x^{x_1}(1-c_1x^{x_1})^{-1})$ into one term $(1-c_1x^{x_1})^{-1}$.

Only the first two rows and first column (of zeros) changed which means all cycles, their delays, and the corresponding matrix representations for $Mx_2, \ldots Mx_n$ are not affected. The reason they are not affected is because we could have removed all the rows and columns corresponding to Mx_1 and this would just have removed Mx_1, leaving Mx_2 as the new "first" cycle. Thus, the only impact of the aforementioned modifications is on Mx_1.

The first term, the +1 that was previously realized immediately in Figure 4 Version A, no longer exists on its own. Instead, of a cycle of length 1 beginning immediately and everything else time delayed, what is realized immediately in Figure 5 version B is the entire first cycle, that is, Mx_1. In the example for Version B, we can see the value of 1 in cell i = 1, j = 3 corresponds horizontally to row 1 subsection 0, which means the cycle of length 2 begins immediately. This corresponds to the generating function $1/(1-2x^2)$ and not to the time delayed expression that is found in Version A, which was $1 + 2x^2/(1-2x^2)$.

Incidentally, the generating function for Mx_1 begins with a "1" and it is then followed by a number of zeros equal to the cycle length X_1 minus one, which in this case is $2 - 1 = 1$. At that point, based on the fact that $x_1 \leq x_k$ for $1 < k$ the impact of Mx_2 through Mx_n may begin to affect the tabulator.

9 Additional Information Contained in the Adjacency Matrix

It is worthwhile to note that, when using the matrix form to compute $q(x)^{-1}$, one also simultaneously computes the sequences produced by restricting $q(x)^{-1}$ to the first k terms. If we let $q(x)$ be expressed as in 3, then $q(x)$ restricted to the first k terms, denoted $q_k(x)$, is of the form: $(1 - c_1 x^{x_1}) \ldots (1 - c_k x^{x_k})$. The matrix used to produce the sequence for $q(x)^{-1}$ also contains cells that tabulate the sequence for either $q_k(x)^{-1}$ or for $c_k q_k(x)^{-1}$. For all k less than n, the position of these tabulators correspond to row one subsection zero and column X_k subsection zero.

One way to determine if there is a c_k in front of $q_k(x)^{-1}$ or not is by inspection. If the value in the tabulator position corresponding to $q_k(x)$ within M is a 1, then we have the sequence for $q_k(x)^{-1}$. Otherwise, we have the sequence for $c_k q_k(x)^{-1}$. Equivalently, we can observe that for any q_k, the largest cycle is of length Y_k and the number in the tabulator position for q_k will be 1 if $Y_k > 1$ and will be c_k if $Y_k = 1$ by the very process of the matrix's construction.

The reason we automatically know the tabulators for all the $q_k(x)$ is because the construction of the matrix proceeded by appending cycles to an already functioning square matrix. We could just as well have removed the final Y_n rows and columns and stopped our construction at cycle n-1, or removed more rows and columns and stopped our construction at cycle k. The position of the tabulators will stay the same.

Another way to think about it is that the smaller cycles never produce larger cycles. This is codified in the adjacency matrix construction because the cells below each cyclical subsection are all zero. Thus the tabulators of the smaller cycles are never impacted by anything happening within the longer cycles.

The only possible change in the construction of the matrix as it grows is the position of c_k. The impact of the positioning of c_k is felt in the tabulator for $q_k(x)$ only as potentially c_k times the base sequence for $q_k(x)$. Therefore, the cell corresponding to row X_1 subsection zero and column X_k subsection zero tabulates either $q_k(x)$ or $c_k q_k(x)$.

Consequently, we have the useful result that any time a sequence for $q(x)^{-1}$ is

computed so too are the sequences for $q_k(x)^{-1}$.[5]

10 Node Form for Rational Generating Functions

The name, node form, is to emphasize that the adjacency matrix built using the construction of Figure 1 has a self-similar quality when represented as a graph that allows for an algorithm to perfrom the matrix multiplication at a faster than typical speed. To connect the node form representation with the matrix representation, it is sufficient to show the time update, cycles, and growth of the cycles behave identically. These characteristics are found in Section 16 Python Code in the definition "choices." Note that several of the If / Else statements are designed to exclude function calls outside of the display matrix's range. These out-of-bound calls can be thought of as adding zero and are thus unrelated to the underlying mathematics.

Time update: causality runs rightward from lower columns to higher columns and upward from higher rows to lower rows. In particular, the column to the right is one step forward in time compared to its neighbor on the left. Consequently, as we move rightward from the position corresponding to x^N, we read the coefficient in front of x^{N+1} in the expansion of $q(x)^{-1}$. Each step right mimics matrix multiplication.

Growth: The value of a cell depends on two components. The first, and easiest to understand, is the input from its own cycle. As the matrix moves to the next state, each cycle permutes its values by one step. In other words, the value stored in position k of a cycle of length Y_i moves to position $(k+1) \bmod Y_i$.

The second component involves the input from all larger cycles in the form of a running sum. Importantly, the sum of the input to X_{i+1} (cycle i+1), is also applied to X_i (cycle i), which allows for a tabulation of a running sum going from the higher (larger) cycles to the lower (smaller) cycles. This fact can be seen by conceptualizing the higher cycles as "sequence generators" and then noting that each sequence of larger length creates the sequences of smaller length. Equivalently, it can be seen in the growth columns (these are the columns of ones corresponding to cycle i+1 subsection zero) in the matrix form. Each nonzero cell of the growth component of these columns is identically 1, or identically c_{i+1}. The growth component of cycle i+1 is identical to that of cycle i except for an additional 1 (or c_{i+1}) located at row X_i subsection 1 and column X_{i+1} subsection 0. This can be seen most clearly in figure 6. It is this similarity throughout the adjacency matrix that allows for the running sum trick to work.[6]

Taking both the input from its own cycle and that of the input from the larger cycles culminates with the final value for that cell. Notably, only the values of the realized part of the sub sequences (those with zero time delay) impact the growth component of the smaller cycles.

The above points are all codified in the definition for "choices." These points can also be implemented in a calculator file using If / Else statements and the Indirect command.

[5] Since ultimately the ordering of the cycles by their length from least to greatest was unnecessary, we can arrange the cycles in any way we please and make use of the tabulators for the rearranged $q_k(x)^{-1}$.

[6] By shifting one's perspective, one could just as well view the matrix representation as smaller cycles producing larger cycles. There is symmetry in how one can interpret the representation because x_i does not necessarily need to be less than or equal to x_{i+1}. Ultimately the only thing that matters is that the connections in the adjacency matrix are preserved.

Recall equation 1: $R(x) = p(x)/q(x) + r(x)$. Let K_p equal the number of nonzero coefficients of $p(x)$, K_q equal the degree of $q(x)$ when it is written as an expanded polynomial, and K_r equal the number of nonzero coefficients in $r(x)$. Our objective is to find the coefficient in front of x^N.

The running sum is what speeds the node form calculation from regular matrix runtime of (side length)$^3 = K_q^3$ computations per step to only $2K_q$ computations. Each step forward in time involves K_q cycle computations and K_q running sum computations for a total of $2K_q$ computations. (If we do not use the running sum trick and instead sum all realized values per time step then we would have K_q cycle computations and $1 + 2 + 3 + \cdots + K_q = K_q(K_q + 1)/2$ additional sums.) There are N steps to the N^{th} coefficient, and possibly K_p shifts of the underlying sequence, and possibly a single entry from $r(x)$ per step, thus yielding a runtime for the node form of:[7]

$$\text{Node Form Runtime } = \Omega(2K_pK_qN + K_r) \tag{13}$$

11 Memoization of Matrix Powers Yields Logarithmic Runtime on Sequences Produced by RGFs

Speeding up calculations of sequences produced by rational generating functions has widespread application. For example, for a finite group, the Molien series corresponding to any finite-dimensional representation is a rational function of the formal variable.[20][2] Any situation where the denominator $q(x)$ can be written in the form of 3 where the powers x_i and x_j can be, but do not have to be, equal is thus amenable to both the matrix and node form representation as described earlier in the paper. Because we can add sequences to each other, we can also segregate the underlying matrix and node form representations for multiple $q(x)$'s and then add their outputs together.

Memoization is used to speed up computer programs by storing the results of expensive function calls and returning the cached results when the same inputs reoccur. However, if we can write the Read vector as a linear combination of eigenvectors of M, then the eigenvector method will likely triumph. Nonetheless, there are cases where the eigenvalues have multiplicities greater than one and thus the span of the eigenvectors does not cover the full space in which the Read vectors live. For these situations, memoization of matrix powers may prove useful.

Memoization allows for bootstrapping matrix powers as 2^k where k is the number of iterations that have occurred. The matrix powers produced by this method are $M^1, M^2, M^4, M^8 \ldots M^N$ with $N = 2^k$ and where the next result is the previous result times itself. This method works for generic matrices. The worst case scenario to find M^n for $n < 2^k$ occurs when every intermediate result must be used (i.e. multiplied together). This happens at the cases of $n = 2k - 1$. Because straightforward matrix multiplication[8] of square matrices has a runtime of (L = side length)^3, we find using the memoization technique that approximately $\log_2(n)$ computations each of which

[7] The included Python program has a limitation in that there are extra zeros padding the sequence output. The number of zeros may be as long as the length of all the cycles added together, which is K_q. In such cases the runtime is $\Omega(2K_p(K_q(K_q+N)+K_r)$. It is likely that an optimized version of the program could reach maximum efficiency at the runtime shown in 13.

[8] There are matrix multiplication algorithms with better asymptomatics than (side length)^3 but here we assume the unvarnished straightforward method.

has L^3 sub computations must be performed for a total computational complexity of $L^3 \cdot \log_2(n)$ calculations.[18]

The runtime using the memoized matrix method becomes:

$$\text{Matrix Form Runtime } = \Omega(K_p K_q^3 \cdot \log_2 N + K_r) \tag{14}$$

We first analyze the component of the expression that is independent of N. To determine if $r(x)$ contributes to the coefficient in front of N, it is sufficient to check if there is a nonzero entry in $r(x)$ in front of the position corresponding to x^N. This takes one unit of time, but is repeated K_r times.

Next, assume for the moment that the runtime of the memoized denominator $q(x)$ is known. Then to find $p(x)/q(x)$ one needs to add the underlying sequence $q(x)$ multiple times after shifting appropriately. The number of times this must be done corresponds exactly to the number of nonzero coefficients of $p(x)$. In other words, we must find $q(x)$ in K_p places. This takes K_p units of time multiplied by however long it takes to find $q(x)$.

Finally, let us examine the contribution of $q(x)$ to the runtime. The side length of the matrix is the sum of the powers of the polynomials composing $q(x)$. For example, with $q(x) = (1 - 2x^2)(1 - 2x^3)$ as in Figure 5, the side length of the matrix is 2+3 = 5. Another way of expressing this is by finding the degree of the polynomial $q(x)$ when written as an expansion in x, and this is K_q.

We can decompose every Read / seed vector as a linear combination of its basis vectors, which are the traditional basis vectors $[1, 0, ...0]$, $[0, 1, 0, ...0]$, $\ldots$ $[0, 0, ...0, 1]$, and where the total length of the vector is K_q. Each of these basis vectors produces a new read vector under multiplication by the matrix. We can then rebuild the original vector's output by adding together the outputs of the original linear combination of basis vectors. Each rebuilding cycle is equivalent to matrix multiplication. As discussed before, the runtime for matrix multiplication is (side length)^3 which in this case results in K_q^3 computations.

Thus, the general procedure of memoization allows for K_q^3 calculations per time step which allows for reproducing the sequence for $q(x)^{-1}$. Putting this together to find $p(x)/q(x) + r(x)$ yields the result in 14.

At any point in time we can transfer from the matrix multiplication algorithm to the node form algorithm. However, there is a loss of information when converting to the node form algorithm because we lose track of what happens to every particular basis vector and only see what happens to the ensemble. A specific use case of when changing perspective might be useful would be when calculating outward to find the coefficient in front of x^N where $N = 2^n + 1$ and where $0 < n$. In this case, we perform the matrix bootstrap method n times to get to M^{N-1} and then switch to the node form algorithm for the final step. Depending on K_q, N, and how many coefficients we want to find, and how close they are to each other, there may be situations where the bootstrapping matrix method can be applied first and then a series of updates interleaving the node form method can be applied.[9]

Memoization is not always the best option for finding the coefficient in front of x^N. For example, if K_q is large relative to N, then 13 is faster than 14. The best situations for applying memoization is when N is very large relative to K_q.

[9] It seems reasonable that if one needs to simultaneously calculate the restricted $q_k(x)^{-1}$ then the matrix method will be superior to the explicit solution for RGFs. However, it is not resolved in this paper if the matrix or node form approaches may be faster in certain cases than explicit calculation of multiple coefficients using the closed form solution for RGFs.

M		1	2	2	3	3	3	4	4	4	4
		0	0	1	0	1	2	0	1	2	3
1	0	1	1	0	1	0	0	1	0	0	0
2	0	0	0	1	0	0	0	0	0	0	0
2	1	0	1	0	1	0	0	1	0	0	0
3	0	0	0	0	0	0	1	0	0	0	0
3	1	0	0	0	1	0	0	1	0	0	0
3	2	0	0	0	0	1	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	1
4	1	0	0	0	0	0	0	1	0	0	0
4	2	0	0	0	0	0	0	0	1	0	0
4	3	0	0	0	0	0	0	0	0	1	0

Fig. 6: Restricted Integer Partition, Matrix Form $P_4(N)$. Light gray refers to the cyclical components. Gray refers to the "creators" of smaller cycles.

12 Matrix Representation Example for Restricted Integer Partitions, $P_k(N)$

We will now associate the generating functions for restricted integer partitions, $P_k(N)$, to the matrix representation.[22] $P_k(N)$ refers to partitions of N which have elements less than or equal to k. For example, the partitions of 4 are: $4 = \{1,1,1,1\}$, $\{2,1,1\}$, $\{3,1\}$, $\{4\}$, $\{2,2\}$. Restricting to k = 3 eliminates 4 from the enumeration, so $P_3(4)$ = 4.

The matrix in Figure 6 is the initially seeded adjacency matrix needed to find the integer partitions restricted to four parts, $P_4(N)$. The generating function is $q(x)^{-1} = ((1-x)(1-x^2)(1-x^3)(1-x^4))^{-1}$. As each additional power of N is taken, the corresponding result can be read by multiplying $M^N \cdot v_2$ where v_2 is the Read vector and then reading the uppermost entry of the resultant column vector. It is interesting to note that the side length of the matrix is $k(k+1)/2$. In this case, with k = 4, the side length is 10.

The adjacency matrix can be conceptualized as the top row on the traditional x-axis producing nodes on the traditional y-axis. Thus, (2,0) produces a (1,0) and a (2,1).

Per the general method for constructing a matrix, we can exhibit an M corresponding to generating a sequence produced by $((1-x)(1-x^2)\dots(1-x^k))^{-1}$, which is the RGF for $P_k(N)$. Incidentally, this is equivalent to having a matrix M with unlimited size (that is, letting lim $k \to \infty$) but whose read vector has only a single 1 corresponding to $X_k = k$ subsection 0, and zeros elsewhere.

If we let $q(x)$ equal $(1-x)(1-x^2)(1-x^3)\dots$ then we can appreciate the connection between $P_k(N)$ and $q_k(x)^{-1}$ because, for all $i < k$, the matrix used to tabulate the sequence for $P_k(N)$ automatically includes the tabulators for $P_i(N)$. These tabulators correspond to row $(1,0)$ and columns $(X_i,0)$.

Hence, if we calculate $P_N(N)$ (that is, the N^{th} partition number) using the matrix

method we also derive all $P_k(N)$ simultaneously for $k < N$.

13 Example of Memoization Using the Molien Series for the Alternating Group A_4

The technique of memoization will be applied to find the coefficient in front of x^{10} for the Molien series for the alternating group A_4. This will show in practice how the steps are applied. However, in principle, the technique would only be superior to brute force expansion of the underlying rational function $p(x)/q(x)$ when looking for the N^{th} coefficient when N is large. (In case only a single coefficient needs to be calculated for a large N, applying the exact solution may be fastest.)

In order to preserve space in this paper and yet not jeopardize showing the technique in action, let us assume the outputs for the basis vectors are already known for N = 1, 2, 4, and 8 steps forward.

We commence by taking the generating function for A_4 as a given, which is $(1 + x^6) \cdot ((1-x)(1-x^2)(1-x^3)(1-x^4))^{-1}$, and placing the $q(x)$ portion into matrix form such as in Figure 6. The read vector is a single 1 in front of the row labeled (4,0) corresponding to the largest cycle, which is of length 4, and which creates the smaller cycles.

The cycles that have been created by this initial basis vector after eight steps are shown in Figure 7 under $N = 8$. The expression P(n) refers to the number of realized values "produced" at time n. We then pull the cached results corresponding to the new set of basis vectors requiring analysis. The cached results correspond to 10 – 8 = 2 steps of production; they are located in Figure 8. The entries under $N = 8$ form the linear combination of basis vectors that need updating. We take the middle matrix and multiply the linear combination by it to form the Resultant. In other words, Basis Vector Production · Linear Combination = Resultant. The Resultant corresponds to $q(x)$ at $N = 10$. Since $p(x)$ has two non zero coefficients, we now add the sequence for $q(x)$ to itself shifted by 6 steps. This final step translates to taking the Resultant and adding what the sequence looks like at step 10 – 6 = 4. Serendipitously, this is the column under $N = 4$. These two columns added together produce the final column A_4 at $N = 10$. Finally, we sum the realized values corresponding to X_i subsection 0 which is +17 (from 1,0), +2 (from 2,0), and +2 (from 3,0) for a total of 21. (Recall that the unrealized values X_i subsection $j > 0$ correspond to the coefficient in front of x^{10+X_i-j} and thus we do not count them now.)

For the purpose of verification, the Molien series for A_4 begins: 1, 1, 2, 3, 5, 6, 10, 12, 17, 21 and the final entry of 21 is the 10th coefficient.[8]

The memoized runtime for finding the coefficient in front of x^N for A_4 can be found by using the runtime analysis equation with $K_p = 2, K_q = 1 + 2 + 3 + 4 = 10$, and $K_r = 0$. Then the memoized runtime is $\Omega(K_p K_q^3 \cdot log_2 N + K_r) = 2000 \cdot log_2 N$. Clearly this method is only superior to constant time algorithms when N is very large.

Incidentally, in the process of finding the coefficient of x^{10} for A_4, we simultaneously solved for $P_k(N)$ with $k = 4$ and $N = 10$. We have from the Resultant column that $P_4(10)$ equals +15 (from 1,0), +2 (from 2,0), and +1 (from 3,0) which yields a total of 18.

Memoization for Alternating Group A4

N =	1	2	4	8
P(N) =	1	1	3	11
1,0	0	1	2	9
2,0	0	0	0	1
2,1	0	1	1	3
3,0	0	0	1	1
3,1	0	1	0	1
3,2	0	0	0	0
4,0	1	0	0	0
4,1	0	1	0	0
4,2	0	0	0	0
4,3	0	0	1	1

Fig. 7: Memoization Procedure to Find Coefficient of x^{10} for A_4 Part 1.

Basis Vector Production in Two Steps						Linear Combination	Resultant q(N=10)	A_4 at N = 10
(1,0)	(2,0)	(2,1)	(3,0)	(3,1)	(4,3)			P(N)=21
1	1	1	1	0	1	9	15	17
0	1	0	1	0	0	1	2	2
0	0	1	0	0	1	3	4	5
0	0	0	0	1	0	1	1	2
0	0	0	0	0	1	1	1	1
0	0	0	1	0	0	1	1	1
0	0	0	0	0	0		0	0
0	0	0	0	0	1		1	1
0	0	0	0	0	0		0	0
0	0	0	0	0	0		0	1

Fig. 8: Memoization Procedure to Find Coefficient of x^{10} for A_4 Part 2.

14 Additional Avenues of Analysis / Open Questions

14.1 Runtime for Matrices Akin to those that Model RGFs

Let M be a matrix modeling an RGF. That is, M models $q(x)^{-1}$. The node form method whose runtime is described by 13 may have application to a broader class of matrices than M if it can be shown that the matrix multiplication operation for this broader class of matrices is equivalent to tabulating a running sum. Recall that tabulating a running sum was the main requirement for the runtime of 13 because it reduced the number of computations for each time step to $2K_q$ because K_q came from the cycle component and K_q came from the running sum component.

Of course, the memoization technique with a runtime of 14 will also work for any matrix because one can calculate M^K with $K = 2^N$ in only N steps by iteratively multiplying by the previous result.

14.2 Fast Converging Series for RGFs

A fast converging series for $P_N(N)$, whose generating function has every $(1 - x^n)$ appear with multiplicity one in the denominator, was discovered by Hans Rademacher.[6] There may be such formula for rational generating functions with multiplicities different than one. Another example would be to take every even (or odd) n in the denominator up till $n = N$. A possible avenue of attack would be through analyzing the underlying matrix representation.

Several methods of analysis are possible. Eigenvector analysis of matrices representing RGFs may reveal new proofs of old concepts or new insights into additional avenues of exploration. Also, it may be interesting to explore the Jordan normal form for these matrices in case this perspective proves useful.[17]

Additionally, something like the Perron-Frobenius Theorem for non-negative real square matrices may apply.[23] The theorem already has application to the study of partition numbers, which suggests a possible link to these types of matrices.[10] Unfortunately, the adjacency matrix looks like it is not strongly connected (because smaller cycles can never impact the larger cycles) and thus the requirements for applying the theorem may be invalid. Nonetheless, to make the graph strongly connected, one would only need to add a single positive entry connecting the smallest cycle with the largest cycle. This can be done by placing a positive entry of size epsilon in the bottom left corner of the matrix. Perhaps studying the limiting behavior of such a matrix will prove fruitful.

15 Conclusion

This paper has demonstrated how to construct a matrix such that, as successive matrix powers are calculated, one of its internal cells mimics the sequence produced by a rational generating function. The format of the adjacency matrix is a sequence of connected cycles. For certain RGFs, the denominator $q(x)$ can be represented more easily, from a computational standpoint, as a matrix of connected cycles than as a traditional matrix representing a linear recurrence relation. Because the adjacency matrix has a self-similar quality, it lends itself to being thought of as an array of nodes upon which an algorithm with a runtime of 13 can operate. Even when the matrix is thought of specifically as a matrix, memoization allows for a runtime of

14. Therefore, nothing essential is lost by taking this alternative perspective that emphasizes connected cycles.

Perhaps further analysis of the techniques described in this paper will lead to new ideas related to integer partition numbers or rational generating functions. In essence, one is moving the study of rational generating functions into the study of cyclical graphs that mimic the sequences or vice versa.

16 Python Code to Calculate Coefficients of RGFs

```
# -*- coding: utf-8 -*-
"""
Running on Python 3.8
Created on Wed Feb 24 7:00:00 2021
@author: Yonah and Ariel Berwaldt
"""
#Start Program
#This program calculates the coefficients of rational
    generating functions
import numpy as np
import time

def x_12(maxnum): # create 1st and 2nd column for later
    reference
    #Cycles must be arranged from smallest to largest.
    #Constants multiply the row each time the corresponding
        cycle repeats.
    #If there are multiple cycles of the same length, arrange
        so the larger constants come first.
    cycle_vec = [1,1,2,3]
    const_vec = [2,1,3,2]
    x_1vector = np.array([])
    x_2vector = np.array([])
    x_3vector = np.array([])
    #The Read vector rvect must be manually populated so its
        length equals the sum of all entries in cycle_vec.
    #To get [(1-c_1 x^x_1)*...(1-c_n x^x_n)]^(-1) a 1 must be
        placed in the rvect corresponding to the final cycle
        in the row corresponding horizontally to position (X_n
        ,0).
    #Time delays are not supported in this version.
    #Imaginary numbers are not supported in this version.
    rvect = [0,0,0,0,1,0,0]

    for i in range(0,len(cycle_vec)): #create first and second
         reference column
        count = 0
        newvec2 = np.zeros(cycle_vec[i],dtype = int)
        for j in range (0,cycle_vec[i]):
            newvec1 = np.zeros(1)
```

```
            newvec1[0] = cycle_vec[i]
            x_1vector = np.concatenate((x_1vector,newvec1),
                axis = 0)
            newvec2[count] = j
            count = count+1
        x_2vector = np.concatenate((x_2vector,newvec2), axis =
            0)

count = 0
final_range = len(x_1vector)-cycle_vec[len(cycle_vec)-1]

for i in range(0,final_range): #create third (constant
    multiplier) reference column
    if x_2vector[i] == x_1vector[i]-1:
        newvec1 = np.zeros(1)
        newvec1[0] = const_vec[count]
        x_3vector = np.concatenate((x_3vector,newvec1),
            axis = 0)
        count = count + 1
    else:
        newvec1 = np.zeros(1)
        newvec1[0] = 1
        x_3vector = np.concatenate((x_3vector,newvec1),
            axis = 0)

for i in range(final_range,len(x_1vector)):
    if x_2vector[i] == 0:
        newvec1 = np.zeros(1)
        newvec1[0] = const_vec[count]
        x_3vector = np.concatenate((x_3vector,newvec1),
            axis = 0)
        count = count + 1
    else:
        newvec1 = np.zeros(1)
        newvec1[0] = 1
        x_3vector = np.concatenate((x_3vector,newvec1),
            axis = 0)

#prepend a [0] to the first through fourth reference
    columns to make an empty row for the SumIf tabulation
x_1vector = np.concatenate(([0],x_1vector), axis = 0)
x_2vector = np.concatenate(([0],x_2vector), axis = 0)
x_3vector = np.concatenate(([0],x_3vector), axis = 0)
rvect = np.concatenate(([0],rvect), axis = 0)

maxrow = (len(x_1vector)) #The length of x_1vector is the
    sum of all entries in cycle_vec.

if maxrow != len(rvect):
    print("Please␣manually␣change␣the␣rvect␣so␣its␣length␣
```

```
            is␣the␣same␣as␣maxrow␣and␣len(x1_vector).")

    return(x_1vector,x_2vector,x_3vector,rvect, maxrow)

def choices(j,i,maxrow,mat,update_vec): # several
    possibilities to how next cell is updated

    if mat[i][1]>1:
        update_vec[i]=mat[i-1][j-1]*mat[i][2]
    if mat[i][1]==1:
        k = int(i+mat[i][0])
        if k > maxrow:
            update_vec[i]=mat[i-1][j-1]*mat[i][2]
        else:
            update_vec[i]=mat[i-1][j-1]*mat[i][2]+mat[k][j]
    if mat[i][1]==0:
        if mat[i][0]==1:
            k = int(i+mat[i][0])
            if k >= maxrow:
                update_vec[i] = mat[i][j-1]*mat[i][2]
            else:
                update_vec[i] = mat[i][j-1]*mat[i][2]+mat[k][j
                    -1]
        else:
            k = int(i+mat[i][0]-1)
            update_vec[i]=mat[k][j-1]*mat[i][2]
    mat[i][j]=update_vec[i]
    return()

def main():
    begintime = time.time()
    maxnum = 9 #if you want P(N) change constant to N+1;
    x_1vector,x_2vector, x_3vector, rvect, maxrow = x_12(
        maxnum)
    mat = np.zeros(shape = (maxrow, maxnum+3)) # note the +3
        is for the three extra reference columns on the left

    for n in range(0,maxrow): #seed the matrix's 3 leftmost
        columns
        mat[n][0]=x_1vector[n] #first reference column
        mat[n][1]=x_2vector[n] #second reference column
        mat[n][2]=x_3vector[n] #third reference column for
            multiplying by constants
        mat[n][3]=rvect[n] #read vector

    for j in range(4,maxnum+3,1): #move low to high along the
        columns
        update_vec = np.zeros(maxrow)
        for i in range(maxrow-1,0,-1): #move high to low along
            rows
```

```
            choices(j,i,maxrow,mat, update_vec) #populate
                matrix with cycles and growth
        for i in range(maxrow-1,0,-1):
            mat[i][j]=update_vec[i]

    for j in range(3,maxnum+3,1): #move low to high along the
        columns
        sum_if = 0 #tabulate all realized elements in a given
            column
        for i in range(maxrow-1,0,-1): #move high to low along
             rows
            if mat[i][1] == 0:
                sum_if = sum_if + mat[i][j]
        mat[0][j] = sum_if


    print("If there are no cycles of length 1, read the
        uppermost row. Else read the second row after removing
        the prepended 0's.")
    print (mat)
    finishtime = time.time()
    totaltime = round(finishtime-begintime,3)
    print ("cumulative computation time =", totaltime)
    #np.savetxt("cycle_type_matrix.csv", mat, delimiter = ",")
         #remove the leading comment to save output
    return()

if __name__ == "__main__": main()
#End Program
```

References

[1] Neil Calkin, Jimena Davis, Kevin James, Elizabeth Perez, and Charles Swannack. *Computing the Integer Partition Function.* Mathematics of Computation. Volume 76, Number 259, July 2007, Pages 1619–1638. http://www.ams.org/journals/mcom/2007-76-259/S0025-5718-07-01966-7/S0025-5718-07-01966-7.pdf Visited 12/16/2020

[2] Groupprops. https://groupprops.subwiki.org/wiki/Molien_series Visited 1/30/2021

[3] Formula Search Engine. https://formulasearchengine.com/wiki/Recurrence_relation Visited 4/25/2021

[4] Fredrik Johansson. *New partition function record: $p(10^{20})$ computed.* March 2, 2014. http://fredrikj.net/blog/2014/03/new-partition-function-record/ Visited 12/16/2020

[5] Jerome Kelleher and Barry O'Sullivan. *Generating All Partitions: A Comparison of Two Encodings.* May 5, 2014. https://arxiv.org/pdf/0909.2331.pdf Visited 12/16/2020

[6] James Mc Laughlin and Scott Parsell. Mathematics Department, West Chester University, 25 University Avenue, West Chester, PA 19383. *A Hardy-Ramanujan-Rademacher-type formula for (r; s)-regular partitions.* The Ramanujan Journal. 2000. https://www.wcupa.edu/sciences-mathematics/mathematics/jMcLaughlin/documents/HRRAug112011.pdf Visited 3/2/2021

[7] Oscar Levin. *Discrete Mathematics: An Open Introduction.* http://discrete.openmathbooks.org/dmoi2/section-27.html Visited 2/11/2021

[8] OEIS. Sequence A008627. https://oeis.org/A008627 Visited 1/30/2021

[9] OEIS. Sequence A103221. https://oeis.org/A103221 Visited 2/4/21

[10] Ken Ono. *Math Encounters – Ken Ono - Enigmatic Figures: The Ramanujan Legacy.* National Museum of Mathematics. https://www.youtube.com/watch?v=Qvoou69SNGI Time 51:55. Visited 1/2/2021.

[11] Jay Pantone. *Lecture 7 – Coefficient Extraction for Rational Generating Functions.* http://jaypantone.com/courses/winter16math118/lecture-notes/lecture-notes-07.pdf Visited 3/1/2021

[12] Stanley R.P. (1986). *Rational Generating Functions. In: Enumerative Combinatorics.* The Wadsworth & Brooks/Cole Mathematics Series, vol 1. Springer, Boston, MA. https://doi.org/10.1007/978-1-4615-9763-6_4 Visited 3/2/2021.

[13] Wikipedia https://en.wikipedia.org/wiki/Adjacency_matrix Visited 12/14/2020

[14] Wikipedia https://en.wikipedia.org/wiki/Binomial_theorem Visited 3/2/2021

[15] Wikipedia https://en.wikipedia.org/wiki/Fundamental_theorem_of_algebra Visited 2/20/21

[16] Wikipedia https://en.wikipedia.org/wiki/Generating_function Visited 2/10/2021

[17] Wikipedia https://en.wikipedia.org/wiki/Jordan_normal_form Visited 1/5/2021

[18] Wikipedia https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm Visited 2/8/2021

[19] Wikipedia https://en.wikipedia.org/wiki/Memoization Visited 1/28/2021

[20] Wikipedia https://en.wikipedia.org/wiki/Molien's_formula Visited 2/21/2021

[21] Wikipedia https://en.wikipedia.org/wiki/Partial_fraction_decomposition Visited 3/1/2021

[22] Wikipedia https://en.wikipedia.org/wiki/Partition_(number_theory) Visited 2/11/2021

[23] Wikipedia https://en.wikipedia.org/wiki/Perron\OT1\textendashFrobenius_theorem Visited 1/2/2021

[24] Wikipedia https://en.wikipedia.org/wiki/Strongly_connected_component Visited 2/10/21

[25] Herbert S. Wilf. *generatingfunctionology.* Academic Press Inc. (1994). https://www2.math.upenn.edu/~wilf/gfologyLinked2.pdf Visited 2/11/2021