Ordered Matrices of a Prescribed Row and Column Sum

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mathematical Science

---

by
Janine E. Janoski
May 2008

Accepted by:
Dr. Neil J. Calkin, Committee Chair
Dr. Gretchen L. Matthews
Dr. Colin M. Gallagher

Abstract

Let $\mathcal{M}(n, s)$ be the number of $n \times n$ matrices with binary entries, row and column sum s, and whose rows are in lexicographical order. Let $\mathcal{S}(n)$ be the number of $n \times n$ matrices with entries from $\{0, 1, 2\}$, symmetric, with trace 0, and row sum 2. (The sequence $S(n)$ appears as A002137 in N.J.A. Sloane's Online Encyclopedia of Integer Sequences.)

We give two proofs to show that $|\mathcal{M}(n, 2)| = |\mathcal{S}(n)|$. First, we show they satisfy the same recurrence. Second, we give an explicit bijection between the two sets. We also show that the bijection maintains the cycle structure of our matrices.

Let $\mathcal{M}_s(n, 2)$ be the set of symmetric matrices in $\mathcal{M}(n, 2)$. We will show $|\mathcal{M}_s(n, 2)|$ satisfies the Fibonacci sequence.

Acknowledgments

I would like to acknowledge the advice and guidance of my committee chair Neil Calkin. I would also like to thank Dr. Colin Gallagher and Dr. Gretchen Matthews for being members of my committee and taking the time to work with me on this endeavor.

Table of Contents

Chapter 1

Introduction

The enumeration of integer matrices is a topic which arises in several areas of mathematics including statistics, symmetric function theory, combinatorics, representation theory, and the enumeration of permutations with respect to descents. In particular, there has been a considerable amount of study of integer matrices with a prescribed row and column sum. We will use the notation of [11], that is let $f(m, n, s, t)$ be the number of $m \times n$ binary matrices with row sum s and column sum t. In the case that $m = n, s = t$ we will use the notation $f_s(n)$.There are 4 different ways in which the above problem can be viewed.

1. This problem can be viewed as the enumeration of 2-way contingency tables of order $m \times n$ over $\{0, 1, 2, \ldots\}$ such that the row marginal is s and the column marginal is t, so that $ms = nt$. This problem is of great interest to statisticians.

2. Let $f(m, n, s, t)$ be the number of semiregular labeled bipartite multigraphs with m vertices of degree s and n vertices of degree t.

3. Suppose there are $t \times n$ balls with t balls labeled A_i for $i = 1, \ldots, n$. Distribute these balls into m distinct boxes, such that each box contains s different balls. How many distributions are there? [22].

4. Suppose there are $t \times n$ letters in a row, letter A_i appears t times for $i = 1, \ldots n$. We define the following rules for the placement of these letters. There is only one letter in each position for any given row. No letter appears in more then one of the following positions: the $(sk+1)^{th}$ position, $(sk+2)^{th}$ position, $\ldots, (sk+s)^{th}$ position, $k = 0, \ldots m-1$. If $A_{i_1}, A_{i_2}, \ldots A_{i_s}$ are in positions $(sk+1), (sk+2), \ldots, (sk+s)$ then $i_1 < i_2 < \ldots < i_s$. How many arrangements are there? [22].

MacMahon [15] was one of the earliest to study such matrices while studying coefficients of functions that were expanded as the standard bases of symmetric functions. In particular MacMahon was interested in the number of permutations of n different letters. His procedure corresponds to a lattice of m rows and n columns such that there is one and only one unit in each column and a specified number of units in each row, say $\pi_1, \pi_2, \ldots \pi_n$. MacMahon also studied the magic square which was later examined by Stanley [23]. The magic square, also known as an integer stochastic matrix, is an $n \times n$ matrix over $\{0, 1, 2, \ldots\}$ with row and column sum s. It has been shown that $f_0(n) = 1, f_1(n) = n!, f_s(1) = 1$ and $f_s(2) = s + 1$. It was shown by Anand, Dumir, Gupta [2] that

$$f_s(3) = \binom{s+2}{2} + 3\binom{s+3}{4}.$$

We now state some other asymptotic results to the above problem. Everett and Stein [9] showed that

$$f_s(n) \sim \frac{(rn)!}{(r!)^{2n}} e^{\frac{1}{2}(r-1)^2}.$$

Bekessy, Bekessy, and Komlos [5] proved for specific row sums $S = \{r_1, \ldots r_n, \}$ and specific column sums $T = \{c_1, \ldots c_n\}$, that

$$f(n, n, S, T) \sim \frac{n!}{r_1! \cdots r_n! c_1! \cdots c_n!} exp\left\{ \frac{2}{n^2} \sum_{i,j} \binom{r_i}{2} \binom{c_j}{2} \right\}.$$

Barvinok [4] extended this result by assuming every table had a given weight and found the exact number of such tables is

$$\frac{n!}{r_1!\cdots r_n!c_1!\cdots c_n!}.$$

The following theorem was proven by Canfield and McKay [7]. Let s,t be positive integers satisfying $ms = nt$. Define $\lambda = \frac{s}{n} = \frac{t}{m}$. Let $a, b > 0$ be constants such that $a+b < \frac{1}{2}$. Suppose that $m, n \to \infty$ in such a way that

$$\frac{(1+2\lambda)^2}{4\lambda(1+\lambda)}\left(1+\frac{5m}{6n}+\frac{5n}{6m}\right) \le a\log n.$$

Then

$$f(m,n,s,t) = \frac{\binom{n+s-1}{s}^m \binom{m+t-1}{t}^n}{\binom{mn+\lambda mn-1}{\lambda mn}} exp\left\{\frac{1}{2}+O(n^{-b})\right\}.$$

Another result of Greenhill and McKay[13] deals with sparse integer matrices. Let $\mathcal{F}(s,t)$ be the set of $m \times n$ matrices with nonnegative integer entries such that the i^{th} row has sum s_i and the j^{th} column has sum t_j. Then clearly we must have $S = \sum_{i=1}^{m} s_i = \sum_{i=1}^{n} t_j$. Define $S_k = \sum_{i=1}^{m}[s_i]_k$ and $T_k = \sum_{j=1}^{n}[t_j]_k$ for $k \ge 1$, where $[x]_k = x(x-1)\cdots(x-k+1)$. Suppose that $m, n \to \infty, S \to \infty$ and $1 \le st = o(S^{2/3})$. Then

$$\begin{aligned}\mathcal{F}(s,t) &= \frac{S!}{\prod_{i=1}^m s_i! \prod_{j=1}^n t_j!} \exp\left\{\frac{S_2T_2}{2S^2}+\frac{S_2T_2}{2S^3}+\frac{S_3T_3}{3S^3}-\right.\\ &\left.\frac{S_2T_2(S_2+T_2)}{4S^4}-\frac{S_2^2T_3+S_3T_2^2}{2S^4}+\frac{S_2^2T_2^2}{2S^5}+O\left(\frac{s^3t^3}{S^2}\right)\right\}.\end{aligned}$$

Bender [6] considered the following generalization. Let $H(f,s,t,r)$ be the set of $m \times n$ matrices (h_{ij}) over $\{0,1,\ldots,r\}$ with row sum s, column sum t, and $(m_{ij}) \in f(m,n,s,t)$ so that $h_{ij} = 0$ whenever $m_{ij} = 0$. Then

$$H(f,s,t,r) \sim \frac{(\sum r_i)!}{\prod r_i! \prod c_j!} e^{\epsilon a - b},$$

where $\epsilon = -1$ if $r = 1$, $\epsilon = 1$ if $r > 1$, $a = \frac{(\sum r_i(r_i-1))(\sum c_j(c_j-1))}{2(\sum r_i)^2}$ and $b = \sum_{m_{ij}=0} \frac{r_i c_j}{\sum r_i}$.

McKay and Wang [18] have proven some asymptotic results for binary matrices. Suppose $m, n \to \infty$ with $sm = nt$ and $1 \leq s, t = O((sm)^{1/4})$, then

$$f(m,n,s,t) = \frac{(sm)!}{(s!)^m (t!)^n} \exp\left(-\frac{(s-1)(t-1)}{2} + O(\frac{(s+t)^4}{sm}\right).$$

Greenhill and McKay [12] have extended these results to binary matrices with forbidden positions.

Now we turn our attention to closed form formulas for $m \times n$ binary matrices. This problem has been solved for small s and t, but is still an open problem for large s and t. The case $s = t = 2$ was first solved by Anand, Dumir and Gupta[2]. Tan, Gao, Niederhausen [19] have proven several other formulas for $s = t = 2$. The known formulas include:

$$f_2(n) = \frac{1}{4^n}\left((2n)! + \sum_{k=1}^{n} (-2)^k \binom{n}{k}^2 k! 2(n-k))!\right)$$

$$f_2(n) = 2\binom{n}{2} f_2(n-1) + \binom{n}{2}(n-1) f_{n-2}(2)$$

$$f_2(n) = 4^{-n} \sum_{i=0}^{n} \frac{(-2)^i (n!)^2 (2n-2i)!}{i!(n-1)!^2}$$

$$f_2(n) = \frac{1}{4^n}\left((2n)! - \sum_{i=1}^{n-2} \binom{n}{i} P_n^i f_2(n-i) 2^{n-i} - n!\right),$$

where $\binom{n}{i} P_n^i f_2(n-i) 2^{n-i}$ represents the arrangements of $2 \times n$ letters arranged in a row when there are exactly i pairs of equal letters, each pair in consecutive positions.

One of the other early results, $s = t = 3$, was due to Reed [20] and again studied by Stanley [24]. We have

$$f_3(n) = 6^{-n} \sum \frac{(-1)^\beta n!^2 (\beta + 3\gamma)! 2^\alpha 3^\beta}{\alpha! \beta! \gamma!^2 6^\gamma},$$

where the sum is over all $\frac{(n+2)(n+1)}{2}$ solutions to $\alpha+\beta+\gamma=n$ in nonnegative integers. Tan and Gao [22] gave the following formulas for $f_3(n)$. Let

$$\begin{aligned}
f_3(n) &= \binom{n}{3}\left(\binom{n-1}{2}f_3(n-1)+18\binom{n-1}{3}t_1(n-3)+6(n-1)f_3(n-2)+\right.\\
&\quad 6\binom{n-1}{3}t_n(n-3)+90\binom{n-1}{4}t_3(n-3)+180\binom{n-1}{5}t_4(n-3)\\
&\quad \left.+90\binom{n-1}{6}t_5(n-3)\right)\\
&\quad \text{where } f_3(0)=1, f_3(1=f_3(2)=0\\
&\quad \text{where}\\
t_1(n) &= 3f_3(n)+n(n-1)t_1(n-1)\\
&\quad \text{where } t_1(1)=t_1(2)=0, t_1(-n)=0,\\
t_2(n) &= 6f_3(n)+nf_3(n-1)+3n(n-1)t_1(n-1)\\
&\quad \text{where } t_1(1)=1, t_2(2)=0, t_2(-n)=0,\\
t_3(n) &= 2nt_1(n-1)+6f_3(n)/(n-1)+2f_3(n+1)/\binom{n+1}{3}\\
&\quad \text{where } t_3(1)=0, t_3(2)=2, t_3(-n)=0,\\
t_4(n) &= 6nt_3(n-1)+4n(n-3)t_4(n-1)+H(n)\\
&\quad \text{where } H(n)=3f_3(n+1)/\binom{n+1}{3}-3n(n-3)t_4(n-1)-3nt_3(n-1), H(n)=0, n\le 3\\
&\quad \text{and } t_4(1)=t_4(2)=0, t_4(-n)=0,\\
&\quad \text{and}\\
t_5(n) &= \frac{2H(n+1)}{(n+1)(n-2)(n-3)}\\
&\quad \text{where } t_5(-n)=0,
\end{aligned}$$

$$f_3(n)=6^{-n}\sum_{\alpha=0}^{n}\sum_{\beta=0}^{n-\alpha}\frac{(-1)^\beta 2^\alpha 3^\beta (n!)^2(3n-3\alpha-2\beta)!}{\alpha!\beta!(n-\alpha-\beta)!^2 6^{n-\alpha-beta}},$$

and

$$
\begin{aligned}
f_3(n) &= \binom{n}{3}\left\{6\binom{n-1}{3}(n-3)f_3(n-4)+\right. \\
&\left(\frac{(n-1)(n-2)(1-6(n-3))}{2}-\frac{18(n-1)(n-3)}{(n-5)}\right)f_3(n-3)+ \\
&\left(33(n-1)+\frac{18(n-1)}{n-2}\right)f_3(n-2)-\frac{3}{2}(n-1)(t_1(n-2)+ \\
&\frac{3}{4}(n-1)(n-3)(n-4)(2(n-2)(n-5)-1)(n-3)!(n-6)!\times \\
&\left.\left\{\frac{18(n-5)f_3(n-2)}{(n-3)!(n-2)!}+18\sum_{i=3}^{n-3}\frac{(i-2)f_3(i)}{i!(i-1)!}-\sum_{j=3}^{n-4}\frac{f_3(j)}{j!(j-1)!}\sum_{i=j+1}^{n-3}\frac{6}{i}\right\}\right\}
\end{aligned}
$$

Tan, Gao, Mathesis, Niederhausen, have results for $s \leq 8$, and $t \leq 5$ [16], [17]. The following are just a few examples of their results:

$$f(m,n,2,3) = 2^{-m}\sum_{i=0}^{n}\frac{(-1)^i m!n!(2m-2i)!}{i!(m-i)1(n-i)!6^{n-i}}$$

$$f(m,n,4,2) = 24^{-m}\sum_{\alpha=0}^{m}\sum_{\beta=0}^{m-\alpha}\frac{(-1)^{m-\alpha-\beta)}3^{\alpha}6^{(m-\alpha-\beta)}m!n!(4\beta+2(m-\alpha-\beta)!}{\alpha!\beta!(m-\alpha-\beta)(2\beta+(m-\alpha-\beta))!2^{(2\beta+(m-\alpha-\beta))}}$$

$$
\begin{aligned}
f(m,n,4,4) &= 24^{-n}\sum_{\alpha+\beta+\gamma+\mu+\nu=n}\left(\frac{3^{\gamma}(-6)^{\beta+\nu}8^{\mu}(n!)^2(4\alpha+2\gamma+\mu)!(\beta+2\gamma)!}{\alpha!\beta!\gamma!\mu!\nu!}\times\right. \\
&\left.\sum_{i=0}^{\lfloor(\beta+2\gamma)/2\rfloor}\frac{1}{24^{(\alpha-\gamma+i}2^{\beta+2\gamma-i}i!(\beta+2\gamma-2i)!(\alpha-\gamma-i)!}\right),
\end{aligned}
$$

where the sum is over all $(n+4)(n+3)(n+2)(n+1)/24$ solutions of $\alpha+\beta+\gamma+\mu+\nu=n$.

$$f(m,n,5,3) \;=\; 120^{-m} \sum_{\alpha+\beta+\gamma+\mu+\nu=n} \frac{(-1)^{\beta+\nu}10^{\beta}15^{\gamma}2^{\,-\mu+\nu}\, m!n!(5\alpha+3\beta+\gamma+2\mu)!}{\alpha!\beta!\gamma!\mu!\nu!(n-\beta-2\gamma-\mu-2\nu)!2^{(n-\beta-2\gamma-\mu-2\nu)}}$$

where the sum is over all $(n+4)(n+3)(n+2)(n+1)/24$ solutions of $\alpha+\beta+\gamma+\mu+\nu=n$.

$$\begin{aligned} f(m,n,7,5) \;=\; & \frac{m!n!}{(7!)^m} \sum_{i_1+\ldots+i_{13}=m} \sum_{2i+j=i_2+2i_3+i_4+i_6+2i_7+i_{10}+i_{13}} \sum_{k=0}^{j} \\ & \frac{(-1)^{i_2+i_4+i_6+i_9+i_{11}+i_{13}}21^{i_2}105^{i_3+i_4}70^{i_5}}{i_1!i_2!\cdots i_{13}!2^{(i_5+i_6+i_7+2i_8+i_{11}+i-k)}} \times \\ & \frac{420^{i_6+i_{11}}210^{i_7+i_9}280^{i_8}630^{i_{10}}504^{i_{12}+i_{13}}(i_5+i_6+i_7+2i_8+i_{11})!}{k!(j-k(!i!(n-i_5-i_6-i_7-2i_8-i_9-i_{10}-2i_{11}-i_{12}-i_{13}-i-j+k)!} \times \\ & \frac{(2i+j)!(7i_1+5i_2+3i_3+i_4+4i_5+2i_6+i_8+3i_9+i_{10}+2i_{12})!}{6^{(j+k)}(i_5+i_6+i_7+2i_8+i_{11}-k)!120^{(n-i_5-i_6-i_7-2i_8-i_9-i_{10}-2i_{11}-i_{12}-i_{13}-i-j+k)}} \end{aligned}$$

where the sum is over all $\binom{m+12}{12}$ solutions of $i_1+\ldots+i_{13}=m$ in nonnegative integers.

Our interest in this area of research began with the following question: What is the probability that a binary square matrix with a fixed row and column sum is invertible? We want to randomly generate a such a matrix using a markov chain simulation that uses an alternative rectangle switch.

Definition 1. An alternating rectangle in $(a_{ij}) = A^{(n)}$ is a set of four distinct entries $\{a_{ii}, a_{ij}, a_{ji}, a_{jj}\}$ such that the entries alternate 0's and 1's as we go around the rectangle in either direction.

We perform a switch on an alternating rectangle by interchanging the 0's and 1's. For example in $(a_{ij}) = A^{(4)}$ shown below we can choose $a_{10}, a_{11}, a_{20}, a_{21}$.

$$A^4 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

It has been proven that any binary matrix with given row and columns sums can be obtained from any other by a finite sequence of switches along alternating rectangles, Ryser [21]. We are interested in empircally studying the mixing rate of this markov chain and comparing this to the upper bounds of Greenhill and McKay. We are interested in finding a uniform distribution of these matrices. One method towards studying the length of such chains is to compress the amount of data for the markov chain. In order to compress the data we will consider ordered rows.

Let $\mathcal{M}(n, 2)$ be the set of $n \times n$ binary matrices with row and column sum 2 and rows in lexicographical order. We say R_i, R_j are in lexicographical order, i.e. $R_i < R_j$, if the first column that R_i and R_j differ must contain a 1 in R_i and a 0 in R_j. Let $M(n, 2) = |\mathcal{M}(n, 2)|$. Example:

$$\mathcal{M}(4,2) = \left\{ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \right.$$

$$\left. \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \right\}$$

For example

$$\left\{ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \right.$$

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\left\{ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \right\}$$

is the set of matrices which will be transformed into

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \in \mathcal{M}(4,2)$$

with the lexicographical ordering.

We want to determine is the number of steps in the markov chain walk that will give us a random matrix in $\mathcal{M}(n,2)$. While we preform our markov walk we will not consider the ordering, we will just be concerned with binary matrices with row and column sum 2. At the end of our markov walk we will then reorder the matrix.

This inner loop of our algorithm is the alternative rectangle switch. The outer loop will run the inner loop a fixed number of times and record the results. We will alter the length of the inner loop and determine the appropriate length of a markov walk to find a random matrix.

We wish to examine the following problems as open research.

Open Problem 1. How much faster than the current known upper bounds given by Greenhill does the alternating rectangle switch converge?

Open Problem 2. What is the probability that a binary square matrix with row and column sum s is invertible?

Open Problem 3. What is the probability that a matrix in $\mathcal{M}(n,s)$ is invertible?

In this paper we will be interested in enumerating three different set of matrices, $\mathcal{M}(n,2), \mathcal{S}(n), \mathcal{M}_(n,2)$. We define $\mathcal{S}(n)$ be the set of $n \times n$ symmetric matrices over $\{0,1,2\}$ with row sum 2 and trace 0. We define $\mathcal{M}_s(n,2)$ be the set of symmetric matrices contained in $\mathcal{M}(n,2)$.

In particular in Lemma 2 we give a recurrence for $|\mathcal{M}(n,2)|$. In Lemma 3 we show that $|\mathcal{S}(n)|$ satisfies that same recurrence. In Theorem 5 we give a bijection between $\mathcal{M}(n,2)$ and $\mathcal{S}(n)$. In Theorem 7 we show $M_s(n,2)$ is the Fibonacci numbers, suitably indexed.

Chapter 2

Enumeration of $\mathcal{M}(n, 2)$

We are interested in enumerating the set $\mathcal{M}(n, 2)$. Let $M(n, 2) = |\mathcal{M}(n, 2)|$.

We begin by considering all possible rows of size n with a row sum of 2, say R. Let $x_i x_j$ represent the row with ones in column C_i and column C_j. Then $(x_i x_j)^k$ represents the case where we have used k copies of R. We sum all possible k values to find $\sum_{k=0}^{n} x_i x_j = \frac{1}{1-x_i x_j}$. We are interested in all possible combinations of such rows, that is, we want the union of these sets of rows. We define

$$P_2(x_1, \ldots, x_n) = \frac{1}{\prod_{i \leq j}(1 - x_i x_j)}$$

to represent all possible combinations of rows with sum 2.

We are interested in the coefficient of x_i^2 so that C_i will have a sum of 2. To find this coefficient we will take the second derivative and set $x_i = 0$. The lower order terms x_i^0, x_i will be differentiated out and the higher terms $x_i^3, x_i^4, \ldots$ will still have an x_i after differentiation and will be set to 0. We note that during this process we will have an extra factor of 2.

We perform this differentiation for all x_i. This will ensure that every column will have a sum of 2. Dividing by 2^n we find the cardinality of the set.

We begin with a simple example to find $M(3,2)$. Let

$$P_2(x_1,x_2,x_3)=\frac{1}{(1-x_1x_2)(1-x_1x_3)(1-x_2x_3)}.$$

We will now take the second derivative with respect to x_1 to get:

$$\begin{aligned}\frac{d^2P_2(x_1,x_2,x_3)}{dx_1^2} &= \frac{2x_2^2}{(1-x_1x_2)^3(1-x_1x_3)(1-x_2x_3)}+\\ &\quad \frac{2x_2x_3}{(1-x_1x_2)^2(1-x_1x_3)^2(1-x_1x_3)^2(1-x_2x_3)}+\\ &\quad \frac{2x_3^2}{(1-x_1x_2)(1-x_1x_3)^3(1-x_2x_3)}.\end{aligned}$$

Setting $x_1=0$ we have

$$\begin{aligned}\left.\frac{d^2P_2(x_1,x_2,x_3)}{dx_1^2}\right|_{x_1=0} &= \frac{2x_2^2}{1-x_2x_3}+\frac{2x_2x_3}{1-x_2x_3}+\frac{2x_3^2}{1-x_2x_3}\\ &= P_2(x_2,x_3)((x_2+x_3)^2+x_2^2+x_3^2).\end{aligned}$$

Next we take the second derivative with respect to x_2 and find:

$$\begin{aligned}\frac{d^2P_2(x_2,x_3)((x_2+x_3)^2+x_2^2+x_3^2)}{dx_2^2} &= \frac{4x_2^2x_3^2}{(1-x_2x_3)^3}+\frac{8x_2x_3}{(1-x_2x_3)^2}+\frac{4}{1-x_2x_3}+\\ &\quad \frac{4x_2x_3^3}{(1-x_2x_3)^3}+\frac{4x_3^2}{(1-x_2x_3)^2}+\frac{4x_3^4}{(1-x_2x_3)^3}.\end{aligned}$$

Setting $x_2=0$ we have

$$\left.\frac{d^2P_2(x_2,x_3)((x_2+x_3)^2+x_2^2+x_3^2)}{dx_2^2}\right|_{x_2=0}=4+4x_3^2+4x_3^4.$$

Taking the second derivative with respect to x_3 and then setting $x_3 = 0$ we have

$$\left.\frac{d^2(4+4x_3^2+4x_3^4)}{dx_3^2}\right|_{x_3=0} = 8.$$

To find the total number of matrices in $\mathcal{M}(n,2)$ we must divide by $2^3 = 8$. Thus for $n = 3$, $M(3,2) = 1$,

$$\mathcal{M}(3,2) = \left\{ \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \right\}.$$

As we see from above the notation gets cumbersome. We define the following notation to help generalize the above procedure for large $M(n,2)$. Define

$$P_2(x_r, \ldots, x_k) = \frac{1}{\prod_{r \le i < j \le k}(1 - x_i x_j)},$$

$$\epsilon_2^{(1)}(x_r; x_{r+1}, \ldots, x_k) = \sum_{i=r+1}^{k} \frac{x_i}{1 - x_r x_i},$$

and

$$\epsilon_2^{(2)}(x_r; x_{r+1}, \ldots, x_k) = \sum_{i=r+1}^{k} \frac{x_i^2}{(1 - x_r x_i)^2}.$$

Note:

$$\frac{d}{dx_r}\epsilon_2^{(1)}(x_r; x_{r+1}, \ldots, x_k) = \epsilon_2^{(2)}(x_r; x_{r+1}, \ldots, x_k).$$

Finally we define:

$$e_1(x_{r+1}, \ldots, x_k) = \frac{d}{dx_r}\, \epsilon_2^{(1)}(x_r; x_{r+1}, \ldots, x_k)\Big|_{x_r=0} = \sum_{i=r+1}^{k} x_i,$$

$$e_2(x_{r+1}^2, \ldots, x_k^2) = \frac{d}{dx_r}\, \epsilon_2^{(2)}(x_r; x_{r+1}, \ldots, x_k)\Big|_{x_r=0} = \sum_{i=r+1}^{k} x_i^2.$$

Note:

$$\frac{d(P_2(x_r,\ldots,x_k))}{dx_r})\Big|_{x_r=0} = P_2(x_{r+1},\ldots,x_k).$$

When we are interested in a specific n we will let $k=n$

With the above notation we now hope to generalize our results. We begin by taking the first derivative with respect to x_1 to get

$$\frac{dP_2}{dx_1} = P_2(x_1,\ldots,x_k){\epsilon_2}^{(1)}(x_1;x_2,\ldots,x_k).$$

Taking the second derivative we find

$$\frac{d^2P_2}{dx_1^2} = P_2(x_1,\ldots,x_k)({\epsilon_2}^{(1)}(x_1;x_2,\ldots,x_k)^2 + {\epsilon_2}^{(2)}(x_1;x_2,\ldots,x_k)).$$

Setting $x_1=0$ we have

$$\frac{d^2P_2}{dx_1^2}\Big|_{x_1=0} = P_2(x_2,\ldots,x_k)[e_1(x_2,\ldots,x_k)^2 + e_1(x_2^2,\ldots,x_k^2)] = P_2^{(1)}.$$

We now take the derivative with respect to x_2 to find

$$\begin{aligned}\frac{dP_2^{(1)}}{dx_2} &= P_2(x_2,\ldots,x_k)[{\epsilon_2}^{(1)}(x_2;x_3,\ldots,x_k)[e_1(x_2,\ldots,x_k)^2 + e_1({x_2}^2,\ldots,{x_k}^2)] + \\ &\quad 2e_1(x_2,\ldots,x_k) + 2x_2].\end{aligned}$$

We now take the second derivative with respect to x_2 to find

$$\begin{aligned}\frac{d^2P_2^{(1)}}{dx_2^2} &= P_2(x_2,\ldots,x_k)[{\epsilon_2}^{(1)}(x_2;x_3,\ldots,x_k)[{\epsilon_2}^{(1)}(x_2;x_3,\ldots,x_k)[e_1(x_2,\ldots,x_k)^2 + \\ &\quad e_1(x_2^2,\ldots,x_k^2)] + 2e_1(x_2,\ldots,x_k) + 2x_2] + \epsilon_2^{(2)}(x_2;x_3,\ldots,x_k)[e_1(x_2,\ldots,x_k)^2 + \\ &\quad e_1(x_2^2+\ldots,x_k^2)] + \epsilon_2^{(1)}(x_2;x_3,\ldots,x_k)[2e_1(x_2,\ldots,x_k) + 2x_2] + 4].\end{aligned}$$

Setting $x_2 = 0$ we have

$$\begin{aligned}
\left.\frac{d^2 P_2^{(1)}}{dx_2^2}\right|_{x_2=0} &= P_2(x_3,\ldots,x_k)[e_1(x_3,\ldots,x_k)[e_1(x_3,\ldots,x_k)[e_1(x_3,\ldots,x_k)^2 + e_1(x_3^2,\ldots,x_k^2)] + \\
&\quad 2e_1(x_3,\ldots,x_k)] + e_1(x_3^2,\ldots,x_k^2)[e_1(x_3,\ldots,x_k)^2 + e_1(x_3^2+\ldots,x_k^2)] + \\
&\quad 2e_1(x_3,\ldots,x_k)^2 + 4] \\
&= P_2(x_3,\ldots,x_k)[e_1(x_3,\ldots,x_k)^4 + 2e_1(x_3,\ldots,x_k)^2 e_1(x_3^2,\ldots,x_k^2) \\
&\quad +4e_1(x_3,\ldots,x_k)^2 + e_1(x_3^2,\ldots,x_k^2)^2 + 4] \\
&= P_2^{(2)}.
\end{aligned}$$

Continuing in this manner, the second derivative with respect to x_3 and setting $x_3 = 0$ gives

$$\begin{aligned}
\left.\frac{d^2}{dx_3^2}\right|_{x_3=0} &= P_2(x_4,\ldots,x_k)[e_1(x_4,\ldots,x_k)^6 + 3e_1(x_4,\ldots,x_k)^4 e_1(x_4^2,\ldots,x_k^2) + \\
&\quad 12e_1(x_4,\ldots,x_k)^4 + 3e_1(x_4,\ldots,x_k)^2 e_1(x_4^2,\ldots,x_k^2)^2 + \\
&\quad 36e_1(x_4,\ldots,x_k)^2 + 12e_1(x_4,\ldots,x_k)^2 e_1(x_4^2,\ldots,x_k^2) + \\
&\quad e_1(x_4^2,\ldots,x_k^2)^3 + 12e_1(x_4^2,\ldots,x_k^2) + 8] \\
&\quad P_2^{(3)}
\end{aligned}$$

The second derivative with respect to x_4 and setting $x_4 = 0$ gives

$$\begin{aligned}
\left.\frac{d^2}{dx_4^2}\right|_{x_4=0} &= P_2(x_5,\ldots,x_k)[e_1(x_5,\ldots,x_k)^8 + 4e_1(x_5,\ldots,x_k)^6 e_1(x_5^2,\ldots,x_k^2) + \\
&\quad 24e_1(x_5,\ldots,x_k)^6 + 6e_1(x_5,\ldots,x_k)^4 e_1(x_5^2,\ldots,x_k^2)^2 + 168e_1(x_5,\ldots,x_k)^4 + \\
&\quad 48e_1(x_5,\ldots,x_k)^4 e_1(x_5^2,\ldots,x_k^2) + 4e_1(x_5,\ldots,x_k)^2 e_1(x_5^2,\ldots,x_k^2)^3 + \\
&\quad 320e_1(x_5,\ldots,x_k)^2 + 24e_1(x_5,\ldots,x_k)^2 e_1(x_5^2,\ldots,x_k^2)^2 + \\
&\quad 144e_1(x_5,\ldots,x_k)^2 e_1(x_5^2,\ldots,x_k^2) + 24e_1(x_5^2,\ldots,x_k^2)^2 + \\
&\quad 32e_2(x_5^2,\ldots,x_k^2) + 96 + e_1(x_5^2,\ldots,x_k^2)^4] \\
&\quad P_2^{(4)}
\end{aligned}$$

The second derivative with respect to x_5 and setting $x_5 = 0$ gives

$$\begin{aligned}
\left.\frac{d^2}{dx_5^2}\right|_{x_5=0} &= P_2(x_6,\ldots,x_k)[e_1(x_6,\ldots,x_k)^{10} + 5e_1(x_6,\ldots,x_k)^8 e_1(x_6^2,\ldots,x_k^2) + \\
&\quad 40e_1(x_6,\ldots,x_k)^8 + 10e_1(x_6,\ldots,x_k)^6 e_1(x_6^2,\ldots,x_k^2)^2 + 520e_1(x_6,\ldots,x_k)^6 + \\
&\quad 120e_1(x_6,\ldots,x_k)^6 e_1(x_6^2,\ldots,x_k^2) + 10e_1(x_6,\ldots,x_k)^4 e_1(x_6^2,\ldots,x_k^2)^3 + \\
&\quad 2480e_1(x_6,\ldots,x_k)^4 + 120e_1(x_6,\ldots,x_k)^4 e_1(x_6^2,\ldots,x_k^2)^2 \\
&\quad +840e_1(x_6,\ldots,x_k)^4 e_1(x_6^2,\ldots,x_k^2) + 1600e_1(x_6,\ldots,x_k)^2 (x_6^2,\ldots,x_k^2) + \\
&\quad 3680e_1(x_6,\ldots,x_k)^2 + 5e_1(x_6,\ldots,x_k)^2 e_1(x_6^2,\ldots,x_k^2)^4 + \\
&\quad 40e_1(x_6,\ldots,x_k)^2 e_1(x_6^2,\ldots,x_k^2)^3 + 360e_1(x_6,\ldots,x_k)^2 e_1(x_6^2,\ldots,x_k^2)^2 + \\
&\quad 40e_1(x_6^2,\ldots,x_k^2)^3 + 80e_2(x_6^2,\ldots,x_k^2)^2 + 480e_1(x_6^2,\ldots,x_k^2) + e_1(x_6^2,\ldots,x_k^2)^5 \\
&\quad +704] \\
&= P_2^{(5)}.
\end{aligned}$$

Dividing by 2^n we have found,

n	1	2	3	4	5
$M(n,2)$	0	1	1	6	22

Let $C(n,j,l)$ be the coefficient of $e_2(x_{n+1},\ldots,x_k)^{2j}e_2(x_{n+1}^2,\ldots,x_k^2)^l$. Let $u = e_2(x_{n+1},\ldots,x_k)$ and $v = e_2(x_{n+1}^2,\ldots,x_k^2)$.

Number of choices for j,l	Degree of $e_2(x_{n+1},\ldots,x_k)^{2j}e_2(x_{n+1}^2,\ldots,x_k^2)^l$
$n+1$	2^n
n	2^{n-1}
$n-1$	2^{n-2}
$\vdots$	$\vdots$
2	2^1
1	0

The above table suggests a natural arrangement of our terms based on the degree. For the 3×3 case we have

$$C(3,0,0)u^0v^0$$

$$C(3,0,1)v \qquad C(3,1,0)u^2$$

$$C(3,0,2)v^2 \qquad C(3,1,1)vu \qquad C(3,2,0)u^4$$

$$C(3,0,3)v^3 \qquad C(3,1,2)v^2u^2 \qquad C(3,2,1)vu^4 \qquad C(3,3,0)u^6$$

$$8u^0v^0$$

$$12v \qquad 36u^2$$

$$0v^2 \qquad 12vu \qquad 12u^4$$

$$1v^3 \qquad 3v^2u^2 \qquad 3vu^4 \qquad 1u^6$$

In the 4×4 case we have

$$C(4,0,0)$$

$$C(4,0,1)v \quad C(4,1,0)u^2$$

$$C(4,0,2)v^2 \quad C(4,1,1)vu^2 \quad C(4,2,0)u^4$$

$$C(4,0,3)v^3 \quad C(4,1,2)v^2u^2 \quad C(4,2,1)vu^4 \quad C(4,3,0)u^6$$

$$C(4,0,4)v^4 \quad C(4,1,3)v^3u^2 \quad C(4,2,2)v^2u^4 \quad C(4,3,1)vu^6 \quad C(4,4,0)u^8$$

$$96$$

$$32v \qquad 320u^2$$

$$24v^2 \qquad 144vu^2 \qquad 168u^4$$

$$0v^3 \qquad 24v^2u^2 \qquad 48vu^4 \qquad 24u^6$$

$$1v^4 \qquad 4v^3u^2 \qquad 6v^2u^4 \qquad 4vu^6 \qquad 1u^8$$

From this small example we notice a recursive relationship between the coefficients $C(n,j,l)$.

Theorem 1. *$C(n,j,l)$ can be defined recursively as $C(n,j,l) = C(n-1,j-1,l)+4jC(n-1,j,l)+C(n-1,j,l-1)+2(j+1)(2j+1)C(n-1,j+1,l)+2(l+1)C(n-1,j,l+1)$ for $j \leq n, l \leq n, j+l \leq n$*

Proof: We will give a proof by induction. For the base case we have that

$$\frac{d^2}{d^2x_1}P_2(x_1,\ldots,x_k)|_{x_1=0} = P_2(x_2,\ldots,x_k)[e_1(x_2,\ldots,x_k)^2 + e_1(x_2^2,\ldots,x_k^2)].$$

Thus $C(1,1,0) = 1, C(1,0,1) = 1, C(1,j,l) = 0$ for all other j,l. We have seen

$$\begin{aligned}\frac{d^2}{d^2x_2}|_{x_2=0} &= P_2(x_3,\ldots,x_k)[e_1(x_3,\ldots,x_k)^4 + 2e_1(x_3,\ldots,x_k)^2e_1(x_3^2,\ldots,x_k^2) + \\ &\quad 4e_1(x_3,\ldots,x_k)^2 + e_1(x_3^2,\ldots,x_k^2)^2 + 4.\end{aligned}$$

Thus

$$\begin{aligned}C(2,0,0) = 4 &= 0+0+0+2+2 \\ &= C(1,-1,0) + 0C(1,0,0) + C(1,0,-1) + 2C(1,1,0) + 2C(1,0,1), \\ C(2,1,0) = 4 &= 0+4+0+0+0 \\ &= C(1,0,0) + 4C(1,1,0) + C(1,1,-1) + 12C(1,2,0) + 2C(1,1,1), \\ C(2,2,0) = 1 &= 1+0+0+0+0 \\ &= C(1,1,0) + 8C(1,2,0) + C(1,2,-1) + 30C(1,3,0) + 2C(1,2,1), \\ C(2,0,1) = 0 &= 0+0+0+0+0 \\ &= C(1,-1,1) + 0C(1,0,1) + C(1,0,0) + 2C(1,1,1) + 4C(1,0,2), \\ C(2,0,2) = 1 &= 0+0+1+0+0 \\ &= C(1,-1,2) + 0C(1,0,2) + C(1,0,1) + 2C(1,1,2) + 6C(1,0,3),\end{aligned}$$

and

$$\begin{aligned} C(2,1,1) = 2 &= 1+0+1+0+0 \\ &= C(1,0,1) + 4C(1,1,1) + C(1,1,0) + 12C(1,2,1) + 4C(1,1,2). \end{aligned}$$

Thus the base case holds. Assume that this recursion holds for all $m \leq n$.

We wish to show that the recursion will hold for $n+1$. Suppose we have

$$P_2(x_n, \ldots, x_k) \left[\sum_{j=0}^{n} \sum_{l=0}^{n} C(n,j,l) e_1(x_n, \ldots x_k)^{2j} e_1(x_n^2, \ldots, x_k^2)^l \right].$$

We take the derivative with respect to x_n and find

$$\begin{aligned} \frac{d}{dx_n} &= P_2(x_n, \ldots, x_k)[\epsilon_2^{(1)}(x_n, \ldots, x_k) \left[\sum_{j=0}^{n} \sum_{l=0}^{n} C(n,j,l) e_1(x_n, \ldots x_k)^{2j} e_1(x_n^2, \ldots, x_k^2)^l \right] + \\ &\sum_{j=0}^{n} \sum_{l=0}^{n} 2jC(n,j,l) e_1(x_n, \ldots, x_k)^{2j-1} e_1(x_n^2, \ldots, x_k^2)^+ l \\ &\sum_{j=0}^{n} \sum_{l=0}^{n} 2lx_n C(n,j,l) e_1(x_n, \ldots, x_n)^{2j} e_1(x_n^2, \ldots, x_k^2)^{l-1}]. \end{aligned}$$

Taking the second derivative with respect to x_n we have

$$
\begin{aligned}
\frac{d^2}{dx_n^2} \;=\;& P_2[\epsilon_2^{(1)}(x_n,\ldots,x_k)[\epsilon_2^{(1)}(x_n,\ldots,x_k)\left[\sum_{j=0}^{n}\sum_{l=0}^{n} C(n,j,l)e_1(x_n,\ldots x_k)^{2j}e_1(x_n^2,\ldots,x_k^2)^l\right] + \\
& \sum_{j=0}^{n}\sum_{l=0}^{n} 2jC(n,j,l)e_1(x_n,\ldots,x_k)^{2j-1}e_1(x_n^2,\ldots,x_k^2)^l + \\
& \sum_{j=0}^{n}\sum_{l=0}^{n} 2lx_nC(n,j,l)e_1(x_n,\ldots,x_n)^{2j}e_1(x_n,\ldots,x_k^2)^{l-1}]] + \\
& \epsilon_2^{(2)}(x_n,\ldots,x_k)\left[\sum_{j=0}^{n}\sum_{l=0}^{n} C(n,j,l)e_1(x_n,\ldots x_k)^{2j}e_1(x_n^2,\ldots,x_k^2)^l\right] + \\
& \epsilon_2^{(1)}(x_n,\ldots,x_k)[\sum_{j=0}^{n}\sum_{l=0}^{n} 2jC(n,j,l)e_1(x_n,\ldots,x_k)^{2j-1}e_1(x_n^2,\ldots,x_k^2)^l + \\
& \sum_{j=0}^{n}\sum_{l=0}^{n} 2lx_nC(n,j,l)e_1(x_n,\ldots,x_n)^{2j}e_1(x_n^2,\ldots,x_k^2)^{l-1}]] + \\
& \sum_{j=0}^{n}\sum_{l=0}^{n} 2j(2j-1)C(n,j,l)e_1(x_n,\ldots,x_k)^{2(j-1)}e_1(x_n^2,\ldots,x_k^2)^l + \\
& \sum_{j=0}^{n}\sum_{l=0}^{n} 4jlx_nC(n,j,l)e_1(x_n,\ldots,x_n)^{2j-1}e_1(x_n^2,\ldots,x_k^2)^{l-1}] + \\
& \sum_{j=0}^{n}\sum_{l=0}^{n} 4jlx_nC(n,j,l)e_1(x_n,\ldots,x_n)^{2j-1}e_1(x_n^2,\ldots,x_k^2)^{l-1}] + \\
& \sum_{j=0}^{n}\sum_{l=0}^{n} 4l(l-1)x_n^2C(n,j,l)e_1(x_n,\ldots,x_n)^{2j}e_1(x_n^2,\ldots,x_k^2)^{l-2}] + \\
& \sum_{j=0}^{n}\sum_{l=0}^{n} 2lC(n,j,l)e_1(x_n,\ldots,x_n)^{2j}e_1(x_n^2,\ldots,x_k^2)^{l-1}].
\end{aligned}
$$

Setting $x_n = 0$ we have

$$
\begin{aligned}
\frac{d^2}{dx_n^2}|_{x_n=0} &= P_2(x_{n+1},\ldots,x_k)[e_1^2(x_{n+1},\ldots,x_k) \\
&\quad \left[\sum_{j=0}^{n}\sum_{l=0}^{n} C(n,j,l)e_1(x_{n+1},\ldots x_k)^{2j}e_1(x_{n+1}^2,\ldots,x_k^2)^l\right] + \\
&\quad e_1(x_{n+1},\ldots,x_k)[\sum_{j=0}^{n}\sum_{l=0}^{n} 2jC(n,j,l)e_1(x_{n+1},\ldots,x_k)^{2j-1}e_1(x_{n+1}^2,\ldots,x_k^2)^l] + \\
&\quad e_1(x_{n+1}^2,\ldots,x_k^2)\left[\sum_{j=0}^{n}\sum_{l=0}^{n} C(n,j,l)e_1(x_{n+1},\ldots x_k)^{2j}e_1(x_{n+1}^2,\ldots,x_k^2)^l\right] + \\
&\quad e_1(x_{n+1},\ldots,x_k)[\sum_{j=0}^{n}\sum_{l=0}^{n} 2jC(n,j,l)e_1(x_{n+1},\ldots,x_k)^{2j-1}e_1(x_{n+1}^2,\ldots,x_k^2)^l] + \\
&\quad \sum_{j=0}^{n}\sum_{l=0}^{n} 2j(2j-1)C(n,j,l)e_1(x_{n+1},\ldots,x_k)^{2(j-1)}e_1(x_{n+1}^2,\ldots,x_k^2)^l + \\
&\quad \sum_{j=0}^{n}\sum_{l=0}^{n} 2lC(n,j,l)e_1(x_{n+1},\ldots,x_n)^{2j}e_1(x_{n+1},\ldots,x_k^2)^{l-1}]] \\
&= P_2(x_{n+1},\ldots,x_k)[\sum_{j=0}^{n+1}\sum_{l=0}^{n+1} nC(n,j,l)e_1(x_{n+1},\ldots x_k)^{2(j+1)}e_1(x_{n+1}^2,\ldots,x_k^2)^l + \\
&\quad \sum_{j=0}^{n+1}\sum_{l=0}^{n+1} 4jC(n,j,l)e_1(x_{n+1},\ldots,x_k)^{2j}e_1(x_{n+1}^2,\ldots,x_k^2)^l + \\
&\quad \sum_{j=0}^{n+1}\sum_{l=0}^{n+1} C(n,j,l)e_1(x_{n+1},\ldots x_k)^{2j}e_1(x_{n+1}^2,\ldots,x_k^2)^{l+1} + \\
&\quad \sum_{j=0}^{n+1}\sum_{l=0}^{n+1} 2j(2j-1)C(n,j,l)e_1(x_{n+1},\ldots,x_k)^{2(j-1)}e_1(x_{n+1}^2,\ldots,x_k^2)^l + \\
&\quad \sum_{j=0}^{n+1}\sum_{l=0}^{n+1} n2lC(n,j,l)e_1(x_{n+1},\ldots,x_n)^{2j}e_1(x_{n+1},\ldots,x_k^2)^{l-1}]
\end{aligned}
$$

$$
\begin{aligned}
&= P_2(x_{n+1},\ldots,x_k)[\sum_{j=1}^{n+1}\sum_{l=0}^{n+1} C(n,j-1,l)e_1(x_{n+1},\ldots x_k)^{2j}e_1(x_{n+1}^2,\ldots,x_k^2)^l + \\
&\sum_{j=0}^{n+1}\sum_{l=0}^{n+1} 4jC(n,j,l)e_1(x_{n+1},\ldots,x_k)^{2j}e_1(x_{n+1}^2,\ldots,x_k^2)^l + \\
&\sum_{j=0}^{n+1}\sum_{l=1}^{n+1} C(n,j,l-1)e_1(x_{n+1},\ldots x_k)^{2j}e_1(x_{n+1}^2,\ldots,x_k^2)^l + \\
&\sum_{j=1}^{n+1}\sum_{l=0}^{n+1} 2(j+1)(2(j+1)-1)C(n,j+1,l)e_1(x_{n+1},\ldots,x_k)^{2j}e_1(x_{n+1}^2,\ldots,x_k^2)^l + \\
&\sum_{j=0}^{n+1}\sum_{l=1}^{n+1} 2(l+1)C(n,j,l+1)e_1(x_{n+1},\ldots,x_n)^{2j}e_1(x_{n+1},\ldots,x_k^2)^l]
\end{aligned}
$$

$$
\begin{aligned}
= \; & P_2(x_{n+1},\ldots,x_k)[\sum_{j=0}^{n+1}\sum_{l=0}^{n+1}[C(n,j-1,l)e_1(x_{n+1},\ldots x_k)^{2j}e_1(x_{n+1}^2,\ldots,x_k^2)^l + \\
& 4jC(n,j,l)e_1(x_{n+1},\ldots,x_k)^{2j}e_1(x_{n+1}^2,\ldots,x_k^2)^l + \\
& C(n,j,l-1)e_1(x_{n+1},\ldots x_k)^{2j}e_1(x_{n+1}^2,\ldots,x_k^2)^l + \\
& 2(j+1)(2j+1)C(n,j+1,l)e_1(x_{n+1},\ldots,x_k)^{2j}e_1(x_{n+1}^2,\ldots,x_k^2)^l + \\
& 2(l+1)C(n,j,l+1)e_1(x_{n+1},\ldots,x_n)^{2j}e_1(x_{n+1},\ldots,x_k^2)^l]] \\
= \; & P_2(x_{n+1},\ldots,x_k)[\sum_{j=0}^{n+1}\sum_{l=0}^{n+1}[C(n,j-1,l)+4jC(n,j,l)+C(n,j,l-1)+ \\
& 2(j+1)(2j+1)C(n,j+1,l)+2(l+1)C(n,j,l+1)e_1(x_{n+1},\ldots,x_n)^{2j}e_1(x_{n+1},\ldots,x_k^2)^l]].
\end{aligned}
$$

■

Note that $\frac{C(n,0,0)}{2^n} = M(n,2)$. We implemented the recursion first using maple and then using C++ (see Appendix).

n	$M(n,2)$	n	$M(n,2)$
1	0	11	5238370
2	1	12	60222844
3	1	13	752587764
4	6	14	10157945044
5	22	15	147267180508
6	130	16	2282355168060
7	822	17	37655004171808
8	6202	18	658906772228668
9	52552	19	12188911634495388
10	499194	20	237669544014377896

Chapter 3

Relationship between $\mathcal{M}(n,2)$ and $\mathcal{S}(n)$

3.1 A recurrence for $M(n,2)$

Recall $\mathcal{M}(n,2)$ is the set of $n \times n$ binary matrices with row sum and column sum 2 and rows in lexicographical order. We define $M(n,2) = |\mathcal{M}(n,2)|$.

Lemma 2. *Let $M(n,2)$ be defined as above. Then $M(n,2)$ satisfies*

$$M(n,2) = (n-1)M(n-1,2) - \frac{(n-1)(n-2)}{2}M(n-3,2) + (n-1)M(n-2,2).$$

Proof: First we wish to consider how we can create a matrix in $\mathcal{M}(n,2)$ from a matrix in $\mathcal{M}(n-1,2)$, by the following algorithm, denoted algorithm M_{n-1}:

1. Let $A^{(n)}$ be an empty $n \times n$ matrix and let $A^{(n-1)} \in \mathcal{M}(n-1,2)$.
2. Choose $R_i^{(n-1)}$ with ones in $C_j^{(n-1)}$ and $C_k^{(n-1)}$.
3. Split $R_i^{(n-1)}$ into $R_0^{(n)}$ with ones in $C_0^{(n)}$ and $C_{j+1}^{(n)}$ and $R_1^{(n)}$ with ones in $C_0^{(n)}$ and $C_{k+1}^{(n)}$.
4. Fill $R_0^{(n)}$ and $R_1^{(n)}$ into $A^{(n)}$.

5. Fill $C_0^{(n)}$ with 0's.

6. Fill $A^{(n)}$ with the remaining rows of $A^{(n-1)}$.

Example: Let

$$A^{(5)} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Choose

$$R_2^{(5)} = [\ 0 \quad 1 \quad 1 \quad 0 \quad 0\ ].$$

Then we spilt $R_2^{(5)}$ into

$$R_0^{(6)} = [\ 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0\ ] \text{ and } R_1^{(6)}[\ 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0\ ].$$

Filling in $C_0^{(6)}$ with 0s we have

$$A^{(6)} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & & & & & \\ 0 & & & & & \\ 0 & & & & & \\ 0 & & & & & \end{bmatrix}.$$

Filling in the remaining entries we have

$$A^{(6)} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Since there are $n-1$ rows of $A^{(n-1)}$ this process will generate $(n-1)$ possible $A^{(n)}$ from every matrix in $\mathcal{M}(n-1,2)$, that is, this process will contribute a total of

$$(n-1)M(n-1,2)$$

matrices.

Note: If $R_i^{(n-1)} = R_j^{(n-1)}$ then $A_{R_i}^{(n)} = A_{R_j}^{(n)}$. That is if there is a repeated row when we preform this algorithm we will create the same matrix in $\mathcal{M}(n,2)$. For example let

$$A^{(5)} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

If we choose $R_0^{(5)}, R_1^{(5)}$ we will get

$$A^{(6)} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

In order to fix this double counting, consider the number of matrices in $\mathcal{M}(n-1,2)$ with repeated rows.

We can form a new matrix in $\mathcal{M}(n-1,2)$ with a repeated row from $\mathcal{M}(n-3,2)$ by the following algorithm, algorithm M_{n-3}.

1. Let $A^{(n-1)}$ be an empty $(n-1) \times (n-1)$ matrix. Let $A^{(n-3)} \in \mathcal{M}(n-3,2)$.
2. Pick $R^{(n-1)}$ of length $n-1$. Suppose $R^{(n-1)}$ has ones in $C_j^{(n-1)}$ and $C_k^{(n-1)}$.
3. Set $R_0^{(n-1)} = R_1^{(n-1)} = R^{(n-1)}$.
4. Fill $C_j^{(n-1)}$ and $C_k^{(n-1)}$ with zeros.
5. Fill the remaining of $A^{(n-1)}$ with the entries from $A^{(n-3)}$.
6. Reorder the rows of $A^{(n-1)}$.

Example: Let

$$A^{(3)} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Let

$$R^{(5)} = [\ 0 \quad 1 \quad 0 \quad 0 \quad 1\ ].$$

Then

$$A^{(5)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ & 0 & & & 0 \\ & 0 & & & 0 \\ & 0 & & & 0 \end{bmatrix}.$$

Filling in $A^{(5)}$ with the rows from $A^{(3)}$ we have

$$A^{(5)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Reordering the rows we have:

$$A^{(5)} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

There are

$$\frac{(n-1)(n-2)}{2}$$

different rows of size $n-1$. We have

$$\frac{(n-1)(n-2)}{2}M(n-3,2)$$

matrices in $\mathcal{M}(n-1,2)$ which have have repeated rows. That is from $\mathcal{M}(n-1,2)$ the total contributed $n \times n$ matrices is

$$(n-1)M(n-1,2) - \frac{(n-1)(n-2)}{2}M(n-3,2).$$

There is one more case to consider. We can create new matrices by adding a repeated row. In order for these to be distinct we will choose a repeated row which has a one in $C_0^{(n)}$. We can create these matrices from the following procedure, algorithm M_{n-2}:

1. Let $A^{(n)}$ be an empty $n \times n$ matrix and $A^{(n-2)} \in \mathcal{M}(n-2,2)$
2. Choose $R^{(n)}$ of length n with ones in $C_0^{(n)}$ and $C_j^{(n)}$.
3. Let $R_0^{(n)} = R_1^{(n)} = R^n$ in $A^{(n)}$.
4. Fill $C_0^{(n)}$ and $C_j^{(n)}$ with zeros.
5. Fill $A^{(n)}$ with the elements in $A^{(n-2)}$.

Example: Let

$$A^{(4)} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}.$$

Let

$$R^{(6)} = [\ 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0\ ].$$

Then

$$A^{(6)} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & & & 0 & & \\ 0 & & & 0 & & \\ 0 & & & 0 & & \\ 0 & & & 0 & & \end{bmatrix}.$$

Filling in $A^{(6)}$ with $A^{(4)}$ we have

$$A^{(6)} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

There are $n-1$ rows of size n with a one in column $C_0^{(n)}$. Thus we have

$$(n-1)M(n-2,2)$$

matrices in $\mathcal{M}(n,2)$. In total we have

$$M(n,2) = (n-1)M(n-1,2) - \frac{(n-1)(n-2)}{2}M(n-3,2) + (n-1)M(n-2,2).$$

■

3.2 A recurrence for $S(n)$

Definition 2. Let $\mathcal{S}(n)$ be the set of $n \times n$ matrices satisfying the following properties:

1. The elements are taken from the set $\{0, 1, 2\}$.
2. The row sum is two.
3. The matrix is symmetric.
4. The trace is 0.

Note: The sequence S(n) has been studied by MacMahon, Etherington, Aitken and can be found as sequence A002137 In N.J.A. Sloans Online Encyclopedia of Integer Sequences [1], [8], [14]. Let $S(n) = |\mathcal{S}(n)|$.

Lemma 3. *$S(n)$ satisfies*

$$S(n) = (n-1)S(n-1) - \frac{(n-1)(n-2)}{2}S(n-3) + (n-1)S(n-2)$$

Proof: Let $B^{(n-1)} \in \mathcal{S}(n-1)$. If we wish to replace a row that contains only the elements $0, 1$. We can create $B^{(n)} \in \mathcal{S}(n)$ by the following procedure, algorithm $S_{n-1,1}$.

1. Let $(b_{ij}) = B^{(n-1)} \in \mathcal{S}(n-1)$.
2. Choose $R_i^{(n-1)}$ with ones in $C_j^{(n-1)}, C_k^{(n-1)}$ and $R_j^{(n-1)}$ with ones in $C_i^{(n-1)}, C_l^{(n-1)}$, such that $i \neq j$.
3. Replace b_{ij} and b_{ji} with 0s. Prepend a 1 to the front of $R_i^{(n-1)}$ and $R_j^{(n-1)}$.
4. Prepend a zero to the front of all other rows.
5. Prepend $R_0^{(n)}$ with 1s in $C_{i+1}^{(n)}$ and $C_{j+1}^{(n)}$ to the top of $B^{(n-1)}$.

6. Let $B^{(n)} = B^{(n-1)}$

Example: Let

$$B^{(5)} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Choose

$$R_1^{(5)} = [\ 1 \quad 0 \quad 1 \quad 0 \quad 0\ ]$$

and

$$R_2^{(5)} = [\ 0 \quad 1 \quad 0 \quad 0 \quad 1\ ].$$

Then we have

$$B^{(5)} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

We prepend

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

to the top of $B^{(5)}$ to find

$$B^{(6)} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Let $b_{ii}^{(n)}, i > 0$ be a diagonal entry for $B^{(n)}$. Then $b_{ii}^{(n)} = b_{ii}^{(n-1)} \in B^{(n-1)}$, as $i \neq j$. From algorithm $S_{n-1,1}$ we see that this holds as we do not change any diagonal entries in $B^{(n-1)}$. Also from our process we have $b_{00}^{(n)} = 0$. Thus we have $Tr(B^{(n)}) = 0$.

Also we are replacing $b_{i,j}$, and $b_{j,i}$ in matrix $B^{(n-1)}$ with a 0, maintaining the symmetry of $B^{(n-1)}$. Then we add $b_{0i} = b_{0j} = b_{i0} = b_{0j} = 1$ and all other $b_{0k} = b_{k0} = 0$, that is, $\left(R_0^{(n)}\right)^T = C_0^{(n)}$. Thus $B^{(n)}$ is symmetric.

Now suppose we want to switch a row which contains a 2. We can create a new matrix with the following algorithm, algorithm $S_{n-1,2}$.

1. $B^{(n-1)} \in \mathcal{S}(n-1)$.
2. Choose $R_i^{(n-1)}$ with a 2 in $C_j^{(n-1)}$. Choose to switch $R_i^{(n-1)}, R_j^{(n-1)}$.
3. Replace the 2 in $R_i^{(n-1)}$ with a 1, and prepend a 1 to the front of $C_0^{(n-1)}$.
4. Replace the 2 in $R_j^{(n-1)}$ with a 1, and prepend a 1 to the front of $C_0^{(n-1)}$.
5. Prepend a 0 to the front of all remaining rows.
6. Prepend

$$[0, \ldots, 0, 1, 0, \ldots, 0, 1, 0, \ldots 0]$$

with ones in $C_{i+1}^{(n)}$ and $C_{j+1}^{(n)}$ to the top of the matrix.

7. Let $B^{(n)} = B^{(n-1)}$

Example: Let

$$B^{(5)} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix}.$$

Choose

$$R_2^{(5)} = [\ 0 \quad 0 \quad 0 \quad 0 \quad 2\ ]$$

and

$$R_4^{(5)} = [\ 0 \quad 0 \quad 2 \quad 0 \quad 0\ ].$$

Then we have

$$R_2^{(5)} = [\ 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1\ ]$$

and

$$R_4^{(5)} = [\ 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0\ ].$$

So we have

$$B^{(5)} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

We prepend

$$[\ 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1\ ]$$

to the top of $A^{(5)}$ and find

$$B^{(6)} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

Let $b_{ii}^{(n)}, i > 0$ be a diagonal entry for $B^{(n)}$. Then $b_{ii}^{(n)} = b_{ii}^{(n-1)} \in B^{(n-1)}$, as $i \neq j$. From algorithm $S_{n-1,2}$ we see that this holds as we do not change any diagonal entries in $B^{(n-1)}$. Also from our process we have $b_{00}^{(n)} = 0$. Thus we have $Tr(B^{(n)}) = 0$.

Also we are replacing $b_{i,j}$, and $b_{j,i}$ in matrix $B^{(n-1)}$ with a 1, maintaining the symmetry of $B^{(n-1)}$. Then we add $b_{0i} = b_{0j} = b_{i0} = b_{0j} = 1$ and all other $b_{0k} = b_{k0} = 0$, that is, $\left(R_0^{(n)}\right)^T = C_0^{(n)}$. Thus $B^{(n)}$ is symmetric.

We want to count the total number of matrices in $\mathcal{S}(n)$ formed by the two algorithms above. We will refer to these algorithms together as $S_{(n-1)}$ and will use choose the appropriate algorithm based on the elements in the row we are transforming. Let $B^{(n-1)} \in \mathcal{S}(n-1)$ and consider walking through each of the rows of $B^{(n-1)}$. Suppose we are at a row which contains $\{0, 1\}$. By removing the restriction that $i < j$, each row of this form will create 2 matrices in $\mathcal{S}(n)$. By removing this restriction we will count each pair twice. So the total number contributed will be the number of rows with entries from $\{0, 1\}$.

If we choose a row with 2 in $C_i^{(n-1)}$ it will contribute one new matrix. But by our above algorithm will produce the same matrix when we use $R_i^{(n-1)}$. So we can consider counting each of these, so we will have

$$(n-1)S(n-1).$$

And we must now remove the double counting we used for the rows that contained a 2.

We can create a matrix $B^{(n-1)} \in \mathcal{S}(n-1)$ with a row containing a 2 by the following algorithm, algorithm S_{n-3}

1. Let $B^{(n-3)}) \in \mathcal{S}(n-3)$ and $B^{(n-1)}$ be an empty $(n-1) \times (n-1)$ matrix.
2. Choose i, j such that $j > i$.
3. Set $b_{ij}^{(n-1)} = b_{ji}^{(n-1)} = 2$.
4. Fill the remaining entries of $R_i^{(n-1)}, R_j^{(n-1)}, C_i^{(n-1)}, C_j^{(n-1)}$ with zeros.
5. Fill $B^{(n-1)}$ with the entries from $B^{(n-3)}$

Example: Let

$$B^{(3)} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

Let $i = 2, j = 4$. Then we have

$$B^{(5)} = \begin{bmatrix} & & 0 & & 0 \\ & & 0 & & 0 \\ 0 & 0 & 0 & 0 & 2 \\ & & 0 & & 0 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix}.$$

Filling in the remaining elements we have

$$B^{(5)} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix}.$$

In the first row of a matrix of size $(n-1) \times (n-1)$ we have $(n-2)$ choices where we can add a 2 to create a $\mathcal{S}(n-1)$ matrix. In row 2 we have already seen the matrix produced by placing a 2 in the first column, and cannot place a 2 in the second column. So there are $(n-3)$ choices. Continuing in this manner the total number is

$$(n-2) + (n-3) + \ldots + 1 = \frac{(n-1)(n-2)}{2}.$$

So that we have double counted

$$\frac{(n-1)(n-2)}{2} S(n-3)$$

matrices. That is from $\mathcal{S}(n-1)$ the total contributed matrices is

$$(n-1)S(n-1) - \frac{(n-1)(n-2)}{2} S(n-3).$$

Finally we can form new matrices by adding two rows with elements from $\{0, 2\}$ into a matrix from $\mathcal{S}(n-2)$. In order for these to be distinct from the matrices we have already created, we must add a 2 to the first row. And then by symmetry picking where we place the two will determine which rows and columns we are adding. Denote this as algorithm S_{n-2}

1. Let $B^{(n-2)} \in \mathcal{S}(n-2)$. Let $B^{(n)}$ be an empty $n \times n$ matrix.

2. Choose $R^{(n)}$ to be a row of length n with a 2 in $C_j^{(n)}$ where $j \neq 0$.

3. Let $C_0^{(n)} = R^{(n)T}$ and $R_0^{(n)} = R$.

4. Fill the remaining entries of $C_j^{(n)}$ and $R_j^{(n)}$ with zeros.

5. Fill the remaining entries of $B^{(n)}$ with the entries from $B^{(n-2)}$.

Example : Let

$$B^{(4)} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

Let

$$R^{(6)} = [\ 0 \quad 0 \quad 2 \quad 0 \quad 0 \quad 0\ ].$$

Then

$$B^{(6)} = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & & 0 & & & \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & & 0 & & & \\ 0 & & 0 & & & \\ 0 & & 0 & & & \end{bmatrix}.$$

Filling in the remaining elements we have

$$B^{(6)} = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

There are $n-1$ ways we can add this row, giving us $(n-1)S(n-2)$ additional matrices.

So our total number of matrices is

$$S(n) = (n-1)S(n-1) - \frac{(n-1)(n-2)}{2}S(n-3) + (n-1)S(n-2).$$

■

It has been shown that $S(n)$ satisfies the exponential generating function

$$(1-x)^{-\frac{1}{2}}e^{(-\frac{x}{2}+\frac{x^2}{4})}.$$

We will give a second proof to Lemma 4 using the techniques of exponential generating functions.

Proof: Let

$$f(x) = \sum a_n \frac{x^n}{n!}.$$

Note:

$$f'(x) = \sum \frac{a_n x^{n-1}}{(n-1)!} = \sum a_{n+1}\frac{x^n}{n!}.$$

So we have

$$f(x) = (1-x)^{-\frac{1}{2}} e^{(-\frac{x}{2}+\frac{x^2}{4})}$$

and

$$f'(x) = f(x)\left[\frac{1}{2(1-x)} + \frac{x}{2} - \frac{1}{2}\right].$$

Rearrange our terms we have

$$2f'(x)(1-x) = f(x)(2x - x^2).$$

Note: we are interested in the coefficient of x^n, denoted $[x^n]$.

$$\frac{2a_{n+1}}{n!} - \frac{2a_n}{(n-1)!} = \frac{2a_{n-1}}{(n-1)!} - \frac{a_{n-2}}{(n-2)!}.$$
$$a_{n+1} - na_n = na_{n-1} - \frac{n(n-1)}{2}a_{n-2}.$$
$$a_{n+1} = na_n + na_{n-1} - \frac{n(n-1)}{2}a_{n-2}.$$

Relabeling our indices we have

$$a(n) = (n-1)a(n-1) - \frac{(n-1)(n-2)}{2}a(n-3) + (n-1)a(n-2).$$

■

3.3 Bijection between $\mathcal{M}(n,2)$ and $\mathcal{S}(n)$

We have the following theorem.

Theorem 4. $S(n) = M(n,2)$

Proof: Combining Lemmas 3 and 4 we see that $M(n,2) = S(n)$. ■

We now offer an alternative proof.

Proof: We give a bijection between $\mathcal{M}(n,2)$ and $\mathcal{S}(n,2)$.

Let $A^{(n)} \in \mathcal{M}(n,2)$. Let $\sigma_i : A^{(k-1)} \to A^{(k)}$ be the map defined by splitting $R_l^{(l-1)}$ in algorithm $M_{(n-1)}$. Define $\sigma_i^{-1} : A^{(k)} \to A^{(k-1)}$ by combining $R_0^{(k)}$ and $R_1^{(k)}$, deleting $C_0^{(n)}, R_0^{(n)}$ and reordering the matrix. Then we see $\sigma_i^{-1}(\sigma_i(A^{(k-1)})) = A^{(k-1)}$. Let $\sigma_j : A^{(k-2)} \to A^k$ by σ_j be defined by adding $R^{(k)}$ with ones in $C_0^{(k)}$ and $C_m^{(k)}$ twice to the top of $A^{(k-2)}$as in algorithm $M_{(n-2)}$. Define $\sigma_j^{-1} : A^{(k)} \to A^{(k-2)}$ by removing $R_0^{(k)}$ and $R_1^{(k)}$ and deleting $C_0^{(k)}$ and $C_m^{(k)}$. So we have $\sigma_j^{-1}(\sigma_j(A^{(k-2)})) = A^{(k-2)}$.

We wish to map $A^{(n)}$ to the appropriate base case. We note that the notation we use will depend on each specific matrix. This happens because we have two choices for the map, either jumping to a matrix with one less dimension or two less dimensions. We denote $A^{(n)}, A^{n-i_1}, \ldots, A^4, A^{4-i_k}$ to be the sequence of matrices from applying the σ_i^{-1}. Note that $i_k = 1$ or $i_k = 2$, and this depends on if we are using algorithm M_{n-1} or M_{n-2}. We have

$$\sigma_1^{-1}\sigma_2^{-1}\ldots\sigma_{l-1}^{-1}\sigma_l^{-1}(A^{(n)}) = \sigma_1^{-1}\sigma_2^{-1}\ldots\sigma_{l-1}^{-1}A^{(n-1)} = \ldots = \sigma_1^{-1}A^{(n-4)} = A^{(n-3)}$$

or

$$\sigma_1^{-1}\sigma_2^{-1}\ldots\sigma_{r-1}^{-1}\sigma_r^{-1}(A^{(n)}) = \sigma_1^{-1}\sigma_2^{-1}\ldots\sigma_{r-1}^{-1}A^{(n-1)} = \ldots = \sigma_1^{-1}A^{(n-4)} = A^{(n-2)}$$

For each $\sigma_i^{-1}(A^{k-i})$ we keep track of the top row of $A^{(k)}$ before we reorder the matrix.

Once we have reached the base case we map $A^{(3)}$ to $B^{(3)} \in \mathcal{S}(3)$ or $A^{(2)}$ to $B^{(2)} \in \mathcal{S}(2)$.

Let $\tau_i : B^{(k-1)} \to B^{(k)}$ be the map defined by splitting $R_l^{(k-1)}$ and $R_m^{(k-1)}$ as in algorithm $S_{(n-1)}$. Define $\tau_i^{-1} : B^{(k)} \to B^{(k-1)}$ by removing $R_0^{(k)}$ and $C_0(k)$ and adding one to b_{lm} and b_{ml}. So we have $\tau_i^{-1}(\tau_i(B^{(k-1)})) = B^{(k-1)}$. Let $\tau_j : B^{(k-2)} \to B^{(k)}$ be the map defined by adding$R_0^{(k)}$ with a two in $C_m^{(k)}$ as in algorithm $S_{(n-2)}$. Define τ_j^{-1} by deleting $C_0^{(k)}, C_m^{(k)}, R_0^{(k)}, R_m^{(k)}$. So we have $\tau_j^{-1}(\tau_j(B^{(k-2)})) = B^{(k-2)}$.

Using the rows we recorded from the σ_l we perform the appropriate operation to build

$B^{(n)}$. In particular, the operation of splitting $R_l^{(k-1)} = [\ 0 \quad \dots \quad 0 \quad 1 \quad 0 \quad \dots \quad 0 \quad 1 \quad 0 \quad \dots \quad 0\ ]$ with ones in $C_l^{(k-1)}$ and $C_m^{(k-1)}$ corresponds to applying the operation $S_{(n-1)}$ to $R_l^{(k-1)}$ and $R_m^{(k-1)}$. We see in the base case that our choices for splitting a row in $A^{(2)}$ and $A^{(3)}$ correspond exactly to the possible pairs we can pick in $B^{(2)}$ and $B^{(3)}$. By the recursive nature of our algorithm this will hold for all $A^{(k)}$ and $B^{(k)}$.

So we define τ_i by the row that was spilt in σ_i. That is

$$\tau_1\tau_2\dots\tau_{r-1}\tau_r(B^{(2)}) = B^{(n)}$$

or

$$\tau_1\tau_2\dots\tau_{l-1}\tau_l(B^{(3)}) = B^{(n)}$$

We note this map has an inverse. We begin with $B^{(n)}$ and apply $\tau_l^{-1}, \tau_{l-1}^{-1}, \dots \tau_2^{-1}\tau_1^{-1}(B^{(n)})$. Then we map $B^{(2)}$ to $A^{(2)}$ or $B^{(3)}$ to $A^{(3)}$. Then we apply $\sigma_1, \sigma_2, \dots \sigma_l$.

Thus this process completely defines an invertible mapping between $\mathcal{M}(n,2)$ and $\mathcal{S}(n)$.

■

Example: Let

$$A^7 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

We notice that the top row is not a repeated row, so we are going to transform to A^6. That

is σ_3 is operation M_{n-1} applied to A^6 to get to A^7. The inverse of this operation is to reverse the splitting of the row. An easy way to think of this is to add the top two rows together and delete the first column and first row. Then

$$A^6 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

We must record the top row of A^6 before we reorder the matrix. Let $a_3 = [\ 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0\ ]$. Reordering A^6 we have

$$A^6 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

In this case we see that the top row is repeated. That means we will be looking for σ_2 that will be the map to A^6 from $A^{(4)}$. To reverse this mapping we just remove the top two rows and the columns that contain the ones. But we need to record the "top" row. If we add the

top two rows we have $a_2 = \begin{bmatrix} 0 & 2 & 0 & 0 & 0 \end{bmatrix}$. Now we delete the top two rows and find:

$$A^4 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

We are in the case where σ_1 will be the map from A^3 to A^4. Thus we have $a_1 = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$ and

$$A^3 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Now we map A^3 to B^3, where

$$B^3 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

Recall $a_1 = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$. So we want to perform a switch on $R_0^{(3)}$ and $R_1^{(3)}$. So

$$B^{(4)} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

Now $a_2 = \begin{bmatrix} 0 & 2 & 0 & 0 & 0 \end{bmatrix}$. That is we want to add the row $\begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$ as the

top row of B^6. Thus

$$B^6 = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Recall $a_3 = [\ 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0\ ]$. So we will be switching R_1^6, R_3^6. Thus

$$B^7 = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Now we will show the inverse operations to transform $B^{(7)}$ to $A^{(7)}$. We start with

$$B^7 = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

We see that $R_0^{(7)}$ has ones in $C_2^{(7)}$ and $C_4^{(7)}$. So we will choose τ_3^{-1} to be the operation of adding a 1 to b_{24} and b_{42} and deleting $R_0^{(7)}$ and $C_0^{(7)}$. We note in this direction we do not want to record the top row, rather we want to record $R_0^{(7)}$ without the element in b_{00}. That is $b_3 = [\ 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0\ ]$. Note that $a_3 = b_3$. Thus we have

$$B^6 = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

We notice $R_0^{(6)}$ contains a 2, so we will delete $R_0^{(6)}, R_2^{(6)}, C_0^{(6)}, C_2^{(6)}$. So

$$B^{(4)} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

Again we record the top row of $b_2 = [\ 0 \quad 2 \quad 0 \quad 0 \quad 0\ ]$. We see that $R_0^{(4)}$ has ones in $C_1^{(4)}$ and $C_2^{(4)}$. So we will choose τ_1^{-1} to be the operation of adding a 1 to b_{12} and b_{21} and deleting $R_0^{(4)}$ and $C_0^{(4)}$. Let $b_1 = [\ 1 \quad 1 \quad 0\ ]$. Then

$$B^3 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

We map $B^{(3)}$ to $A^{(3)}$, where

$$A^3 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Recall $b_1 = [\ 1 \quad 1 \quad 0\ ]$, so that we are switching $R_0^{(3)}$ to find

$$A^4 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

Recall $b_2 = [\ 0 \quad 2 \quad 0 \quad 0 \quad 0\ ]$ so we will add two copies of $[\ 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0\ ]$ to find

$$A^6 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Recall $b_3 = [\ 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0\ ]$ so we will split $R_2^{(6)}$ to find

$$A^7 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

3.4 The Cycle structure of $\mathcal{M}(n, 2)$ and $\mathcal{S}(n)$

We can view the matrices in $\mathcal{M}(n, 2)$ as incidence matrices and the matrices in $\mathcal{S}(n)$ as adjacency matrices. Recall an incidence matrix is a representation of a graph where the rows represent each vertex, the columns represent the edges, and $(v, e) = 1$ if and only if v is incident upon edge e. An adjacency matrix is a matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position (v_i, v_j) according to whether v_i and v_j are adjacent or not.

We wish to examine the cycle structure, in particular the number of cycles and the length of each cycle, of these two sets. For $\mathcal{M}(n, 2)$ we will consider the rows as vertices labeled $a, b, c, d, \ldots$. There is an edge between two rows if there is some column that has a 1 in both rows. Example:

$$A^{(4)} = \begin{array}{c} a \\ b \\ c \\ d \end{array} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

There are edges between $(a, b), (b, d), (d, c), (a, c)$.

For $\mathcal{S}(n)$ we will label the columns and rows $0, 1, 2, \ldots$. There is an edge between two vertices if in some row there is a 1 in the column. Example:

$$B^{(4)} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \left[\begin{array}{cccc} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right] \end{array}$$

There are edges between $(0,1), (0,2), (1,3), (2,3)$.

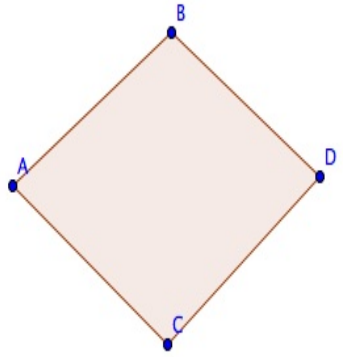

Figure 3.1: The graph for the incidence matrix $A^{(4)}$

Lemma 5. *The cycle structure of $\mathcal{M}(n,2)$ is the same as the cycle structure between $\mathcal{S}(n)$*

Proof. We begin with the base case for each set.

$$\mathcal{M}(2,2) = \left\{ \left[\begin{array}{cc} 1 & 1 \\ 1 & 1 \end{array}\right] \right\} \qquad \mathcal{S}(2) = \left\{ \left[\begin{array}{cc} 0 & 2 \\ 2 & 0 \end{array}\right] \right\}$$

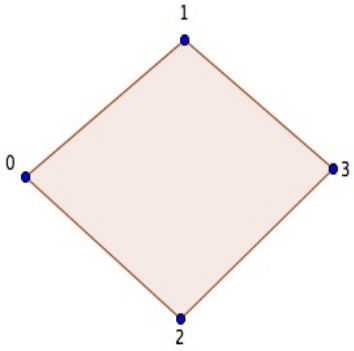

Figure 3.2: The graph for the adjacency matrix $B^{(4)}$

We can consider this as a a cycle between two nodes. Consider

$$\mathcal{M}(3,2) = \left\{ \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \right\} \qquad \mathcal{S}(3) = \left\{ \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \right\}$$

We notice that both of these have one cycle of length three. Recall from our bijection that there is a natural mapping between these two sets. So it remains to show that the bijection will preserve the cycle structure.

Let $A^{(3)}, A^{(4)}, \ldots, A^{(n)}$ be a sequence of matrices in $\mathcal{M}(n,,2)$. Let $B^{(3)}, B^{(4)}, \ldots, B^{(n)}$ be a sequence of matrices in $\mathcal{S}(n)$ such that $A^{(k)} \to B^{(k)}$. Recall we can go to $A^{(k)}$ from $A^{(k-2)}$ by adding a repeated row. This is the same as going to $B^{(k)}$ from $B^{(k-2)}$ by adding a pair of rows both containing a 2. Performing this operation will not touch the current cycle structure, it will add a new cycle containing two nodes.

Now we must worry about going from $A^{(k-1)}$ to $A^{(k)}$ by splitting $R_i^{(k-1)}$ with ones in $C_j^{(k-1)}$ and $C_l^{(k-1)}$. When we perform this split we are adding an extra vertex to the cycle

that contains $R_i^{(k-1)}$. Example: Let us split $R_1^{(4)}$ from $A^{(4)}$ given above.

$$A^{(5)} = \begin{array}{c} \\ e \\ a \\ b \\ c \\ d \end{array} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

(a, b) becomes edges $(a, e), (e, b)$.

From Theorem 5 we know this is the same as the map from $B^{(k-1)}$ to $B^{(k)}$ by splitting $R_j^{(k-1)}$ and $R_l^{(k-1)}$.

$$B^{(5)} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Again we see that this operation will add a node to the cycle which contains $R_k^{(k-1)}$ and $R_l^{(k-1)}$. By our bijection we know that this cycle is the same cycle as in $A^{(k-1)}$. □

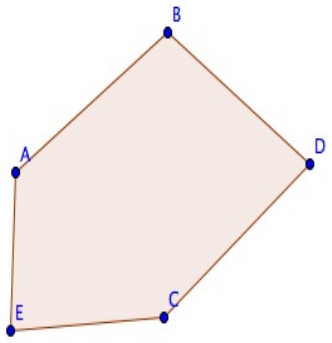

Figure 3.3: The graph for the incidence matrix $A^{(5)}$

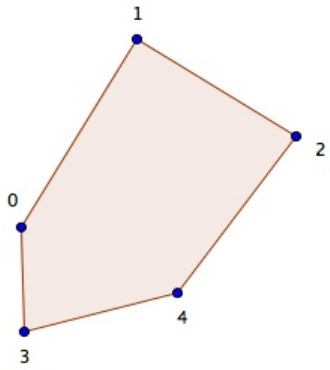

Figure 3.4: The graph for the adjacency matrix $B^{(5)}$

Chapter 4

Subsets and Generalizations

4.1 Subsets of $\mathcal{M}(n, 2)$

Let $\mathcal{M}_s(n, 2)$ be the set of symmetric matrices in $\mathcal{M}(n, 2)$. Define $M_s(n, s) = |\mathcal{M}_s(n, 2)|$. Example: $M_s(5, 2) = 3$

$$\mathcal{M}_2(5,2) = \left\{ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \right\}$$

Theorem 6. *For $n \geq 4$,*

$$M_s(n, 2) = M_s(n-1, 2) + M_s(n-2, 2),$$

where $M_s(2, 2) = 1, M_s(3, 2) = 1$. That is $M_s(n, 2)$ satisfies the Fibonacci sequence.

Proof: Let $f_n = f_{n-1} + f_{n-2}$ represent the Fibonacci sequence. Then we can write

$$\begin{aligned} f_n &= f_{n-1} + f_{n-2} \\ &= f_{n-2} + f_{n-3} + f_{n-3} + f_{n-4} \\ &= f_{n-2} + f_{n-3} + f_{n-4} + f_{n-5} + f_{n-5} + f_{n-6} \\ &\vdots \\ &= \sum_{i=2}^{n} f_{n-i} + f(0) = \sum_{i=2}^{n} f_{n-i} + 1 \end{aligned}$$

We will prove $M_s(n,2) = \sum_{i=2}^{n-2} M_s(n-i,2) + 1$. Since our rows are ordered and the matrix must be symmetric there is only one choice for the first row,

$$R_0^{(n)} = [\ 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad \ldots \quad \ldots \quad 0 \quad 0 \quad 0\ ].$$

Now we want to consider$R_1^{(n)}$. The first choice is that we can have

$$R_1^{(n)} = [\ 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad \ldots \quad \ldots \quad 0 \quad 0 \quad 0\ ].$$

In this case the sum of $R_0^{(n)}, R_1^{(n)}, C_0^{(n)}$ and $C_1^{(n)}$ is two. So we must only worry about the $M_s(n-2,2)$ submatrix. The only other choice is

$$R_1^{(n)} = [\ 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad \ldots \quad \ldots \quad 0 \quad 0 \quad 0\ ].$$

If we choose any other row for $R_1^{(n)}$ it will violate the ordering since our matrix must be symmetric. If we choose this row we must now consider $R_2^{(n)}$. The first option is to choose

$$R_2^{(n)} = [\ 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad \ldots \quad \ldots \quad 0 \quad 0 \quad 0\ ].$$

In this case the sum of $R_0^{(n)}, R_1^{(n)}, R_2^{(n)}, C_0^{(n)}, C_1^{(n)}$ and $C_2^{(n)}$ is two. So we must worry about the $M_s(n-3,2)$ submatrix. The only other possible row choice is to let

$$R_2^{(n)} = [\ 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad \ldots \quad \ldots \quad 0 \quad 0 \quad 0\ ].$$

We continue this process and notice

$$R_k^{(n)} = [\ 0 \quad \ldots \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad \ldots \quad 0 \quad 0\ ]$$

or

$$R_k^{(n)} = [\ 0 \quad \ldots \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad \ldots \quad 0 \quad 0\ ],$$

for $1 \leq k \leq n-3$, where the first one is in $C_{k-1}^{(n)}$. If we choose

$$R_k^{(n)} = [\ 0 \quad \ldots \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad \ldots \quad 0 \quad 0\ ]$$

then we are concerned with the $M_s(k-1,2)$ submatrix. So for $1 \leq k \leq n-3$ we have counted $\sum_{i=2}^{n-2} M_s(n-i,2)$ matrices.

We notice if

$$R_{n-3}^{(n)} = [\ 0 \quad 0 \quad 0 \quad 0 \quad \ldots \quad 0 \quad 1 \quad 0 \quad 1 \quad 0\ ],$$

then we cannot choose

$$R_{n-2}^{(n)} = [\ 0 \quad 0 \quad 0 \quad 0 \quad \ldots \quad 0 \quad 1 \quad 1 \quad 0\ ],$$

since this would force the sum of $R_1^{(n)}, \ldots, R_{n-2}^{(n)}, C_1^{(n)}, \ldots, C_{n-2}^{(n)}$ to be 2 forcing $a_{n-1,n-1} = 2$.

So the only choice we have is

$$R_{n-3}^{(n)} = [\ 0\ \ 0\ \ 0\ \ 0\ \ \ldots\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ ,$$

$$R_{n-2}^{(n)} = [\ 0\ \ 0\ \ 0\ \ 0\ \ \ldots\ \ 0\ \ 1\ \ 0\ \ 1\ ],$$

and

$$R_{n-1}^{(n)} = [\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ \ldots\ \ \ldots\ \ 0\ \ 1\ \ 1\ ].$$

That is there is only 1 matrix in this case.

Thus

$$M_s(n,2) = \sum_{i=2}^{n-2} M_s(n-i,2) + 1 = M_s(n-1,2) + M_s(n-2,2).$$

■

Definition 3. $\mathcal{M}_{s,tr(0)}(n,2)$ be the set of symmetric binary matrices with row and column sum 2, trace 0, and partially ordered. That is we will impose our lexicographical ordering on all but the first row. Let $M_{s,tr(0)}(n,2) = |\mathcal{M}_{s,tr(0)}(n,2)|$.

Theorem 7. *For all* $n \geq 4$, $M_{s,t}(n,2) = 1$.

Proof: We will show that there is exactly one choice for each row. Let $(a_{ij}) = A^{(n)} \in \mathcal{M}_{s,tr(0)}(n,2)$. In $R_0^{(n)}$ we want to have $a_{00} = 0$ but need to ensure the remainder of the matrix will be ordered. Thus we must have

$$R_0^{(n)} = [\ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ \ldots\ \ \ldots\ \ 0\ \ 0\ \ 0\ ].$$

In $R_1^{(n)}$ we now have $a_{10} = 1, a_{11} = 0$. If $a_{12} = 1$ then $R_2^{(n)} < R_1^{(n)}$, contradicting our ordering. Thus $a_{12} = 0, a_{13} = 1$ so we have

$$R_1^{(n)} = [\ 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad \dots \quad \dots \quad 0 \quad 0 \quad 0\ ].$$

In $R_2^{(n)}$ we have $a_{20} = 1, a_{21} = a_{22} = 0$. If $a_{23} = 1$ we will have

$$= \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & \dots & \dots & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & \dots & \dots & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & \dots & \dots & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & \dots & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & & & & & & \\ 0 & 0 & 0 & 0 & & & & & & \\ \vdots & \vdots & \vdots & \vdots & & & & & & \\ 0 & 0 & 0 & 0 & & & & & & \end{bmatrix}.$$

Thus we a $(n-4) \times (n-4)$ submatrix. In order to keep trace 0, we would have to have the top row of the submatrix out of order. Thus $a_{23} = 0$ and $a_{24} = 1$. That is

$$R_2^{(n)} = [\ 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad \dots \quad 0 \quad 0 \quad 0\ ].$$

In $R_3^{(n)}$ we have $a_{30} = 0, a_{31} = 1, a_{32} = 0, a_{33} = 0$. If $a_{34} = 1$ we would have the same argument as above, with a $(n-5) \times (n-5)$ submatrix. Thus $a_{34} = 0, a_{35} = 1$. So

$$R_3^{(n)} = [\ 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad \dots \quad 0\ ].$$

This pattern holds for $R_k^{(n)}$ for $1 \leq k \leq n-3$. In $R_{n-2}^{(n)}$ we have $a_{n-2,0} = a_{n-1,1} = \dots = a_{n-2,n-5} = 0, a_{n-2,n-4} = 1, a_{n-2,n-3} = a_{n-2,n_2} = 0$. Since we must have a sum of 2, $a_{n-2,n-1} =$

1, so

$$R_{n-2}^{(n)} = [\ 0 \quad 0 \quad 0 \quad \dots \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0\ ].$$

This forces

$$R_{n-1}^{(n)} = [\ 0 \quad 0 \quad 0 \quad 0 \quad \dots \quad \dots \quad 0 \quad 1 \quad 1 \quad 0\ ].$$

So each row only has one possible choice, thus $M_{s,tr(0)}(n,2) = 1$. For example

$$A^{(9)} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

■

Definition 4. $\mathcal{M}_{s,tr(t)}(n,2)$ be the set of symmetric binary matrices with row and column sum 2, trace t, and lexicographical ordered. Let $M_{s,tr(t)}(n,2) = |\mathcal{M}_{s,tr(t)}(n,2)|$

Open Problem 4. Does $M_{s,tr(t)}(n,2)$ satisfy a recurrence relation when we fix t?

4.2 Generalizations

Definition 5. Let $\mathcal{M}(n,s)$ be the set of binary matrices with row and column sum s in lexicographical order.

Example: $\mathcal{M}(5,3) =$

$$\left\{ \begin{bmatrix} 1&1&1&0&0\\ 1&1&1&0&0\\ 1&0&0&1&1\\ 0&1&0&1&1\\ 0&0&1&1&1 \end{bmatrix} \quad \begin{bmatrix} 1&1&1&0&0\\ 1&1&0&1&0\\ 1&1&0&0&1\\ 0&0&1&1&1\\ 0&0&1&1&1 \end{bmatrix} \quad \begin{bmatrix} 1&1&1&0&0\\ 1&1&0&1&0\\ 1&0&1&0&1\\ 0&1&0&1&1\\ 0&0&1&1&1 \end{bmatrix} \quad \begin{bmatrix} 1&1&1&0&0\\ 1&1&0&1&0\\ 1&0&0&1&1\\ 0&1&1&0&1\\ 0&0&1&1&1 \end{bmatrix} \right.$$

$$\begin{bmatrix} 1&1&1&0&0\\ 1&1&0&0&1\\ 1&0&1&1&0\\ 0&1&0&1&1\\ 0&0&1&1&1 \end{bmatrix} \quad \begin{bmatrix} 1&1&1&0&0\\ 1&1&0&0&1\\ 1&0&0&1&1\\ 0&1&1&1&0\\ 0&0&1&1&1 \end{bmatrix} \quad \begin{bmatrix} 1&1&1&0&0\\ 1&0&1&1&0\\ 1&0&1&0&1\\ 0&1&0&1&1\\ 0&1&0&1&1 \end{bmatrix} \quad \begin{bmatrix} 1&1&1&0&0\\ 1&0&1&1&0\\ 1&0&0&1&1\\ 0&1&1&0&1\\ 0&1&0&1&1 \end{bmatrix}$$

$$\begin{bmatrix} 1&1&1&0&0\\ 1&0&1&0&1\\ 1&0&0&1&1\\ 0&1&1&1&0\\ 0&1&0&1&1 \end{bmatrix} \quad \begin{bmatrix} 1&1&1&0&0\\ 1&0&0&1&1\\ 1&0&0&1&1\\ 0&1&1&1&0\\ 0&1&1&0&1 \end{bmatrix} \quad \begin{bmatrix} 1&1&0&1&0\\ 1&1&0&1&0\\ 1&0&1&0&1\\ 0&1&1&0&1\\ 0&0&1&1&1 \end{bmatrix} \quad \begin{bmatrix} 1&1&0&1&0\\ 1&1&0&0&1\\ 1&0&1&1&0\\ 0&1&1&0&1\\ 0&0&1&1&1 \end{bmatrix}$$

$$\begin{bmatrix} 1&1&0&1&0\\ 1&1&0&0&1\\ 1&0&1&0&1\\ 0&1&1&1&0\\ 0&0&1&1&1 \end{bmatrix} \quad \begin{bmatrix} 1&1&0&1&0\\ 1&0&1&1&0\\ 1&0&1&0&1\\ 0&1&1&0&1\\ 0&1&0&1&1 \end{bmatrix} \quad \begin{bmatrix} 1&1&0&1&0\\ 1&0&1&1&0\\ 1&0&0&1&1\\ 0&1&1&0&1\\ 0&1&1&0&1 \end{bmatrix} \quad \begin{bmatrix} 1&1&0&1&0\\ 1&0&1&0&1\\ 1&0&1&0&1\\ 0&1&1&1&0\\ 0&1&0&1&1 \end{bmatrix}$$

$$\left[\begin{array}{ccccc} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{array}\right] \quad \left[\begin{array}{ccccc} 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{array}\right] \quad \left[\begin{array}{ccccc} 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{array}\right] \quad \left[\begin{array}{ccccc} 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccccc} 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{array}\right] \quad \left[\begin{array}{ccccc} 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{array}\right] \Bigg\}$$

Note: the definition of lexicographical order does not change for a general sum.

Open Problem 5. Does $M(n,3)$ satisfy a recurrence relation?

Let $\mathcal{M}_s(n,3)$ be the subset of $\mathcal{M}(n,3)$ such that the matrices are symmetric. Define $M_s(n,3) = |\mathcal{M}_s(n,3)|$. Example: $|M_s(5,3)| = 3$, $\mathcal{M}_s(5,3) =$

$$\Bigg\{ \left[\begin{array}{ccccc} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{array}\right] \quad \left[\begin{array}{ccccc} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{array}\right] \quad \left[\begin{array}{ccccc} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{array}\right] \Bigg\}$$

Open Problem 6. Does $M_s(n,3)$ satisfy a recurrence relation?

We have found for $\mathcal{M}(n,3)$,

n	2	3	4	5	6	7	8	9	10
$M(n,3)$	0	1	1	22	550	16700	703297	38135272	2584332084

We have found for $\mathcal{M}_s(n,3)$,

n	2	3	4	5	6	7	8	9
$M_s(n,3)$	0	1	1	3	11	35	118	419

Chapter 5

Future Work

We have shown

$$M(n,2) = (n-1)M(n-1,2) - \frac{(n-1)(n-2)}{2}M(n-3,2) + (n-1)M(n-2,2),$$

$$S(n) = (n-1)S(n-1) - \frac{(n-1)(n-2)}{2}S(n-3) + (n-1)S(n-2),$$

and that there exists a bijection between $\mathcal{M}(n,2)$ and $\mathcal{S}(n)$. We have also explored the cycle structure of $\mathcal{M}(n,2)$ viewed as incidence matrices and $\mathcal{S}(n)$ viewed as adjacency matrices. Finally we attempted to place further restrictions on $\mathcal{M}(n,2)$ and found $M_s(n,2)$ satisfies the Fibonacci sequence.

There are several open problems we wish to study as future research.

1. How much faster then the current known upper bounds given by Greenhill does the alternating rectangle switch converge?

2. What is the probability that a binary square matrix with row and column sum s is invertible?

3. What is the probability that a matrix in $\mathcal{M}(n,s)$ is invertible?

4. Does $M_{s,tr(t)}(n, 2)$ satisfy a recurrence relation when we fix t?

5. Does $M(n, 3)$ satisfy a recurrence relation?

6. Does $M_s(n, 3)$ satisfy a recurrence relation?

Appendices

Appendix A Java Code

```
import java.util.Random;
//import java.math.BigInteger;

/**
 * This program does the following:
 * Takes a random walk on our set of matrices
 * It can print the results
 * It can give the matrix a distinct number so that we can just record the
 *   number instead of an nxn matrix
 * It can also take a given number and give back the matrix
 *
 *
 * @Janine Janoski
 * @Version 1 10/10/07
 */
public class nbynGraphs
{
    // instance variables
    public static int N; // the size of our matrix
    public static int M;
    // the matrix on which we will do the markov walk
    private final int[][] matrix;
    private Random rand=new Random();
    // create a list of all the possible rows
    private static int[][] possibleRows;
   // this array stores the number of times each row appears
    private static int[] theArray;
    private static int[][] allMat; // all matrices we've found thus far
    // a matrix to store the combinations we
    private static double[][] combMat;
    // have already found
    private static int theRank; // the number associated with a matrix
    private static int[] longVec; // a vector which will allow us to
    // compute the number
    // a matrix with the edges of our graph
    private static int[][] edgeMatrix;
    private static int[][] myMat;
    protected static int[][] adjMatrix;
    protected static boolean[] rowsFound;
```

```
protected String cycle;
protected int pathLength;
protected String start;

/**
 * Constructor for objects of class nbyn
 * Creates an nxn matrix and a vector to store the matrices we find
 * Here we also initial a temp matrix myMat, and a matrix with stores
 * the edges of a given matrix
 */
public nbynGraphs(int N)
{
    int placeHolder=0;
    this.N=N;
    matrix=new int [N][N];
    for (int i=0; i<N; i++){
        for (int j=placeHolder; j<placeHolder+2; j++){
            matrix[i][j]=1;
        }
        if (i % 2 == 1){
            placeHolder=placeHolder+2;
        }
     }
     M=8000;
     edgeMatrix=new int[N][2];
     myMat=new int[N][N];

}

/*
 * Return the matrix
 */
 public int[][] getMatrix() {
    return matrix;
}

  /**
   * Function that will take a 2x2 submatrix and perform the swtich
   *    10 <-> 01
   *    01     10
   */
    public void mySwitch() {
```

```
    int rowIndex1= rand.nextInt(N);
    int rowIndex2= rand.nextInt(N);
    while (rowIndex2==rowIndex1){
        rowIndex2=rand.nextInt(N);
    }
    int colIndex1= rand.nextInt(N);
    int colIndex2= rand.nextInt(N);
    while (colIndex2==colIndex1){
        colIndex2=rand.nextInt(N);
    }

    if (matrix[rowIndex1][colIndex1]==1
    && matrix[rowIndex1][colIndex2]== 0
            && matrix[rowIndex2][colIndex1]==0
            && matrix[rowIndex2][colIndex2]==1)
        {
            matrix[rowIndex1][colIndex1]=0;
            matrix[rowIndex1][colIndex2]= 1;
            matrix[rowIndex2][colIndex1]=1;
            matrix[rowIndex2][colIndex2]=0;
        }
        else if (matrix[rowIndex1][colIndex1]==0
             && matrix[rowIndex1][colIndex2]== 1 &&
            matrix[rowIndex2][colIndex1]==1
            && matrix[rowIndex2][colIndex2]==0)
           {
            matrix[rowIndex1][colIndex1]=1;
            matrix[rowIndex1][colIndex2]= 0;
            matrix[rowIndex2][colIndex1]=0;
            matrix[rowIndex2][colIndex2]=1;
        }

}


/**
 * A function to compute combinations
 */
    public double nChooseR(int n, int r, int isNew)
{
if (r > n){
        return 0;}//BigInteger.ZERO; }
```

```
else if (r ==n || r==0){
        return 1;}//BigInteger.ONE; }
else {
        if (isNew==1){
            return nChooseR(n-1,r,0)+(nChooseR(n-1,r-1,0));
        }
        else {
            return combMat[n-1][r]+combMat[n-1][r-1];
        }
    }

}

/**
 * Creates a matrix which will store all the combinations we will
 * need to derank our matrices
 */
public void nChooseRMat(){
    combMat=new double[N*(N-1)/2+N-1][N*(N-1)/2+N-1];
for (int i=0; i <N*(N-1)/2+N-1; i++){
        for (int j=0; j<(N*(N-1))/2+N-1;j++){
                combMat[i][j]=nChooseR(i,j,1);}}
}


/**
 * Create all the possible rows with a sum of 2
 */
public void createPossibleRows(){
    int index=0;
    int index1=1;
    possibleRows=new int[N*(N-1)/2][N];
    for (int i=0; i < (N*(N-1)/2); i++){
        possibleRows[i][index]=1;
        possibleRows[i][index1++]=1;
        if (index1==N){
            index=index+1;
            index1=index+1;
        }
    }
}
```

```
/**
 * Create an array which stores which rows are in the matrix
 */
public void myArray(){
    theArray= new int[N*(N-1)/2];
    int test;
    int placeHolder;
    int count;
    for (int i=0; i <N; i++){
        test=0;
        placeHolder=-1;
        while (test==0){
            count=0;
            placeHolder=placeHolder+1;
            for (int j=0; j<N; j++){
                if(matrix[i][j]==(possibleRows[placeHolder][j])){
                    count++;
                }
            }
            if (count==N){
                test=1;
            }
        }
        theArray[placeHolder]++;
    }
}

/**
 * Initiallizes a vertor to store the matrices we have found
 */
public void createAllMat(){
    allMat=new int[M][2];
}

/**
 * Creates a unique number related to each matrix. First we insert a 0
 * after every 1into the array that stores the number of each row.Next
 * we number the columns, and from  right to left we compute x choose
 * y, where x is the column index where there is a 1,
 * and y is the number of 1s we haven't seen yet
 */
public int rank(){
```

```
    int myrank=0;
    int numOnes=N;
    int temp=0;
    longVec= new int[N*(N-1)/2+N-1];
    for (int i=0; i<N*(N-1)/2; i++){
        if (theArray[i]==1){
            longVec[temp]=1;
            temp=temp+2;
        }
        else if (theArray[i]==2){
            longVec[temp++]=1;
            longVec[temp++]=1;
        }
        else {
            temp++;
        }
    }


    int colNum=N*(N-1)/2+N-2;
        while (colNum >=0){
            if (longVec[colNum]==1) {
                myrank=myrank+(int)combMat[colNum][numOnes--];
            }

            colNum--;
        }
        theRank=myrank;
        return myrank;
 }


/**
 * Will take the rank and give back the original matrix.
 * This also prints the vector we used to rank, as well as the array
 * that stores which rows are in our matrix
 */
public void derank(){
int n=N*(N-1)/2+N-2;
int m=N         ;
int temp=N*(N-1)/2+N-2, temp2=0;
double choose;
```

```
double vector[]=new double[N*(N-1)/2+N-1];
double shortVec[]=new double[N*(N-1)/2];

while (n >=0) {
    choose=combMat[n][m];
        if (choose <= theRank) {
                vector[temp]=1;
                theRank=theRank-(int)choose;
                n=n-1;
                temp=temp-1;
                m=m-1;
        }
        else {
                n=n-1;
                temp=temp-1;
                }
}

   System.out.printf("The vector is ");
   System.out.println();
 for (int i=0; i <N*(N-1)/2+N-1; i++) {
        System.out.print((int)vector[i]);
        }
        System.out.println();

int ones=0;
 for (int i=0; i < N*(N-1)/2; i++){
         if (vector[temp2]==1){
                 shortVec[i]=1;
                 ones=ones+1;
                         if (ones ==N){
                                 temp2++;
                                 }
                        else if (vector[temp2+1]==1){
                            if (temp2 != N*(N-1)/2+N-3
                            && vector[temp2+2] != 1){
                                shortVec[i]++;
                                ones=ones+1;
                                temp2=temp2+3;
                                i++;
                                }
                             else{
```

```
                                shortVec[i]++;
                                ones++;
                                temp2=temp2+2;
                            }
                        }
                        else {
                                temp2=temp2+2;
                                }}
        else{
                shortVec[i]=0;
                temp2=temp2+1;  }}

    System.out.printf("The short vector is ");
   System.out.println();
 for (int i=0; i < N*(N-1)/2; i++){
         System.out.print((int)shortVec[i]); }
         System.out.println();

temp2=0;
System.out.printf("The matrix is ");
System.out.println();
for (int i=0; i < N*(N-1)/2; i++){
        if (shortVec[i]==1){
                for (int j=0; j < N; j++){
                        myMat[temp2][j]=possibleRows[i][j];
                        System.out.print((int)myMat[temp2][j]);
                        }
                        System.out.println();
                        temp2=temp2+1; }
        else if (shortVec[i]==2){
                for(int k=0; k < 2; k++){
                for(int j=0; j < N; j++){
                        myMat[temp2][j]=possibleRows[i][j];
                       System.out.print((int)myMat[temp2][j]);
                    }
                        System.out.println();
                        temp2=temp2+1;
                }}}}

                /**
                 * Creates a matrix that will tell which vertices
                 * are connected  by an edge
```

```
              */
public void toEdgeMatrix(){
      int position;
      for (int j=0; j< N; j++){
          position=0;
          for (int i=0; i<N; i++){
              if (myMat[i][j]==1){
                  edgeMatrix[j][position]=i;
                  System.out.print((int)edgeMatrix[j][position]);
                  position++;
                  if (position==2){
                      break; }
              }

          }
          System.out.println();
      }

   rowsFound= new boolean[N];
      for (int i=0; i<N; i++){
          rowsFound[i]=false;
      }
  }

  /**
   * Create an adjacency matrix from our list of edges
   */
  public void createAdjMat(){
      adjMatrix= new int[N][N];
      for (int i=0; i < N; i++){
          adjMatrix[i][i]=0;
      }
      for (int i=0; i<N; i++) {
          int start=edgeMatrix[i][0];
          int end = edgeMatrix[i][1];
          adjMatrix[start][end]++;
          adjMatrix[end][start]++;


      }
      for (int k=0; k < N; k++){
          for (int j=0; j<N; j++){
```

```
                System.out.print(adjMatrix[k][j]+"   ");
            }
            System.out.println();
        }
}

public String getNodeLabel(int a){
        if (a==0)
            return "A";
        else if (a==1)
            return "B";
        else if (a==2)
            return "C";
        else if (a==3)
            return "D";
        else if (a==4)
            return "E";
        else if (a==5)
            return "F";
        else if (a==6)
            return "G";
        else if (a==7)
            return "H";
        else if (a==8)
            return "I";
        else if (a==9)
            return "J";
        else if (a==10)
            return "K";
        else if (a==11)
            return "L";
        else
            return "M";
}

/**
 * Search for cycles
 */
    public void findCycles(){
       for (int j=0; j<N; j++){
            if (rowsFound[j]==false){
                pathLength=0;
```

```
            findCycles(j,j);
        }
    }

}

    /**
     * Search for cycles in our adjacency matrix
     */
public void findCycles(int j, int previous){
    pathLength++;
    if (pathLength==1){
        start=getNodeLabel(j);
        cycle=getNodeLabel(j);
    }

    boolean pathComplete=false;
    rowsFound[j]=true;
    while (pathComplete == false){
        for (int k=0; k<N;k++){
            if (adjMatrix[j][k]==1 && k != previous){
                cycle=cycle+getNodeLabel(k);
                rowsFound[k]=true;
                pathLength++;
                previous=j;
                 j=k;
                if (getNodeLabel(k)==start){
                    pathComplete=true;
                    System.out.println(cycle);
                    System.out.println(--pathLength);
                }
                break;
            }
            else if (adjMatrix[j][k]==2){
                rowsFound[k]=true;
                pathComplete=true;
                System.out.println("There is a
                repeated row in "+getNodeLabel(j)
                +" and "+getNodeLabel(k));
                j=k;
                break;
            }
```

```
            }
        }

    }


/**
  * Main Program that runs the Marcov chain
  */
     public static void main(String [] args){
     N=6;
     //M=500;
     int temp=0;
     nbynGraphs mat= new nbynGraphs(N);

     mat.createPossibleRows();
     mat.nChooseRMat();
     mat.createAllMat();


     for (int l=0; l < 10; l++){ //600000 for 8by8
        for (int i=0; i < 1000; i++){
            mat.mySwitch();
         }

         int[][] matrix=mat.getMatrix();

            mat.myArray();


 /*
  * If we have no doubles derank and look at the cycles
  * This code can be used if we wish to partition into two sets, 1
  * with repeats and one without repeated rows
  */
         double rank=mat.rank();
         for (int k=0; k<=temp; k++){
             if (allMat[k][0]==rank){
                 allMat[k][1]++;
                 break;
```

```
                }
                else if (rank<allMat[k][0]){
                    for (int p=temp-1; p>=k; p--){
                        allMat[p+1][0]=allMat[p][0];
                        allMat[p+1][1]=allMat[p][1];
                    }
                    allMat[k][0]=(int)rank;
                    allMat[k][1]=1;
                }
                else if (allMat[k][0]==0){
                    allMat[k][0]=(int)rank;
                    allMat[k][1]=1;
                    System.out.printf("The rank is ");
                    System.out.println((int)rank);
                    mat.derank();
                    mat.toEdgeMatrix();
                    mat.createAdjMat();
                    mat.findCycles();
                    temp++;
                    break;
                }
            }

            if (temp==5000){
                break;
            }



        }
        System.out.println("Total matrix count");
        System.out.println(temp);

    }


}


/**
 * findAllUs will find all of our matrices in M(n,2)
 *
 * @author Janine Janoski
```

```
 * @version October 20, 2007
 */
public class findAllUS
{
    private static int[][] possibleRows;
    private static int N;
    protected static boolean[][] rowsUsed;
    protected static int count;
    protected static int[][] permMat;
    protected static int[] colSum;


    /**
     * Create all the possible rows with a sum of 2
     */
    public static void createPossibleRows(){
        int index=0;
        int index1=1;
        possibleRows=new int[N*(N-1)/2][N];
        for (int i=0; i < (N*(N-1)/2); i++){
            possibleRows[i][index]=1;
            possibleRows[i][index1++]=1;
            if (index1==N){
                index=index+1;
                index1=index+1;
            }
        }
    }

    /*
     * We start by fixing the first row in our matrix. We then will
     * call a program which recursively calls the find the rest of
     *the matrix
     */
    public static void permInc(){
        int[][] thePermMat;
        for (int i=0; i<N*(N-1)/2; i++){
                rowsUsed[i][0]=true;
                for (int j=0; j<N; j++){
                    permMat[0][j]=possibleRows[i][j];
                    if (permMat[0][j]==1){
                        colSum[j]++;
```

```
                }
            }

            recurPermInc( 1, i);
            rowsUsed[i][0]=false;
            for (int k=0; k<N; k++){
                if (permMat[0][k]==1){
                    colSum[k]--;
                    permMat[0][k]=0;
                }

            }

    }
}

// recursively find the matrix
public static void recurPermInc( int rowNum, int startRow){
    // base case
    boolean done=true;
    int count1=0;

    // first we must check if we are done forming a "good" matrix
    for (int i=0; i< N*(N-1)/2; i++){
        if (rowsUsed[i][0]==true){
            count1++;
        }
       if (rowsUsed[i][1]==true){
            count1++;
        }
    }

    if (count1 != N){
        done=false;
    }

    // if we have found a good matrix we want to print it
    if (done ==true){
        for (int i=0; i<N; i++){
            for (int j=0; j<N;j++){
                System.out.print(permMat[i][j]);
            }
```

```
            System.out.println();
        }
        System.out.println();
        count++;
    }

    // if we do not have a good matrix keep searching for possible rows
    else{
        for (int i=startRow; i<N*(N-1)/2; i++){
            // if we haven't used the row twice yet
            if (rowsUsed[i][0]==false || rowsUsed[i][1]==false){
                boolean test=true;
                for (int k=0; k<N; k++){
                    // if we are trying to put a row in where the column
                    // sum is already 2, we can't add it
                    if (possibleRows[i][k] == 1 && colSum[k] == 2){
                        test=false;
                        break;
                    }
                }
                // if its possible to add the rows, we will try
                if (test==true){
                    int coordinate=0;
                    if (rowsUsed[i][0]==false){
                        rowsUsed[i][0]=true;
                    }
                    else {
                        rowsUsed[i][1]=true;
                        coordinate=1;
                    }

                    for (int k=0; k<N; k++){
                        if (possibleRows[i][k]==1){
                            permMat[rowNum][k]=possibleRows[i][k];
                            colSum[k]++;
                        }
                        //permMat[k][rowNum]=possibleRows[i][k];
                    }
                    recurPermInc(rowNum+1, i);
                    rowsUsed[i][coordinate]=false;
                    for (int k=0; k<N; k++){
                        if (permMat[rowNum][k]==1){
```

```
                                permMat[rowNum][k]=0;
                                colSum[k]--;
                             }
                            // permMat[k][rowNum]=0;
                        }
                    }
                }
            }
        }


     public static void main(String[] args){
         N=Integer.parseInt(args[0]);
         possibleRows= new int[N*(N-1)/2][N];
         rowsUsed=new boolean[N*(N-1)/2][2];
         permMat=new int[N][N];
         colSum=new int[N];

         createPossibleRows();

         for (int i=0; i<N; i++){
             colSum[i]=0;
            }

         for (int i=0; i<N*(N-1)/2; i++){
             for (int j=0; j<2; j++){
                 rowsUsed[i][j]=false;
            }
        }

         permInc();

       System.out.println("The number of matrices is: "+count);
    }
}


/**
 * Write a description of class findAll here.
 *
 * @author (your name)
```

```
 * @version (a version number or a date)
 */
public class findAllThem
{
    private static int[][] possibleRows;
    private static int N;
    protected static boolean[][] rowsUsed;
    protected static int count;
    protected static int[][] permMat;
    public static boolean[] rowsFound;
    protected static String cycle;
    protected static String start;
    protected static int pathLength;
    protected static String cycle1;

    /**
     * Create all the possible rows with a sum of 2
     */
    public static void createPossibleRows(){
        int index=0;
        int index1=1;
        possibleRows=new int[N*(N-1)/2+N][N];
        for (int i=0; i < (N*(N-1)/2+N); i++){
            if (i >= N*(N-1)/2){
                possibleRows[i][i-N*(N-1)/2]=2;
            }
            else {
                possibleRows[i][index]=1;
                possibleRows[i][index1++]=1;
                if (index1==N){
                    index=index+1;
                    index1=index+1;
                }
            }
        }
    }

    public static void permInc(){
        int[][] thePermMat;
        for (int i=0; i<N*(N-1)/2+N; i++){
            if (possibleRows[i][0]==0){
                rowsUsed[i][0]=true;
```

```
            for (int j=0; j<N; j++){
                permMat[0][j]=possibleRows[i][j];
                permMat[j][0]=possibleRows[i][j];
            }

            recurPermInc(/*thePermMat[][],*/ 1);
            rowsUsed[i][0]=false;
            for (int k=0; k<N; k++){
                permMat[0][k]=0;
                permMat[k][0]=0;
            }
        }
    }
}

public static void recurPermInc(int rowNum){
    // base case
    boolean done=true;
    int count1=0;
    for (int i=0; i< N*(N-1)/2+N; i++){
        if (rowsUsed[i][0]==true){
            count1++;
        }
        if (rowsUsed[i][1]==true){
            count1++;
        }
    }
    if (count1 != N){
        done=false;
    }
    if (done ==true){
        for (int i=0; i<N; i++){
            for (int j=0; j<N;j++){
                System.out.print(permMat[i][j]);
            }
            System.out.println();
        }
        findCycles();
        System.out.println();
        count++;
    }
    else{
```

```
        // boolean allFalse=true;
         for (int i=0; i<N*(N-1)/2+N; i++){
             if (rowsUsed[i][0]==false || rowsUsed[i][1]==false){
                 boolean test=true;
                 for (int k=0; k<= rowNum; k++){
                     if (possibleRows[i][k] != permMat[rowNum][k]){
                         test=false;
                         break;
                     }
                 }
                 if (test==true){
                     int coordinate=0;
                     if (rowsUsed[i][0]==false){
                         rowsUsed[i][0]=true;
                     }
                     else {
                         rowsUsed[i][1]=true;
                         coordinate=1;
                     }

                     for (int k=rowNum+1; k<N; k++){
                         permMat[rowNum][k]=possibleRows[i][k];
                         permMat[k][rowNum]=possibleRows[i][k];
                     }
                     recurPermInc(rowNum+1);
                     rowsUsed[i][coordinate]=false;
                     for (int k=rowNum+1; k<N; k++){
                         permMat[rowNum][k]=0;
                         permMat[k][rowNum]=0;
                     }
                 }
             }
         }
    }
}

public static String getNodeLabel(int a){
         if (a==0)
             return "A";
         else if (a==1)
             return "B";
         else if (a==2)
```

```
            return "C";
        else if (a==3)
            return "D";
        else if (a==4)
            return "E";
        else if (a==5)
            return "F";
        else if (a==6)
            return "G";
        else if (a==7)
            return "H";
        else if (a==8)
            return "I";
        else if (a==9)
            return "J";
        else if (a==10)
            return "K";
        else if (a==11)
            return "L";
        else if (a==12)
            return "M";
        else
            return "N";
}

/**
 * Search for cycles
 */
    public static void findCycles(){
        rowsFound= new boolean[N];
        for (int i=0; i<N; i++){
            rowsFound[i]=false;
        }
       for (int j=0; j<N; j++){
            if (rowsFound[j]==false){
                pathLength=0;
                findCycles(j,j);
            }
        }

    }
```

```
    /**
     * Search for cycles in our adjacency matrix
     */
public static void findCycles(int j, int previous){
    pathLength++;
    if (pathLength==1){
        start=getNodeLabel(j);
        cycle=getNodeLabel(j);
    }

    boolean pathComplete=false;
    rowsFound[j]=true;
    while (pathComplete == false){
        for (int k=0; k<N;k++){
            if (permMat[j][k]==1 && k != previous){
                cycle=cycle+getNodeLabel(k);
                rowsFound[k]=true;
                pathLength++;
                previous=j;
                 j=k;
                if (getNodeLabel(k)==start){
                    pathComplete=true;
                    System.out.println(cycle);
                    cycle1=cycle;
                    System.out.println(--pathLength);
                }
                break;
            }
            else if (permMat[j][k]==2){
                rowsFound[k]=true;
                pathComplete=true;
                System.out.println("There is a repeated row in "
                +getNodeLabel(j)+" and "+getNodeLabel(k));
                j=k;
                break;
            }

        }
    }

}
```

```
     public static void main(String[] args){
         N=Integer.parseInt(args[0]);
         possibleRows= new int[N*(N-1)/2+N][N];
         rowsUsed=new boolean[N*(N-1)/2+N][2];
         permMat=new int[N][N];

         createPossibleRows();

         for (int i=0; i<N*(N-1)/2; i++){
             for (int j=0; j<2; j++){
                 rowsUsed[i][j]=false;
            }
        }

        permInc();

       System.out.println("The number of different permuted matrices is:
       "+count);
    }
}


/**
 * This function will generate all matrices of M(n,2) and map them to
 * S(n)
 *
 * @author (Janine Janoski)
 * @version (2/25/08)
 */
public class MtoS
{
    private static int[][] possibleRows;
    private static int N;
    protected static boolean[][] rowsUsed;
    protected static int count;
    protected static int[][] permMat;
    protected static int[] colSum;
    protected static int[][] permMatMinus1;
    protected static int[][] theirMat;
    protected static int[][] rowPairs;
    protected static int[] temp;
```

```
/**
 * Create all the possible rows with a sum of 2
 */
public static void createPossibleRows(){
    int index=0;
    int index1=1;
    possibleRows=new int[N*(N-1)/2][N];
    for (int i=0; i < (N*(N-1)/2); i++){
        possibleRows[i][index]=1;
        possibleRows[i][index1++]=1;
        if (index1==N){
            index=index+1;
            index1=index+1;
        }
    }
}

/*
 * We start by fixing the first row in our matrix. We then will
 * call a program which recursively calls the find the rest
 * of the matrix
 */
public static void permInc(){
    int[][] thePermMat;
    for (int i=0; i<N*(N-1)/2; i++){
            rowsUsed[i][0]=true;
            for (int j=0; j<N; j++){
                permMat[0][j]=possibleRows[i][j];
                if (permMat[0][j]==1){
                    colSum[j]++;
                }
            }

            recurPermInc( 1, i);
            rowsUsed[i][0]=false;
            for (int k=0; k<N; k++){
                if (permMat[0][k]==1){
                    colSum[k]--;
                    permMat[0][k]=0;
                }

            }
```

```
    }
}

// recursively find the matrix
public static void recurPermInc( int rowNum, int startRow){
    // base case
    boolean done=true;
    int count1=0;

    // first we must check if we are done forming a "good" matrix
    for (int i=0; i< N*(N-1)/2; i++){
        if (rowsUsed[i][0]==true){
            count1++;
        }
       if (rowsUsed[i][1]==true){
            count1++;
        }
    }

    if (count1 != N){
        done=false;
    }

    // if we have found a good matrix we want to print it
    if (done ==true){
        for (int i=0; i<N; i++){
            for (int j=0; j<N;j++){
                System.out.print(permMat[i][j]);
            }
            System.out.println();
        }
        System.out.println();
        count++;

        // We now wish to map this good matrix to one of theirs
        mapMtoS();
    }

    // if we do not have a good matrix keep searching for possible rows
    else{
        for (int i=startRow; i<N*(N-1)/2; i++){
```

```
            // if we haven't used the row twice yet
            if (rowsUsed[i][0]==false || rowsUsed[i][1]==false){
                boolean test=true;
                for (int k=0; k<N; k++){
                    // if we are trying to put a row in where the column
                    // sum is already 2, we can't add it
                    if (possibleRows[i][k] == 1 && colSum[k] == 2){
                        test=false;
                        break;
                    }
                }
                // if its possible to add the rows, we will try
                if (test==true){
                    int coordinate=0;
                    if (rowsUsed[i][0]==false){
                        rowsUsed[i][0]=true;
                    }
                    else {
                        rowsUsed[i][1]=true;
                        coordinate=1;
                    }

                    for (int k=0; k<N; k++){
                        if (possibleRows[i][k]==1){
                            permMat[rowNum][k]=possibleRows[i][k];
                            colSum[k]++;
                        }
                        //permMat[k][rowNum]=possibleRows[i][k];
                    }
                    recurPermInc(rowNum+1, i);
                    rowsUsed[i][coordinate]=false;
                    for (int k=0; k<N; k++){
                        if (permMat[rowNum][k]==1){
                            permMat[rowNum][k]=0;
                            colSum[k]--;
                        }
                       // permMat[k][rowNum]=0;
                    }
                }
            }
        }
    }
```

```
}

public static void mapMtoS(){
        int myCount=0;
        int myRow;
        int test1;
        boolean twoBytwo=false;
        boolean minus1=false;
        for (int i=0; i<N; i++){
            for (int j=0; j<N;j++){
                permMatMinus1[i][j]=permMat[i][j];
            }
        }

        // Check to see if we have a repeated row. If there is a repeated
        // row we are in the n-2 case
        // If it is not a repeated row we are in the n-1 case.
        for (int i=0; i<N-3; i++){
            minus1=false;
            for (int j=i; j<N; j++){
                if (permMatMinus1[i][j] != permMatMinus1[i+1][j]){
                    minus1=true;
                    break;
                }

            }


            if (minus1==true){
                test1=1;
                for (int j=i+1; j<N; j++){
                    if (permMatMinus1[i][j]==1){
                        permMatMinus1[i+1][j]=1;
                        break;
                    }
                }
                for (int j=i+1;j<N;j++){
                    if (permMatMinus1[i+1][j]==1){
                        if (test1%2==1){
                            rowPairs[0][myCount]=j;
                            test1++;
                        }
```

```
        else{
            rowPairs[1][myCount++]=j;
            break;
        }

     }
  }


// reorder the rows if nessicary
boolean test=false;
for (int l=i+1; l<N-1; l++){
    for (int j=i+1; j<N; j++){
        if (permMatMinus1[l][j] != permMatMinus1[l+1][j]){
            if (permMatMinus1[l][j]==1
            && permMatMinus1[l+1][j]==0){
                test=true;
            }
            else
            for (int m=i; m<N; m++){
                temp[m]=permMatMinus1[l][m];
                permMatMinus1[l][m]=permMatMinus1[l+1][m];
                permMatMinus1[l+1][m]=temp[m];
            }
        break;
        }
    }
    if (test==true){
        break;
    }

    }
}
else {
    if (i==(N-4)){
        twoBytwo=true;
    }
    for (int j=i+1; j<N; j++){
        if (permMatMinus1[i][j]==1){
            rowPairs[0][myCount]=N+1;
            rowPairs[1][myCount++]=j;
```

```
                for (int k=i; k<N;k++){
                    temp[k]=permMatMinus1[k][j];
                }
                for (int m=j; m>i; m--){
                    for (int k=i; k<N;k++){
       permMatMinus1[k][m]=permMatMinus1[k][m-1];
                    }
                }
                for (int k=i; k<N;k++){
                    permMatMinus1[k][i]=temp[k];
                }
                break;
            }
        }

    i++;
    }
}

for (int i=0; i<N;i++){
    for (int j=0; j<N; j++){
        theirMat[i][j]=0;
    }
}
if (twoBytwo==false){
    theirMat[N-1][N-3]=1;
    theirMat[N-1][N-2]=1;
    theirMat[N-1][N-1]=0;
    theirMat[N-2][N-3]=1;
    theirMat[N-2][N-2]=0;
    theirMat[N-2][N-1]=1;
    theirMat[N-3][N-3]=0;
    theirMat[N-3][N-2]=1;
    theirMat[N-3][N-1]=1;
    myRow=N-4;
}
else{
    theirMat[N-1][N-1]=0;
    theirMat[N-1][N-2]=2;
    theirMat[N-2][N-1]=2;
    theirMat[N-2][N-2]=0;
    myRow=N-3;
```

```
}

myCount--;
while (myRow > -1){
    if (rowPairs[0][myCount]==N+1){
           for (int i=myRow-1; i<N;i++){
               if (i==rowPairs[1][myCount]){
                   theirMat[myRow-1][i]=2;
                   theirMat[i][myRow-1]=2;
                }
                else{
                    theirMat[myRow-1][i]=0;
                    theirMat[i][myRow-1]=0;
                    //theirMat[rowPairs[1][myCount]][i]=0;
                    //theirMat[i][rowPairs[1][myCount]]=0;
                }
            }
            for (int i=myRow; i<N; i++){
                for (int j=myRow; j<N; j++){
                    if (j < rowPairs[1][myCount]
                    && i< rowPairs[1][myCount]){
                        theirMat[i][j]=theirMat[i+1][j+1];
                    }
                    else if (j>rowPairs[1][myCount]
                    && i<rowPairs[1][myCount]){
                        theirMat[i][j]=theirMat[i+1][j];
                    }
                    else if (j<rowPairs[1][myCount]
                    && i>rowPairs[1][myCount]){
                        theirMat[i][j]=theirMat[i][j+1];
                    }
                    else if (j==rowPairs[1][myCount]
                    || i==rowPairs[1][myCount]){
                        theirMat[i][j]=0;
                    }
                }
            }
            myRow=myRow-2;
            myCount--;
    }
    else{
        for (int i=myRow; i<N; i++){
```

```
            for (int j=myRow; j<N; j++){
                if(i==myRow && (j==rowPairs[0][myCount]/*+1*/
                || j==rowPairs[1][myCount]/*+1*/)){
                    theirMat[i][j]=1;
                }
                else if(i==myRow){
                    theirMat[i][j]=0;
                }
                else if((i==rowPairs[0][myCount]/*+1*/
                ||i==rowPairs[1][myCount]/*+1*/)&& j==myRow){
                    theirMat[i][j]=1;
                }
                else if(j==myRow){
                    theirMat[i][j]=0;
                }
                else if((i==rowPairs[0][myCount]/*+1*/
                && j==rowPairs[1][myCount]/*+1*/)||
                        (i==rowPairs[1][myCount]/*+1*/
                        && j==rowPairs[0][myCount]/*+1*/)){
                        if (theirMat[i][j]==2){
                            theirMat[i][j]=1;
                        }
                        else{
                            theirMat[i][j]=0;
                        }
                }

            }
        }
       myRow--;
       myCount--;
    }
}

System.out.println("Maps to: ");
for(int i=0; i<N; i++){
    for (int j=0; j<N;j++){
        System.out.print(theirMat[i][j]);
    }
    System.out.println();
}
System.out.println();
```

```
            }


          public static void main(String[] args){
          N=Integer.parseInt(args[0]);
          possibleRows= new int[N*(N-1)/2][N];
          rowsUsed=new boolean[N*(N-1)/2][2];
          permMat=new int[N][N];
          colSum=new int[N];
             permMatMinus1= new int[N][N];
             theirMat= new int[N][N];
             temp=new int [N];
             rowPairs=new int[2][N];

          createPossibleRows();

          for (int i=0; i<N; i++){
              colSum[i]=0;
             }

          for (int i=0; i<N*(N-1)/2; i++){
              for (int j=0; j<2; j++){
                  rowsUsed[i][j]=false;
             }
         }

         permInc();
     }
}

import java.math.BigInteger;

/**
 * This program finds all matrices in M(n,3)
 *
 * @author (Janine Janoski)
 * @version 1
 */
public class findAllSum3Us
{
    private static int[][] possibleRows;
    private static int N;
```

```
protected static boolean[][] rowsUsed;
protected static BigInteger count;
protected static int[][] permMat;
protected static int[] colSum;

/**
 * Create all the possible rows with a sum of 2
 */
public static void createPossibleRows(){
    int index=0;
    int index1=1;
    int index2=2;
    possibleRows=new int[N*(N-1)*(N-2)/6][N];
    for (int i=0; i < (N*(N-1)*(N-2)/6); i++){
        possibleRows[i][index]=1;
        possibleRows[i][index1]=1;
        possibleRows[i][index2++]=1;
        if (index2==N && index1 != N-2){
            index1=index1+1;
            index2=index1+1;
        }
        else if (index1== N-2){
            index=index+1;
            index1=index+1;
            index2=index1+1;
        }
    }
}

    /*
 * We start by fixing the first row in our matrix. We then will
 * call a program which recursively calls the find the rest of
 * the matrix
 */
public static void permInc(){
    int[][] thePermMat;
    for (int i=0; i<N*(N-1)*(N-2)/6; i++){
            rowsUsed[i][0]=true;
            for (int j=0; j<N; j++){
                permMat[0][j]=possibleRows[i][j];
                if (permMat[0][j]==1){
                    colSum[j]++;
```

```
                }
            }

            recurPermInc( 1, i);
            rowsUsed[i][0]=false;
            for (int k=0; k<N; k++){
                if (permMat[0][k]==1){
                    colSum[k]--;
                    permMat[0][k]=0;
                }

            }

    }
}


public static void AddOneToBigInt(){
    count=count.add(BigInteger.ONE);
}

// recursively find the matrix
public static void recurPermInc( int rowNum, int startRow){
    // base case
    boolean done=true;
    int count1=0;
    //count=BigInteger.ONE;

    // first we must check if we are done forming a "good" matrix
    for (int i=0; i< N*(N-1)*(N-2)/6; i++){
        if (rowsUsed[i][0]==true){
            count1++;
        }
       if (rowsUsed[i][1]==true){
            count1++;
        }
        if (rowsUsed[i][2]==true){
            count1++;
        }

    }
```

```
if (count1 != N){
    done=false;
}

// if we have found a good matrix we want to print it
if (done ==true){
   for (int i=0; i<N; i++){
       if (i==0){
           System.out.print("\\left [ \\begin{array}{ccccc}");
        }
        for (int j=0; j<N;j++){
            System.out.print(permMat[i][j]);
            if (j != N-1){
                System.out.print("&");
            }
            else{
                System.out.print("\\\\");
            }
        }
        System.out.println();
        if (i==N-1){
            System.out.print("\\end{array} \\right ]");
        }

   }
    System.out.println();
    AddOneToBigInt();
   // count=count.add(BigInteger.ONE);
}

// if we do not have a good matrix keep searching for possible rows
// to add
else{
    for (int i=startRow; i<N*(N-1)*(N-2)/6; i++){
        // if we haven't used the row three times yet
        if (rowsUsed[i][0]==false || rowsUsed[i][1]==false
         || rowsUsed[i][2]==false){
            boolean test=true;
            for (int k=0; k<N; k++){
                // if we are trying to put a row in where the column
                // sum is already 2, we can't add it
                if (possibleRows[i][k] == 1 && colSum[k] == 3){
```

```
                        test=false;
                        break;
                    }
                }
                // if its possible to add the rows, we will try
                if (test==true){
                    int coordinate=0;
                    if (rowsUsed[i][0]==false){
                        rowsUsed[i][0]=true;
                    }
                    else if(rowsUsed[i][1]==false){
                        rowsUsed[i][1]=true;
                        coordinate=1;
                    }
                    else {
                        rowsUsed[i][2]=true;
                        coordinate=2;
                    }

                    for (int k=0; k<N; k++){
                        if (possibleRows[i][k]==1){
                            permMat[rowNum][k]=possibleRows[i][k];
                            colSum[k]++;
                        }
                        //permMat[k][rowNum]=possibleRows[i][k];
                    }
                    recurPermInc(rowNum+1, i);
                    rowsUsed[i][coordinate]=false;
                    for (int k=0; k<N; k++){
                        if (permMat[rowNum][k]==1){
                            permMat[rowNum][k]=0;
                            colSum[k]--;
                        }
                       // permMat[k][rowNum]=0;
                    }
                }
            }
        }
    }
}
```

```
    public static void main(String[] args){
        N=Integer.parseInt(args[0]);
        possibleRows= new int[N*(N-1)*(N-2)/6][N];
        rowsUsed=new boolean[N*(N-1)*(N-2)/6][3];
        permMat=new int[N][N];
        colSum=new int[N];
        count=BigInteger.ZERO;

        createPossibleRows();

        for (int i=0; i<N; i++){
            colSum[i]=0;
           }

        for (int i=0; i<N*(N-1)*(N-2)/6; i++){
            for (int j=0; j<3; j++){
                rowsUsed[i][j]=false;
           }
       }

       permInc();

      System.out.println("The number of matrices is: "+count);
   }
}
```

Appendix B C++ Code

The program below will find the coefficients for $C(n, j, l)$. It will also print $C(n, 0, 0) = 2^n M(n, 2)$.

```
/* Janine Janoski
Clemson University
9/10/07
Program to run the recursion
*/

#include<iostream>
#include<fstream>
#include<cstdlib>
#include<cmath>

using namespace std;
const int N=20;
unsigned long long recursion(int n, int j, int l);

int main() {
/*
unsigned int recursionTable[N+1][N+1];
for (int n=0; n<N; n++){
for (int j=0; j <N; j++){
        for (int l=0; l<N; l++){
                if (l+j > n){
                        break; }
                recursionTable[j][l]=recursion(n,j,l);
                cout<<n<<"  "<<j<<"  "<< l<<"  "
                << recursionTable[j][l]<<endl;

}
cout<< endl;
}
}
*/

unsigned long long r;
unsigned long long mat[N][N];

for (int n=0; n<N; n++){
```

```
        cout<< n<< "   "<< recursion(n,0,0)<< endl; }
return 0;
}

unsigned long long  recursion(int n, int j, int l){
if (j < 0 || l < 0 || n<0 || n==0){
        return 0;}
else if (n==1 && ((j==1 && l == 0) || (l==1&& j ==0 ))){
        return 1;
}
else if (n==1){
        return 0;
}
else
        return recursion(n-1,j-1,l)
        +recursion(n-1,j,l-1)+4*j*recursion(n-1,j,l)
        +2*(j+1)*(2*j+1)*recursion(n-1,j+1,l)
        +2*(l+1)*recursion(n-1,j,l+1);
}
```

Appendix C Condor Code

The following program is the necessary class ad to run findSum3 program on Condor. findSum3 is a java program that was compiled on a windows machine.

```
universe=vanilla
requirements = OpSys=="WINNT51" && Arch=="INTEL"
should_transfer_files=YES
transfer_files=ALWAYS
transfer_input_files=findAllSum3Us.class
when_to_transfer_output=ON_EXIT
executable=./findSum3.bat
arguments=findAllSum3Us $(Process)
output=./Output/findSum3.out.$(Process)
error=./Output/findSum3.err.$(Process)
log=./Output/findSum3.log.$(Process)
notification=NEVER
queue 20
```

The following program is the batch file, which is used in the class ad above. This tells the class ad where to find java on the machines across campus.

```
set path=%path%;c:\PROGRA~1\Java\jdk1.5.0_04\bin\

java %*
```

Bibliography

[1] A. C. Aitken. On the number of distinct terms in the expansion of symmetric and skew determinants. *Edinburgh Math. Notes*, 34:1–5, 1944.

[2] H. Anand, V.C. Dumir, and H. Gupta. A combinatorial distribution problem. *Duke Math J.*, 33:757–770, 1966.

[3] Suraj Bandyopadhyay, Rabindranath Jana, and A. Ramachandra Rao. A markov chain monte carlo method for generating random (0-1)-matrices with given marginals. *The Indian Journal of Statistics*, 58:225–242, 1996.

[4] Alexander Barvinok. Low rank approximations of symmetric polynomials and asymptotic counting of contingency tables. arXiv:math/0503170v1, march 2005.

[5] A. Bekessy, P Bekessy, and J. Komlos. Asymptotic enumeration of regular matrices. *Studia Sci. Math. Hungar.*, 7:343–353, 1972.

[6] Edward A. Bender. The asymptotic number of nonnegative integer matrices with given row and column sums. *Discrete Mathematics*, 10:217–223, 1974.

[7] E. Rodney Canfield and Brendan D. McKay. Asymptotic enumeration of contingency tables with constant margins. arXiv:math/0703600v1, March 2007.

[8] I. M. H. Etherington. Some problems of non-associative combinations. *Edinburgh Math. Notes*, 32:1–6, 1940.

[9] C.J. Everett and P.R. Stein. The asymptotic number of integer stochastic matrices. *Discrete Mathematics*, 1(1):55–72, 1971.

[10] Mitchell Gail and Nathan Mantel. Counting the number of $r \times c$ contingency tables with fixed margins. *Journal of the American Statistical Association*, 72(360):859–862, December 1977.

[11] Shanzhen Gao and Zhonghua Tan. Some (0,1)-matrices enumerative problems. *Congressus Numerantium*, 185:209–219, 2007.

[12] Catherine Greenhill and Brendan D. McKay. Asymptotic enumeration of dense 0-1 matrices with specified line sums and forbidden positions. arXiv:0701600v1, January 2007.

[13] Catherine Greenhill and Brendan D. McKay. Asymptotic enumeration with sparse nonnegative integer matrices with specified row and column sums. arXiv:0707.0340v2, February 2008.

[14] P. A. MacMahon. Combinations derived from m identical sets of n different letters and their connexion with general magic squares. *Proc. London Math. Soc.*, 17:25–41, 1917.

[15] P.A. MacMahon. Combinatorial analysis. the foundation of a new theory. *Philos. Trans. Roy. Soc. London. Ser. A*, 194:361–386, 1900.

[16] Ken Matheis, Gao Shanzhen, and Tan Zhonghua. Some formulas of (0,1)-matrices. *Congressus Numerantium*, 182:53–63, 2006.

[17] Ken Matheis, Gao Shanzhen, and Tan Zhonghua. Enumeration of nonnegative integer matrices. *Congressus Numerantium*, 184:193–207, 2007.

[18] Brendan D. McKay and Xiaoji Wang. Asymptotic enumeration of 0-1 matrices with equal row sums and equal column sums. *Linear Algebra and its Applications*, 373:272–287, 2003.

[19] Heinrich Niederhausen, Gao Shanzhen, and Tan Zhonghua. Enumeration of (0,1)-matrices with constant row and column sums. *Appl. Math J. Chinese Univ. Ser. B*, 24(4):479–486, 2006.

[20] R. C. Reed. *Some Enumeration Problems in Graph Theory*. PhD thesis, University of London, 1958.

[21] H.J. Ryser. Combinatorial mathematics. *Carus Mathematical Monogrpahs*, 1963.

[22] Gao Shanzhen and Tan Zhonghua. (0,1)-matrices with constant row and column sums. *Congressus Numerantium*, 177:3–13, 2005.

[23] Richard P. Stanley. *Combinatorics and Commutative Algebra*. Birkhauser, 1983.

[24] Richard P. Stanley. *Enumerative Combinatorics*, volume 2. Cambridge, 1999.